FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Interpretable Recurrent Neural Networks in Continuous-time Control Environments

PhD THESIS

submitted in partial fulfillment of the requirements for the degree of

Doctor of Technical Sciences

within the

Vienna PhD School of Informatics

by

Ramin Hasani
Registration Number 01529745

to the Faculty of Informatics

at the TU Wien

Advisor: Radu Grosu, Technische Universität Wien (TU Wien)
Second advisor: Daniela Rus - Massachusetts Institute of Technology (MIT)
Industry advisor: Dieter Haerle - Kompetenzzentrum Automobil- und Industrieelektronik (KAI)

External reviewers:
Nando De Freitas. Google DeepMind, UK.
Sriram Sankaranarayanan. University of Colorado Boulder, USA.

Vienna, 10th April, 2020

Ramin Hasani

Radu Grosu

Declaration of Authorship

Ramin Hasani

I hereby declare that I have written this Doctoral Thesis independently, that I have completely specified the utilized sources and resources and that I have definitely marked all parts of the work - including tables, maps and figures - which belong to other works or to the internet, literally or extracted, by referencing the source as borrowed.

Vienna, 10th April, 2020

Ramin Hasani

Acknowledgements

I am fortunate that I have been surrounded by many outstanding individuals during my PhD studies. I would like to express my deepest gratitude to my TU-Wien supervisor Prof. Radu Grosu, for his unconditional support and for teaching me priceless principles at work, research and life. I am genuinely thankful of my MIT supervisor Prof. Daniela Rus who has generously provided me with invaluable resources and guidance to learn to play an *Infinite Game*. I express my deepest appreciation to my Infineon supervisor Dipl.-ing. Dieter Haerle, for the continuous support, his patience, motivation and for teaching me work principles and discipline.

I sincerely thank Prof. Thomas Henzinger (IST Austria), Prof. Manuel Zimmer (University of Vienna), Prof. Alessio Lomuscio (Imperial College London), Prof. Ezio Bartocci (TU Wien), Prof. Giorgio Ferrari (Politecnico di Milano), and Dr. Enrico Prati (Italian National Research Council), for our great collaborations and for teaching me how to expand my interdisciplinary research perspectives. I also would like to thank the reviewers of my Thesis Prof. Sriram Sankaranarayanan (University of Colorado) and Prof. Nando De Freitas (Google DeepMind), for their valuable insights and comments.

I thank my dearest friend, colleague and business partner, Guodong Wang, for being a true friend and for the fun we had during the last four years. The most sincere appreciation to my friend, outstanding colleague and business partner, Mathias Lechner, for thousands of hours of research collaborations, without whom this PhD dissertation would not have been possible. I express my profound gratitude to my friend and outstanding colleague, Alexander Amini for our great collaborations during the last years.

I thank all my current and past lab members and collaborators at TU-Wien CPS group with whom my journey was even more fun; Haris Isakovich (bromigo), Denise Ratasich, Christian Hirsch (bromigo), Michael Platzer, Sophie Grünbacher, Prof. Peter Puschner, Gerda Belkhofer, Viktoria Vasalik, Leo Mayerhofer, Magdalena Fuchs, David Lung (masterlung), Ondrej Balun, Marc Javan (son), Julian Posch, Zahra Babaiee, Lukas Esterle and Max Tschaikowski. I also thank my wonderful colleagues and friends at the Distributed Robotics Lab at CSAIL MIT: Mieke Moran, Felix Naser (bromigo), Brandon Araki, Igor Gilitschenski, Joseph DelPreto, Stephanie Gil, Andres Salazar-Gomez and Lucas Liebenwein. I thank the Institute of Computer Engineering group members, special gratitude to Prof. Wolfgang Kasner, Prof. Ulrich Schmidt, and Prof. Andreas Steininger. I would also thank Ms. Barbara Wiesböck from the administration team of the Informatik faculty of TU Wien for the great support in the submission process of my thesis. I thank Stephen Larson and the entire OpenWorm Foundation members, for the exciting collaborations. I would like to acknowledge organizations that provided financial resources throughout this PhD dissertation: EU Horizon-2020 ECSEL Project grant No. 783163 (iDev40), the Austrian Research Promotion Agency (FFG), Project No. 860424, TU-Wien Informatik Department, MIT Computer Science and Artificial Intelligence Lab (CSAIL), Infineon Technologies, the Boeing Company, Microsoft Azure for Research, and Google Cloud Platform.

I want to thank my exceptional friends for the time we had; Annika Nichols from whom I learned courage, passion and motivation, Harris Kaplan for helping me distinguish between *the signal and the noise*, Victoria Beneder for the unique partnership and inspiration, my life-long best friends, Aryan Chalay Amoly, Mohamad Bakharzy, Ehsan Hoseini, Amir Farifteh, Jean-Vicente De Carvalho, examples of excellence in work and life, for their continuous support, fabulous friendships, and being my unique source of inspiration.

An unconditional respect and gratitude to my hero, dearest Bettina Schlager who was a source of light in rough times and for making me feel complete, and most of all, I want to express my genuine respect and appreciation to my exceptional parents, and to my brothers, Amir and Saleh, for their unconditional support. I love you all.

Abstract

Intelligent agents must learn coherent representations of their world, from high-dimensional sensory information, and utilize them to generalize well in unseen situations. Although contemporary deep learning algorithms have achieved noteworthy successes in variform of high-dimensional tasks, their learned causal structure, interpretability, and robustness were largely overlooked. This dissertation presents methods to address interpretation, stability and the overlooked properties of a class of intelligent algorithms, namely recurrent neural networks (RNNs), in continuous-time environments. Accordingly, the contributions of the work lie into two major frameworks:

I) Designing interpretable RNN architectures — We first introduce a novel RNN instance that is formulated by computational models originally developed to explain the nervous system of small species. We call these RNNs liquid time-constant (LTCs) because they possess nonlinear compartments that regulate the state of a neuron through a variable time-constant. LTCs form a dynamic causal model capable of learning causal relationships between the input, their neural state, and the output dynamics directly from supervised training data. Moreover, we demonstrate that LTCs are universal approximators and can be advantageously used in continuous-time control domains. We then combine LTCs with contemporary scalable deep neural network architectures and structural inspirations from the *C. elegans* connectome, to develop novel neural processing units, that can learn to map multidimensional inputs to control commands by sparse, causal, interpretable and robust neural representations. We extensively evaluate the performance of LTC-based neural network instances in a large category of simulated and real-world applications ranging from time-series classification and prediction to autonomous robot and vehicle control.

II) Designing interpretation methods for trained RNN instances — In this framework, we develop a quantitative method to interpret the dynamics of modern RNN architectures. As opposed to the existing methods that are proactively constructed by empirical feature visualization algorithms, we propose a systematic pipeline for interpreting individual hidden state dynamics within the network using response characterization methods. Our method is able to uniquely identify neurons with insightful dynamics, quantify relationships between dynamical properties and test accuracy through ablation analysis, and interpret the impact of network capacity on a network's dynamical distribution. Finally, we demonstrate the scalability of our method by evaluating a series of different benchmark sequential datasets.

The findings of this dissertation notably improves our understanding of neural information processing systems in continuous-time environments.

Contents

Abstract vii

Relationship to published Work xv

1 Introduction 1
- 1.1 Motivation and Problem Statement . . . 1
  - 1.1.1 What do we mean by Interpretability? . . . 2
  - 1.1.2 Why and when do we need Interpretability? . . . 2
  - 1.1.3 How to interpret? . . . 3
  - 1.1.4 Why interpretability of neural networks, specifically RNNs is difficult? 4
- 1.2 Research Questions and Thesis Contributions . . . 5
  - 1.2.1 Designing interpretable neural network architectures . . . 6
  - 1.2.2 Designing interpretation methods for trained neural networks . . . 7
  - 1.2.3 Summary of Contributions . . . 8

2 Background 11
- 2.1 Math Definitions . . . 11
- 2.2 Supervised Learning . . . 11
- 2.3 Optimization . . . 12
- 2.4 Feedforward Neural Networks . . . 13
- 2.5 Universal Approximation Capabilities of Neural Networks . . . 13
- 2.6 Recurrent Neural Networks . . . 14
  - 2.6.1 Standard RNNs . . . 15
  - 2.6.2 RNNs are Difficult to train . . . 15
  - 2.6.3 RNNs with Skip connections . . . 16
    - Time-Delayed Neural Networks . . . 16
    - Nonlinear Auto-regressive Network with Exogenous Input . . . 16
    - Recurrent Identity Networks . . . 17
  - 2.6.4 Long Short-term Memory . . . 18
- 2.7 Dynamical Systems . . . 19
  - 2.7.1 Ordinary Differential Equations . . . 19
    - Autonomous ODE . . . 19
    - Linear ODE . . . 19

Homogeneous ODE . . . 19
Nonlinear ODE . . . 20
2.8 Time-Continuous Networks . . . 20
2.8.1 Neural ODEs . . . 20
2.8.2 Continuous-time RNNs . . . 20
2.8.3 Stability of CT-RNNs in Learning Systems . . . 21
Learning from Demonstration . . . 22
Learning stable Dynamical Systems . . . 23
RNNs for Modeling Dynamical Systems . . . 23
Learning Stable Linear Dynamical Systems . . . 24
LDS as a Continuous-Time RNNs . . . 25
2.8.4 Biologically-inspired Networks . . . 26

3 Liquid Time-Constant Recurrent Neural Networks 29
3.1 Motivation . . . 29
3.2 Defining LTC's Semantics . . . 31
3.2.1 Perceptron Models . . . 31
3.2.2 Integrator Models . . . 31
3.2.3 Membrane Models . . . 33
3.3 LTCs: Liquid Time-constant Recurrent Neural Networks . . . 33
3.4 LTCs and DCMs . . . 37
3.5 LTCs are Universal Approximators . . . 38
3.5.1 Proof of the Theorem 4 . . . 40
3.6 Bounds on the time-constant and the neural state of LTCs . . . 44
3.7 A learning platform for training LTCs by gradient descent . . . 46
3.7.1 LTC cell-update by a Hybrid ODE Solver . . . 46
3.7.2 Training LTC networks by backpropagation through time . . . 49
3.8 LTCs performance in time-series prediction compared to other RNNs . . . 50
3.9 Experimental Setup . . . 51
3.9.1 Gesture segmentation task . . . 51
3.9.2 Room occupancy detection . . . 52
3.9.3 Human activity recognition . . . 52
3.9.4 Traffic volume prediction . . . 52
3.9.5 Ozone level forecasting . . . 52
3.9.6 Model selection procedure . . . 53
3.9.7 Model size . . . 53
3.9.8 ODE solvers setting . . . 53
3.10 Extrapolation with LTCs . . . 53
3.11 Conclusions . . . 55

4 Ordinary Neural Circuits with LTCs 57
4.1 Motivation . . . 57
4.1.1 TW graph realizes the highest maximum flow rate . . . 58
4.1.2 TW can be trained to govern standard control tasks . . . 59

4.1.3 Contributions of this chapter . . . 60
4.2 Design Ordinary Neural Circuits . . . 61
4.2.1 Tap-Withdrawal Neural Circuit . . . 61
4.2.2 Maximum Flow Rate in ONCs versus Other Networks . . . 61
4.3 Sensory inputs and Motor outputs for LTCs . . . 62
4.4 Search-based Optimization Algorithm . . . 63
4.5 Experiments . . . 64
4.5.1 How to map ONCs to environments? . . . 65
4.5.2 Scale the functionality of ONCs to environments with larger observation spaces . . . 66
4.5.3 Transfer learned ONCs to control real robot . . . 66
4.6 Experimental Evaluation . . . 67
4.6.1 Do ONCs perform better than equivalent random circuits? . . . 67
4.6.2 How does ONC + random search compares with policy gradient based RL algorithms? . . . 68
4.6.3 How does ONC compare to deep learning models? . . . 68
4.6.4 Interpretability of the ordinary neural circuits . . . 68
4.7 Conclusions . . . 69

5 Rule-based Design of LTC Networks for Interpretable Robot Control 71
5.1 Motivation . . . 71
5.2 Related Works . . . 73
5.3 Network Simulation Environment . . . 73
5.4 Design Operators . . . 74
5.5 Design a DO-based Neural Network . . . 76
5.6 Synaptic Parametrization . . . 78
5.7 Autonomous Parking of a Mobile Robot . . . 78
5.8 Manipulating a Robotic Arm . . . 79
5.9 Experimental evaluation . . . 81
5.9.1 Emergence of complex dynamics from compact networks . . . 81
5.9.2 DO-based networks are interpretable . . . 81
5.9.3 DO-based networks are highly resilient to noise . . . 82
5.10 Conclusions and Discussions . . . 82

6 Learning High-Fidelity Autonomous Driving agents by LTCs 85
6.1 Motivation . . . 85
6.2 Design and Learn Neural Circuit Policies . . . 87
6.2.1 Design NCPs . . . 89
6.2.2 Numerical implementation of the NCP networks . . . 89
6.2.3 Training Procedure . . . 90
6.3 Experimental setup . . . 92
6.3.1 Vehicle setup . . . 92
6.3.2 Passive test dataset . . . 92
6.3.3 Active test setup. . . . 93

6.4 Results . . . 95
6.4.1 Learning a compact neural representation" . . . 95
6.4.2 Divergence from the road by the increasing input noise . . . 96
6.4.3 Robustness of the output decisions in the presence of input noise . . . 96
Computing saliency maps of convolutional layers . . . 96
Structural Similarity Index . . . 97
6.4.4 Driving with smooth neural activity . . . 97
Lipschitz Continuity Computation . . . 97
6.4.5 Comparing learned causal structures . . . 98
6.4.6 Global network dynamics . . . 98
PCA . . . 98
6.5 Conclusions . . . 101

7 Interpretability of Recurrent Neural Networks 105
7.1 Motivation . . . 105
7.2 Related Works . . . 106
7.3 Dynamics of Recurrent Neural Networks . . . 108
7.4 Methodology for Response Characterization of LSTM cells . . . 108
7.5 Experimental Results . . . 110
7.5.1 Response characterization metrics predict insightful dynamics for individual cells . . . 111
7.5.2 Generalization of response metrics to other sequential datasets . . . 112
7.5.3 Prediction of cell importance using response metrics . . . 114
7.5.4 Network-level Interpretability for Trained LSTMs . . . 115
7.6 Conclusion . . . 115

8 Designing Interpretable RNNs for modeling Analog Integrated Circuits 117
8.1 Motivation . . . 117
8.2 CompNN for MIMO system modeling . . . 118
8.3 Narx Neural-network architecture . . . 119
8.4 Training process and network performance . . . 122
8.5 Recomposition function: A time-delayed neural network layer . . . 126
8.6 Co-simulation of Matlab/Simulink models and analog design environment . . . 127
8.7 Conclusions . . . 127

9 Conclusions 131
9.1 Summary Notes on Chapter 2 . . . 131
9.2 Summary Notes on Chapter 3 . . . 132
9.3 Summary Notes on Chapter 4 and 5 . . . 134
9.4 Summary Notes on Chapter 6 . . . 135
9.5 Summary Notes on Chapter 7 and 8 . . . 136
9.6 Future Directions . . . 136

List of Figures 139

List of Tables **141**

Bibliography **143**

Relationship to published Work

The thesis describes work that has been published in the following venues:

Discussions related to the novel methods for enhancing the stability of continuous-time RNNs, in Chapter 2, are built over the following published work:

- Gershgorin Loss Stabilizes the Recurrent Neural Network Compartment of an End-to-end Robot Learning Scheme
  Mathias Lechner*, **Ramin Hasani***, Daniela Rus, and Radu Grosu. *equal contributions. Accepted at the *IEEE International Conference on Robotics and Automation* (ICRA), 2020

Discussions related to the biological recurrent neural network models in Chapters 2 to 6 are built over the following published work:

- SIM-CE: An Advanced Simulation Platform for Studying the brain of C. elegans
  Ramin Hasani, Victoria Beneder, Magdalena Fuchs, David Lung, and Radu Grosu. *Workshop on Computational Biology* (WCB), 34^{th} *International Conference on Machine Learning* (ICML), 2017
- Modeling a Simple Non-Associative Learning Mechanism in the Brain of C. elegans
  Ramin Hasani, Magdalena Fuchs, Victoria Beneder, Radu Grosu. 2^{nd} *International Workshop on Biomedical Informatics with Optimization and Machine Learning* (BOOM), *In conjunction with* 26^{th} *International Joint Conference on Artificial Intelligence* (IJCAI), 2017.
- Towards Deterministic and Stochastic Computations with Izhikevich Spiking Neuron Model
  Ramin Hasani, Guodong Wang, and Radu Grosu. 14^{th} *International Work-Conference on Artificial Neural Networks* (IWANN), Springer, 2017
- Computing with Biophysical and Hardware-efficient Neural Models
  Konstantin Selyunin, **Ramin Hasani**, Denise Ratasich, Ezio Bartocci, and Radu Grosu. 14^{th} *International Work-Conference on Artificial Neural Networks* (IWANN), Springer, 2017
- c302: a multiscale framework for modelling the nervous system of C. elegans
  Padraig Gleeson, David Lung, Radu Grosu, **Ramin Hasani**, Stephen Larson *Journal of Philosophical Transactions of Royal Society B*, 2018

Chapter 3 is built over the under-revision work:

- Liquid Time-constant Recurrent Neural Networks
  Ramin Hasani*, Mathias Lechner*, Alexander Amini, Daniela Rus, Radu Grosu. *equal contributions. under revision, 2020

Chapter 4 is built over the following published work:

- Worm-level Control through Search-based Reinforcement Learning
  Mathias Lechner, Radu Grosu, **Ramin Hasani**. *Deep Reinforcement Learning Symposium at the* 31^{st} *Neural Information Processing Systems* (NIPS), 2017

Chapter 5 is built over the following published work:

- Designing Worm-inspired Neural Networks for Interpretable Robotics Control
  Mathias Lechner*,**Ramin Hasani***, Manuel Zimmer, Thomas Henzinger, Radu Grosu. *equal contributions, *IEEE International Conference on Robotics and Automation* (ICRA), 2019

Chapter 6 is built over the following under revision work:

- Learning High-fidelity Autonomous Driving Agents by Neural Circuit Policies
  Mathias Lechner*,**Ramin Hasani***, Thomas Henzinger, Daniela Rus, Radu Grosu. *equal contributions. under revision, 2020

Chaspter 7 is built over the following published work:

- Response characterization for auditing cell dynamics in long short-term memory networks
  Ramin Hasani*, Alexander Amini*, Mathias Lechner, Felix Naser, Radu Grosu, Daniela Rus. *equal contributions, *IEEE International Joint Conference on Neural Networks* (IJCNN), 2019

Chapter 8 is built over the following published work:

- Compositional Neural-Network Modeling of Complex Analog Circuits
  Ramin Hasani, Dieter Haerle, Christian F. Baumgartner, Alessio R. Lomuscio and Radu Grosu. *30th International Joint Conference on Neural Networks* (IJCNN), IEEE, 2017
- Efficient Modeling of Complex Analog Integrated Circuits Using Neural Networks
  Ramin Hasani, Dieter Haerle, and Radu Grosu. *12th Conference on PhD Research in Microelectronics and Electronics* (PRIME), IEEE, 2016

Here, I include a list of publications that are loosely connected to the content of the thesis, and are not included:

- Plug-and-Play Supervisory Control Using Muscle and Brain Signals for Real-Time Gesture and Error Detection

  Joseph DelPreto, Andres F. Salazar-Gomez, Stephanie Gil, **Ramin Hasani**, Frank H. Guenther, Daniela Rus *14th Robotics: Science and Systems* (RSS) Conference, Pittsburg, USA, 2018

- OpenWorm: overview and recent advances in integrative biological simulation of C. elegans

  Gopal Sarma, Chee Wai Lee, Tom Portegys, Vahid Ghayoomie, Travis Jacobs, Bradly Alicea, Matteo Cantarelli, Michael Currie, Richard Gerkin, Shane Gingell, Padraig Gleeson, Richard Gordon, **Ramin Hasani**, Giovanni Idili, Sergey Khayrulin, David Lung, Andrey Palyanov, Mark Watts, Stephen Larson *Phil. Trans. Royal Society B*, 2018

- A Machine Learning Suite for Machine Components' Health-Monitoring

  Ramin Hasani, Guodong Wnag, Radu Grosu. *Proceedings of the AAAI Conference on Artificial Intelligence*, 2019

- A Simplified Cell Network for the Simulation of C. elegans' Forward Crawling

  David Lung, Stephen Larson, Andrey Palyanov, Sergey Khayrulin, Padraig Gleeson, Manuel Zimmer, Radu Grosu and **Ramin Hasani**. *Workshop on Worm's Neural Information Processing* (WNIP) *at the 31st Neural Information Processing Systems* (NIPS), 2017

- Searching for Biophysically Realistic Parameters for Dynamic Neuron Models by Genetic Algorithms from Calcium Imaging Recording

  Magdalena Fuchs, Manuel Zimmer, Radu Grosu and **Ramin Hasani**. *Workshop on Worm's Neural Information Processing* (WNIP) *at the 31st Neural Information Processing Systems* (NIPS), 2017

- A generative neural network model for the quality prediction of work in progress products

  Guodong Wang, Anna Ledwoch, **Ramin Hasani**, Radu Grosu, Alexandra Brintrup. *Applied Soft Computing*, 2019

- Probabilistic Reachability Analysis of the Tap-Withdrawal Circuit in C. elegans

  Ariful Islam, Qinsi Wang, **Ramin Hasani**, Ondrej Balun, Edmund M. Clarke, Radu Grosu, and Scott A. Smolka. *18th IEEE International High Level Design Validation and Test Workshop* (HLDVT), 2016

CHAPTER 1

Introduction

Deep learning algorithms have demonstrated outstanding performance in variety of representation learning applications and end-to-end information processing of high-dimensional spaces [Mnih et al., 2015, Silver et al., 2016b, Bojarski et al., 2016, Silver et al., 2017, Schrittwieser et al., 2019, Vinyals et al., 2019]. Despite mastering the ability to learn representations with *supreme performances*, the study of a series of critical attributes of such learning systems was proportionally, largely overlooked. For instance, how a neural network agent comes up with decisions? Can we explain/*interpret* its internal state? Has the true *causal structure* of the task been learned by the agent? are the learned representations *fair*? And how *robust* are such networks under environmental perturbations? Questions of such become more vital where the domain under which the learning system operates, is safety-critical such as controlling robots and autonomous driving. This dissertation aims to investigate the interpretability, stability, and robustness of a class of machine learning algorithms that are spatiotemporal information processing systems, namely recurrent neural networks (RNNs).

1.1 Motivation and Problem Statement

RNNs are artificial neural networks that maintain the history of their input data as an *internal state vector*. This property makes them highly expressive, and as a result, enables them to achieve non-trivial performance on complex sequential tasks.

However, interpretation of the dynamics of the internal state of RNNs similar to many other modern neural network architectures is difficult [Melis and Jaakkola, 2018, Karpathy et al., 2015]. This mainly raises concerns when the RNN agent is deployed in safety-critical application domains such as robot control [Brooks, 1986, Mayer et al., 2008, Latombe, 2012a, Duan et al., 2016, Heess et al., 2017], in which at every stage of agent's autonomy (perception [Elfes, 1989, Beetz et al., 2015], reasoning [Georgeff and Lansky, 1987, Bekey, 2005], and control [Levine et al., 2016]), their underlying mechanisms and structure ideally have to be interpretable to ensure the safety of the system. First, let us define our notion of interpretability:

1.1.1 What do we mean by Interpretability?

Interpretation is the process of providing explanations to human. There is no formal and mathematical definition for interpretability [Molnar, 2019]. However, here we argue that a model is better interpretable than another if its internal dynamics, as well as its decision-making process and its output predictions, are more comprehensible to humans. To delve more rigorously into defining interpretability, we can take advantage of many well-defined machine learning desiderata.

Causality – considers how changes in parameters and elements of the agent (i.e., a neural network's building blocks) alter its decision making process [Pearl, 2009]. Causality has been subjected to rigorous formalization in the research community. When intelligent agents establish a causal structure, our understanding of their dynamics significantly increases. In Chapter 3, we discuss this property for recurrent neural networks, quantitatively.

Fairness – implies that the decisions made by an intelligent agent are not biased and are independent of a selected group of sensitive features (i.e., gender, ethnicity, and image backgrounds) [Hardt et al., 2016, Barocas et al., 2017]. Fairness in machine learning is well-characterized and is closely discussed together with accountability and transparency [Hardt, 2020]. In Chapter 6, we discuss how to design neural processing units which perform *fair* decision making.

Robustness and Reliability – ascertain the functionality of the agent under input or parameters perturbations [Doshi-Velez and Kim, 2017]. In Chapters 4, 5, 6, we perform a series of robustness analysis of intelligent agents in safety critical domains such as robotics and autonomous driving.

Usability – provides users with information eases their task accomplishment process [Doshi-Velez and Kim, 2017]. For instance, how a neural network architecture has to be designed enhances the usability of the machine learning model. In Chapters 5 and 6 we introduce task-specific design principles to build expressive and performant neural network architectures.

Trust – enhances user's confidence in the deployment process of the intelligent agent (e.g., in lane-keeping with self-driving cars) [Huang et al., 2017]. Any form of stability and convergence guarantees for a learning system raises trust. Moreover, trust can be achieved by methodologies that result in a better understanding of the underlying dynamics of the learning system. In Chapters 2 to 7, we develop theoretical, experimental, and quantitative methods to elevate our understanding of a recurrent neural network system to enhance trust.

Conclusively, **we determine *interpretability* as the method that investigates whether discussed machine learning desiderata such as causality, fairness, robustness, reliability, usability, and trust are achieved.** Throughout this thesis, we use interpretability and auditability interchangeably.

1.1.2 Why and when do we need Interpretability?

[Doshi-Velez and Kim, 2017] stated that interpretability is required when any form of *incompleteness* is present in the problem's formalization, optimization, or in the evaluation process. For instance, an end-to-end learning system cannot be tested for all possible risk-element combinations. This means that it is computationally infeasible to assess the output of the system for all possible inputs. Therefore, interpretability could help in *safety* analysis of a learning system.

Furthermore, the selection of decision criteria and inductive biases cannot be done completely for a learning system. For instance, certain kinds of discrimination (such as gender or ethnicity) towards users should be avoided during a social study. However, the selection of such criteria is non-trivial in the general case. Therefore, interpretability can be helpful in *ethical* and fairness analysis of algorithms.

Mismatched objectives can also be improved by the use of interpretability methods [Doshi-Velez and Kim, 2017]. For instance, a learning algorithm with an ill-defined (incomplete) objective can give rise to locally optimal agents. For example, an end-to-end lane-keeping self-driving agent can learn to drive based on the side roads and off-road features, while it ignores a road's horizon. We discuss this phenomenon further in Chapter 6.

Moreover, performance-robustness trade-off always exists for a decision making algorithm. Where one requires to maximize this trade-off, interpretability can be helpful. We discuss this matter further in Chapters 4,5, and 6.

1.1.3 How to interpret?

Interpretability methods have largely been categorized into two major groups: 1. Human-in-the-loop methods, 2. Proxy Methods. The first category includes machine learning models with their application domain being determined by a human expert, such as a doctor diagnosing diseases, or designing an integrated circuit. An ML model, based on human knowledge, is created; thus, its dynamics can be interpreted by the human himself. Several attempts on human-in-the-loop interpretability have been conducted [Suissa-Peleg et al., 2016, Williams et al., 2016, Dong et al., 2017, Lage et al., 2018, Zanzotto, 2019]. While this approach demonstrated promise in task-specific problems, their deployment in large-scale machine learning models such as neural networks is highly laborious and challenging.

The second category, which is scalable to significantly larger machine learning models, focuses on the development of algorithms that interpret a learning system's dynamics by means of a quality measure (proxy). In particular, for the interpretability of neural network models, a large body of work focused on post-training feature visualization to qualitatively understand the dynamics of the neural networks [Erhan et al., 2009, Zeiler and Fergus, 2014, Yosinski et al., 2015, Karpathy et al., 2015, Strobelt et al., 2018, Bilal et al., 2018, Olah et al., 2018]. Alternatively, evaluating input-feature attributions by computing saliency maps [Simonyan et al., 2013, Fong and Vedaldi, 2017, Kindermans et al., 2017, Sundararajan et al., 2017], were effectively deployed for the interpretability of neural networks.

Beyond feature visualization, the choice of the proxy and measures can be arbitrary and challenging [Doshi-Velez and Kim, 2017]. For instance, dimensionality reduction methods [Bishop and Tipping, 1998, Gulrajani et al., 2016, Maaten and Hinton, 2008], recursive dynamical patterns inference inside a network [Strobelt et al., 2018], and robust statistics [Koh and Liang, 2017] were proposed for post-training interpretation of the network dynamics.

More systematic interpretability approaches suggested the design of neural network architectures that, either throughout their optimization process or by nature, are more auditable. Examples

include neural arithmetic logic units (NALU) [Trask et al., 2018] and the attention networks [Vaswani et al., 2017]. However, fundamentally, as the dimensionality of the neural network model increases our understanding of the kinetics of the network's elements drops. Therefore, the interpretability of large-scale machine learning models such as deep neural networks is difficult. This challenge becomes even more severe when the network architectures possess feedback mechanisms such as recurrent neural networks, which are the predominant choice in sequential data processing tasks.

1.1.4 Why interpretability of neural networks, specifically RNNs is difficult?

The difficulty in interpreting the dynamics of deep learning architectures, and more specifically, RNNs, the core models of the current study, have the following fundamental reasons:

1. *Network's size* (number of neurons, synapses, and trainable parameters) – As the network size grows, understanding individual neurons' contribution to the output decision becomes challenging and, in many cases, irrelevant. It has been recently shown that there exist sub-networks within a fully-connected large-scale neural network, that when trained in isolation, achieves comparable performance to that of the original network [Frankle and Carbin, 2018]. This finding suggests that knowledge representation distributed within a dense network more likely arises from groups of neurons in arbitrary-configurations rather than single neurons.
2. *Input features are typically of high-dimensions* (i.e., a sequence of images) – In many application domains such as robotic navigation, autonomous cars, and medicine, the input domain consists of high-dimensional image data realizing an enormous space. Each input dimension (each pixel) maps to the nodes of the first layer of the system. Making sense of the resulting connectome, at the pixel-level, becomes computationally hard and irrelevant [Melis and Jaakkola, 2018].
3. In addition to the difficulty of interpretation of individual input-features, *the parameters of a neural network are highly interleaved* to those high-dimensional features. Therefore their contributions to a network decision cannot be explicitly quantified.
4. *The aggregation of the input features and the network's parameters* are highly nonlinear with conflating of interpretation of impact [Melis and Jaakkola, 2018, Tan et al., 2018]. Therefore, local explanations become non-trivially difficult to extract.
5. *RNN's feedback loop* results in higher-order dynamics. This feature appends another degree of complexity to its internal dynamics interpretation.

There is a high demand for approaching interpretability of neural networks, specially RNNs, which are actively deployed in safety-critical domains. In the present study, we take two primary paths to rigorously understand and interpret the dynamics of recurrent neural network architectures in continuous-time domains: 1) Designing interpretable network architectures. 2) Designing interpretation methods for trained neural networks.

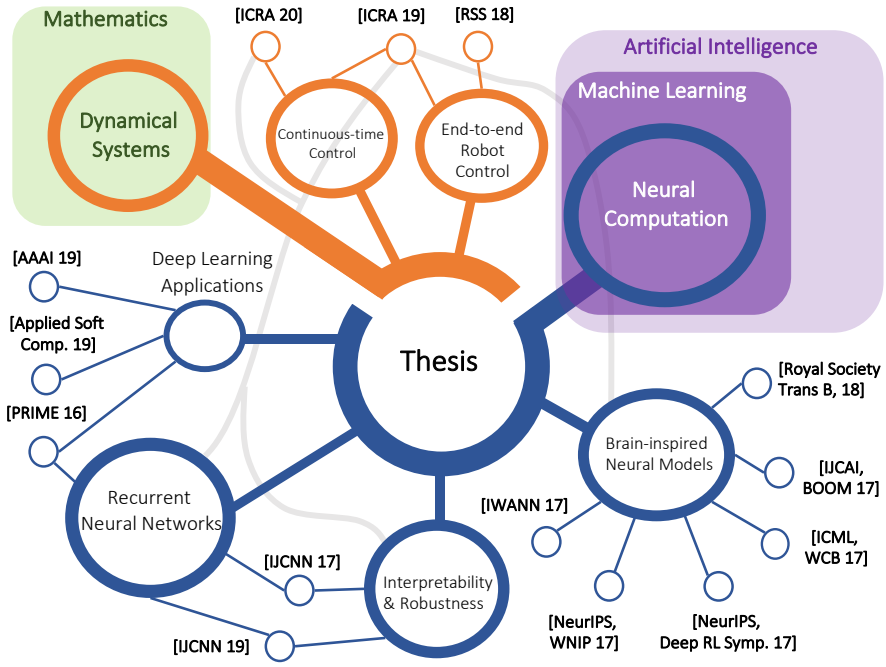


Figure 1.1: Thesis topics and contributions map.

1.2 Research Questions and Thesis Contributions

We aim to take a rigorous computational approach to improve the interpretability challenges of recurrent neural networks, stated in the previous section. The diagram presented in Fig. 1.1, graphically illustrates the structure of the thesis contributions and its relations to the published work.

Our Vision — is to improve our understanding of how RNNs come up with their decisions in continuous sequential data-processing environments, such as robotics. For this purpose, we aim to approach the interpretability of RNNs by designing novel architectures that satisfy many of the machine learning desiderata such as robustness and reliability, usability, causality, and trust. Furthermore, we develop techniques to shed more light on the internal dynamics of contemporary RNN architectures.

Technical insights for getting to the vision — We set out to combine insights from dynamical systems and neural computations, into designing novel neural information processing systems with enhanced interpretability skills, in continuous-time domains. To achieve this, we divide the work into two main frameworks: 1) Designing interpretable network architectures. 2) Designing interpretation methods for trained neural networks.

1.2.1 Designing interpretable neural network architectures

One can design novel architectures equipped with regularization schemes, or computational elements, that naturally address interpretability. The attention mechanism [Vaswani et al., 2017] introduced a novel network architecture that enhances the interpretation of a network's decision-making process. It forces the network to attend to specific cues in the input stream when assigning labels. RNNs equipped with an attention mechanism have been successfully applied in image captioning [You et al., 2016], the fine-alignment in machine translation [Luong et al., 2015], and text extraction from documents [Hermann et al., 2015]. Hidden-state visualization is a shared property of these approaches in order to understand the internal dynamics of the network effectively.

In this thesis, we aim to take a neuroscience-inspired approach to design more transparent learning systems. The architecture of many contemporary neural information processing systems is inspired by what we know about neural computations in biological systems [Hassabis et al., 2017]. Their semantics, however, have diverged dramatically from their biologically plausible roots to reduce complexity and enable scalability. This, however, reduces their ability to express the range of complex dynamical properties of real neural and synaptic models. In this thesis, we aim to bridge this gap by showing that a biophysical neural model, originally developed to explain the nervous system of small species such as *Ascaris* and *C. elegans* can be advantageously used as a performant recurrent neural network (RNN) instance. We call these RNNs liquid time-constant (LTCs) because they possess, like their biological counterparts, nonlinear compartments that regulate the state of a neuron through a variable time-constant. LTCs form a dynamic causal model [Friston et al., 2003] capable of learning causal relationships between the input, their neural state, and the output dynamics directly from supervised training data. In Chapter 3, we show that LTCs also possess universal approximation capabilities as they can learn arbitrary input/output mappings of a given dynamical system with any precision. Moreover, we show that their dynamics are bounded to a finite range and demonstrate LTC's superior performance in a series of real-life time-series prediction and classification tasks compared to other RNN architectures.

We then design neural computational units by the LTC model, in control of the safety-critical domains. More specifically, in Chapter 4 we design an LTC-based neural network, which is obtained by *re-purposing* the function of a biological neural circuit model to govern simulated and real-world control tasks. Inspired by the structure of the nervous system of the soil-worm, *C. elegans*, we introduce *ordinary neural circuits* (ONCs), defined as the model of biological neural circuits reparameterized for the control of alternative tasks. We first demonstrate that ONCs realize networks with higher maximum flow compared to arbitrary wired networks. We then learn instances of ONCs to control a series of robotic tasks, including the autonomous parking of a real-world rover robot. For reconfiguration of the *purpose* of the neural circuit, we adopt a search-based optimization algorithm. Ordinary neural circuits perform on par and, in some cases, significantly surpass the performance of contemporary deep learning models. ONC networks are compact, 77% sparser than their counterpart neural controllers, and their neural dynamics are fully interpretable at the cell-level.

We then introduce an ad-hoc network design algorithm to develop networks of LTC neurons for a particular control task; in Chapter 5, we identify neuron-pair communication motifs as *design operators* and use them to configure compact LTC-based neuronal network structures to govern sequential robotic tasks. The networks are systematically designed to map the environmental observations to motor actions, by their hierarchical topology from sensory neurons, through recurrently-wired interneurons, to motor neurons. The networks are then parametrized in a supervised-learning scheme by a search-based algorithm. We demonstrate that obtained networks realize interpretable dynamics. We evaluate their performance in controlling mobile and arm robots and compare their attributes to other artificial neural network-based control agents. Finally, we experimentally show their superior resiliency to environmental noise, compared to that of existing machine learning methods.

The success of the DO-based on rather simple robotic applications described in Chapter 5, motivated us to expand the use of the LTC-based model into complex high-dimensional environments such as autonomous driving.

In Chapter 6 we combine the LTC-based neural models, scalable deep neural network architectures citelecun1989backpropagation,funahashi1993approximation,chen2018neural, and structural inspirations from the *C. elegans* connectome, to develop a novel neural processing unit, termed a *neural circuit policy*, that can learn to map multidimensional inputs to control commands by sparse, causal, interpretable, and robust neural representations. We test the agent on the safety-critical and real-world domain of end-to-end autonomous control of a self-driving vehicle. We demonstrate that a *neural circuit policy* with a control-network consisting of only 19 sparsely connected neurons, surpasses the driving performance of significantly larger contemporary deep learning [LeCun et al., 2015a] models, while expressing superior stability, causality, interpretability, and robustness.

1.2.2 Designing interpretation methods for trained neural networks

Designing post-training methods to interpret the dynamics of a trained neural network, is another approach to take. This method has been explored by a large body of work, mainly focused on feature visualization methods to empirically understand the dynamics of the network [Erhan et al., 2009, Zeiler and Fergus, 2014, Yosinski et al., 2015, Karpathy et al., 2015, Strobelt et al., 2018, Bilal et al., 2018, Olah et al., 2018]. Alternatively, evaluating attributions by computing saliency maps [Simonyan et al., 2013, Fong and Vedaldi, 2017, Kindermans et al., 2017, Sundararajan et al., 2017], dimensionality reduction method [Bishop and Tipping, 1998, Gulrajani et al., 2016, Maaten and Hinton, 2008], finding recursive dynamical patterns inside a network [Strobelt et al., 2018], robust statistics [Koh and Liang, 2017], information theoretic approaches [Shwartz-Ziv and Tishby, 2017], and gradients in correlation-domain [Hasani et al., 2018c] were effectively proposed.

While these techniques provide rich insight into the dynamics of learned networks, the interpretation of the network often requires detailed prior knowledge about the data content (i.e., in the natural language processing domain). Therefore, such methods may face difficulties in terms of generalization to other forms of sequential data such as time-series forecasting as well as

vision-based classification. Moreover, there is a substantial need for a quantitative approach to measuring interpretation of the dynamics of neural networks, as opposed to the referred empirical qualitative methods.

In Chapter 7, we introduce a novel method to interpret recurrent neural networks (RNNs), particularly their modern gated-variant, long short-term memory networks (LSTMs) at the cellular level. We propose a systematic pipeline for interpreting individual hidden state dynamics within the system using response characterization methods. The ranked contribution of individual cells to the network's output is computed by analyzing a set of interpretable metrics of their decoupled step and sinusoidal responses. As a result, our method is able to uniquely identify neurons with insightful dynamics, quantify relationships between dynamical properties and test accuracy through ablation analysis, and interpret the impact of network capacity on a network's dynamical distribution. Finally, we demonstrate the generalizability and scalability of our method by evaluating a series of different benchmark sequential datasets.

Finally, in Chapter 8, for a real-life application setting, we introduce CompNN, a compositional method for the construction of an interpretable neural-network architecture capturing the dynamical properties of a complex analog multiple-input multiple-output (MIMO) system. CompNN first learns for each input/output pair (i, j), a small-sized nonlinear autoregressive neural network with exogenous input (NARX) [Lin et al., 1996] representing the transfer-function hij . The training dataset is generated by varying input i of the MIMO, only. Then, for each output j, the transfer functions hij are combined by a time-delayed neural network (TDNN) layer, f_j. The training dataset for f_j is generated by varying all MIMO inputs. The final output is $f = (f_1, ..., f_n)$. The neural network's parameters are learned in a supervised learning fashion. We demonstrate the performance of our learned NN in the transient simulation of analog circuits by reducing the simulation time by a factor of seventeen compared to the transistor-level simulations. CompNN allows us to map particular parts of the NN to specific behavioral features of the circuit ergo, enhancing the interpretability of the model. To the best of our knowledge, CompNN is the first method to learn the NN of an analog integrated circuit (MIMO system) in a compositional fashion.

1.2.3 Summary of Contributions

A summary of the dissertation's contributions is provided in the following:

- Development of the Liquid time-constant (LTC) recurrent neural networks as a brain-inspired neural information processing system with continuous-time semantics.
- Theoretical stability and universality analysis of LTCs
- Illustration of the LTC's superior expressivity compared to other types of RNNs in modeling time-series.
- Demonstration of the performance of a compact ordinary neural circuit (ONC) built by the LTC neural model, as an interpretable controller in a series of control tasks and the indication of its superiority compared to similarly structured networks and contemporary deep learning models.

- Experiments with LTC-based ONCs in simulated and physical robot control tasks, including the autonomous parking of a real mobile robot. This is performed by equipping ONCs with a search-based RL optimization scheme.
- Interpretation of the internal dynamics of the learned ONC policies. We introduce a novel computational method to understand continuous-time network dynamics. The technique determines the relation between the kinetics of input neurons and an output decision. We compute the magnitude of a neuron's contribution, of these hidden nodes to the output dynamics in determinable phases of activity, during the simulation.
- We introduce novel network-design principles for the LTC neuronal models, and equipping the designed network with a search-based learning algorithm, to control robotic tasks.
- We deploy Design Operator (DO) based networks in experiments with real and simulated robotic environments. We then Experimentally demonstrate the superiority of DO-based networks in terms of their compactness, robustness to noise, and their interpretable dynamics, compared to contemporary RNNs.
- A central goal of artificial intelligence is to design a single algorithm that simultaneously: 1. expresses attractive generalizability by learning coherent representations of their world, 2. is computationally efficient, 3. realizes robustness to environmental perturbations, and 4. demonstrates interpretability skills to provide sensible and understandable explanations of its learned dynamics, to humans. We combine LTC neural model and scalable deep learning architectures to design an exceedingly compact neural controller for the end-to-end control of the autonomous vehicle.
- We discover that a single algorithm (with a control-network consisting of 19 sparsely connected neurons) learns to map high-dimensional inputs into control commands by superior generalizability, interpretability, and robustness, compared to orders-of-magnitude larger contemporary deep learning models. Such intelligent agents enable high-fidelity autonomy in safety-critical applications.
- We design and implement a novel and lightweight dynamical systems-based algorithm for systematic interpretation of RNNs based on response characterization.
- We Evaluate our interpretation method on a series of sequential datasets, including classification and regression tasks, and perform a detailed interpretation of the trained RNNs on a single cell scale via distribution and ablation analysis as well as on the network scale via network capacity analysis.
- Finally, we discuss an application-oriented method to design compositional recurrent neural networks to model integrated circuits' behavior with enhanced interpretability.

CHAPTER 2

Background

This chapter discusses the necessary background on fundamental principles about dynamical systems and machine learning with the purpose of making the thesis relatively self-contained.

2.1 Math Definitions

In this section, we introduce mathematical principles that have been deployed in the discussions of the thesis.

Differentiability Class. defines the existence of the derivatives of a function. Formally, let $k \in \mathbb{Z}^+$, the function $F : S \to \mathbb{R}$, where S is an open subset on $\mathbb{R}$, is of class C^k, or *smooth*, or C^k-continuous, if $f^{'}$, $f^{''}$, ..., and $f^{(k)}$, exist and are continuous.

Lipschitz. The mapping $F : S \to \mathbb{R}^n$, where S is an open subset of $\mathbb{R}^n$, is called Lipschitz on S, if there exist a constant L (Lipschitz constant), such that:

$$|F(x) - F(y)| \leq L|x - y|, \quad forallx, y \in S. \tag{2.1}$$

Locally Lipschitz. If every point of S has neighborhood S_0 in S, such that the restriction $F \mid S_0$ is Lipschitz, then F is locally Lipschitz.

i.i.d.: Independent and identically distributed random variables. Determines Random variables that are drawn from the same distribution and are mutually independent of each other.

2.2 Supervised Learning

Let D be a dataset that includes (input,output) pairs, from X input space to Y output space. Supervised learning is the process of finding a function $f : X \to Y$, from a training set containing n i.i.d. samples drawn from D; $\{(x_i, y_i)\}_{i=1}^n \sim D^n$, such that the test error $e = E_{(x,y)\sim D}[L(f(x); y)]$,

is minimal. (L is a loss function measuring the error between a predicted output and a labeled output). The learning process terminates once we obtain an f whose test error is minimal enough for a problem's requirement.

2.3 Optimization

Given a set of functions $\mathscr{F}$, from which we choose $f \in \mathscr{F}$, with a low training error e_{trian}, the optimization problem can be formulated as finding the best minimizer of the training error as follows:

$$f^* = \underset{f \in \mathscr{F}}{\operatorname{argmin}}\, e_{Train_S}(f) \tag{2.2}$$

Now if f is parametrized by θ number of parameters, and $f_\theta \in \mathscr{F}$ is differentiable, given a differentiable loss function L, then the function

$$e_{Train_S}(\theta) \equiv e_{Train_S}(f_\theta) = \mathbf{E}_{(x,y)\sim S}[L(f_\theta(x); y)], \tag{2.3}$$

is also differentiable. One can use the gradient descent method, which is a function minimization algorithm applicable to differentiable functions, to solve the optimization problem. Given the function f_θ, the algorithm operates, as shown below:

for epoch **do**
  $\theta_{t+1} \leftarrow \eta \bigtriangledown f_{\theta_t}$
  $t \leftarrow t+1$
end for

Here, the parameter η represents the learning rate, which is problem-dependent and tunes the speed of the optimization steps. A line of work demonstrated improved performance of the gradient descent learning algorithm by adaptive learning rate methods [Duchi et al., 2011, Tieleman and Hinton, 2012, Zeiler, 2012].

Stochastic gradient descent (SGD) is an effective variant of the gradient descent algorithm, in which each optimization step operates on a batch, s, randomly selected from the training data, S. SGD computational steps are considerably faster than gradient descent, especially in the case of large models and large datasets. This property makes SGD to be attractive in contemporary machine learning algorithms. Equipping an SGD optimizer with the Momentum method [Hinton, 1977, Nesterov, 1983] forces the optimizer to take gradient steps in the most effective direction instead of the steepest direction [Goh, 2017].

During the last years, a series of fundamental works have been proposed to improve the performance of the Momentum-based SGD algorithms by introducing adaptive momentum strategies, such as Adam [Kingma and Ba, 2014], AdaMax [Kingma and Ba, 2014], Nadam [Dozat, 2016] and AMSGrad [Reddi et al., 2019].

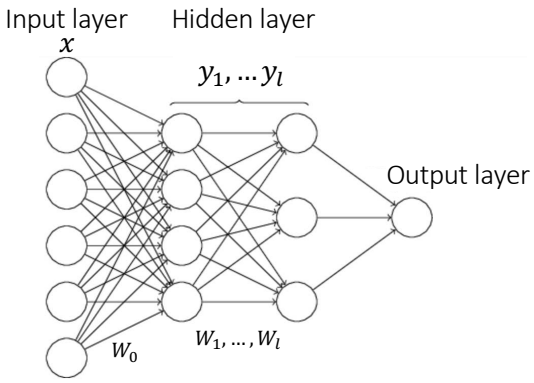


Figure 2.1: The feedforward neural network.

2.4 Feedforward Neural Networks

A Feedforward neural network (FNN) is defined as the configuration of a series of layers composed of artificial neurons (activation functions), as shown in Fig. 2.1. FNNs have successfully been used widely in perceptual problems such as speech recognition and achieved the state-of-the-art compared to the hand-tuned models [Mohamed et al., 2011].

Formally, Given an input x, the output state ,y, of a FFN with l hidden layer, parameterized by weight matrices $W_0, \dots, W_l$ and bias vectors $b_1, \dots, b_{l+1}$ is computed as follows:

$y_0 \leftarrow x$
for i **from** 1 **to** $l+1$ **do**
  $x_i \leftarrow W_{i-1} * y_{i-1} + b_i$
  $y_i \leftarrow f(x_i)$
end for
output $y \leftarrow y_l$

$f(.)$ is typically a sigmoidal activation function (e.g., $f(x) = 1/(1+e^{-x})$, and is applied element-wise. In a supervised-learning setting, FNNs (up to hundreds of layers) can be trained by gradient methods to minimize a loss function with respect to their parameters, thanks to the significant improvements achieved by the development of novel learning algorithms and regularization methods [Hinton and Salakhutdinov, 2006, Hinton et al., 2006, Kingma and Ba, 2014, Srivastava et al., 2014, Ioffe and Szegedy, 2015], architectures [Krizhevsky et al., 2012, Simonyan and Zisserman, 2014, Szegedy et al., 2015, He et al., 2016], and tool-kits [Abadi et al., 2016, Paszke et al., 2017] proposed during the last 2 decades.

2.5 Universal Approximation Capabilities of Neural Networks

In this section, we briefly introduce how neural networks are universal approximators [Cybenko, 1989]. The *fundamental universal approximation* theorem [Hornik et al., 1989] suggests that three-layer feedforward neural networks (input layer, one hidden layer, output layer) can approximate

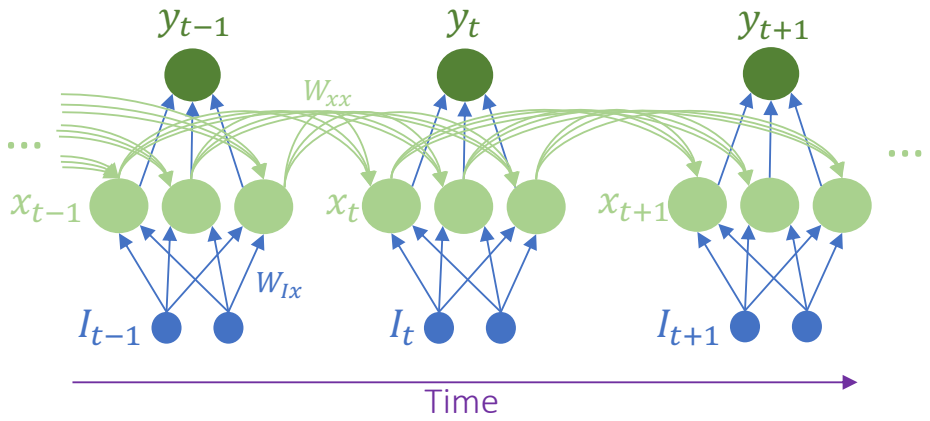


Figure 2.2: A standard recurrent neural network resembles a very deep feedforward network with a hidden layer at every time-step. Note that the weights of the network are preserved across time.

any continuous mapping $f : \mathbb{R}^n \to \mathbb{R}^m$ on a compact set. More precisely [Hornik et al., 1989]:

Theorem 1. *(The fundamental approximation theorem) [Funahashi, 1989]. Let $x = x_1 to x_n$ be the n-dimensional Euclidean space $\mathbb{R}^n$. Let $\sigma(x)$ be a sigmoid function (a non-constant, monotonically increasing and bounded continous function in $\mathbb{R}$). Let K be a compact subset of $\mathbb{R}^n$, and $f(x_1,...,x_n)$ be a continuous function on K. Then, for an arbitrary $\varepsilon > 0$, there exist an integer N, real constants c_i, $\theta_i (i = 1,...,N)$ and $w_{ij} (i = 1,...,N; j = 1,...,n)$, such that*

$$\max_{x \in K} |f(x_1,...,x_n) - \sum_{i=1}^{N} c_i \sigma(\sum_{j=1}^{n} w_{ij} x_j - \theta_i)| < \varepsilon \tag{2.4}$$

holds.

Theorem 1 illustrates that three-layer feedforward neural networks (Input-hidden layer-output), can approximate any continuous mapping $f : \mathbb{R}^n \to \mathbb{R}^m$ on a compact set. We utilize this theorem in Chapter 3, to prove the universal approximation capability of a newly introduced neural network instance.

2.6 Recurrent Neural Networks

A recurrent neural network (RNN) is a class of artificial neural networks which computes the sequential dependencies presented in data, by a state-dependent feedback (memory) mechanism. Therefore, they are a suitable choice for modeling sequential data. RNN variants have achieved state-of-the-art performance in a large variety of applications, ranging from speech recognition [Robinson et al., 1996, Graves et al., 2013, Graves and Jaitly, 2014, Sak et al., 2014], autonomous robot control [OpenAI, 2018, Lechner et al., 2019], to natural language processing (NLP) [Mikolov et al., 2011, Sutskever et al., 2011, Liu et al., 2014, Sutskever et al., 2014].

2.6.1 Standard RNNs

Formally, given an input sequence, $I_1, \ldots, I_T$, with the length T, the network calculates a sequence of hidden states $x_1, \ldots, x_T$, and their corresponding outputs, $y_1, \ldots, y_T$, as follows:

for t **from** 1 **to** T **do**
$u_t \leftarrow W_{Ix} * I_t + W_{xx} * x_{t-1} + b_x$
$x_t \leftarrow f(u_t)$
$o_t \leftarrow W_{xo} * x_t + b_o$
$y_t \leftarrow g(o_t)$
end for

Here, weight matrices W_{Ix}, W_{xx}, W_{xo}, and bias vectors b_x, b_o, are the network parameters and are preserved at every sequence-step as depicted in Fig. 2.2. $f(.)$ and $g(.)$ are the nonlinearities of the RNN for their hidden state and the output state, respectively. Note that at the very first step of the computation, the hidden state of an RNN, a vector x_0, is also an input parameter to the system.

The loss function for RNNs in a supervised learning setting can be determined as the sum of losses computed at every time-steps $L(y, y_{label}) = \sum_{i=1}^{T} L(y_i, y_{label_i})$, and their parameters can be learned by the backpropagation through time (BPTT) algorithm with their derivatives computed as follows [Rumelhart et al., 1986, Werbos et al., 1990]:

for t **from** T **downto** 1 **do**
$do_t \leftarrow g'(o_i)\ dL(y_t, y_{label_t})/dy_t$
$db_o \leftarrow db_o + do_t$
$dW_{xo} \leftarrow dW_{xo} + do_t x_t^T$
$dx_t \leftarrow dx_t + W_{xo}^T do_t$
$dy_t \leftarrow f'(u_t) dx_t$
$dW_{Ix} \leftarrow dW_{Ix} + du_t I_t^T$
$db_x \leftarrow db_x + du_t$
$dW_{xx} \leftarrow dW_{xx} + du_t x_{t-1}^T$
$dx_{t-1} \leftarrow W_{xx}^T du_t$
end for
Return $d\theta = [dW_{xo}, dW_{Ix}, dW_{xx}, db_x, db_o, dx_o]$.

2.6.2 RNNs are Difficult to train

The computation of the derivatives for a recurrent network is pretty straight forward. However, the feedback loop mechanism can naturally result in difficulties to learn long-term dependencies [Bengio et al., 1994, Martens and Sutskever, 2011]. As the gradients of the hidden state at later time-steps are sensitively dependent on the ones at initial time-steps, their values can exponentially grow and lead to an unstable learning process (This is known as the exploding gradient problem). Similarly, if the hidden state gradients at earlier steps are smaller than one, the resulting next step gradient can converge to zero. This phenomenon is known as the vanishing gradient problem [Hochreiter, 1991, Bengio et al., 1994], and is the main reason for an RNN failure to capture long-term temporal structures.

The likelihood of the vanishing gradient problem can be reduced by proper initialization of the network weights. The choice of an activation function with a larger gradient (such as the rectified linear units (ReLU) [Nair and Hinton, 2010], or *tanh* can also enhance the quality of the learning long-term temporal dependencies. Moreover, careful considerations on the growth or the shrinking of the gradient during the learning process by means of regularization methods could also significantly give rise to a performance. Methods include adding noise to the gradient [Neelakantan et al., 2015], clipping the gradients specifically to avoid the exploding gradient problem [Pascanu et al., 2012] and stabilizing the activation functions [Krueger and Memisevic, 2015].

Moreover, many variants of the standard RNN structure have been proposed to tackle the exploding and the vanishing gradient problems fundamentally and to achieve better performance on data with temporal dependencies. We will introduce them briefly in the following sections.

2.6.3 RNNs with Skip connections

To avoid the infinite feedback loop in standard RNN structures, and as a result, preventing the vanishing gradient problem, one can unfold the RNN loop and directly provide a finite number of past states to perform inference of the current state. This process can be seen as the use of skip connections to make the temporal dependencies of the gradients less deep and, therefore, easier to train.

Time-Delayed Neural Networks

The first version of these forms of networks was introduced as the time-delayed neural networks (TDNN) [Waibel et al., 1989]. The output state of a TDNN is computed as follows:

$$y(t) = f(x(t-1), x(t-2), ..., x(t-n)). \tag{2.5}$$

Here, $f(.)$ is the network's nonlinearity and n stands for the number of delayed versions of input signal $x(t)$. The structure of a TDNN is presented in Fig. 2.3. Although this architecture can reduce the likelihood of the vanishing gradient problem for the learning of sequential data with short-term temporal dependencies, they would still suffer from higher-order nonlinearities as n increases. Moreover, TDNNs are not recurrent architectures and do not possess a mechanism to take into account the historical output state at the inference time. Therefore, their ability to learn dynamics with output-temporal dependencies are limited.

Nonlinear Auto-regressive Network with Exogenous Input

An improved variant of the TDNNs was introduced as the nonlinear autoregressive network with exogenous input (NARX)[Lin et al., 1996], where in addition to the skipped connections from the input signal, a finite number of historical output steps are used to define the current output state of the network. Such an output state is then computed as follows:

$$y(t) = f(x(t-1), x(t-2), ..., x(t-n), y(t-1), ..., y(t-m)). \tag{2.6}$$

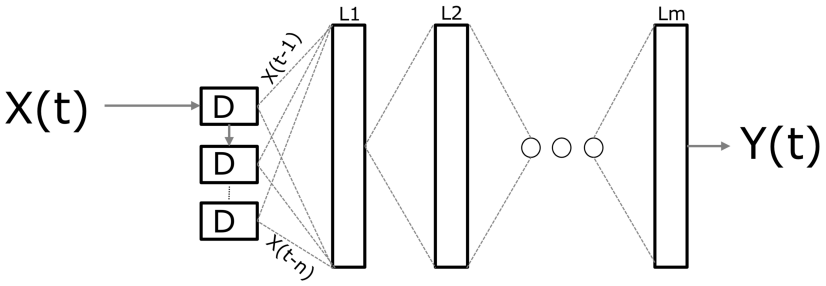


Figure 2.3: Time-delayed neural network structure. D blocks represent delay elements.

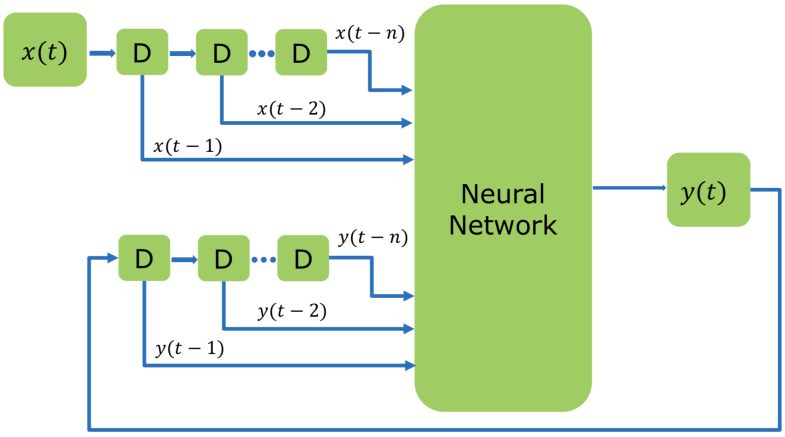


Figure 2.4: NARX network architecture. D blocks represent delay elements.

Here, $f(.)$ represents the network's nonlinearity, n is the number of input signal delays, and m is the number of output signal delays. Fig. 2.4, schematically shows the structure of a NARX network. Recently, variants of this auto-regressive architecture achieved great performance in modeling text-to-speech tasks, phoneme recognition, music generation [Oord et al., 2016], and speech denoising [Rethage et al., 2018]. The architecture is called Wavenet, and at its core, it utilizes dilated causal convolutional layers, where a convolutional filter is applied with a pre-defined input skip-length.

Recurrent Identity Networks

A standard RNN can be equipped with skip connections at their hidden state in order to overcome the vanishing gradient problem, as follows [Hu et al., 2018]:

$$x_t = f(W_{Ix}I_t + (W_{xx} + Identity)x_{t-1} + b), \tag{2.7}$$

where $Identity$ is a non-trainable identity weight matrix that stands for a surrogate memory

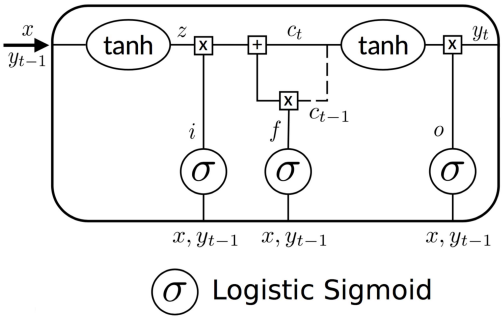


Figure 2.5: Long short-term memory (LSTM) cell structure

component. This representation, which is inspired by the successful feedforward network architecture with skip connections (Residual networks) [He et al., 2016], outperformed modern variants of RNNs in a series of sequential data processing tasks [Hu et al., 2018].

2.6.4 Long Short-term Memory

Long short term Memory (LSTM) [Hochreiter and Schmidhuber, 1997], are gated-recurrent neural networks architectures specifically designed to tackle the training challenges of RNNs such as the vanishing gradient problem. In addition to memorizing the state representation, they realize three gating mechanisms to read from input (i), write to output (o) and forget what the cell has stored (f). Activity of the cell can be formulated as follows [Greff et al., 2017]:

$$c_t^l = z \odot i + f \odot c_{t-1}^l \tag{2.8}$$

$$y_t^l = o \odot \tanh(c_t^l) \tag{2.9}$$

$$\begin{pmatrix} z \\ i \\ f \\ o \end{pmatrix} = \begin{pmatrix} \tanh \\ \sigma \\ \sigma \\ \sigma \end{pmatrix} W^l \begin{pmatrix} y_t^{l-1} \\ y_{t-1}^l \end{pmatrix} \tag{2.10}$$

where c_t^l is layer l's cell state at time t, W^{4n*2n} is the weight matrix, z stands for the input block, and y_t^l denotes the cell's output state.

LSTM networks enhance the learning capability of RNNs by separating their hidden dynamics from their output and by having an additive state temporal dependencies, compared to a multiplicative mechanism in standard RNNs. Moreover, the differentiable gating mechanisms allow LSTM cells to learn temporal dependencies automatically throughout the training process. LSTMs have shown remarkable performance in sequential data processing tasks; accordingly, they are the predominant choice of networks in such application domains. Their complex dynamics, however, makes their interpretability to be challenging and limited to empirical methods such

as feature visualizations [Karpathy et al., 2015, Strobelt et al., 2018]. In Chapter 7, we introduce a quantitative method to gain more insights into the dynamics of LSTM networks.

2.7 Dynamical Systems

Dynamical systems can be defined as the time-evolution of a particular phenomenon, given the environment and the circumstance under which it operates. The evolution of dynamical systems is typically formulated by differential equations. In our context, for instance, a time-continuous neural network's state is the phenomenon with its time-evolution expressed by Eq. 2.14. This particular dynamical system is represented by the flow of an input-dependent differential equation.

2.7.1 Ordinary Differential Equations

Given F, a function of input, I and state x and its derivatives, the following equation,

$$x^{(n)} = F(I, x, x^{(1)}, x^{(2)}, ..., x^{(n-1)}) \tag{2.11}$$

is an explicit representation of an ordinary differential equation (ODE) and

$$F(I, x, x^{(1)}, x^{(2)}, ..., x^{(n-1)}, x^{(n)}) = 0 \tag{2.12}$$

is an implicit representation of an ODE [Teschl, 2012].

Autonomous ODE

An ODE which has no dependency on its inputs, I, and its F is solely a function if its states and its derivatives.

Linear ODE

An ODE is considered to be linear if F can be formatted as the linear combination of the state's derivatives as:

$$x^{(n)} = \sum_{i=0}^{n-1} a_i(z) x^{(i)} + r(z), \tag{2.13}$$

where $a_i(z)$ is a differentiable function of the variable z and is one coefficient of the ODE, which can be nonlinear. $r(z)$ is a function of z, and it defines the homogeneity of the ODE.

Homogeneous ODE

An ODE is considered to be homogeneous when $r(z) = 0$. This property induces one trivial solution of $x = 0$. When $r(z) \neq 0$, the equation is called inhomogeneous.

Nonlinear ODE

If F cannot be separated into a linear combination of its derivatives, the ODE is nonlinear.

2.8 Time-Continuous Networks

RNNs and their gated variants such as LSTMs are successfully deployed in sequential data processing tasks, and continuous-time environments, with a discretized computational modality. The hidden neural state of a neural network can also be determined in continuous time, by linear ODEs as follows:

$$\dot{x}(t) = f(I, x(t), \theta), \tag{2.14}$$

where the continuous variables $x(t)$ determine the hidden state, $f(.)$ is a nonlinear function of inputs I, hidden states x, and parameters θ. Such representation can make the resulting neural network express certain computational benefits over discretized RNN models [Mozer et al., 2017, Chen et al., 2018]. For instance, they express adaptive computation with the use of various ODE solvers [Runge, 1895, Kutta, 1901, Deuflhard et al., 1987] make them suitable for sequential events sampled irregularly in time [Rubanova et al., 2019]. They possess parameter efficiency and are inherently capable of modeling continuous time-series [Rubanova et al., 2019], given a proper learning setting.

2.8.1 Neural ODEs

One can enable infinite computational steps for a neural network $f(.)$ in the limit, by adopting an Euler numerical discretization [Lu et al., 2017] of Eq. 3.3, [Lu et al., 2017, Chen et al., 2018]. The architecture is called Neural ordinary differential equations (Neural ODE) and is effectively utilized in the sequential data processing. Neural ODEs can bring several advantages, compared to discretized RNN models, such as parameter efficiency and superior capability of learning continuous-time dynamics, which arrive at arbitrary time-step [Mozer et al., 2017]. The representation of such models, however, is limited to that of deep learning models since f in Eq. 3.3, is a multi-layer perceptron.

2.8.2 Continuous-time RNNs

In contrast, a rather old variant of neural networks known as the continuous-time (CT) RNN defines its neural state, differently. Its output neural state, $x_i(t)$, is described as the solution of the initial-value problem shown below [Funahashi and Nakamura, 1993]:

$$\dot{x}_i(t) = -x_i(t)/\tau_i + \sum_{j=1}^{m} w_{ij}\sigma(A_{ij}x_j(t) + B_i), \tag{2.15}$$

where τ_i is the time-constant (equivalent to the self-connectivity matrix or self-coupling) of the hidden node, A_{ij} and B_i are the weights and bias, respectively.

CT-RNN were shown to be universal approximators (can approximate arbitrary chosen input-output mappings to an *epsilon* level of precision) [Funahashi and Nakamura, 1993]:

Theorem 2. *(Approximation of dynamical systems by continuous-time recurrent neural networks) [Funahashi and Nakamura, 1993]. Let $D \subset \mathbb{R}^n$ and $F : D \to \mathbb{R}^n$ be an autonomous ordinary differential equation and C^1-mapping, and let $\dot{x} = F(x)$ determine a dynamical system on D. Let K denote a compact subset of D and we consider the trajectories of the system on the interval $I = [0,\ T]$. Then, for an arbitrary positive ε, there exist an integer N and a recurrent neural network with N hidden units, n output units, and an output internal state $u(t) = (U_1(t), ..., U_n(t))$, expressed as:*

$$\frac{du_i(t)}{dt} = -\frac{u_i(t)}{\tau_i} + \sum_{j=1}^{m} w_{ij}\sigma(u_j(t)) + I_i(t), \tag{2.16}$$

where τ_i is the time constant, w_{ij} are the weights, $I_i(t)$ is the input, and σ is a C^1-sigmoid function ($\sigma(x) = 1/(1+exp(-x))$, such that for any trajectory $\{x(t); t \in I\}$ of the system with initial value $x(0) \in K$, and a proper initial condition of the network the statement below holds:

$$\max_{t \in I} |x(t) - u(t)| < \varepsilon.$$

Theorem 2 was proved for the case where the time constants, τ, were kept constant for all hidden states, and the RNN was without inputs ($I_i(t) = 0$) [Funahashi and Nakamura, 1993].

CT-RNNs have been comprehensively evaluated in regard to their stability properties [Funahashi and Nakamura, 1993, Beer, 1995, Zhang et al., 2014], as they are able to forget the neural state with speed τ. The model can capture the temporal kinetics of a dynamical system.

Recently, a number of useful features of such integrator models in the context of Neural ODEs were introduced [Chen et al., 2018]. In fact, many deep learning architectures have been shown to be an approximation of Neural ODEs [Lu et al., 2017]. Integrator models come with many advantages such as adaptive computation where various ODE solvers [Runge, 1895, Kutta, 1901, Deuflhard et al., 1987] can be utilized to model continuous time-series. For a CT-RNN architecture, for instance, the connectivity matrix, τ_i, is kept fixed to ensure stability and improve the quality of the learning process [Funahashi and Nakamura, 1993], while restraining the expressivity of the model from capturing higher-order dynamics given a fixed network size. In Chapter 3, we describe how to overcome the limitations imposed by the CT-RNN and Neural ODE models, by introducing a novel bio-inspired RNN architecture.

2.8.3 Stability of CT-RNNs in Learning Systems

(This Section is entirely reprinted from [Lechner et al., 2020] - © 2020 IEEE) — RNNs are of nonlinear sequential models that have shown great success in modeling sequences in a broad range of application domains, specifically in robotics learning tasks such as maximum likelihood estimation of dynamical systems [Levine and Koltun, 2013], continuous control [Lillicrap et al., 2015, Zhang et al., 2016, Lechner et al., 2019] and simulation to real-world, end-to-end reinforcement learning [Rusu et al., 2016, Hasani et al., 2018c]. Despite their empirically represented

effectiveness, their nonlinear dynamical properties are yet to be discovered. It has been recently shown that linear dynamical systems can be learned through gradient descent with polynomial sample complexity [Hardt et al., 2018], in contrast to the prior works [Vidyasagar and Karandikar, 2006], which suggested an exponential complexity. To take a step forward towards the understanding of RNNs in continuous-time spaces, in this section, we remove the nonlinearity of an RNN's internal state and express its dynamics by the state transition of a time-invariant linear dynamical system (LDS).

The Backpropagation-Through-Time (BPTT) algorithm [Werbos et al., 1990] used for training RNNs, does not scale well with increasing sequence length, due to the sequential workload that cannot be parallelized [Rodgers, 1985]. As discussed, In order to utilize parallel computing hardware effectively, training sequences are usually split into fixed-length sub-sequences. Though this technique significantly improves training efficiency, it creates a training-testing discrepancy when learned RNNs are deployed on arbitrary-length environments. An example of such an issue observed in practice is the explosion of the RNN's internal memory, caused by test episodes that are much longer than the training sequences. Contemporary nonlinear RNN architectures tackle this problem by contracting the RNN state, e.g., the LSTM implementation of TensorFlow has an optional clipping operation applied to the memory variables[1].

In the context of LDS, such nonlinear state-contractors are inapplicable, as they would interfere with the linearity of the system. In this section, we illustrate that careful stability considerations have to be taken into account when an LDS model is learned over fixed-length sequences. For instance, we show that an LDS trained to make a mobile robot avoid obstacles from short episodes of imitation can easily go unstable. To avoid such divergent behavior, we equip a gradient descent-based learning platform with a new regularization loss component-driven from the *Gershgorin circle theorm* [Golub and Van Loan, 1996]. We prove that the resulting loss function ensures the stability of the autonomous LDS by pushing all its eigenvalues to be negative real numbers.

The learning scheme enables end-to-end training of stacked convolutional neural networks (CNN)s or multilayer perceptron (MLP)s kernels, together with the LDS, in simulated and real-life robotic control environments. A video of the performance of the algorithm compared to others in the obstacle avoidance experiment can be viewed at `https://youtu.be/mhEsCoNao5E` and on the Half-Cheetah experiment at `https://youtu.be/MIUGkGPxCdY`.

Learning from Demonstration

Learning from demonstration is a method in which robots learn and generalize well from a set of observations of represented tasks [Atkeson and Schaal, 1997, Biggs and MacDonald, 2003]. Let $f(x) : \mathbb{R}^N \rightarrow \mathbb{R}^N$, be a mapping function from the observations to actions, and $\dot{x} = f(x)$ with $x \in \mathbb{R}^N$ be the state variable of the robotic system, $f(x)$ can be estimated from data formulated as a regression problem [Figueroa and Billard, 2018, Ravichandar et al., 2017]. Several machine learning methods have been introduced for the approximation of $f(x)$ such as Gaussian Mixture Regression [Calinon et al., 2007], Gaussian Processes [Shon et al., 2005], Bayesian Non-Parametric Mixture Models [Figueroa and Billard, 2018], and Neural Networks

[1] `https://www.tensorflow.org/api_docs/python/tf/nn/rnn_cell/LSTMCell`

[Lemme et al., 2014]. While these approaches, employing Learning from Demonstrations, often learn only a single component of the entire control stack, i.e., usually, the trajectory controller [Khansari-Zadeh and Billard, 2011, Khansari-Zadeh and Billard, 2014, Singh et al., 2017], the successes of Deep Learning has made it possible to learn the complete control suit in an end-to-end fashion [Zhang et al., 2016, Finn et al., 2017, Zeng et al., 2017, Hardt et al., 2018]. Here, we extend the existing end-to-end imitation learning scheme to the context of linear dynamical systems.

Learning stable Dynamical Systems

Learning a provably stable system is a desired property for most control environments. Naive approaches for learning a stable system formulate the learning task as a *Constrained Optimization* problem, where a stability condition is added as a constraint to the main objective. For instance, [Khansari-Zadeh and Billard, 2011, Khansari-Zadeh and Billard, 2014] proposed to stabilize a trajectory controller realized by a Gaussian mixture regression by introducing a stability condition based on a Lyapunov function. The concept of Lyapunov functions was also employed by [Lemme et al., 2014, Richards et al., 2018] to learn stable dynamics by Neural Network models. [Blocher et al., 2017, Singh et al., 2017] derived a stability condition from contraction theory, which is then used as a constraint.

For such approaches to work, the stability constraint must imply the stability of the system. However, this implication does not necessarily hold in the other direction; in order to employ a gradient-based optimization, the stability condition must be continuous and differentiable and is therefore often a bound to the true stability condition of the system. The learning algorithm distinguishes between the differentiable stability condition and the true stability property of the autonomous LDS. The constraint is only optimized when the system is non-stable.

An alternative approach was introduced by [Ijspeert et al., 2013], which leverages linear system theory to learn a stable system. [Ijspeert et al., 2013] employs fixed nonlinear *basis functions* to enhance the expressiveness of the learned controller. One way to generalize this method is to stack up an LDS model with more flexible nonlinearity modules such as deep learning layers equipped with convolutional and fully-connected layers).

RNNs for Modeling Dynamical Systems

RNNs presented great performance in robotic control environments; examples include the maximum likelihood estimation of a dynamical system [Levine and Koltun, 2013, Hasani et al., 2019], continuous control [Lillicrap et al., 2015, Zhang et al., 2016] and simulation to real-world reinforcement learning [Rusu et al., 2016]. Fully-connected LSTM networks have been used for the learning of unsupervised video representations [Srivastava et al., 2015]. [Long et al., 2018a] proposed the combination of data-driven and model-based learning to predict the behavior of dynamical systems; Authors stacked a convolutional LSTM [Xingjian et al., 2015], an architecture which performs the convolution operation as the input-to-state transitions instead of dense connections, to a Cellular Neural Networks [Chua and Yang, 1988], an algorithm for solving partial differential equations (PDE) computationally efficient. They achieved state-of-the-art

performance on two dynamical systems test-beds. Here, we propose a novel learning scheme to learn structures by CNNs or MLPs and to learn the temporal dependencies by continuous-time dynamical systems (RNNs) in an end-to-end fashion.

Learning Stable Linear Dynamical Systems

The hidden state transition $x(t)$, and the output dynamics $y(t)$ of a time-invariant *linear dynamical system* (LDS) can be determined as follows:

$$\begin{aligned} \dot{x}(t) &= Ax(t) + Bu(t) \\ y(t) &= Cx(t) \\ x(0) &= x_0 \end{aligned} \tag{2.17}$$

, where $u(t)$ is the input, $A \in \mathbb{R}^{n \times n}$, $B \in \mathbb{R}^{n \times k}$ and $C \in \mathbb{R}^{m \times n}$ are linear transformation matrices which are denoted as the parameters of the LDS to be learned.

Properties of LDS systems of the form Eq. (2.17), specifically their stability, have been studied for decades within the Control Theory community [Hardt et al., 2018]. For instance, the closed-form solution of the autonomous sub-system $\dot{x(t)} = Ax(t)$ is a linear combination of complex exponential functions [Teschl, 2012]. The stability of such autonomous sub-system, i.e. $\lim_{x \to \infty} x(t) < \infty$, can be identified by computing the eigenvalues of A. The system is considered stable if all eigenvalues of A are negative [Teschl, 2012].

In this section, we aim at modeling the input and output of an RNN with standard nonlinear functions, but simplifying the state-transition representation (sequential dependencies as a result of the recurrent connections), as a linear dynamical system. The motivation is to discover unknown dynamical system properties of RNNs, which might be identifiable by the simplification.

In order to efficiently apply gradient-descent to the ODE solution, we discretize the ODE using Euler's explicit method [Press et al., 2007a]:

$$x(t + \Delta) = x(t) + \Delta \dot{x}(t), \tag{2.18}$$

which essentially translates the LDS into an RNN.

As we discussed earlier, many RNN architecture suffer from the *vanishing and exploding gradient* problems [Bengio et al., 1994, Hochreiter et al., 2001, Pascanu et al., 2013]. Learning long-term dependencies becomes challenging with the vanishing gradient effect. In this case, the RNN can solely learn to correlate events that happen close in time. On the other hand, the explosion of the gradient results in an unstable learning process. Truncated back-propagation through time [Williams and Zipser, 1989, Pascanu et al., 2013] is a commonly used method to ease the exploding gradient problem. Despite the choice of the ODE solver, in case of the Euler discretization of the LDS, the described challenges of the gradient computations become a stability issue of the linear dynamical system. The learned dynamical system, post-training, operates in a continuous loop, therefore assuring its stability is vital. Accordingly, we introduce

the *Gershgorin circle loss* as follows [Golub and Van Loan, 1996]:

$$\mathscr{L}_{gc}(A) := \sum_{i=1}^{n} \max\{0, A_{i,i} + \sum_{j \neq i} |A_{i,j}| + \varepsilon\}, \qquad \varepsilon > 0 \tag{2.19}$$

Where A is the state transition matrix of the LDS, in the following, we prove that by minimizing the Gershgorin circle loss, eigenvalues of the matrix A are forced to be negative real numbers, Therefore, if the Gershgorin circle loss is zero, all eigenvalues must have a negative real part.

Lemma 1 (Gershgorin circle theorem)**.** *Every eigenvalue of A lies within at least one of the Gershgorin discs* $D(A_{ii}, R_i)$*, with* $R_i = \sum_{j \neq i} |A_{i,j}|$ *and* $D(a,b) := \{x \in \mathbb{C} \mid |x-a| \leq b\}$

Proof. See [GERSCHGORIN, 1931]. □

Theorem 3 (Gershgorin circle loss ensures stability)**.** *Let* $A \in \mathbb{R}^{n \times n}$ *and*

$$\mathscr{L}_{gc}(A) := \sum_{i=1}^{n} \max\{0, A_{i,i} + R_i + \varepsilon\}, \tag{2.20}$$

with $R_i = \sum_{j \neq i} |A_{i,j}|$ *and* $\varepsilon > 0$*. If* $\mathscr{L}_{gc}(A) \leq 0$ *then all eigenvalues of A have negative real part.*

Proof. Given the definition of D, for every $x \in D(A_{ii}, R_i) \subseteq \mathbb{C}$ it holds that $Re(x) < A_{i,i} + R_i + \varepsilon$ for arbitrary $i = 1, \ldots n$. We assumed $\mathscr{L}_{gc}(A) \leq 0$, ergo $A_{i,i} + R_i + \varepsilon \leq 0$ for every $i = 1, \ldots n$. Using the triangular inequality, it follows that for every $x \in D(A_{ii}, R_i)$ $Re(x) \leq -\varepsilon$ for every Gershgorin disc $D(A_{ii}, R_i)$. According to Lemma 1, every eigenvalue of A must lie within at least one Gershgorin disc; therefore, every eigenvalue must have a negative real part. □

Note that a learned linear dynamical system can be stable even if the Gershgorin circle loss is greater than zero. Consequently, it does not make sense to use the Gershgorin circle loss as regularizer during every optimization step. Therefore, we introduce Algorithm 1, which checks if at least one eigenvalue has a non-negative real part and only then perform a gradient update step with respect to the Gershgorin circle loss. Algorithm 1 declares a stable learning process for LDS equipped with the Gershgorin loss.

LDS as a Continuous-Time RNNs

The representation of the described discretization of ODEs as recurrent neural networks highly resembles the state-space dynamics of CT-RNNs. More specifically, a CT-RNN can be formulated as the Euler simulation of an ODE of the form:

$$\dot{x} = f_\theta(x, u) - x, \tag{2.21}$$

where $f : \mathbb{R}^n \times \mathbb{R}^k \to \mathbb{R}^n$ is a neural net parametrized by θ. The major difference between the state-space variable dynamics of a linear dynamical system and a CT-RNN's state representation

Algorithm 1 Training algorithm for linear dynamical systems where all eigenvalues are guaranteed to be negative - Entirely reprinted from [Lechner et al., 2020] © 2020 IEEE.

Input Maximum number of training epochs N, Training loss $\mathscr{L}_{train}$, Validation loss $\mathscr{L}_{valid}$, Gershgorin circle loss $\mathscr{L}_{gc}$, parameter θ with $A \in \theta$, learning rate α
while at least one eigenvalue of A has non-negative real part **do**
  $\theta \leftarrow \theta - \alpha \frac{\partial \mathscr{L}_{gc}(A)}{\partial A}$
end while
$\theta_{best} \leftarrow \theta$
$v_{best} \leftarrow \mathscr{L}_{valid}(\theta)$
for $1 \ldots N$ **do**
  $\theta \leftarrow \theta - \alpha \frac{\partial \mathscr{L}_{train}(\theta)}{\partial \theta}$
  while at least one eigenvalue of A has positive real part **do**
    $\theta \leftarrow \theta - \alpha \frac{\partial \mathscr{L}_{gc}(A)}{\partial A}$
  end while
  $v \leftarrow \mathscr{L}_{valid}(\theta)$
  if $v < v_{best}$ **then**
    $v_{best} \leftarrow v$
    $\theta_{best} \leftarrow \theta$
  end if
end for
return θ_{best}

is in the nonlinearity introduced by the neural network f in equation (2.21). This difference makes CT-RNNs arguably more expressive but simultaneously increases the complexity of the system, and correspondingly reduces the provability of a system's characteristics such as stability and the closed-form solution.

Note that the transition state-stability of any recurrent model can be feasibly enforced by clipping the RNN state to a bounded range (e.g., between -10 and 10), after every update. However, this approach introduces nonlinearity into the feedback operation, which is non-permissible for linear dynamical systems.

2.8.4 Biologically-inspired Networks

The way nervous systems of living creatures process information has been extensively used in continuous control spaces (i.e., robotic control) as a source of inspiration [Brabazon et al., 2015, LeCun et al., 2015b, Folgheraiter et al., 2006, Capuozzo and Livingston, 2011].

From the computational units' perspective, biologically plausible instances were of time-continuous models predominantly utilized to explain the mechanisms underlying computations in natural nervous systems [Szigeti et al., 2014, Sarma et al., 2018]. Such models are typically constructed by ODEs that mimic the electrophysiological mechanism of ion channels, which form the membrane potential dynamics of a neuron [Koch and Segev, 1998, Hasani et al., 2017a]. Note that in this thesis, we target bio-inspired neural models that realize differentiable dynamics and

are non-spiking, such as the patterns observed in the nervous system of small species (i.e., *C. elegans* and *Ascaris*). The motivation behind this choice is two-fold: 1) we aim to formulate core bio-inspired computational units which are analogous to the activation functions in contemporary deep learning topologies, to be able to compare performances. 2) The principles of neural computation in spiking neural networks (SNNs) [Maass, 1997], and their adaptive dynamics are yet to be understood from the biological perspective. Therefore, we aim to avoid speculations on their essence, and alternatively, will focus on what is better known, which are the analog dynamics of neurons as membrane models in primary nervous systems [Kaplan et al., 2019]. In Chapter 3, we formulate such semantics and rigorously investigate their computational benefits compare to their state-of-the-art deep learning counterparts.

From the network architecture perspective, networks of biophysically modelled neurons [Hasani et al., 2017a, Gleeson et al., 2018] are deployed in applications such as navigation of mobile robots [Folgheraiter et al., 2006, Hagras et al., 2004], control of unmanned aerial vehicles (UAV) [Westphal et al., 2013] and legged robots [Beer et al., 1992, Szczecinski et al., 2015, Szczecinski et al., 2017]. Obtained networks can be topologically divided into two categories: 1) Networks that are put together by hand in a *piece-by-piece* and *trial-and-error* fashion [Beer et al., 1992, Szczecinski et al., 2017, Folgheraiter et al., 2006, Westphal et al., 2013]. These approaches lack fundamental design principles. 2) Networks that deploy fully-connected structures and rely purely on the learning phase to determine functions. Similar to Deep learning models, interpreting the dynamics of these networks becomes a challenge [Olah et al., 2018, Hasani et al., 2018a]. In Chapter 5 and Chapter 6, we address both challenges by incorporating systematic design principles together with a set of rules that improve interpretability and the generalizability of biophysical models.

CHAPTER 3

Liquid Time-Constant Recurrent Neural Networks

3.1 Motivation

Deep learning architectures are constructed by progressive composition of layers of activation functions, which at a high level of abstraction, are believed to emulate the dynamics of biological neurons. The representation of the activation function, however, has been drastically simplified compared to the computational models of neural and synaptic interactions. The reason for such simplification is to reduce the algorithm's complexity and scale the size of the computational graph. Additionally, from an engineering standpoint, strict biological plausibility is not necessary to build up a functional artificial intelligence (AI) system [Hassabis et al., 2017]. However, building greater computational capabilities with increased biological plausibility remains an open research question. For instance, communication between two neurons in biological neural circuits occurs by nonlinear synaptic modulators, where a neuron can adapt its dynamics as a function of the input stimulation it receives. In contrast, inter-neuron connections in contemporary deep learning models are fixed static functions after training. Would a better performance be achievable if we enable more extensive calculations inside a neuron?

Neural information processing in the brain of small species such as *Ascaris* [Davis and Stretton, 1989], *Leech* [Lockery and Sejnowski, 1992], and *C. elegans* [Wicks et al., 1996, Hasani et al., 2017b], is continuous, happens electrotonically (graded charge propagation, *Non-spiking neurons*) [Kato et al., 2015a], and can be modeled by nonlinear ordinary differential equations (ODEs) [Gleeson et al., 2018]. The equivalent counterparts of such models in the deep learning field are arguably recurrent neural networks (RNNs) which are of suitable algorithms for spatiotemporal information processing. RNNs and their gated variants i.e., long short-term memory (LSTM) [Hochreiter and Schmidhuber, 1997], are successfully deployed in sequential data processing tasks, and continuous-time environments, with a discretized computational modality. The neural state of the neural networks can also be determined by ODEs (i.e. continuous-time (CT) RNNs

[Funahashi and Nakamura, 1993] and Neural ODEs [Chen et al., 2018]), which can make the resulting neural network express certain computational benefits over discretized RNN models [Mozer et al., 2017, Chen et al., 2018]. For instance, they express adaptive computation with the use of various ODE solvers [Runge, 1895, Kutta, 1901, Deuflhard et al., 1987], are parameter efficient, and are naturally capable of modeling continuous time-series [Rubanova et al., 2019]. In contrast to Neural ODEs [Chen et al., 2018], standard CT-RNNs [Funahashi and Nakamura, 1993] possess a fixed time-constant (intrinsic self-coupling of a neuron), which allows them to control (i.e. forget) the neural state. This is an essential property not only for learning but also for their dynamical stability. However, in biological neural circuits, the nonlinear synaptic propagation model enforces an additional input-dependent, variable time-constant (i.e., liquid time-constant) for every neuron. Moreover, this form of the synaptic transmission mechanism equips the network with regulatory excitation and inhibition operators, in addition to weighted communications.

These properties result in a learning system that supports efficient learning of neural computational units while capturing nonlinear transformations of input to output signals I) accurately and II) succinctly as possible. Accuracy is important, for example, in a supervised learning context, or equivalently in a teacher-based setting. Succinctness increases the interpretability of the resulting learning system.

We demonstrate that activating these operators yields a more expressive machine learning model and originates more compact representations (e.g., in continuous robotic control settings [Lechner et al., 2019]), thus allowing for greater interpretability of the learned parameters and overall network dynamics.

Furthermore, these properties give rise to a rich set of dynamical systems whose characteristics can be formulated in a dynamic causal modeling (DCM) [Friston et al., 2003, Penny et al., 2004] framework. More precisely, the resulting dynamical system can be identified by the following parameter sets: I) parameters that moderate the influence of the exogenous inputs on the neural state, II) parameters that control the internal interaction of the neural states, and III) parameters that enable input stimulation influence the coupling state of the neuron.

Here we show that a biophysical model, originally developed to describe the neural interactions in the brain of small species such as *Ascaris* and *C. elegans* provides a more compact and more expressive platform for learning the dynamics of complex nonlinear systems. We call this model liquid time-constant (LTC) RNNs. We demonstrate several desirable properties of these models including the incorporation of a natural causal structure in their semantics in which the set of parameters II and III are linked nonlinearly. This feature enriches an LTC network to realize a range of dynamics not possible to be reached by Neural ODEs and CT-RNNs, given an equivalent experimental condition, with the same inputs and the same parameter setting, (See i.e., Fig. 3.3). Moreover, we demonstrate the expressive representation-learning ability of the LTC bio-inspired neural models in continuous time-series domains and thoroughly characterize their attributes as an alternative recurrent neural network instance. We accomplish this by designing an end-to-end learning platform using gradient descent for such models. We prove that any arbitrary-chosen continuous mapping of a dynamical system can be approximated with any precision by the neural state of an LTC network. We formulate upper and lower bounds on their neural state and

the reaction-speed of individual LTC neurons. We show how LTCs can effectively be used in time-series prediction and classification tasks, and perform better than existing RNN models. Finally, we discuss an exciting property of LTCs, that we observed empirically, on their capability to explore beyond their training regime, namely their ability to perform fair and symmetrical extrapolation.

3.2 Defining LTC's Semantics

We design a biologically plausible neural network that is equipped with synaptic regulatory compartments and possess a varying coupling sensitivity for learning continuous-time temporal relationships, inspired by the nervous system of small species. To formally express their representations, we discuss the evolution of the neural networks, from discretized models to time-continuous models.

3.2.1 Perceptron Models

Given a multi-layer perceptron (MLP), f, with sigmoidal activation functions, the neural state of a recurrent neural network, $x(t)$, is constructed by a discretized sequence of transformations defined as follows [Rumelhart et al., 1988]:

$$x(t+1) = f(x(t), \theta), \tag{3.1}$$

where θ are the weights of the MLP. Using this update-state equation, the neural state of the RNN is determined by a finite set of unfolded layers (transformations), enabling back-propagation through time algorithm to be applied as a learning scheme. Discretized RNNs and their variants have been predominantly successful in sequential data processing tasks [Hochreiter and Schmidhuber, 1997, Mikolov et al., 2010, Graves et al., 2013, Pascanu et al., 2013, Chung et al., 2014]. The finite discretization of their neural state, however, limits their expressivity [Chen et al., 2018]. Moreover, as the number of the unfolding steps or the number of layers of the underlying MLP grows (deep network), the degradation problem (i.e., inability of the perceptron to accurately learn the identity function) arises [He et al., 2016]. This problem gave rise to a successful class of deep neural networks known as residual neural networks (ResNets) [He et al., 2016], where the neural state can be computed by [He et al., 2016]:

$$x(t+1) = x(t) + f(x(t), \theta). \tag{3.2}$$

This representation, gave rise to the next generation of (recurrent) neural networks, the so called Neural ODEs [Chen et al., 2018].

3.2.2 Integrator Models

One can enable infinite computational steps in the limit, if we determine Eq. 3.2, as an Euler neumerical discretization [Lu et al., 2017] of an ODE, of the form:

$$\dot{x}(t) = f(x(t), t, \theta). \tag{3.3}$$

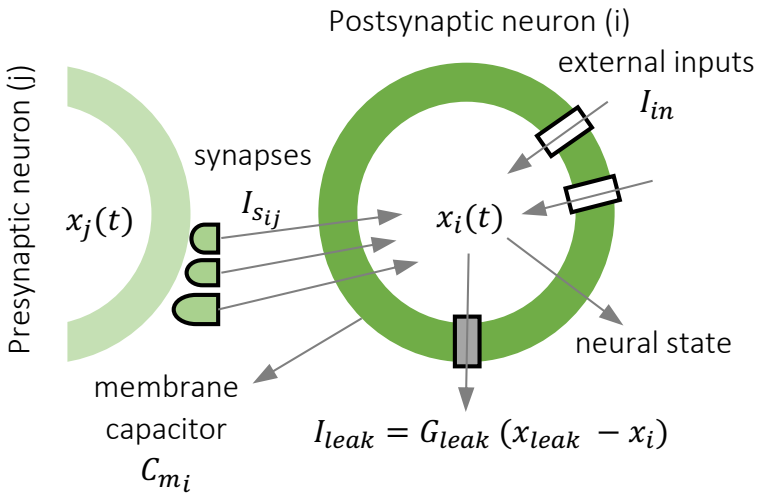


Figure 3.1: Schematic view of a membrane neural model. Postsynaptic neural state $x_i(t)$ is determined by the sum of all inward (I_{in}, $I_{s_{ij}}$) and outward currents (leakage I_{leak}). Synaptic strength from neuron j to i is adaptable and is a nonlinear function of the presynaptic neural state, $x_j(t)$, as depicted in Eq. 3.5.

In this ODE, $x(t)$ is the hidden layers' state at time t and f is the hidden neuron nonlinearity with parameters θ [Lu et al., 2017, Chen et al., 2018].

The architecture was called Neural ordinary differential equations (Neural ODE) and was effectively utilized in the sequential data processing. Neural ODEs can bring several advantages, compared to discretized RNN models, such as parameter efficiency and superior capability of learning continuous-time dynamics, which arrive at arbitrary time-step [Mozer et al., 2017]. The representation of such models, however, is limited to that of deep learning models since f in Eq. 3.3, is a multi-layer perceptron.

In contrast, a rather old variant of neural networks known as the continuous-time (CT) RNN [Funahashi and Nakamura, 1993] defines its neural state, differently. Its output neural state, $x_i(t)$, is described as the solution of the initial-value problem shown in Eq. 2.15. CT-RNN were shown to be universal approximators (can approximate arbitrary chosen input-output mappings to an *epsilon* level of precision) [Funahashi and Nakamura, 1993]. CT-RNNs have been comprehensively evaluated in regard to their stability properties [Funahashi and Nakamura, 1993, Beer, 1995, Zhang et al., 2014], as they are able to forget the neural state with the speed τ. The model can capture the temporal kinetics of a dynamical system. Recently, a number of useful features of integrator models in the context of Neural ODEs were introduced [Chen et al., 2018]. In fact, many deep learning architectures have been shown to be an approximation of Neural ODEs [Lu et al., 2017]. Integrator models come with many advantages such as adaptive computation where various ODE solvers [Runge, 1895, Kutta, 1901, Deuflhard et al., 1987] can be utilized to model continuous time-series. For a CT-RNN architecture, for instance, the connectivity matrix, τ_i, is kept fixed to ensure stability and improve the quality of the learning process [Funahashi and Nakamura, 1993], while restraining the expressivity of the model from capturing higher-order

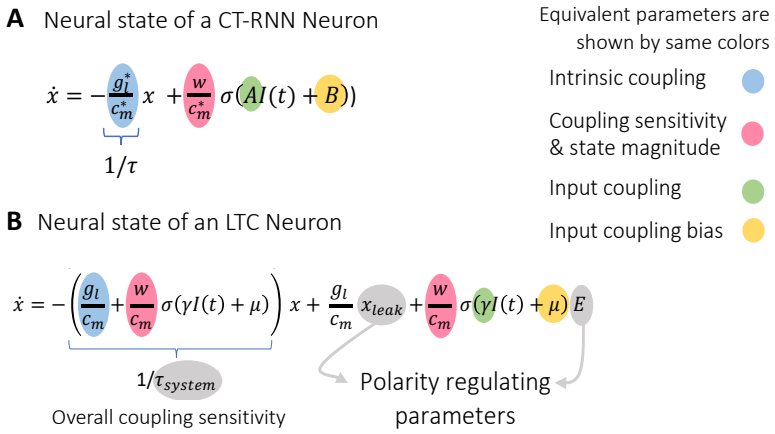


Figure 3.2: T-RNN vs LTC architecture. **A**) Semantics of a CT-RNN neuron taken from Eq. 2.15, compared to **B**) an LTC neuron taken from Eq. 3.7, when they are exposed to an input $I(t)$.

dynamics given a fixed network size.

3.2.3 Membrane Models

In the following, we introduce a biophysical neural network model that comes with natural regulatory elements and relaxes model limitations imposed by neural ODEs and CT-RNNs. Liquid time-constant networks form a causal structure amongst its rich set of control parameters which can notably enhance the expressivity of the neural model.

3.3 LTCs: Liquid Time-constant Recurrent Neural Networks

We formulate LTCs whose neural dynamics are inspired by the kinetics of neurons and synapses in small species. Fig. 3.1 symbolically illustrates how the neural state of a postsynaptic neuron i is determined.

Neurons: the state of the i'th neuron, $x_i(t)$, can be modeled as a *membrane integrator* with the following ordinary differential equation (ODE) [Koch and Segev, 1998]:

$$C_{m_i}\dot{x}_i = g_{l_i}\Big(x_{Leak_i} - x_i(t)\Big) + \sum_{j=1}^{n} I_{in}^{(ij)}, \tag{3.4}$$

with neuronal parameters C_{m_i}, g_{l_i} and x_{Leak_i}. $I_{in}^{(ij)}$ representing the external currents to the cell.

Synapses: synaptic transmission from neuron j to i, is modeled by a sigmoidal nonlinearity (μ_{ij},γ_{ij}), which is a function of the presynaptic node's state, $x_j(t)$, and has a maximum weight of w_{ij} [Koch and Segev, 1998, Wicks et al., 1996]:

$$I_{s_{ij}} = w_{ij}(E_{ij} - x_i(t))/(1+e^{-\gamma_{ij}(x_j(t)+\mu_{ij})}). \tag{3.5}$$

The synaptic current, $I_{s_{ij}}$, then linearly depends on the state of the neuron i. The value of E_{ij}, determines the sign of $I_{s_{ij}}$ making the synapse either excitatory or inhibitory. Such synaptic transmission mechanism allows a post-synaptic neuron to regulate its state based on an intrinsic signed dynamics, and not just a scalar synaptic weight. More precisely, the state dynamics of a neuron i, $x_i(t)$, receiving one synapse from neuron j, can be written as:

$$\dot{x}_i = g_{l_i}\Big(x_{Leak_i} - x_i(t)\Big)/C_{m_i} + w_{ij}\sigma_i(x_j(t))(E_{ij} - x_i)/C_{m_i}, \tag{3.6}$$

where $\sigma_i(x_j(t)) = 1/(1+e^{-\gamma_{ij}(x_j+\mu_{ij})})$. If we set the time-constant of the neuron i to be $\tau_i = C_{m_i}/g_{l_i}$, we can reform this equation as follows:

$$\dot{x}_i = -\Big(\frac{1}{\tau_i} + \frac{w_{ij}}{C_{m_i}}\sigma_i(x_j)\Big)x_i + \Big(\frac{x_{leak_i}}{\tau_i} + \frac{w_{ij}}{C_{m_i}}\sigma_i(x_j)E_{ij}\Big). \tag{3.7}$$

Equation 3.7 presents the ODE as a system with a nonlinearly varying time-constant (coupling sensitivity) defined by $\tau_{system} = \frac{1}{1/\tau_i + w_{ij}/C_{m_i}\sigma_i(x_j)}$, and it is therefore named as *Liquid Time-constant RNN* (LTC).

To build up an intuitive understanding of LTC models, we conduct a series of simple experiments as follows: A single CT-RNN neuron and an LTC neuron are stimulated by a cosine input signal $I(t)$, as shown in Fig. 3.3A and 3.3B and their corresponding semantics are represented in Fig. 3.2.

We then alter their equivalent parameters (shown by colors in Fig. 3.2 in both systems, and compare their generated output states. For both systems, we also present their time-constant (coupling sensitivity) evolution over time, in order to illustrate their differences, empirically. In the experiments shown in Fig. 3.3C to 3.3F, we observe that changing a coupling sensitivity parameter, w, results in a drastic alternation of the output dynamics of the LTC neuron, whereas it minimally changes the shape of the amplified output signal for the CT-RNN neuron. The reason is that the parameter w in LTCs explicitly influences the system's time-constant (coupling sensitivity), τ_{system}, as well as the neuron's state, simultaneously, resulting in the development of an inherent causal structure between neuron's input, its liquid time-constant, and its output state.

Such causal effect can further be illustrated, when the inherent neural time-constant C_m/g_l and τ in LTC and CT-RNN, respectively, varies while w is kept fixed. We observe a range of nonlinear behaviors realized by the LTC neuron as its inherent neural time-constant rises. The effect of such change is represented in terms of the neuron's overall coupling sensitivity, τ_{system}. This attribute for the CT-RNN networks is stationary, which imposes a limitation on the range of possible dynamics the neuron can realize. To elaborate further, we conduct an experiment in which CT-RNN networks with different network sizes are trained to map a cosine input time-series to one of the outputs generated by the LTC neuron presented in Fig. 3.3F). This output signal is highly nonlinear; it is phase-shifted, it includes cosine-wave distortion on the uphill and is asymmetric. Fig. 3.3G indicates that a CT-RNN network requires at least ten neurons to learn the target output generated by a single LTC neuron. Therefore, while CT-RNNs are universal, one pays a dear price in the number of neurons to produce complex dynamics making the resulting network less auditable.

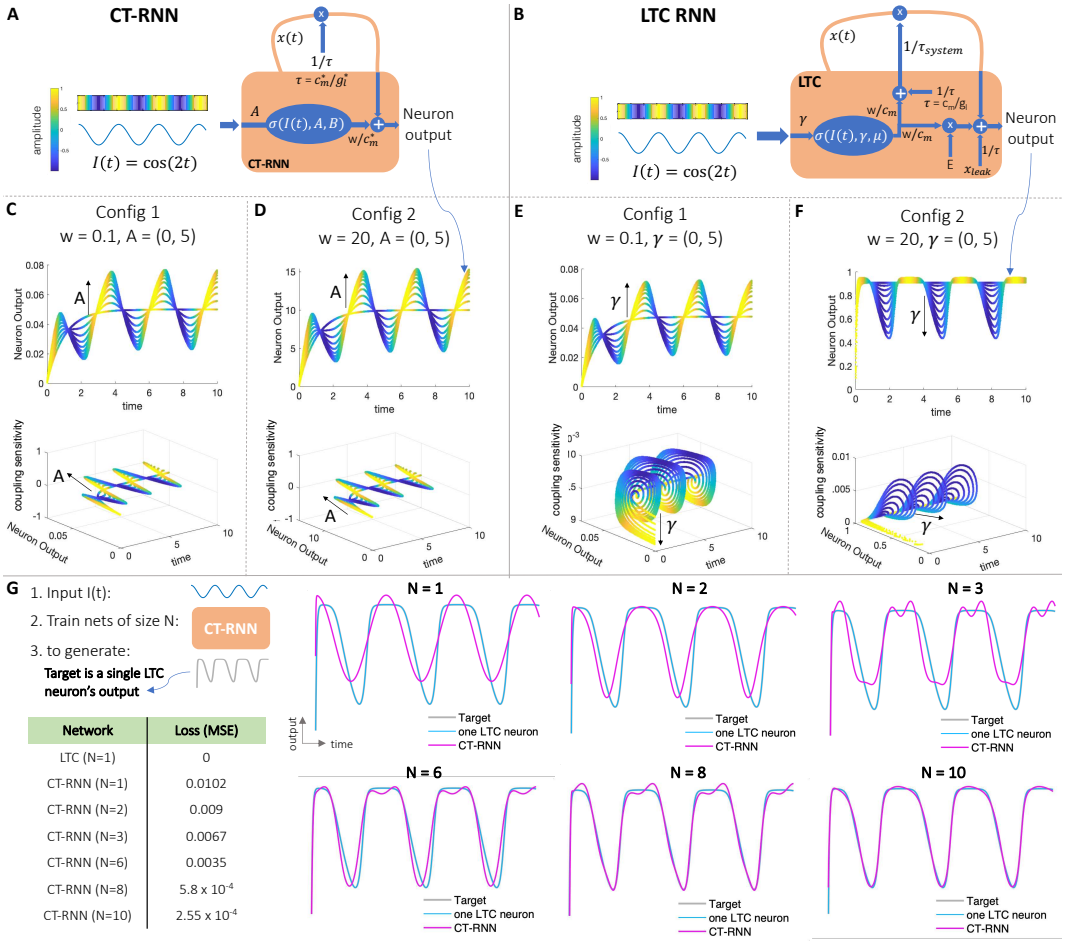
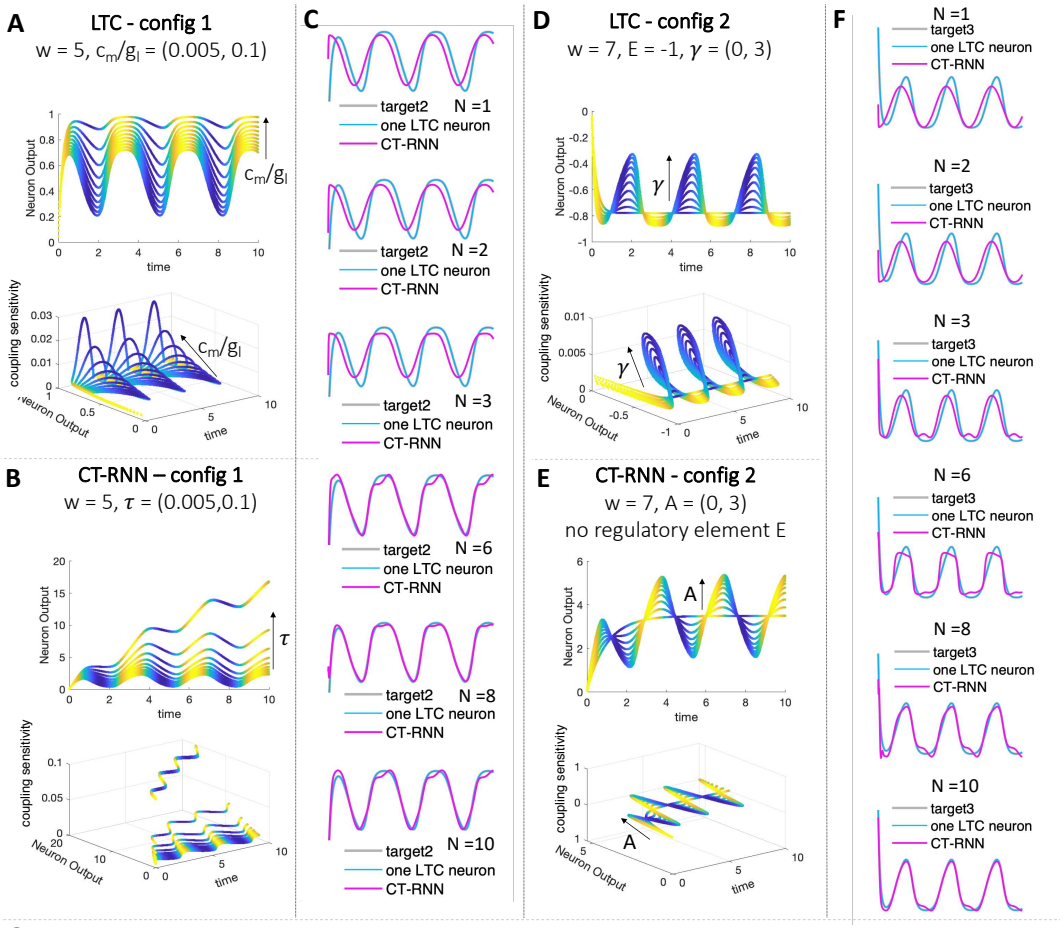


Figure 3.3: How a liquid time-constant (coupling sensitivity) results in a more expressive dynamics? **A**) A CT-RNN neuron's schematic exposed to an input signal $I(t)$. The schematic stands for Eq. 2.15. The color in all plots corresponds to the amplitude of the input signal which is shown by the color bar. **B**) An LTC neuron's symbolic representation exposed to the same input stimulus. The schematic stands for Eq. 3.7. C to F illustrate two experiments performed on a CT-RNN neuron (C-D) and an LTC neuron (E-F), in a comparable parametrization setting, when they are both stimulated by $I(t)$. Each subsection presents the projection of the neuron's output response (the top plot) and its coupling sensitivity (time constant) (the bottom plot) over time. The objective of these experiments is to demonstrate the expressivity of the LTC dynamics compared to the CT-RNN when we alter an equivalent parameter in both architectures. **C**) Parameter w, is set to a low value, and ten output trajectories corresponding to 10 different input coupling parameter A varying in the specified range, for the CT-RNN neuron. **D**) In the second configuration, w is set to a higher value, and the same ten trajectories are plotted. The only observable change in the output response of the CT-RNN compared to case C is the amplification of the signal. **E**) When parameter w is set to a low value equivalently for an LTC neuron, similar dynamics to that of CT-RNN is observed since the effect of the coupling sensitivity is weakened by a low w. **F**) The dynamics of the LTC undergo significant alternations when w is large; LTC's dynamics are directly caused by its varying coupling sensitivity (τ_{system}), whereas the shape of the signal for a CT-RNN neuron (case D) stays intact with an amplified output. Note the change in the coupling sensitivity kinetics for the LTC neuron from case E to Case F. **G**) An experiment demonstrating the fitting performance of a CT-RNN (stimulated by $I(t)$, and with different network sizes, N), which is trained to generate a particular output of a single LTC neuron (shown in part F). One can observe that a CT-RNN requires at least ten neurons to realize the desired dynamics produced by only one LTC cell. From this set of experiments, we can intuitively infer how the coupling sensitivity (*time-constant*) of the LTC neuron is varied (*is liquid*) over time, and how its semantics result in an increased expressivity of dynamics, compared to standard CT-RNNs.

From these set of experiments, we can intuitively take away how the coupling sensitivity (*time-constant*) of the LTC neuron is varied (*is liquid*) over time, and how its semantics results in an increased expressivity of its output dynamics, compared to standard CT-RNNs. Moreover, having a *liquid* time-constant allows the neuron to determine its reaction speed at various neural states, to enhance its output (decision) more flexibly.



G

Network	LTC (N=1)	CTRNN (N=1)	CTRNN (N= 2)	CTRNN (N=3)	CTRNN (N=6)	CTRNN (N=8)	CTRNN (N=10)
Loss (MSE)	0	0.012	0.011	0.011	0.0018	8×10^{-4}	9×10^{-4}

H

Network	LTC (N=1)	CTRNN (N=1)	CTRNN (N= 2)	CTRNN (N=3)	CTRNN (N=6)	CTRNN (N=8)	CTRNN (N=10)
Loss (MSE)	0	0.009	0.0086	0.0064	0.0029	0.0014	4.6×10^{-4}

Figure 3.4: **A-B**) The influence of the alternation of the inherent neural time-constant C_m/g_l and τ in LTC and CT-RNN, respectively when w is set to a constant value for both networks. The nonlinear behavior of the LTC neuron is originated as a result of its input dependent liquid coupling sensitivity, while this attribute for the CT-RNN cell is stationary, limiting its range of realizable dynamics. **C**) Training of a CT-RNN network with size N to reproduce one of the outputs generated by one LTC neuron shown in part A. **D-E**) The influence of the regulatory parameter E in LTCs. By setting $E < x_{leak}$, the neuron naturally originates an inhibitory effect on its output. We also observe that the modification of the dynamics of the LTC neuron is explicitly caused by its coupling sensitivity parameters expressed in Eq. 3.7 as τ_{system}. **F**) Training of a CT-RNN network with size N to reproduce one of the outputs generated by one LTC neuron shown in part D. **G**) The mean squared error corresponding to the output traces of part C. **H**) The mean squared error corresponding to the output traces of part F.

In the experiments shown in Fig. 3.3 and Fig. 3.4, one neuron receives an input signal $I(t) = cos(2t)$ with $t \in [0, 10]s$ sampled at $f = 1000Hz$. We introduced a scaling parameter c_m^* in the CT-RNN semantics, and set it to be equal to c_m in LTCs, in order to make the range of the parameters in both models comparable. Correspondingly, the intrinsic coupling parameter τ in CT-RNN is defined by $\frac{c_m^*}{g_l^*}$, and is set to be equal to the $\frac{c_m}{g_l}$ term in LTCs. Table 3.1 represent the parameters used in each experiment presented in Fig. 3.3, and Fig. 3.4.

Table 3.1: Full list of parameter setting in the experiments of Fig. 3.3 and Fig. 3.4

Model	Param	Fig. 3.3C Fig. 3.3E	Fig. 3.3D Fig. 3.3F	Fig. 3.4A Fig. 3.4B	Fig. 3.4D Fig. 3.4E
LTC	g_l	1	1	(0.01,2)	1
	c_m	0.01	0.01	0.01	0.01
	w	0.1	20	5	7
	γ	(0,5)	(0,5)	3	(0,3)
	μ	0	0	0	0
	x_{leak}	0	0	0	0
	E	1	1	1	-1
CT-RNN	g_l^*	1	1	(0.01,2)	1
	c_m^*	0.01	0.01	0.01	0.01
	w	0.1	20	5	7
	A	(0,5)	(0,5)	3	(0,3)
	B	0	0	0	0

This feature distinguishes the dynamics of LTC cells from the standard CT-RNN and Neural ODEs, and make them fit into a *dynamic causal modeling* framework;

3.4 LTCs and DCMs

Dynamic causal modeling (DCM) describes a framework to identify models in continuous-time and fit them to data by statistical learning methods [Friston et al., 2003]. DCMs were effectively used in modeling fMRI time-series where the nature of the identification problem matches that of the used bilinear system [Penny et al., 2004]. When describing neural states, $x = [x_1, x_2, ..., x_n]$, by a DCM, their standard form can be written as follows [Friston et al., 2003]:

$$\dot{x} = f(x, I, \theta), \tag{3.8}$$

f is a nonlinear function, I is the set of inputs to the system and θ are the system's parameters. The DCM's bilinear approximation [Penny et al., 2005] can be expanded as follows [Friston et al., 2003]:

$$\dot{x} \approx Ax + \sum I_j B^j x + CI = (A + \sum I_j B^j)x + CI \tag{3.9}$$

with:

$$A = \frac{\partial f}{\partial x} = \frac{\partial \dot{x}}{\partial x}, \quad B^j = \frac{\partial^2 f}{\partial x \partial I_j} = \frac{\partial}{\partial I_j}\frac{\partial \dot{x}}{\partial x}, \quad C = \frac{\partial f}{\partial I}. \tag{3.10}$$

The resulting three sets of parameters, A, B_j, and C, determine a natural and useful medium for modeling dynamics of continuous time-series. A mediates the coupling of the nodes of the system in the absence of inputs. B_j induced by the jth input, varies the states' coupling behavior, and correspondingly the time-constant of the system. C controls the direct input influence over the network's dynamics.

Interestingly, the overall network dynamics of the LTCs with $x(t) = [x_1(t), ..., x_{n+N}(t)]$ representing the internal states of N hidden units and n output units, realizes dynamics akin to the bilinear DCM sketched in Eq. 3.9. LTCs can be written in matrix format as follows:

$$\dot{x}(t) = -(1/\tau + W\sigma(x(t)))x(t) + A + W\sigma(x(t))B, \tag{3.11}$$

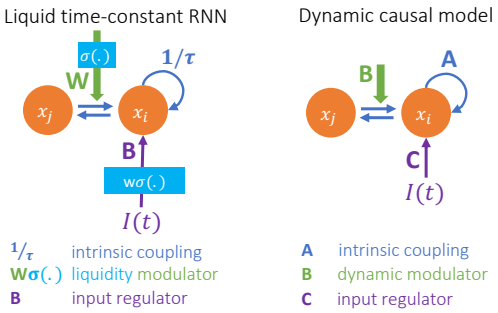


Figure 3.5: LTC vs DCM. Schematic representation of the similarities of LTC model to the dynamical causal models. In both modeling frameworks, three sets of parameters are deployed to identify a proper model, Intrinsic coupling, dynamic modulator (liquid time-constant) and the input control parameters.

in which $\sigma(x)$ is a C^1-sigmoid functions and is applied element-wise. $\tau^{n+N} > 0$ includes all neuronal time-constants, A is an $n+N$ vector of resting states, B depicts an $n+N$ vector of synaptic reversals, and W is a $n+N$ vector produced by the matrix multiplication of a weight matrix of shape $(n+N) \times (n+N)$ and an $n+N$ vector containing the reversed value of all C_{m_i}s. Both A and B entries are bound to a range $[-\alpha, \beta]$ for $0 < \alpha < +\infty$, and $0 \leq \beta < +\infty$. A contains all x_{leak_i}/C_{m_i} and B presents all E_{ij}s.

Fig. 3.5 schematically depicts the similarities of both modeling frameworks. The set of Liquid parameters which control the coupling sensitivity of the neuron results in a more expressive form of a dynamical system with a causal structure. In order to demonstrate the performance of the model, we first theoretically prove that they possess universal approximation capabilities. We then show that LTCs realize a more compelling performance in many real-life time-series prediction and classification tasks.

3.5 LTCs are Universal Approximators

Universal approximation theory suggests that any finite set of a continuous mapping can be realized by three-layer feed-forward neural networks with any precision [Hornik et al., 1989, Funahashi, 1989, Cybenko, 1989]. Such global approximation capability has also been extended to the standard recurrent neural networks (RNN)s [Funahashi, 1989], and continuous-time RNNs [Funahashi and Nakamura, 1993] on their ability to fit arbitrarily spatiotemporal dynamics.

In this section, we prove that any given finite trajectory of an n-dimensional dynamical system can be approximated by the internal and output states of an LTC RNN, with n outputs, N hidden nodes, and a proper initial condition. Let $x = [x_1, ..., x_n]^T$ be the n-dimensional Euclidean space on $\mathbb{R}^n$.

Theorem 4. *Let $S \subset \mathbb{R}^n$ and let $\dot{x} = F(x)$ be an autonomous ordinary differential equation defining a dynamical system, where $F : S \to \mathbb{R}^n$ is a C^1-mapping on S. Let D denote a compact subset of S and assume the simulation of the system is bounded in the interval $I = [0, T]$. Then,*

for a positive ε, there exist an integer N and an LTC RNN with N hidden units, n output units, and an output internal state $u(t) = (U_1(t), ..., U_n(t))^T$, described as in Equation (8.1):

$$\dot{u}(t) = -(1/\tau + W\sigma(u(t)))u(t) + A + WB\sigma(u(t)), \tag{3.12}$$

such that for any rollout $\{x(t)|t \in I\}$ of the system with initial value $x(0) \in D$, and a proper initial condition of the network, $max_{t \in I}|x(t) - u(t)| < \varepsilon$. In (8.1) $\sigma(x)$ is a C^1-sigmoid function (bounded, non-constant, continuous and monotonically increasing), $\tau > 0$ is the time constant, A is an $n+N$ vector of resting states, B is an $n+N$ vector of synaptic reversal values, and W is an $(n+N) \times (n+N)$ weight matrix, The entries of A and B are bounded in $[-\alpha, \beta]$ for $0 < \alpha < \infty$ and $0 \leq \beta < \infty$.

Proof sketch – We base the proof on the fundamental universal approximation theorem [Hornik et al., 1989] on feedforward neural networks [Funahashi, 1989, Cybenko, 1989, Hornik et al., 1989], recurrent neural networks (RNN) [Funahashi, 1989, Schäfer and Zimmermann, 2006] and continuous-time RNNs [Funahashi and Nakamura, 1993]. Moreover, we utilize additional concepts such as Lipschitzness, locally Lipschitzness, uniqueness of the solution of CT-RNN models, and Lemma 6, to prove the universality of LTCs.

Lemma 2. *Let a mapping $F : S \to \mathbb{R}^n$ be C^1. Then F is locally Lipschitz. Also, if $D \subset S$ is compact, then the restriction $F \mid D$ is Lipschitz. (Proof in [Hirsch and Smale, 1973], chapter 8, section 3).*

Lemma 3. *Let $F : S \to \mathbb{R}^n$ be a C^1-mapping and $x_0 \in S$. There exists a positive a and a unique solution $x : (-a, a) \to S$ of the differential equation*

$$\dot{x} = F(x), \tag{3.13}$$

which satisfies the initial condition $x(0) = x_0$. (Proof in [Hirsch and Smale, 1973], chapter 8, section 2, Theorem 1.)

Lemma 4. *Let S be an open subset of $\mathbb{R}^n$ and $F : S \to \mathbb{R}^n$ be a C^1-mapping. On a maximal interval $J = (\alpha, \beta) \subset \mathbb{R}$, let $x(t)$ be a solution. Then for any compact subset $D \subset S$, there exists some $t \in (\alpha, \beta)$, for which $x(t) \notin D$. (Proof in [Hirsch and Smale, 1973], Chapter 8, section 5, Theorem).*

Lemma 5. *For an $F : \mathbb{R}^n \to \mathbb{R}^n$ which is a bound C^1-mapping, the differential equation*

$$\dot{x} = -\frac{x}{\tau} + F(x), \tag{3.14}$$

where $\tau > 0$ has a unique solution on $[0, \infty)$. (Proof in [Funahashi and Nakamura, 1993], Section 4, Lemma 4).

Lemma 6. *For an $F : \mathbb{R}^n \to \mathbb{R}^{+n}$ which is a bounded C^1-mapping, the differential equation*

$$\dot{x} = -(1/\tau + F(x))x + A + BF(x), \tag{3.15}$$

in which τ is a positive constant, and A and B are constants coefficients bound to a range $[-\alpha, \beta]$ for $0 < \alpha < +\infty$, and $0 \leq \beta < +\infty$, has a unique solution on $[0, \infty)$.

Proof. Based on the assumptions, we can take a positive M, such that

$$0 \leq F_i(x) \leq M (\forall i = 1, ..., n) \tag{3.16}$$

by looking at the solutions of the following differential equation:

$$\dot{y} = -(1/\tau + M)y + A + BM, \tag{3.17}$$

we can show that

$$min\{|x_i(0)|, \frac{\tau(A+BM)}{1+\tau M}\} \leq x_i(t) \leq max\{|x_i(0)|, \frac{\tau(A+BM)}{1+\tau M}\}, \tag{3.18}$$

if we set the output of the max to C_{max_i} and the output of the min to C_{min_i} and also set $C_1 = min\{C_{min_i}\}$ and $C_2 = max\{C_{max_i}\}$, then the solution $x(t)$ satisfies

$$\sqrt{n}C_1 \leq x(t) \leq \sqrt{n}C_2. \tag{3.19}$$

Based on Lemma 3 and Lemma 4 a unique solution exists on the interval $[0, +\infty)$. □

Lemma 6 demonstrates that an LTC-RNN defined by Eq. 3.15, has a unique solution on $[0, \infty)$, since the output function is bound and C^1.

Lemma 7. *Let two continuous mapping $F, \tilde{F} : S \to \mathbb{R}^n$ be Lipschitz, and L be a Lipschitz constant of F. if $\forall x \in S$,*

$$|F(x) - \tilde{F}(x)| < \varepsilon, \tag{3.20}$$

holds, if $x(t)$ and $y(t)$ are solutions to

$$\dot{x} = F(x), \tag{3.21}$$

$$\dot{y} = \tilde{F}(x), \tag{3.22}$$

on some interval J, such that $x(t_0) = y(t_0)$, then

$$|x(t) - y(t)| \leq \frac{\varepsilon}{L}(e^{L|t-t_0|} - 1). \tag{3.23}$$

(Proof in [Hirsch and Smale, 1973], chapter 15, section 1, Theorem 3).

3.5.1 Proof of the Theorem 4

Proof. Using the above definitions and lemmas, we prove that LTC-RNNs are universal approximators. For proving Theorem 4, we adopt similar steps to that of Funahashi and Nakamura on the approximation ability of continuous-time RNNs [Funahashi and Nakamura, 1993], to define a dynamical system and try to approximate it with a larger dynamical system. The second one is an LTC RNN. The fundamental difference of the proof of LTC's universal approximation to that of CT RNNs lies in the distinction of the semantics of both systems where the LTC network contains a nonlinear input term in its coupling sensitivity module represented in Eq. 3.12.

Part 1. We choose an η which is in range $(0, min\{\varepsilon, \lambda\})$, for $\varepsilon > 0$, and λ the distance between $\tilde{D}$ and boundary δS of S. D_η is set:

$$D_\eta = \{x \in \mathbb{R}^n; \exists z \in \tilde{D}, |x - z| \leq \eta\}. \tag{3.24}$$

D_η stands for a compact subset of S, because $\tilde{D}$ is compact. Thus, F is Lipschitz on D_η by Lemma 2. Let L_F be the Lipschitz constant of $F|D_\eta$, then, we can choose an $\varepsilon_l > 0$, such that

$$\varepsilon_l < \frac{\eta L_F}{2(e^{L_F T} - 1)}. \tag{3.25}$$

Based on the universal approximation theorem, there is an integer N, and an $n \times N$ matrix B, and an $N \times n$ matrix C and an N-dimensional vector μ such that

$$max|F(x) - B\sigma(Cx + \mu)| < \frac{\varepsilon_l}{2}. \tag{3.26}$$

We define a C^1-mapping $\tilde{F} : \mathbb{R}^n \to \mathbb{R}^n$ as:

$$\tilde{F}(x) = -(1/\tau + W_l \sigma(Cx + \mu))x + A + BW_l \sigma(Cx + \mu), \tag{3.27}$$

with parameters matching that of Eq. 3.12 with $W_l = W$.

We set system's time constant (coupling sensitivity), τ_{sys} as:

$$\tau_{sys} = \frac{1}{1/\tau + W_l \sigma(Cx + \mu)}. \tag{3.28}$$

We chose a large τ_{sys}, conditioned with the following:

$$(a) \;\; \forall x \in D_\eta; \;\; |\frac{x}{\tau_{sys}}| < \frac{\varepsilon_l}{2} \tag{3.29}$$

$$(b) \;\; |\frac{\mu}{\tau_{sys}}| < \frac{\eta L_{\tilde{G}}}{2(e^{L_{\tilde{G}} T} - 1)} \text{ and } |\frac{1}{\tau_{sys}}| < \frac{L_{\tilde{G}}}{2}, \tag{3.30}$$

where $L_{\tilde{G}}/2$ is a lipschitz constant for the mapping $W_l \sigma : \mathbb{R}^{n+N} \to \mathbb{R}^{n+N}$ which we will determine later. To satisfy conditions (a) and (b), $\tau W_l << 1$ should hold true.

Then by Eq. 3.26 and 3.27, we can prove:

$$\max_{x \in D_\eta} |F(x) - \tilde{F}(x)| < \varepsilon_l \tag{3.31}$$

Let's set $x(t)$ and $\tilde{x}(t)$ with initial state $x(0) = \tilde{x}(0) = x_0 \in D$, as the solutions of equations below:

$$\dot{x} = F(x), \tag{3.32}$$

$$\dot{\tilde{x}} = \tilde{F}(x). \tag{3.33}$$

Based on Lemma 7 for any $t \in I$,

$$|x(t) - \tilde{x}(t)| \leq \frac{\varepsilon_l}{L_F}(e^{L_F t} - 1) \tag{3.34}$$

$$\leq \frac{\varepsilon_l}{L_F}(e^{L_F T} - 1). \tag{3.35}$$

Thus, based on the conditions on ε,

$$\underset{t \in I}{max} |x(t) - \tilde{x}(t)| < \frac{\eta}{2}. \tag{3.36}$$

Part 2. Let's Considering the following dynamical system defined by $\tilde{F}$ in Part 1:

$$\dot{\tilde{x}} = -\frac{1}{\tau_{sys}}\tilde{x} + A_1 + W_l B\sigma(C\tilde{x} + \mu). \tag{3.37}$$

Suppose we set $\tilde{y} = C\tilde{x} + \mu$; then:

$$\dot{\tilde{y}} = C\dot{\tilde{x}} = -\frac{1}{\tau_{sys}}\tilde{y} + E\sigma(\tilde{y}) + A_2 + \frac{\mu}{\tau_{sys}}, \tag{3.38}$$

where $E = CW_l B$, an $N \times N$ matrix. We define

$$\tilde{z} = (\tilde{x}_1, ..., \tilde{x}_n, \tilde{y}_1, ..., \tilde{y}_n), \tag{3.39}$$

and we set a mapping $\tilde{G} : \mathbb{R}^{n+N} \to \mathbb{R}^{n+N}$ as:

$$\tilde{G}(\tilde{z}) = -\frac{1}{\tau_{sys}}\tilde{z} + W\sigma(\tilde{z}) + A + \frac{\mu_1}{\tau_{sys}}, \tag{3.40}$$

where;

$$W^{(n+N)\times(n+N)} = \begin{pmatrix} 0 & B \\ 0 & E \end{pmatrix}, \tag{3.41}$$

$$\mu_1^{n+N} = \begin{pmatrix} 0 \\ \mu \end{pmatrix}, \quad A^{n+N} = \begin{pmatrix} A_1 \\ A_2 \end{pmatrix}. \tag{3.42}$$

Now using Lemma 3, we can show that solutions of the following dynamical system:

$$\dot{\tilde{z}} = \tilde{G}(\tilde{z}), \quad \tilde{y}(0) = C\tilde{x}(0) + \mu, \tag{3.43}$$

are equivalent to the solutions of the Eq. 3.37.

Let's define a new dynamical system $G : \mathbb{R}^{n+N} \to \mathbb{R}^{n+N}$ as follows:

$$G(z) = -\frac{1}{\tau_{sys}} z + W\sigma(z) + A, \tag{3.44}$$

where $z = (x_1, ..., x_n, y_1, ..., y_n)$. Then the dynamical system below

$$\dot{z} = -\frac{1}{\tau_{sys}} z + W\sigma(z) + A, \tag{3.45}$$

can be realized by an LTC-RNN, if we set $h(t) = (h_1(t), ..., h_N(t))$ as the hidden states, and $u(t) = (U_1(t), ..., U_n(t))$ as the output states of the system. Since $\tilde{G}$ and G are both C^1-mapping and $\sigma'(x)$ is bound, therefore, the mapping $\tilde{z} \mapsto W\sigma(\tilde{z}) + A$ is Lipschitz on $\mathbb{R}^{n+N}$, with a Lipschitz constant $L_{\tilde{G}}/2$. As $L_{\tilde{G}}/2$ is lipschitz constant for $-\tilde{z}/\tau_{sys}$ by condition (b) on τ_{sys}, $L_{\tilde{G}}$ is a Lipschitz constant of $\tilde{G}$.

From Eq. 3.40, Eq. 3.44, and condition (b) of τ_{sys}, we can derive the following:

$$|\tilde{G}(z) - G(z)| = |\frac{\mu}{\tau_{sys}}| < \frac{\eta L_{\tilde{G}}}{2(e^{L_{\tilde{G}}T} - 1)}. \tag{3.46}$$

Accordingly, we can set $\tilde{z}(t)$ and $z(t)$, solutions of the dynamical systems:

$$\dot{\tilde{z}} = \tilde{G}(z), \quad \begin{cases} \tilde{x}(0) = x_0 \in D \\ \tilde{y}(0) = Cx_0 + \mu \end{cases} \tag{3.47}$$

$$\dot{z} = G(z), \quad \begin{cases} u(0) = x_0 \in D \\ \tilde{h}(0) = Cx_0 + \mu \end{cases} \tag{3.48}$$

By Lemma 7, we achieve

$$\max_{t \in I} |\tilde{z}(t) - z(t)| < \frac{\eta}{2}, \tag{3.49}$$

and therefore we have:

$$\max_{t \in I} |\tilde{x}(t) - u(t)| < \frac{\eta}{2}, \tag{3.50}$$

Part3. Now by using Eq. 3.36 and Eq. 3.50, for a positive ε, we can design an LTC-RNN with internal dynamical state $z(t)$, with τ_{sys} and W. For x(t) satisfying $\dot{x} = F(x)$, if we initialize the network by $u(0) = x(0)$ and $h(0) = Cx(0) + \mu$, we obtain:

$$\max_{t \in I} |x(t) - u(t)| < \frac{\eta}{2} + \frac{\eta}{2} = \eta < \varepsilon. \tag{3.51}$$

□

REMARKS. LTC-RNNs allow the elements of the hidden layer to have recurrent connections to each other. However, it assumes a feed-forward connection stream from hidden nodes to output units. We assumed no inputs to the system and principally showed that the hidden nodes' together with output units, could approximate any finite trajectory of an autonomous dynamical system.

Theorem 4 proves the universal approximation capability of LTC networks. In the following sections, we determine the stability of their hidden state and the bounded dynamics of their varying time-constant.

3.6 Bounds on the time-constant and the neural state of LTCs

We prove that the liquid time-constant and the state of neurons in an LTC-RNN are bounded to a finite range, as described in Lemmas 8 and 9, respectively.

Lemma 8. *Let x_i denote the state of a neuron i, receiving N synaptic connections as in Eq. 3.5 from the other neurons of an LTC network G. If Eq. 3.4 determines the dynamics of each neuron's state, then the time constant of the activity of the neuron, τ_i, is bounded to the following range:*

$$C_i/(g_i + \sum_{j=1}^{N} w_{ij}) \leq \tau_i \leq C_i/g_i, \tag{3.52}$$

Proof. The sigmoidal nonlinearity in Eq. 3.6, is a monotonically increasing function, bound to a range 0 and 1:

$$0 < S(Y_j, \sigma_{ij}, \mu_{ij}, E_{ij}) < 1 \tag{3.53}$$

By replacing the upper-bound of S, in Eq. 3.6 and then substituting the synaptic current of Eq. 3.5, we will have:

$$C_i \frac{dx_i}{dt} = g_i.(x_{leak} - x_i) + \sum_{j=1}^{N} w_{ij}(E_{ij} - x_i), \tag{3.54}$$

$$C_i \frac{dx_i}{dt} = \underbrace{(g_i x_{leak} + \sum_{j=1}^{N} w_{ij} E_{ij})}_{\text{A}} \tag{3.55}$$

$$- \underbrace{(g_i + \sum_{j=1}^{N} w_{ij})}_{\text{B}} x_i, \tag{3.56}$$

$$C_i \frac{dx_i}{dt} = A - B x_i. \tag{3.57}$$

Eq. 3.57 is an ordinary differential equation with solution of the form:

$$x_i(t) = k_1 e^{-\frac{B}{C_i} t} + \frac{A}{B}. \tag{3.58}$$

From this solution, one can derive the lower bound of the system's time constant, τ_i^{min}:

$$\tau_i^{min} = \frac{C_i}{B} = \frac{C_i}{g_i + \sum_{j=1}^{N} w_{ij}}. \tag{3.59}$$

By replacing the lower-bound of S, in Eq. 3.54, the term $\sum_{j=1}^{N} w_{ij}(E_{ij} - x_i)$ becomes zero, therefore:

$$C_i \frac{dx_i}{dt} = \underbrace{g_i x_{leak}}_{\text{A}} - \underbrace{g_i}_{\text{B}} x_i. \tag{3.60}$$

Thus, we can derive the upper-bound of the time constant, τ_i^{max}:

$$\tau_i^{max} = \frac{C_i}{g_i}. \tag{3.61}$$

□

Having a stable varying time-constant significantly enhances the expressiveness of this form of CT-RNNs. Next, we show that the neural state of LTCs is bounded to a finite range.

Lemma 9. *Let x_i denote the state of a neuron i, receiving N synaptic connections as in Eq. 3.5 from the other nodes of a network G. If the dynamics of each neuron is determined by Eq. 3.4, then the hidden state of the neurons on a finite trajectory, $I = [0,T] (0 < T < +\infty)$, is bounded as follows:*

$$\min_{t \in I}(x_{leak_i}, E_{ij}^{min}) \leq x_i(t) \leq \max_{t \in I}(x_{leak_i}, E_{ij}^{max}), \tag{3.62}$$

Proof. Let us insert $M = max\{x_{leak_i}, E_{ij}^{max}\}$ as the membrane potential $x_i(t)$ into Eq. 3.54:

$$C_i \frac{dx_i}{dt} = \underbrace{g_i(x_{leak} - M)}_{\leq 0} + \underbrace{\sum_{j=1}^{N} w_{ij}\sigma(x_j)(E_{ij} - M)}_{\leq 0}. \tag{3.63}$$

The right-hand side of Eq. 3.63, is negative based on the conditions on M, positive weights and conductances, and the fact that $\sigma(x_i)$ is also positive in $\mathbb{R}^N$. Therefore, the left-hand-side must also be negative and if we conduct an approximation on the derivative term:

$$C_i \frac{dx_i}{dt} \leq 0, \quad \frac{dx_i}{dt} \approx \frac{x(t+\delta t) - x(t)}{\delta t} \leq 0, \tag{3.64}$$

holds. by Substituting $x(t)$ with M, we have the following:

$$\frac{x(t+\delta t) - M}{\delta t} \leq 0 \ \rightarrow \ x(t+\delta t) \leq M \tag{3.65}$$

and therefore:

$$x_i(t) \leq \underset{t \in I}{max}(x_{leak_i}, E_{ij}^{max}). \tag{3.66}$$

Now if we substitute the membrane potential, $x_{(i)}$ with $m = min\{x_{leak_i}, E_{ij}^{min}\}$, following the same methodology used for the proof of the upper bound, we can derive

$$\frac{x(t+\delta t) - m}{\delta t} \leq 0 \ \rightarrow \ x(t+\delta t) \leq m, \tag{3.67}$$

and therefore:

$$x_i(t) \geq \underset{t \in I}{min}(x_{leak_i}, E_{ij}^{min}). \tag{3.68}$$

□

Lemma 9 illustrates a fundamental property of LTC-RNNs, namely *state stability* by which we mean that the output of an LTC will never explode as its input rises to infinity. Next, we design a learning platform to train and evaluate LTC instances on real-life examples.

3.7 A learning platform for training LTCs by gradient descent

LTCs' model represented by Eq. 3.11, describes an initial value problem. Solving Eq. 3.11, analytically, is non-trivial due to the nonlinearity of the LTC semantics. The neuronal states of the ODE, however, at any time point T, can be computed by a numerical ODE solver that simulates the system starting from a trajectory $x(0)$, to $x(T)$.

3.7.1 LTC cell-update by a Hybrid ODE Solver

The ODE solver breaks down the continuous simulation interval $[0,T]$ to a temporal discretization, $[t_0, t_1, \ldots t_n]$. As a result, a solver's step involves only the update of the neuronal states from t_i to t_{i+1}.

The explicit Euler method computes the neuronal state at time t_{i+1} by interpreting the right-hand side of Eq 3.3 as the slope of x, i.e.,

$$x(t_{i+1}) = (t_{i+1} - t_i) f(x(t_i)) + x(t_i). \tag{3.69}$$

While this approach is guaranteed to converge to the true solution when deploying an infinitely fine discretization [Gerald, 2012], in practice it is often inaccurate on a finite trajectory of timestamps. Alternatively, more complex approximation methods such as Runge-Kutta [Runge, 1895] provide a better approximation of the system, by the weighted averaging of neuronal state at points between $[t_i, t_{i+1}]$. In practice, the Runge-Kutta method achieves a decent precision on many initial value problems and is considered a standard choice for an ODE solver [Press et al., 2007b]. Described methods struggle with *stiff* ODE instances [Press et al., 2007b], a

Algorithm 2 LTC Cell Update by the Hybrid ODE Solver

Parameters: $\theta = (W, \gamma, \mu, E, C_m, g_l, x_{leak})$,
k = Number of unfolding steps, Δ = time step, N = Number of Neurons
Inputs: Current inputs $I(t)$, Current LTC neurons' state $x(t)$
Output: Next LTC neuron state $x(t+\Delta) = x_{next}$
Function: ODEsolverStep($x(t)$, $I(t)$, Δ, θ)

$$x_{next} = \begin{bmatrix} \frac{x_1(t)C_{m_1}/\Delta + g_{l_1}x_{leak_1} + \sum_{j\in I_{in}} w_{1j}\ \sigma_1\left(\gamma_{1j}(x_j(t)-\mu_{1j})\right)E_{1j}}{C_{m_1}/\Delta + g_{l_1} + \sum_{j\in I_{in}} w_{1j}\ \sigma_1\left(\gamma_{1j}(x_j(t)-\mu_{1j})\right)} \\ \cdots \\ \cdots \\ \frac{x_i(t)C_{m_i}/\Delta + g_{l_i}x_{leak_i} + \sum_{j\in I_{in}} w_{ij}\ \sigma_i\left(\gamma_{ij}(x_j(t)-\mu_{ij})\right)E_{ij}}{C_{m_i}/\Delta + g_{l_i} + \sum_{j\in I_{in}} w_{ij}\ \sigma_i\left(\gamma_{ij}(x_j(t)-\mu_{ij})\right)} \\ \cdots \\ \cdots \\ \frac{x_N(t)C_{m_N}/\Delta + g_{l_N}x_{leak_N} + \sum_{j\in I_{in}} w_{Nj}\ \sigma_N\left(\gamma_{Nj}(x_j(t)-\mu_{Nj})\right)E_{Nj}}{C_{m_i}/\Delta + g_{l_N} + \sum_{j\in I_{in}} w_{Nj}\ \sigma_N\left(\gamma_{Nj}(x_j(t)-\mu_{Nj})\right)} \end{bmatrix}$$

$x_{next} = x(t)$
for $i = 1 \ldots k$ **do**
  x_{next} = ODEsolverStep($x(t)$, $I(t)$, Δ, θ)
end for
return x_{next}

numerical phenomenon that can occur when several variables mutually influence each other. In such situations, both methods either become unstable or require a very fine discretization to compute an accurate solution, ergo exponentially increase the complexity of the algorithm.

On the contrary, the family of implicit ODE solvers provide stable solutions even on stiff ODE instances [Press et al., 2007b]. The implicit Euler method modifies Eq. 3.69 by substituting $f(x(t_i))$ by $f(x(t_{i+1}))$. The resulting equation

$$x(t_{i+1}) = (t_{i+1} - t_i) f(x(t_{i+1})) + x(t_i), \tag{3.70}$$

describes the solution for $x(t_{i+1})$ in an *implicit* format.

We design a new ODE solver that combines the explicit and the implicit Euler method. To this end, we substitute $f(x(t_i))$ in Eq 3.69 by $f(x(t_i), x(t_{i+1}))$, which results in the following expression:

$$x(t_{i+1}) = (t_{i+1} - t_i) f(x(t_i), x(t_{i+1})) + x(t_i). \tag{3.71}$$

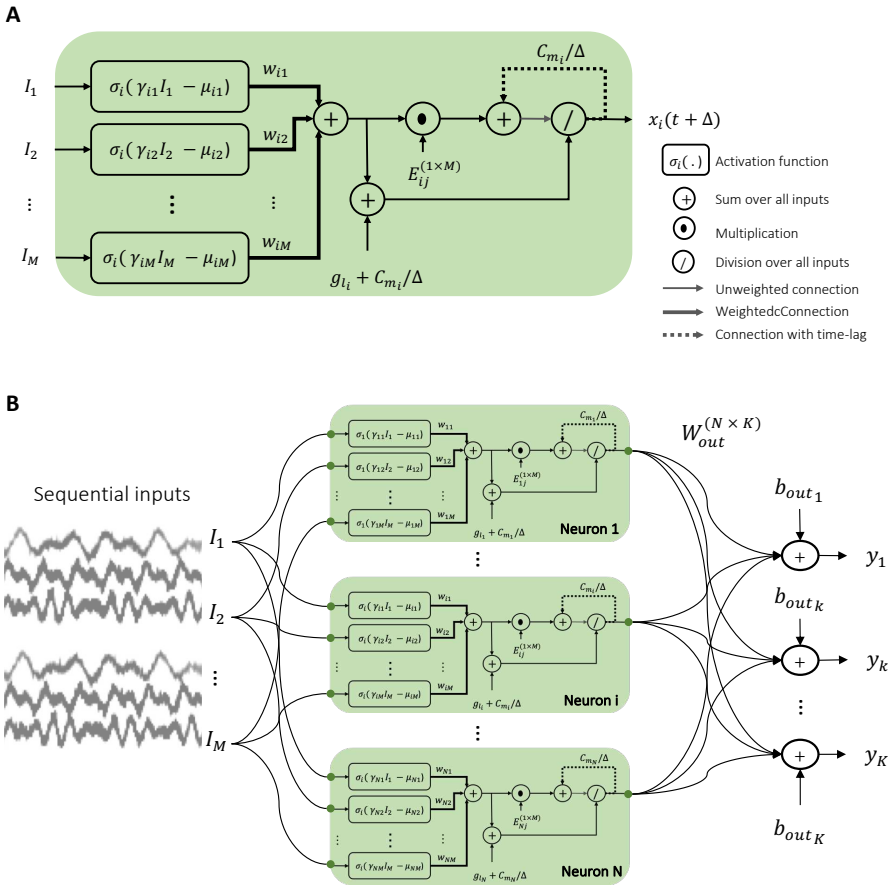


Figure 3.6: A) LTC cell realized by a Hybrid ODE solver. Note that for each input connection there exists an activation function associated to it. This feature implies that the nonlinearity of a neuron is chiefly resulted by the nonlinear synaptic transmissions. B) An LTC network realized by a Hybrid ODE solver. Note that for each input connection there exists an activation function associated to it. One can add arbitrary synaptic connections to this networks, (e.g. recurrent connection between cells), by passing the output of a cell i through a synaptic nonlinearity of the form $\sum_{j\in x} w_{ij}^{recurrent}\ \sigma_i(\gamma_{ij}^{recurrent}(x_i(t) - \mu_{ij}^{recurrent}))$.

we refer to this approach as the *hybrid* Euler method. In particular, we replace only the $x(t_i)$ that occur linearly in f by $x(t_{i+1})$. As a result, Eq 3.71 can be solved for $x(t_{i+1})$, symbolically.

Based on Eq. 3.71, given a system of ordinary differential equations of LTC neurons, the hybrid solver computes an update for the next state of each neuron i, by the following expression:

$$x_i(t+\Delta) := \frac{x_i(t)C_{m_i}/\Delta + g_{l_i}x_{leak_i} + \sum_{j\in I_{in}} w_{ij}\ \sigma_i\Big(\gamma_{ij}(x_j(t) - \mu_{ij})\Big)E_{ij}}{C_{m_i}/\Delta + g_{l_i} + \sum_{j\in I_{in}} w_{ij}\ \sigma_i\Big(\gamma_{ij}(x_j(t) - \mu_{ij})\Big)}. \tag{3.72}$$

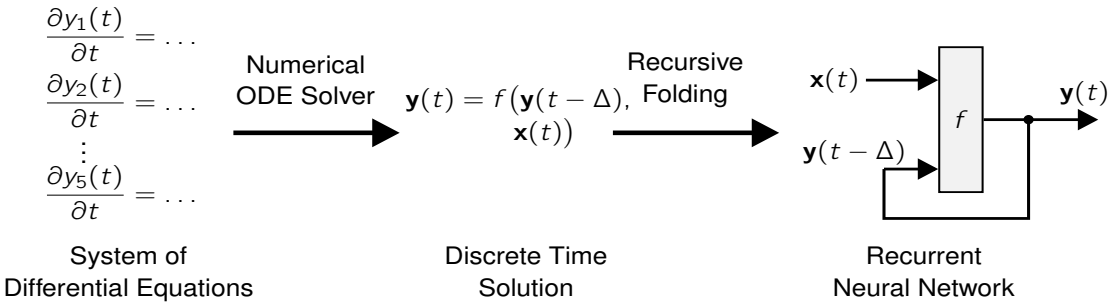


Figure 3.7: ODE to recurrent cell. Each hidden neuron's state dynamics has been discretized by an ODE solver, and then unfolded in a recursive fashion to build a recurrent network cell.

In this expression, the set I_{in} represents the set of inputs that are presynaptic to neuron i, and Δ accounts for the solver's step. Algorithm 2 represents how to implement an LTC network, given the parameter space θ.

The time complexity of the algorithm is $O(N^2 \times k)$, where N is the number of neurons and k the number of unfolding steps. N is usually much larger than k, which can be neglected (e.g. in our experiments $k = 6$). Intuitively, a dense version of an LTC network with N neurons, and a dense version of an LSTM network with N cells, would be of the same complexity degree.

Accordingly, the structure of a single LTC neuron realized by the hybrid ODE solver introduced, can be sketched as shown in Fig. 3.6A. This cell can be utilized to structure networks of LTCs, with a fully connected network representation. An implementation note for having cells of a network recurrently synapse into each other is to consider that every additional synapse to the architecture comes with its own activation function as illustrated in Fig. 3.6B. For instance, in order to add recurrent connection between neurons, one can pass the output of a cell i through a synaptic nonlinearity of the form $\sum_{j \in x} w_{ij}^{recurrent} \ \sigma_i(\gamma_{ij}^{recurrent}(x_i(t) - \mu_{ij}^{recurrent}))$.

3.7.2 Training LTC networks by backpropagation through time

We aim to train the LTC models realized by ODE solvers, by the backpropagation through time algorithm. Therefore, it is vital to investigate how well a solver enables backward error propagation. It has been previously shown that the magnitude of the error backpropagating from one time-step to the previous time-step significantly influences the learnability of RNNs [Pascanu et al., 2013, Hochreiter and Schmidhuber, 1997].

We investigate this property, experimentally, by comparing the performance of LTC models realized by the introduced hybrid solver, to the vanilla explicit and the Runge-Kutta solvers in a group of time-series classification tasks. The solver's output which corresponds to a vector of neural states can be recursively folded to build up a recurrent neural network instance, as depicted in Fig. 3.7. Back-propagation through time learning algorithm is deployed to train the resulting LTC-RNN network. The corresponding learning algorithm is then summarized in Algorithm 3.

Algorithm 3 uses a vanilla Stochastic gradient decent (SGD), to obtain network parameters. One

Algorithm 3 LTC Training Algorithm for time-step sequence classification and regression tasks

Inputs: Dataset of traces $[I(t), y(t)]$ of length T, RNN $= f(I,x)$
Parameter: Loss function $L(\theta)$, initial parameters θ_0, learning rate α, Output weights $= W_{out}$, output bias $= b_{out}$
Output: Training parameters θ
for $i = 1 \ldots$ number of training steps **do**
  $(I_b, y_b) =$ Sample training batch
  $x =$ All zeros initial neural state
  for $j = 1 \ldots T$ **do**
    $x = f(I(t), x)$
    $\hat{y}(t) = W_{out}.x + b_{out}$
    $L_{total} = \sum_{j=1}^{T} L(y_j(t), \hat{y}_j(t))$
    $\nabla L(\theta) = \frac{\partial L_{total}}{\partial \theta}$
    $\theta = \theta - \alpha \nabla L(\theta)$
  end for
end for
return θ

can easily substitute this with a more performant variant of the SGD algorithm such as Adam [Kingma and Ba, 2014], which we use for our experimentation. Correspondingly, the training algorithm has a complexity of $O(N^2 \times k \times t)$, with N neurons, k ODE steps, and a sequence length of t.

3.8 LTCs performance in time-series prediction compared to other RNNs

We evaluate the performance of LTCs modeled by different ODE solvers against the state-of-the-art discretized RNN, the LSTM networks, and CT-RNNs, in a series of real-life time-series processing tasks from the UCI Machine Learning Repository [Dua and Graff, 2017]. The results of our experimentation are summarized in Table 3.2. Here, we first briefly describe the objective of each task and discuss results subsequently.

The first test-case concerns the temporal segmentation of hand gestures. The dataset consists of seven recordings of individuals performing a sequence of hand gesticulations [Wagner et al., 2014]. The input features at each time step are comprised of 32 data points recorded from a motion detection sensor. The output, at each time step, represents one of the five possible hand gestures; rest position, preparation, stroke, hold, and retraction. The objective is to train a classifier to detect hand gestures from the motion data.

The objective of the second task is to detect whether a room is occupied by observations recorded from five physical sensor streams, such as temperature, humidity, and CO2 concentration sensors [Candanedo and Feldheim, 2016]. Input data and labels are sampled in a one-minute interval.

Table 3.2: Test performance of RNN models. Numbers depict the mean + standard deviation of the validation set performance of RNN models with the dedicated metric in each experiment. n=5.

Dataset	Metric	LSTM	CT-RNN	LTC-RNN (hybrid euler) **(ours)**
Hand gesture segmentation	(accuracy)	64.57% ± 0.59	59.01% ± 1.22	**68.04**% ± 2.35
Room occupancy detection	(accuracy)	93.18% ± 1.66	**94.54**% ± 0.54	94.47% ± 0.53
Human activity recognition	(accuracy)	**95.85**% ± 0.29	95.73% ± 0.47	95.67% ± 0.57
Traffic volume prediction	(squared error)	0.169 ± 0.004	0.224 ± 0.008	**0.099** ± 0.009
Ozone level forecasting	(F1-score)	0.284 ± 0.025	0.236 ± 0.011	**0.302** ± 0.015

The third task involves the recognition of human activities, such as walking, sitting, and standing, from inertial measurements of the user's smartphone [Anguita et al., 2013]. Data consists of recordings from 30 volunteers performing activities form six possible categories. Input variables are filtered and are pre-processed to obtain a feature column of 561 items at each time step.

The goal of the fourth experiment is to predict the hourly westbound traffic volume at the US Interstate 94 highway between Minneapolis and St. Paul. Input features consist of weather data and date information such as local time and flags indicating the presence of weekends, national, or regional holidays. The output variable represents the hourly traffic volume, which we normalized by its annual mean and standard deviation.

The objective of the final task is to forecast ozone days, i.e., days when the local ozone concentration exceeds a critical level. Input features consist of wind, weather, and solar radiation readings. The dataset is imbalanced with a 1:15 ratio, due to the low number of ozone days present in the dataset. Consequently, we assign an importance rate of 15 times more substantial for the ozone day samples compared to the other samples. Moreover, we report the F_1-score instead of standard accuracy (higher score is better). Read more about the experimental setup in the Supporting Information.

As it is illustrated in Table 3.2, we quantify the improvement of the LTCs implemented by the hybrid solver, in three test-cases and their highly competitive performance to the other RNNs compared to the other two tasks. In these real-life test-cases, a performance improvement between 5% to 70% is achieved by the LTC networks compared to the best performing state-of-the-art LSTM architecture.

3.9 Experimental Setup

In this section, we provide a detailed description of the experimental setup.

3.9.1 Gesture segmentation task

The dataset consists of seven recordings of humans performing hand gestures [Wagner et al., 2014]. The input features at each time step consist of 32 items, and the output represents one of the five mutually exclusive categories. We use the categorical classification accuracy as our performance metric. We cut each of the seven recordings into overlapping sub-sequences of exactly 32 time-steps. We randomly separate all sub-sequences into non-overlapping training

(75%), validation (10%), and test (15%) set. Input features are normalized to have zero mean and unit standard deviation.

3.9.2 Room occupancy detection

The original dataset consists of a pre-defined training and test set [Candanedo and Feldheim, 2016]. Input data and binary labels are given in a one-minute interval. We use the binary classification accuracy as our performance metric. We cut the sequences of each of the two sets into a training and test set of overlapping sub-sequences of exactly 32 time-steps. Note that no item from the test set is leaking into the training set during this process. Input features of all data are normalized by the mean and standard deviation of the training set, such that the training set has zero mean and unit standard deviation. We select 10% of the training set as the validation set.

3.9.3 Human activity recognition

Like with the occupancy experiment, the dataset is split into a training and test set [Anguita et al., 2013]. The output variable represents one of six activity categories at each time step. We employ the categorical classification accuracy as our performance metric. The original data is already preprocessed by temporal filters. The accelerometer and gyroscope sensor data were transformed into 561 features in total at each time step. We cut the sequences of the training and test set into overlapping sub-sequences of exactly 32 time-steps. We select 10% of the training set as the validation set.

3.9.4 Traffic volume prediction

The original data consists of hourly recordings between October 2012 and October 2018, provided by the Minnesota Department of Transportation and OpenWeatherMap. We select the seven columns of the data as input features: 1. Flag indicating whether the current day is a holiday, 2. The temperature in Kelvin normalized by annual mean, 3. Amount of rainfall, 4. Amount of snowfall, 5. Cloud coverage in percent, 6. Flag indicating whether the current day is a weekday, and 7. time of the day preprocessed by a sine function to avoid the discontinuity at midnight. The output variable was normalized to have zero mean and unit standard deviation. We used the mean-squared-error as training loss and evaluation metric. We split the data into overlapping sequences lasting 32 hours. We randomly separate all sequences into non-overlapping training (75%), validation (10%), and test (15%) set.

3.9.5 Ozone level forecasting

The original dataset consists of daily data points collected by the Texas Commission on Environmental Quality (TCEQ). We split the 6-years period into overlapping sequences of 32 days. A day was labeled as ozone day if, for at least 8 hours, the exposure to ozone exceeded 80 parts per billion. Inputs consist of 73 features, including wind, temperature, and solar radiation data. The binary predictor variable has a prior of 6.31%, i.e., expresses a 1:15 imbalance. For the training procedure, we weighted the cross-entropy loss at each day, depending on the label.

Labels representing an ozone day were assigned 15 times the weight of a non-ozone day. In roughly 27% of all samples, some of the input features were missing. To not disrupt the continuity of the collected data, we set all missing features to zero. Note that such zeroing of some input features potentially negatively affects the performance of our RNN models compared to non-recurrent approaches and filtering out the missing data. Consequently, ensemble methods and model-based approaches, i.e., methods that leverage domain knowledge [Zhang and Fan, 2008], can outperform the end-to-end RNNs studied in our experiment. We randomly split all sub-sequences into training (75%), validation (10%), and test (15%) set.

3.9.6 Model selection procedure

Each training procedure consists of 200 training epochs. At each training epoch, we create a backup of the model weights and evaluate the score of the model on the validation set. At the end of the training process, we restore the model weights that scored the best validation score and evaluated this model on the test set. We repeat the experiment above five times for each model with different weight initialization.

3.9.7 Model size

In order to have a fair comparison between different RNN model, the size of all RNN hidden state is fixed to 32 units.

3.9.8 ODE solvers setting

We compared the explicit Euler, Runge-Kutta with four intermediate evaluations (RK4), and our hybrid Euler method. For our UCI benchmark, we simulated our LTC ODE at a temporal discretization that is three times finer than the input and output sampling frequency. In essence, each RNN time-step consists of exactly three ODE solver steps.

3.10 Extrapolation with LTCs

Throughout our experimentation, we also observed that LTCs possess special skills in learning the underlying dynamics of a system beyond the domains and ranges they have been trained over. In order to quantify this, we took the first steps towards investigating their extrapolation capabilities in the classification of unseen input time-series to unseen domains. We designed a novel experiment to assess how well RNNs can classify a new time-series with unseen frequency properties to unseen classes. The motivation behind performing the experiment is that neural networks have been typically shown to successfully generalize to new data drawn from their original training distribution [Novak et al., 2018, Keskar et al., 2016, Kawaguchi et al., 2017]. However, they often have difficulty with extrapolation or modeling data from outside of the training distribution [Haley and Soloway, 1992]. Typically, since these networks are trained to be consistent with a given dataset, there are no guarantees of their behavior outside of that dataset. While this variability is useful for specific applications, such as epistemic uncertainty estimation [Lakshminarayanan

et al., 2017, Pearce et al., 2018], learning to successfully extrapolate also presents the promise of temporal forecasting into the future as well as increased sample efficiency.

In this experiment, as shown in Fig. 3.8, a neural network with two inputs, one output, and k hidden nodes, successively receives the elements $x_1(t)$ and $x_2(t)$ of two time series of length N, generated from two sinusoidal signals, $x_1(t) = sin(f_1 * t)$ and $x_2(t) = sin(f_2 * t)$. The time increment is dt, the time horizon is T, and $N = T/dt$. For a given geometric shape as a classification boundary (e.g., a half disc in the f_1-f_2 coordinate system), the neural network has to first learn the frequencies f_1 and f_2 from x_1 and x_2, and than output *class one* if (f_1,f_2) is a point within the geometric shape, and *class 0* if it is not. For training, the frequency f_1 is chosen in the domain $[0,5,2.25]Hz$ and f_2 from $[0.5,4]Hz$. We then train LTCs, discretized (LSTM) and continuous-time RNNs to fit the input sequences to their corresponding class-domain.

The trained networks are then evaluated on time series x_1 and x_2 generated as before from the sinusoidal input signals, respectively, but this time with frequencies both inside and outside the range of their training domain; that is, f_1 now ranges in $[0.5,4]Hz$ and f_2 in $[0.5,7.5]Hz$. The question aimed to be elucidated, is how these networks classify the unseen time series in unseen classification domains. A human observer that would be able to learn the frequencies of the sinusoidal signals from their respective time series x_1 and x_2 would intuitively predict symmetry for classifying the signals. In other words, one would guess that the hidden part of the geometric shape is symmetric to the one used for training. Hence, one classifies the time series whose frequencies (f_1, f_2) lie within the symmetric geometric shape as *class 1*, and the other as *class 0*.

Now for the trained networks, the result of their predictions is presented in Fig. 3.8. While LSTM and CT-RNN can perfectly fit the training data, predictions in the unseen domain appear arbitrary. The predictive output of the LTC-RNNs, on the other hand, appears to be symmetric and in most cases and is closer to what a human observer would expect. This is a surprising observation about the extrapolation performance of LTCs which opens up prospective research questions to be addressed. In the context of the extrapolation capability of neural networks, a recent line of work [Martius and Lampert, 2016, Sahoo et al., 2018] reformulates the deep network outputs to instead model analytical equations governing the output variables, in order to perform extrapolation in the output space. Additionally, similar approaches have been used to learn analytical partial differential equations directly from data [Long et al., 2018b]. LTCs can presumably leverage the extrapolation capability of neural networks not only to the time-series beyond range values but also to time-series classification of unseen data to unseen domains. Moreover, in comparison to the other methods which alter the network's computational units [Martius and Lampert, 2016, Sahoo et al., 2018, Trask et al., 2018], LTCs suggests a brain-inspired universal approximator model for governing novel scenarios, without the need to varying the network's fundamental units.

For each model, we empirically set the number of training epochs to achieve a training loss lower than 0.05. For instance, while larger LSTM models reach such threshold after 1000 iterations, the LTC networks require up to 50,000 training steps. For each configuration, i.e., (model, size, and shape) tuple, we created ten figures starting from random initial parameters and selected an appropriate model, as shown in Figure 3.8. Furthermore, for each configuration, we generated five figures starting from different random initialization without applying any filtering.

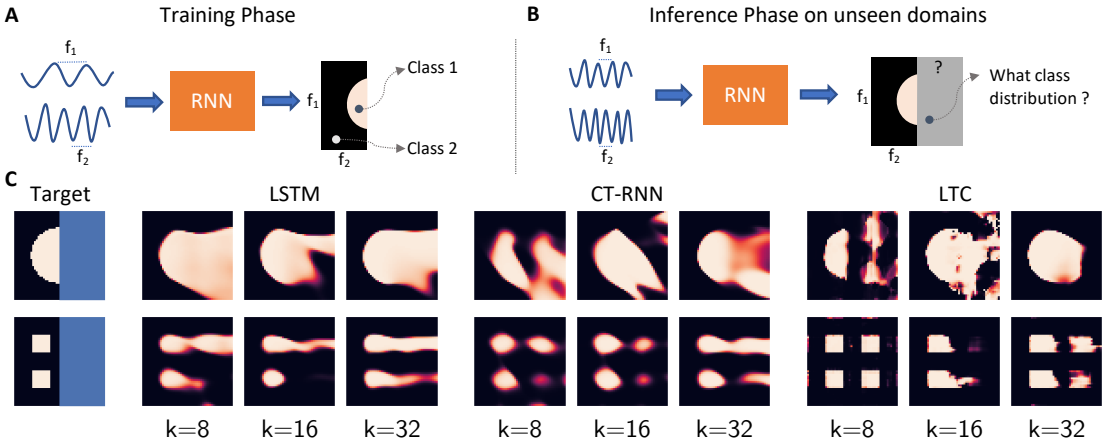


Figure 3.8: Frequency extrapolation experiment. **A**) Two sinusoidal inputs with two different frequencies are applied to LSTM, CT-RNN, and LTC networks. The target is composed of the frequencies of the two inputs, and the class domain distribution is structured in 2 different scenarios. **B**) The objective is to see the extrapolation performance of RNNs after training. **C**) Each column represents the output class prediction of a trained recurrent network with k hidden nodes. The columns also show the network's extended predictions in the domains they have not been explored during training. The predictions of the LTCs in the unseen domain appear closer to the expectations of a human observer, compared to that of other networks.

3.11 Conclusions

We formulated a new brain-inspired RNN instance, namely the liquid-time constant networks (LTC). We analyzed their properties and found activity bounds on their neural state dynamics. We illustrated how their dynamics are more expressive than CT-RNNs. We showed their superior performance compared to the other RNNs in time-series processing tasks and experimentally demonstrated that LTCs possess compelling generalization capabilities, due to their biologically plausible dynamical representation, which resembles a dynamic causal model.

Due to their expressive dynamics and their causal semantics, we speculate that LTCs might be able to learn dynamics beyond the range and distribution they have been trained over. To investigate this phenomenon further, we took the first steps and conducted a toy experiment, in which we evaluate the extrapolation capability of different RNN models compared to that of LTCs. The experiment demonstrates that symmetrical extrapolation was achievable by LTC models. This property demands more rigorous attention, which is going to be the focus of our prospective research on LTCs.

We believe that the LTCs establish a step forward towards the realization of creative neural information processing systems, with computational models closer to that of natural learning systems. Next, we explore their ability to control dynamical systems.

CHAPTER 4

Ordinary Neural Circuits with LTCs

4.1 Motivation

In this chapter, we wish to deploy LTC neural model into the development of a new class of machine learning algorithms for robot control[1]. Through natural evolution, the subnetworks within the nervous system of the nematode, *C. elegans*, structured a near-optimal wiring diagram from the *wiring economy principle*[2] perspective [White et al., 1986, Pérez-Escudero and de Polavieja, 2007]. Its stereotypic brain composed of 302 neurons connected through approximately 8000 chemical and electrical synapses [Chen et al., 2006]. The wiring diagram therefore, establishes a 91% sparsity level and outstandingly gives rise to high-degrees of controllability, to process complex chemical stimulations [Bargmann, 2006], express adaptive behavior [Ardiel and Rankin, 2010], and to control muscles [Wen et al., 2012].

This property is particularly attractive to the machine learning community that aims at reducing the size of fully-connected neural networks to sparser representations while maintaining the great output performance [LeCun et al., 1990b, Hassibi and Stork, 1993, Han et al., 2015, Hinton et al., 2015, Frankle and Carbin, 2018]. In this regard, the lottery ticket hypothesis [Frankle and Carbin, 2018], suggested an algorithm to find sparse subnetworks (winning tickets) within a dense, randomly initialized feedforward neural network, which can achieve comparable (and sometimes greater) performance to the original network, when trained separately [Frankle and Carbin, 2018, Zhou et al., 2019, Morcos et al., 2019]. The lottery ticket hypothesis motivated us to investigate whether subnetworks (neural circuits) within the natural nervous systems are already formulation of *winning tickets* originated from the natural evolution?

To study this question fundamentally, we take a computational approach to analyze neural circuit models from the nervous system of the worm. The reason is that the function of many

[1] Supplementary materials and code for all experiments of this chapter is available online at: `https://github.com/raminmh/ordinary-neural-circuits`

[2] Under proper functionality of a network, the wiring economy principle proposes that its morphology is organized such that the cost of wiring its elements is minimal [Pérez-Escudero and de Polavieja, 2007].

circuits within its nervous system have been identified [Wicks and Rankin, 1995, Chalfie et al., 1985a, Li et al., 2012, Nichols et al., 2017, Kaplan et al., 2019], and simulated [Islam et al., 2016, Sarma et al., 2018, Gleeson et al., 2018], which makes it a suitable model organism for further computational investigations.

The general network architecture in *C. elegans* establishes a hierarchical topology from sensory neurons (source nodes) through upper interneuron and command interneurons down to motor neurons, sink nodes, (See Fig. 4.1A). In these neuronal circuits, typically, interneurons establish highly recurrent wiring diagrams with each other while sensors and command neurons mostly realize feedforward connections to their downstream neurons. An example of such a structure is a neural circuit shown in Fig. 4.1B, the Tap-withdrawal (TW) [Rankin et al., 1990], which is responsible for inducing a forward/backward locomotion reflex when the worm is mechanically exposed to touch stimulus on its body. The circuit has been characterized in terms of its neuronal dynamics [Chalfie et al., 1985a]. It comprises eleven neuron classes which are wired by thirty chemical and electrical synapses. Is TW a *Winning Ticket* compared to networks of the same size, from any perspective?

4.1.1 TW graph realizes the highest maximum flow rate

Let us first define the *maximum flow problem* [Shiloach and Vishkin, 1982]:

Definition 5. *For a given graph $G(V,E)$, with $s,t \in V$ source and sink nodes, respectively:*

- *The **capacity (weight)** of an edge is the mapping $c : E \rightarrow \mathbb{R}^+$, declared by c_e,*
- *A **Flow** is a mapping $f : E \rightarrow \mathbb{R}^+$, denoted by f_e, from node u to v, if: 1) $f_e \leq c_e$ for each $e \in E$. 2) $\sum_{Inputs\ to\ v} f_e = \sum_{output\ \ from\ v} f_e$ for all $v \in V$ except source and sink nodes,*
- *The **flow rate** is denoted by $|f| = \sum_{s \rightarrow v} f_{sv}$, where s is the source of G. This value depicts the amount of flow passing from a source node to a chosen sink node.*
- *The **maximum flow problem** is to maximize $|f|$.*

The maximum flow problem is typically used for sparse directed networks in order to assess their input/output propagation performance. The TW circuit is a sparse directed network, and therefore, we chose to evaluate its propagation properties by computing the maximum flow rate.

The TW neural circuit realizes a higher maximum flow rate from arbitrary chosen source to sink node, compared to randomly wired networks of the same size. Formally, given a directed weighted graph $G(V,E)$, with V vertices, $E \ll V^2$ edges and:

- $S \subset V$, $S = \{s_1, ..., s_k\}$ source (sensory neurons),
- $T \subset V$, $T = \{t_1, ..., t_n\}$ sink (motor neurons),
- $I \subset V$, $I = \{i_1, ..., i_{N_i}\}$ interneurons,
- $C \subset V$, $I = \{c_1, ..., c_{N_c}\}$ command neurons,

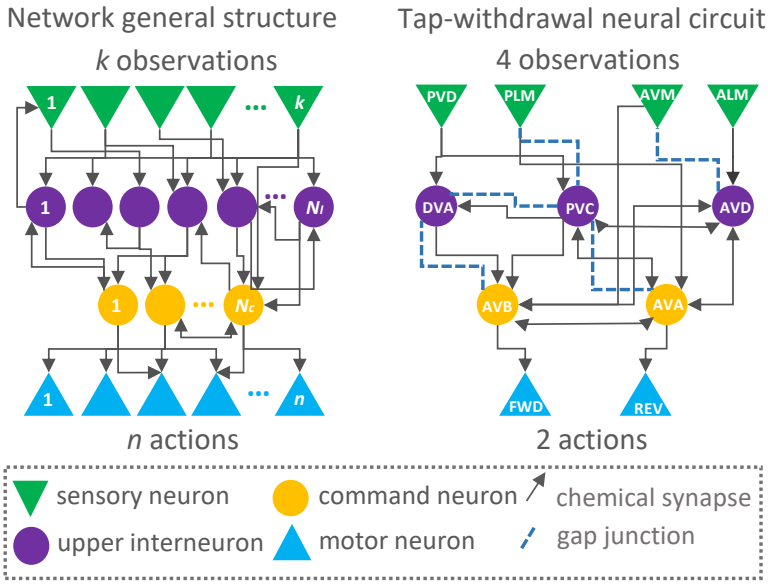


Figure 4.1: Left: C. elegans' general neuronal circuit structure. Right: Tap-Withdrawal (TW) neural circuit schematic. Total number of interneurons $= N_i + N_C$. We preserve the TW circuit wiring topology, model its dynamics by computational models, and deploy a search-based reinforcement learning algorithm to control robots.

then, the highest maximum flow rate is achievable for randomly-weighted and -wired networks (with E edges, when the architecture of the network approaches that of randomly-weighted TW. To demonstrate this claim quantitatively, we construct 40000 randomly-wired networks and compare their max-flow rate to randomly-weighted TW. We witnessed an enhanced max-flow rate between 1% and 17% when a network is constrained to be wired similar to TW. (See details in Section 4.2). Accordingly, this finding motivated us to explore the dynamics of the TW circuit from a control theory perspective.

4.1.2 TW can be trained to govern standard control tasks

The behavior of the TW reflexive response is substantially similar to the control agent's reaction in standard control settings such as a controller acting on driving an underpowered car, to go up on a steep hill, known as the *Mountain Car* [Singh and Sutton, 1996], or a controller operating on the navigation of a rover robot that plans to go from point A to B.

As discussed in Chapter 3, the biophysical neuronal and synaptic models such as LTC express useful properties: I) In addition to the nonlinearities expressed by the neurons' hidden state, synapses possess additional nonlinearity. This property results in realizing complex dynamics with a fewer number of neurons [Hasani et al., 2018b]. II) Their dynamics are set by grounded biophysical properties, which ease the interpretation of the network's hidden dynamics.

We construct instances of the TW network obtained by learning its parameters and define these

learning systems as *ordinary neural circuits* (ONC). We experimentally investigate ONC's properties in terms of their learning performance, their ability to solve tasks in different reinforcement learning domains, and introduce ways to interpret their internal dynamics. For this purpose, we preserve the wiring structure of an example ONC (the TW circuit) and adopt a search-based optimization algorithm for learning the neuronal and synaptic parameters of the network. We discover that sparse ONCs (*Natural lottery winners*) not only establish a higher maximum flow rate from any arbitrary source to sink node but also when trained in isolation for control tasks, significantly outperform randomly wired networks of the same size and in many cases contemporary deep learning models with larger capacities.

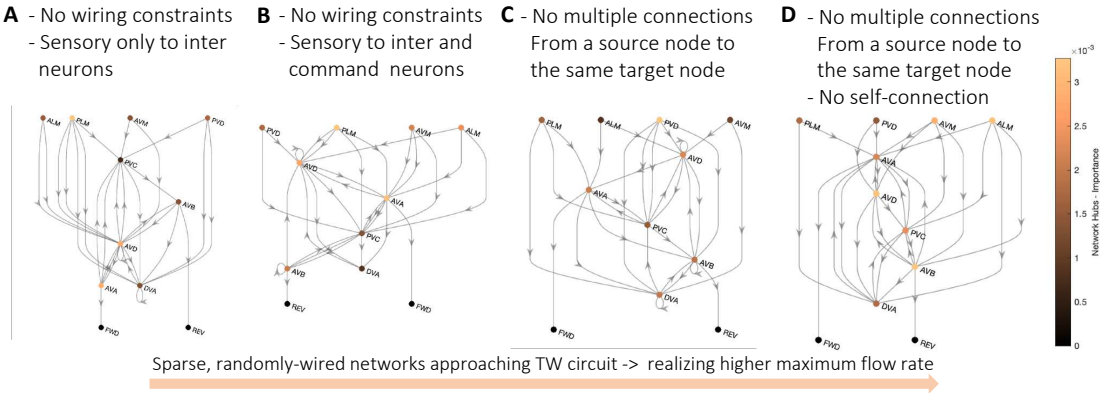


Figure 4.2: Sparse and randomly wired network samples. A to D indicate random neural circuits with the same number of elements as in TW, wired with modified constraints in Algorithm 4: A) In Step 1, target $=$ Rand($^{I}P_1$) B) In Step 1, target $=$ Rand($^{I\&C}P_1$) C) In Step 1 and Step 2, if the tuple $(source_e, target_e)$ is repeated, remove and loop again. D) In Step 1 and Step 2, if $(source_e, target_e)$ is repeated, remove and loop again. In Step 2, if $source_e = target_e$, remove the selection, and loop again. The colorbar represents network hubs – nodes with highest number of inward/outward edges.

4.1.3 Contributions of this chapter

- Quantitative illustration of achieving the highest maximum flow rate for randomly wired sparse networks, when their architecture gets closer to an instance of ONCs, the TW.
- Demonstration of the performance of a compact ONC as an interpretable controller in a series of control tasks and the indication of its superiority compared to similarly structured networks and contemporary deep learning models.
- Experiments with ONCs in simulated and physical robot control tasks, including the autonomous parking of a reak mobile robot. This is performed by equipping ONCs with a search-based RL optimization scheme.
- Interpretation of the internal dynamics of the learned policies. We introduce a novel computational method to understand continuous-time network dynamics. The technique (Definition 6) determines the relation between the kinetics of sensory/interneurons and a

motor neuron's decision. We compute the magnitude of a neuron's contribution (positive or negative), of these hidden nodes to the output dynamics in determinable phases of activity, during the simulation.

4.2 Design Ordinary Neural Circuits

In this section, we first briefly describe the structure and dynamics of the tap-withdrawal neural circuit as an instance of ONCs. We then delve into the graph theory properties of the network to motivate the choice of the TW circuit as the *natural lottery winner* for control. We then introduce the mathematical neuron and synapse models utilized to build up the circuit, as an instance of ordinary neural circuits.

4.2.1 Tap-Withdrawal Neural Circuit

A mechanically exposed stimulus (i.e., tap) to the petri dish in which the worm inhabits, results in the animal's reflexive response in the form of a forward or backward movement. This response has been named as the *tap-withdrawal reflex*, and the circuit identified to underlay such behavior is known as the *tap-withdrawal* (TW) neural circuit [Rankin et al., 1990]. The circuit is shown in Fig. 4.1. It is composed of four sensory neurons, PVD and PLM (posterior touch sensors), AVM and ALM (anterior touch sensors), five interneuron classes (AVD, PVC, AVA and AVB, DVA), and two subgroups of motor neurons which are abstracted as forward-locomotory neurons, FWD, and backward locomotory neurons, REV. Interneurons recurrently synapse into each other with excitatory and inhibitory synaptic links. TW consists of 28 synapses connecting 11 neurons.

4.2.2 Maximum Flow Rate in ONCs versus Other Networks

The TW neural circuit, is wired with a set of network-design constraints. Formally, given V vertices and E edges,I) it realizes a 77% network sparsity. II) The structure exclusively determines four distinct layers of neurons: $S \subset V$, $S = \{s_1, ..., s_k\}$ source (sensory neurons), $T \subset V$, $T = \{t_1, ..., t_n\}$ sink (motor neurons), $I \subset V$, $I = \{I_1, ..., I_{N_i}\}$ interneurons, and $C \subset V$, $I = \{C_1, ..., C_{N_c}\}$ command neurons. III) Sensory nodes unidirectionally synapse into upper interneurons with 40% of the total number of connections. IV) interneurons and command neurons recurrently synapse into each other (without any self-connections) by 53% of the total number of connections. V) Command neurons exclusively synapse into motor neurons by the rest of the synapses.

We discovered that with the construction of randomly-wired sparse networks while applying the aforementioned TW constraints, we can achieve the highest maximum flow rate for such networks. To demonstrate this quantitatively, we developed Algorithm 4 to design random networks with a series of assumptions gradually increased to satisfy TW constraints. We then compute the ratio of the average maximum flow (computed by a tree-search max-flow algorithm [Boykov and Kolmogorov, 2004]) from sensory nodes to motor neurons of the TW circuit, to the obtained networks and report results in Table 4.1. The ratio approaches 1, which indicates that networks

Algorithm 4 Design ONC-like random networks

S =**sensory,** T =**motor,** I =**interneuron,** C =**command,** E =**No. of synapses**
Generate E synapse weights, $W \sim Binomial(E, \rho)$
Step 1
for e in range $[1, 40\%E]$ **do**
  source = Rand($^{S}P_1$), target = Rand($^{I\&C}P_1$)
end for
connect source and target
Step 2
$E_{ic} = 53\%E$ selected from the remainder of the synapses
for e in E_{ic} **do**
  source = Rand($^{I\&C}P_1$), target = Rand($^{I\&C}P_1$)
  connect source and target
end for
Step 3
Connect $C = \{c_1, ..., c_{N_c}\}$, one-to-one to $T = \{t_1, ..., t_n\}$
Return $Random_{TW}$ Graph

Table 4.1: Ratio of the average maximum flow rate of TW compared to variations of other random networks shown in Fig. 4.2. The ratio of the max flow of the Random networks of each subcategory to the max flow of TW has been simulated for 10000 times. Total No. of networks tested = 40000.

Networks	Average $\frac{MaxFlow_{TW}}{MaxFlow}$ of FWD neuron	Average $\frac{maxflow_{TW}}{MaxFlow}$ of REV neuron
Fig. 4.2A	1.15 ± 0.01	1.14 ± 0.01
Fig. 4.2B	1.12 ± 0.005	1.10 ± 0.01
Fig. 4.2C	1.03 ± 0.003	1.04 ± 0.005
Fig. 4.2D	1.01 ± 0.001	1.01 ± 0.001

designed based on the TW constraints would benefit from a better max-flow rate, compared to less-constrained randomly connected networks.

4.3 Sensory inputs and Motor outputs for LTCs

For interacting with the environment, We introduced sensory and motor neuron models. A *sensory component* consists of two neurons S_p, S_n and an input variable, x. S_p gets activated when x has a positive value, whereas S_n fires when x is negative. The potential of the neurons S_p, and S_n, as a function of x, are defined by an affine function that maps the region $[x_{min}, x_{max}]$ of the system variable x, to a membrane potential range of $[-70mV, -20mV]$. Mathematically, the potential of

the neurons S_p, and S_n, as a function of x, can be expressed as

$$S_p(x) := \begin{cases} -70mV & \text{if } x \leq 0 \\ -70mV + \frac{50mV}{x_{max}} x & \text{if } 0 < x \leq x_{max} \\ -20mV & \text{if } x > x_{max} \end{cases} \tag{4.1}$$

$$S_n(x) := \begin{cases} -70mV & \text{if } x \geq 0 \\ -70mV + \frac{50mV}{x_{min}} x & \text{if } 0 > x \geq x_{min} \\ -20mV & \text{if } x < x_{min}. \end{cases} \tag{4.2}$$

This maps the region $[x_{min}, x_{max}]$ of system variable x, to a membrane potential range of $[-70mV, -20mV]$. Note that the potential range is selected to be close to the biophysics of the nerve cells, where the resting potential is usually set around -70 mV, and a neuron can be considered to be active when it has a potential around -20 mV [Hasani et al., 2017b].

Similar to sensory neurons, a *motor component* is composed of two neurons M_n, M_p and a controllable motor variable y. Values of y is computed by $y := y_p + y_n$ and an affine mapping links the neuron potentials M_n and M_p, to the range $[y_{min}, y_{max}]$, as follows:

$$y_p(M_p) := \begin{cases} y_{max}, & \text{if } M_p > -20mV \\ \frac{y_{max}(M_p + 70mV)}{50mV}, & \text{if } M_p \in [-70, -20]mV \\ 0, & \text{if } M_p < -70mV \end{cases} \tag{4.3}$$

$$y_n(M_n) := \begin{cases} y_{min}, & \text{if } M_n > -20mV \\ \frac{y_{min}(M_n + 70mV)}{50mV}, & \text{if } M_n \in [-70, -20]mV \\ 0, & \text{if } M_n < -70mV \end{cases} \tag{4.4}$$

FWD and REV motor classes (Output units) in Fig. 4.1B, are modeled in this fashion.

We use the hybrid ODE solver to simulate the ONCs realized by the LTC model. The solver's complexity for each time step Δ_t is $\mathcal{O}(|\#\text{ neurons}| + |\#\text{ synapses}|)$.

In the next section, we introduce the optimization algorithm used to re-parametrize the tap-withdrawal circuit.

4.4 Search-based Optimization Algorithm

In this section we formulate a *Reinforcement learning* (RL) setting for tuning the parameters of a given neural circuit to control robots. The behavior of a neural circuit can be expressed as a policy $\pi_\theta(o_i, s_i) \mapsto \langle a_{i+1}, s_{i+1} \rangle$, that maps an observation o_i, and an internal state s_i of the circuit, to an action a_{i+1}, and a new internal state s_{i+1}. This policy acts upon a possible stochastic environment

Algorithm 5 Adaptive Random Search

Input: A stochastic objective indicator f and a starting parameter θ, noise scale σ, adaption rate $\alpha \geq 1$
Output: Optimized parameter θ
$f_\theta \leftarrow f(\theta)$
for $k \leftarrow 1$ **to** maximum iterations **do**
  $\theta' \leftarrow \theta + rand(\sigma)$; $f_{\theta'} \leftarrow f(\theta')$;
  if $f_{\theta'} < f_\theta$ **then** $\theta \leftarrow \theta'$; $f_\theta \leftarrow f_{\theta'}$; $i \leftarrow 0$; $\sigma \leftarrow \sigma \cdot \alpha$ **else** $\sigma \leftarrow \sigma / \alpha$ **end if**
  $i \leftarrow i+1$
  if $i > N$ **then** $f_\theta \leftarrow f(\theta)$ **end if**;
end for
return θ

$Env(a_{i+1})$, that provides an observation o_{i+1}, and a reward, r_{i+1}. The stochastic return is given by $R(\theta) := \sum_{t=1}^{T} r_t$. The objective of the RL is to find a θ that maximizes $\mathbb{E}\Big(R(\theta)\Big)$.

Simple search based RL [Spall, 2005], as suggested in [Salimans et al., 2017], in [Duan et al., 2016], and very recently in [Mania et al., 2018], can scale and perform competitively with gradient-based approaches, and in some cases even surpass their performance, with clear advantages such as skipping gradient scaling issues. Accordingly, we adopted a simple search-based algorithm to train ONCs. Our approach combines a *Adaptive Random Search* (ARS) optimization [Rastrigin, 1963], with an *Objective Estimate* (OE) function $f : \theta \mapsto \mathbb{R}^+$. The OE generates N rollouts with π_θ on the environment and computes an estimate of $\mathbb{E}(R_\theta)$ based on a filtering mechanism on these N samples. We compared two filtering strategies in this context; 1) taking the average of the N samples, and 2) taking the average of the worst k samples out of N samples. The first strategy is equivalent to the *Sample Mean estimator* [Salimans et al., 2017], whereas the second strategy aims to avoid getting misled by outlying high samples of $\mathbb{E}(R_\theta)$. The objective for realizing the second strategy was the fact that a suitable parameter θ enforces the policy π_θ control the environment in a reasonable way even in challenging situations (i.e., rollouts with the lowest return). The algorithm is outlined in Algorithm 5. We treat the filtering strategy as a hyperparameter.

4.5 Experiments

The goal of our experimentation is to answer the following questions: 1) How would an ONC with a preserved biological connectome, perform in basic standard control settings, compared to that of a randomly-wired circuit? Are ONCs *natural lottery ticket winners*? 2) When possible, how would the performance of our learned circuit compare to the other methods? 3) Can we transfer a policy from a simulated environment to a real environment? 4) How can we interpret the behavior of the neural circuit policies?

We use four benchmarks for measuring and calibrating the performance of this approach, including one robot application to parking for the TW sensory/motor neurons and then deployed our RL algorithm to learn the parameters of the TW circuit and optimize the control objective. The environments include I) Inverted pendulum of Roboschool [Schulman et al., 2017], II) Mountain

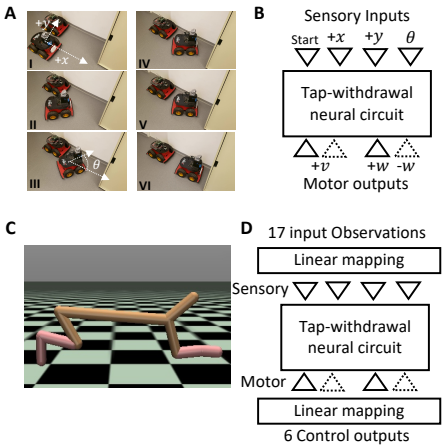


Figure 4.3: Mapping the environments to the TW circuit in A) Parking task, B) mapping for the parking. C) half-cheetah, and C) mapping for the half-cheetah experiment. See Table S3 in the Supplementary Material for more details.

car of OpenAI Gym, III) Half-CHeetah from Mujoco, and IV) Parking a real rover robot with a transferred policy from a simulated environment. The code is available online. The TW neural circuit (cf. Fig. 4.1B) allows us to incorporate four input observations and to take two output control actions. We evaluate our ONC in environments of different toolkits on a variety of dynamics, interactions, and reward settings.

4.5.1 How to map ONCs to environments?

The TW neural circuit is shown in Fig. 4.1B, contains four sensory neurons. It, therefore, allows us to map the circuit to four input variables. Let us assume we have an inverted pendulum environment which provides four observation variables The position of the cart x, together with its velocity $\dot{x}$, the angle of the pendulum φ.[3] along with its angular velocity $\dot{\varphi}$. Since the main objective of the controller is to balance the pendulum in an upward position and make the car stay within the horizontal borders, we can feed φ (positive and negative values), and x (positive and negative), as the inputs to the sensors of the TW circuit. Control commands can be obtained from the motor neuron classes, FWD and REV. Likewise, any other control problem can be feasibly mapped to an ONC. We set up the search-based RL algorithm to optimize neurons' and synapses' parameters $\omega, \hat{\omega}, \sigma, C_m, E_L$ and G_L, within their corresponding range, shown in Table S2. A video of different stages of the learned ordinary neural circuit for the inverted pendulum can be viewed at `https://youtu.be/cobEtJVw3A4`

In a simulated Mountaincar experiment, the environmental variables are the car's horizontal position, x, together with its linear velocity. The control signal applies force to the car to build up momentum until finally reaching the top of the hill. The TW circuit can then be learned by the

[3]Remark: The environment further splits φ into $sin(\varphi)$ and $cos(\varphi)$ to avoid the $2\pi \rightarrow 0$ discontinuity

search-based RL algorithm. A video illustrating the control of the car at various episodes during the optimization process can be viewed at https://youtu.be/J7vXFsZz7EM.

4.5.2 Scale the functionality of ONCs to environments with larger observation spaces

We extend the application of the TW circuit as an instance of ordinary neural circuits, to handle tasks with more observation variables. We choose the HalfCheetah-v2 test-bed of Mujoco. The environment consists of 17 input and six output variables. We add a linear layer that maps an arbitrary number of input variables to two continuous variables that are then fed into the four sensory neurons of the TW circuit, as shown in Fig. 4.3D. Similarly, we add a linear layer that maps the neuron potentials of the two motor neurons to the control outputs. A video of this experiment can be viewed at https://youtu.be/zG_L4JGOMbU.

4.5.3 Transfer learned ONCs to control real robot

In this experiment, we generalized our TW ordinary neural circuit to a real-world control test-bed. We let the TW circuit learn to park a rover robot on a determined spot, given a set of checkpoints form a trajectory, in a deterministic simulated environment. We then deploy the learned policy on a mobile robot in a real environment shown in Fig. 4.3A. The key objective here is to show the capability of the method to perform well in a transformation from a simulated environment to a real setting. For doing this, we developed a *custom deterministic simulated RL environment*. The rover robot provides four observational variables (start signal, position (x, y) and angular orientation θ), together with two motor actions (linear and angular velocity, v and w). We mapped all four observatory variables, as illustrated in Fig. 4.3B, to the sensors of the TW. Note that here the geometric reference of the surrounding space is set at the initial position of the robot. Therefore, observation variables are positive. We mapped the linear velocity (which is a positive variable throughout the parking task) to one motor neuron and the same variable to another motor neuron. We determined two motor neurons for the positive and negative angular velocity. (See Table S3 in Supplementary for mapping details). This configuration implies that the command neuron, AVA, controls two motor neurons responsible for the turn-right and forward motion-primitives, and AVB to control the turn-left and also forward motor neurons.

Optimization setup for the parking task – A set of checkpoints on a pre-defined parking trajectory were determined in the custom simulated environment. For every checkpoint, a deadline was assigned. At each deadline, a reward was given as the negative distance of the rover to the current checkpoint. The checkpoints are placed to resemble a real parking trajectory composed of a sequence of motion primitives: Forward, turn left, forward, turn right, forward, and stop. We then learned the TW circuit by the RL algorithm. The learned policy has been mounted on a Pioneer AT-3 mobile robot and performed a reasonable parking performance. The Video of the performance of the TW ordinary neural circuit on the parking task can be viewed at https://youtu.be/p0GqKf0V0Ew.

Table 4.2: ONC versus random circuits. Results are computed for 10 random circuits, and 10 weight initialization runs of the TW circuit - High standard deviations are due to the inclusion of unsuccessful attempts of each type of network.

Env / Method	Random Circuit	**ONC**
Inverted Pendulum	138.1± 263.2	**866.4 ±418**
Mountain car	54 ±44.6	**91.5±6.6**
Half-Cheetah	1742.9±642.3	**2891.4±1016**

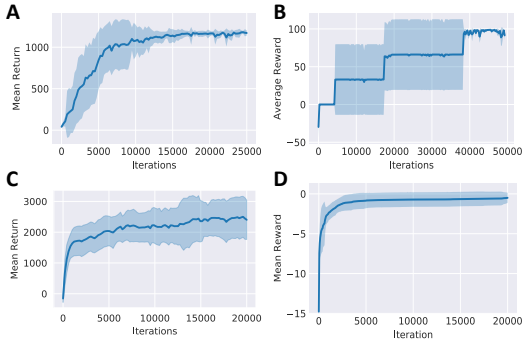


Figure 4.4: Learning curves for the TW circuit in standard RL tasks. A) Inverted pendulum B) Mountain car (OpenAI Gym) C) Half-Cheetah D) The parking task. The shadows around the learning curves represent the standard deviation in learning each task for 10-times repetitions.

4.6 Experimental Evaluation

In this section, we thoroughly assess the results of our experimentation. We qualitatively and quantitatively explain the performance of our ordinary neural circuits. We then benchmark our results where possible, with the existing methods, and describe the main attributes of our methodology. Finally, we quantitatively interpret the dynamics of the learned policies.

4.6.1 Do ONCs perform better than equivalent random circuits?

We conducted an experiment in which we designed circuits with randomly wired connectomes, with the same number of neurons and synapses used in the TW circuit. The initial polarity of the synapses is set randomly (excitatory, inhibitory, or electrical synapse) with a simple rule that no synapse can be fed into a sensory neuron, which is a property of ONCs as well. The random circuits were then trained over a series of control tasks described earlier, and their performance is reported in Table 4.2. We observe that ONCs significantly outperform the randomly wired networks, which is empirical evidence for ONCs being the lottery ticket winners.

Performance. The training algorithm was able to solve all the tasks, after a reasonable number of iterations, as shown in the learning curves in Fig. 4.4A-D. Jumps in the learning curves of the mountain car (Fig. 4.4B) are the consequence of the sparse reward. For the deterministic parking trajectory, the learning curve converges in less than 5000 iterations.

Table 4.3: Comparison of ONC to artificial neural networks with policy gradient algorithms

Method	Inverted Pendulum	MountainCar
MLP + PPO [Schulman et al., 2017]	1187.4±51.7	94.6±1.3
MLP + A2C [Mnih et al., 2016]	1191.2±45.2	86.4±18.3
ONC + RS (ours)	1168.5±21.7	91.5±6.6

Table 4.4: Compare ONC with deep learning models. numbers show the Mean, standard deviation, and success rate for 10 runs. $N = 10$

Agent	**Inverted Pendulum**	**Mountaincar**	**HalfCheetah**	Sparsity
LSTM	629.01 ± 453.1 (40.0%)	**97.5** ± 1.25 (100.0%)	1588.9 ± 353.8 (10.0%)	0% (fully connected)
MLP	**1177.49** ± 31.8 (100.0%)	95.9 ± 1.86 (100.0%)	1271.8 ± 634.4 (0.0%)	0% (fully connected)
ONC (ours)	**1168.5**± 21.7 (90.0%)	91.5 ± 6.6 (80.0%)	**2587.4** ± 846.8 (72.7%)	**77% (28 synapses)**
Random circuit	138.10 ± 263.2 (10.00%)	54.01± 44.63 (50.0%)	1743.0 ± 642.3 (50.0%)	77% (28 synapses)

4.6.2 How does ONC + random search compares with policy gradient based RL algorithms?

ONCs + Random search algorithm demonstrates comparable performance to the state-of-the-art policy gradient RL algorithms such as Proximal Policy optimization (PPO) [Schulman et al., 2017], and advantage actor critic (A2C) [Mnih et al., 2016]. Table 4.3 reports the performance of the mentioned algorithms compared to NPC+RS.

4.6.3 How does ONC compare to deep learning models?

The final return values for the basic standard RL tasks (provided in Table 4.4), matches that of conventional policies [Heidrich-Meisner and Igel, 2008], and the state-of-the-art deep neural network policies learned by many RL algorithms [Schulman et al., 2017, Berkenkamp et al., 2017]. We compared the performance of the learned TW circuit to long short-term memory (LSTM) recurrent neural networks [Hochreiter and Schmidhuber, 1997], multi-layer peceptrons (MLP), and random circuits. We select the same number of cells (neurons) for the LSTM and MLP networks, equal to size of the tap-withdrawal circuit. LSTM and MLP networks are fully connected while the TW circuit realizes a 77% network sparsity. In simple tasks experiments the TW circuit performs in par with the MLP and LSTM networks, while in HalfCheetah, it significantly achieves a better performance. Results are summarized in Table 4.4.

4.6.4 Interpretability of the ordinary neural circuits

In this section, we introduce a systematic method for interpreting the internal dynamics of an ONC. The technique determines how the kinetics of sensory neurons and interneurons relate to a motor neuron's decision. Fig. 4.5B illustrates how various adaptive time-constants are realized in the parking environment. Interneurons (particularly PVC and AVA) change their time-constants significantly compared to the other nodes. This corresponds to their contribution

to various dynamical modes and their ability to toggle between dynamic phases of an output decision. Fig. 4.5C visualizes the activity of individual TW neurons (lighter colors correspond to a more activation phase) over the parking trajectory. It becomes qualitatively explainable how individual neurons learned to contribute to performing autonomous parking. For instance, AVA, which is the command neuron for turning the robot to the right-hand-side (Motor neuron RGT) while it is moving, gets highly activated during a right-turn. Similarly, AVB and LFT neurons are excited during a left-turning phase. (See Fig. 4.5C). Next, we formalize a quantitative measure of an ONC element's contribution to its output decision.

Definition 6. *Let $I = [0, T]$ be a finite simulation time of an ONC with k input neurons, N interneurons and n motor neurons, (Shown in Fig. 4.1), acting in an RL environment. For every neuron-pair (N_i,n_j), (N_i, N_j) and (k_i, n_j), in a cross-correlation space, let $S = \{s_1, ...s_{T-1}\}$ be the set of the gradients amongst every consecutive simulation time-points, and $\Omega = \{\arctan(s_1), ..., \arctan(s_{T-1}\}$ be the set of all corresponding geometrical angles, bounded to a range $[-\frac{\pi}{2}, \frac{\pi}{2}]$. Given the input dynamics, we quantify the way sensory neurons and interneurons contribute to motor neurons' dynamics, by computing the histogram of all Ωs, with a bin-size equal to l (i.e. Fig 4.5D), as follows:*

- *If sum of bin-counts of all $\Omega > 0$, is more than half of the sum of bin-counts in the $\Omega < 0$, the overall contribution of N_i to n_j is positive.*
- *If sum of bin-counts of all $\Omega < 0$, is more than half of the sum of bin-counts in the $\Omega > 0$, the overall contribution of N_i to n_j is negative,*
- *Otherwise, N_i contributes in phases (switching between antagonistic and phase-alighted) activity of n_j, on determinable distinct periods in I.*

To exemplify the use of the proposed interpretability method, let us consider the neuronal activity of a learned circuit driving a rover robot autonomously on a parking trajectory. Fig. 4.5D presents the histograms computed by using Definition 1 for the RGT motor neuron dynamics (i.e., the neuron responsible for turning the robot to the right) with respect to that of other neurons. Based on Definition 1, we mark AVM, AVD, AVA as positive contributors to the dynamics of the RGT motor neuron. We determine PVD, PLM, and PVC as antagonistic contributors. Neurons such as DVA and AVB realized phase-changing dynamics where their activity toggles between positive and negative correlations, periodically. (For the analysis of the full networks' activities visit Supplementary Materials Section 6). Such study is generalizable to the other environments too. (See Supplementary Materials Section 6). In that case, the algorithm determines principal neurons in terms of neuron's contribution to a network's output decision in computable intervals within a finite simulation time.

4.7 Conclusions

We showed the performance of ONCs in control environments as the natural lottery winner networks. We quantitatively demonstrated that the sub-networks taken directly from the nervous system of the small species realize an attractive max-flow rate and, when trained in isolation,

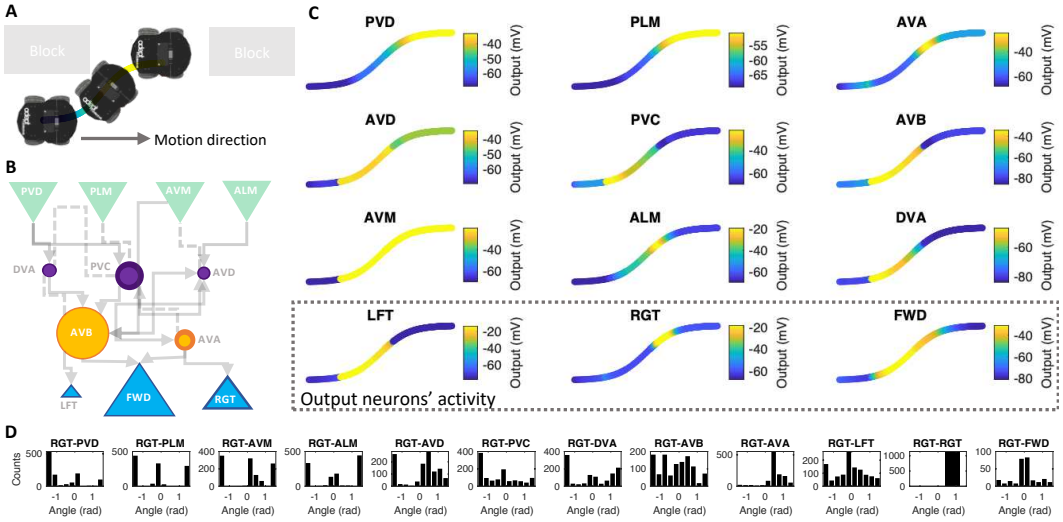


Figure 4.5: Interpretability analysis of the parking task. A) The parking trajectory. B) TW circuit drawn with the range of possible variations of the individual neuron's time-constants; the radius of the darker color circle for each neuron corresponds to the range within which the time-constant varies between τ_{min} and τ_{max} while the robot performs the parking. (Values in Supplementary Materials, Table S7). C) Projection of individual neuron's output over the parking trajectory. The plots demonstrate when neurons get activated while the rover is performing the parking task. D) Histogram of the slopes in manifolds' point-pair angles for a motor neuron in the parking task. (See Supplementary Materials Section 6, for full circuit's analyses, in other experiments.)

perform significantly better than randomly-wired circuits, as well as contemporary deep learning models in simulated and real-life tasks. We experimentally demonstrated the interpretable control performance of the learned circuits in action and introduced a quantitative method to explain networks' dynamics. The proposed method can also be utilized as a building block for the interpretability of recurrent neural networks, which despite a couple of fundamental studies [Karpathy et al., 2015, Chen et al., 2016, Olah et al., 2018, Hasani et al., 2019], is still a grand challenge to be addressed. The next research question arises on what if we set priors in designing NCPs. In particular, can we design a specialized neural network and train it to perform the desired task? We will discuss this in the next chapter.

CHAPTER 5

Rule-based Design of LTC Networks for Interpretable Robot Control

5.1 Motivation

© [A5-Chapter5, 2019] — The *C. elegans* nematode, with a rather simple nervous system composed of 302 neurons and 8000 synapses [Chen et al., 2006], exhibits remarkable controllability in it's surroundings; it expresses behaviors such as processing complex chemical input stimulations [Bargmann, 2006], sleeping [Nichols et al., 2017], realizing adaptive behavior [Ardiel and Rankin, 2010, Hasani et al., 2017c], performing mechano-sensation [Chalfie et al., 1985b], and controlling 96 muscles [Wen et al., 2012]. How does *C. elegans* perform so much with so little? What are the underlying computational principles to gain such high degrees of controllability? And how can we design worm-like artificial intelligent (AI) systems based on these principles to obtain better AI controllers? To answer these questions, we take a computational approach in this chapter.

As shown in Chapter 3, LTCs can capture complex dynamics with a few number of neurons, due to the existence of nonlinearities on their synaptic information processing mechanisms which enable neurons to express varying time-constants. In Chapter 4, we took a network architecture directly from the brain of the worm and demonstrated its compelling performance in robot control. The main open question is now on how to build such network topologies systematically in robotic control domains while enhancing the interpretability of the system?

In this chapter, we propose a network design methodology that utilizes the LTC neural model to create interpretable neural controllers in robotic tasks. The method proposes a set of rules to construct hierarchical architectures, from sensory neurons, through an interleaved set of interneurons, to motor neurons, by means of binary relational structures named *design operators* (DO), illustrated in Fig. 5.1. This design procedure induces a high degree of sparsity (around 80%) and builds up an attention mechanism through distinct network pathways that enhance interpretability. Along with the recent successes of the search-based learning techniques for

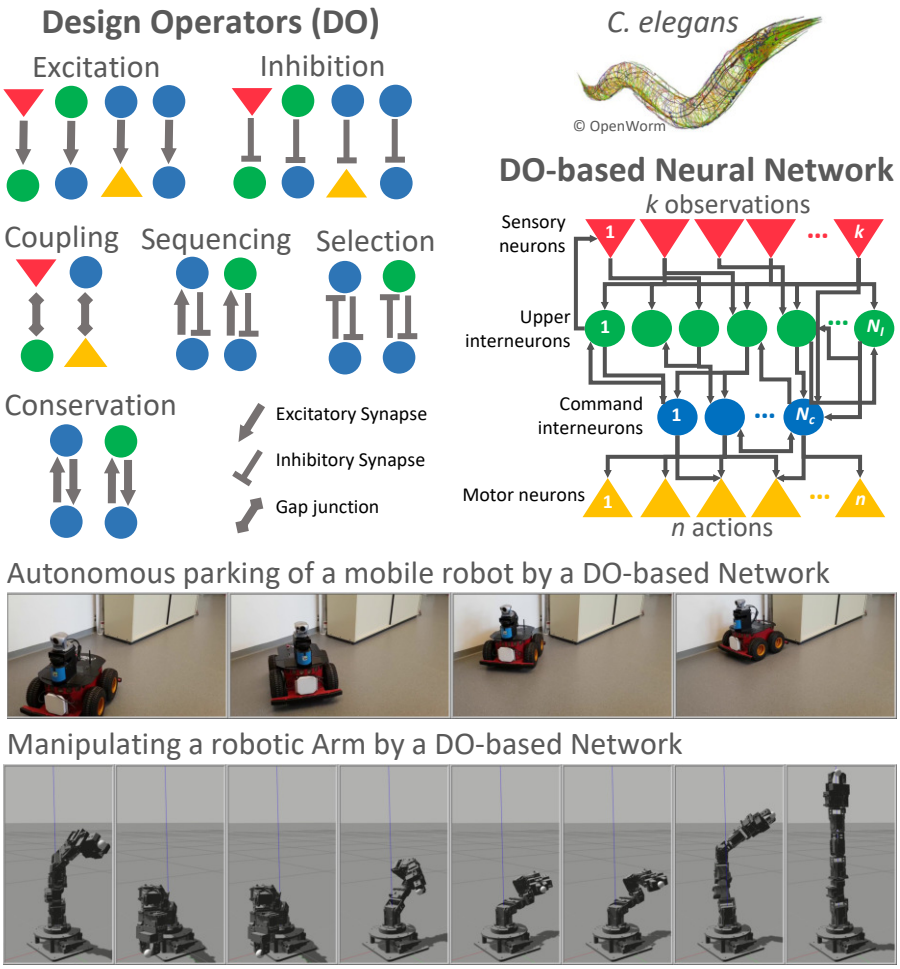


Figure 5.1: We design compact, interpretable and noise-robust neural controllers inspired by the relational structures of the worm's brain, to control robots.

neural networks in control environments [Salimans et al., 2017, Duan et al., 2016, Mania et al., 2018], we adopt a random-search memetic algorithm to parametrize the synaptic weights.

The chapter aims to contribute the following:

1. Introducing novel network-design principles for the LTC neural models, and equipping the designed network with a search-based learning algorithm, to govern robotic tasks.
2. Deploying DO-based networks in experiments with real and simulated robotic environments.
3. Experimental demonstrations of the superiority of the performance of DO-based networks in terms of their compactness, robustness to noise, and their interpretable dynamics, compared to contemporary RNN architectures.

5.2 Related Works

Brain-inspired Robotic Control. The way nervous systems of living creatures process information has been extensively used in robotic control as a source of inspiration [Brabazon et al., 2015, LeCun et al., 2015b, Folgheraiter et al., 2006, Capuozzo and Livingston, 2011]. In particular, networks of biophysically modelled neurons [Hasani et al., 2017a, Gleeson et al., 2018] are deployed in applications such as navigation of mobile robots [Folgheraiter et al., 2006, Hagras et al., 2004], control of unmanned aerial vehicles (UAV) [Westphal et al., 2013] and legged robots [Beer et al., 1992, Szczecinski et al., 2015, Szczecinski et al., 2017]. Obtained networks can be topologically divided into two categories: 1) Networks that are put together by hand in a *piece-by-piece* and *trial-and-error* fashion [Beer et al., 1992, Szczecinski et al., 2017, Folgheraiter et al., 2006, Westphal et al., 2013]. These approaches lack fundamental design principles. 2) Networks that deploy fully-connected structures and rely purely on the learning phase to determine functions. Similar to Deep learning models, interpreting the dynamics of these networks becomes a challenge [Olah et al., 2018, Hasani et al., 2018a]. Our networks address both challenges by incorporating a systematic design together with a set of rules that improves interpretability.

Motion Planning. to (optimally) solve the motion-planning problem, various model driven techniques have been proposed, such as rapidly-exploring random trees [LaValle, 1998, Kavraki and LaValle, 2008, Pokorny et al., 2016], cell decomposition [Kavraki and LaValle, 2008, Latombe, 2012b, Masehian and Sedighizadeh, 2007], potential fields [Kavraki and LaValle, 2008, Stavridis et al., 2017, Latombe, 2012b, Masehian and Sedighizadeh, 2007], satisfiability modulo theories (SMT) [Dantam et al., 2016, Nedunuri et al., 2014, Hung et al., 2014] and model predictive control (MPC) [Cardoso et al., 2017, Camacho and Alba, 2013]. These approaches are often human expert labor-intensive, to distill task-specific solutions. We aim to ease the effort by introducing a combination of systematic design equipped with machine learning techniques. In parallel to model-driven control systems, deep reinforcement learning (RL) has achieved significant successes in agent control [Zhang et al., 2017, Gu et al., 2016, Peng et al., 2017, Salimans et al., 2017]. In a deep RL setting, parameters of the neural network are tuned by a learning algorithm for the control agent to take actions that maximize the total reward. While they perform as good or surpass the performance of the manually designed agents, their explainability becomes a challenge, which is not desirable in safety-critical applications. Our methodology overcomes this challenge by imposing sparse network connectivity and by using an interpretable neuronal model [M. Hasani et al., 2018].

5.3 Network Simulation Environment

We use the LTC neural model described in Chapter 3 to obtain design operator based networks. In order to solve the ODEs in real-time efficiently, we used a fixed step solver [Press et al., 2007b]. The simulator (Algorithm 6) runs with $\mathcal{O}(|\#\text{ neurons}| + |\#\text{ synapses}|)$ time complexity for each time step Δ_t.

Algorithm 6 Network simulator

Input: Network V, Sensory neurons S, Motor neurons M
$v[0 \dots n] \leftarrow V_{leak}$
while True **do**
  $v[s \in S] \leftarrow$ read_sensor_values()
  $I[0 \dots n] \leftarrow 0$
  for $e =$ (pre,post) $\in E$ **do**
    $I[\text{post}] \leftarrow I[\text{post}] + (E_{rev}(e) - v[\text{post}])w(e) \cdot \sigma(v[\text{pre}])$
  end for
  for $i \in 0, \dots n$ **do**
    $v[i] \leftarrow$ ODE_update(v, I)
  end for
  set_output$(v[m \in M])$
end while

5.4 Design Operators

In this section, we characterize a set of relational components with which we construct DO-based networks. The wiring patterns among two neurons, discovered in *C. elegans*, are called binary motifs [Alon, 2006, Milo et al., 2002, Milo et al., 2004]. Equipping these motifs with the neuronal model of Eq. 3.7, results in six design operators (DO)s: Excitation, inhibition, coupling, sequencing, conservation, and selection, presented in Fig. 5.2.

Definition 7. *Given the neurons in circuit $G = \{N_1, N_2, \dots N_n\}$, a Design Operator is a relation formed among the activation state of neurons, based on their distinct connectivity structure and their synaptic parametrization.*

DOs are fundamentally different compared to network motifs. Motifs [Alon, 2007], are frequently repeated structural patterns in biological networks, whereas DOs are both structural and relational dependencies of neurons. Motifs' significance is mainly limited due to the lack of information about the synaptic polarities in biological networks [Alon, 2006]. However, we adopted the concept of DOs from the functional dynamics of neurons, captured by investigating calcium imaging of the neuronal activity of the *C. elegans*' brain [Kato et al., 2015b]. More importantly, the general goal of network motifs is to explain the mechanisms underlying the behavior of a biological network through the interaction of basic building blocks [Alon, 2007]. A design operator, in contrast, may result in the emergence of many behaviors for a given structure due to its output dependencies on the alternation of the synaptic parameters.

We quantify this dependence by performing cross-correlation and bifurcation analyses. A short simulation for each DO is depicted in Fig. 5.2A. An excitation/inhibition DO occurs through one excitatory/inhibitory chemical synapse and leads to the stimulation/inhibition of the postsynaptic neuron. A coupling DO occurs through one electrical synapse and establishes the coupling of the activity of the two neurons. A sequencing DO imposes a sequential activation of the neurons. A conservation/selection realizes a synchronizing/antagonizing activity.

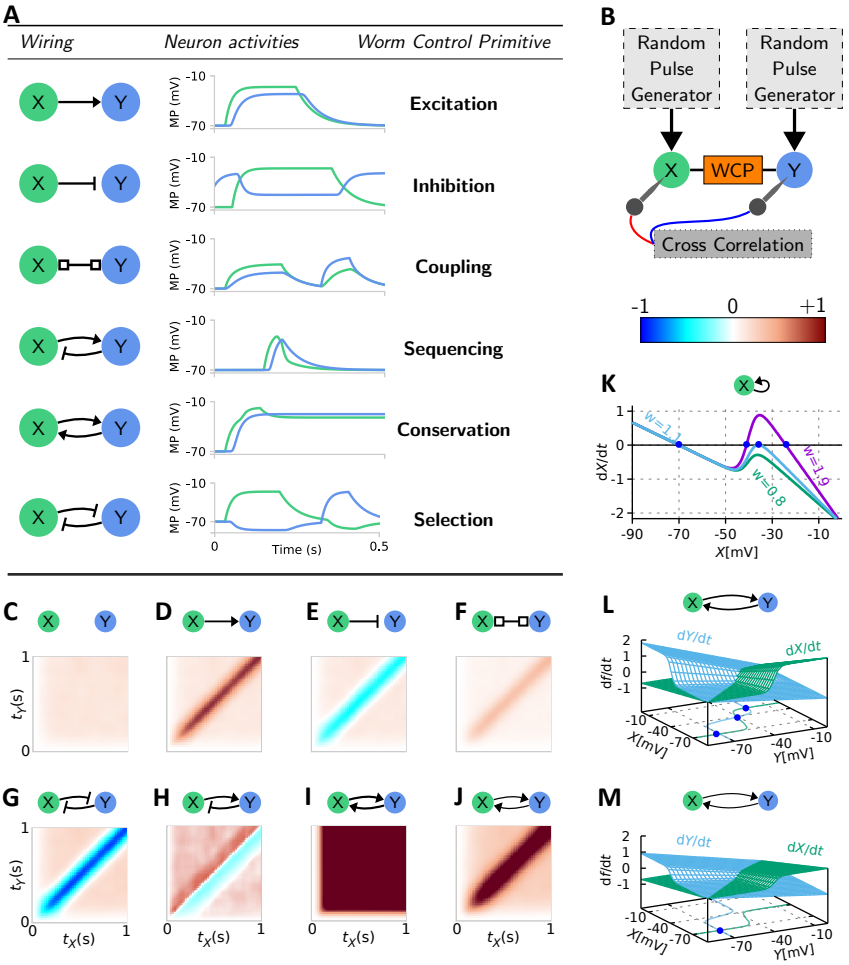


Figure 5.2: Design Operators. (A) Time series of the activity of DOs. (B) Sampling process used in the correlation analysis. (C-J) Cross-correlation. (I-J) Cross-correlation of the conservation DO with strong and weak synaptic weights, respectively. (K) Bifurcation analysis of self-conservation (unary) DO (at most three stationary points). (L) Bifurcation analysis for conservation DO with strong synaptic weights. The isolines for $dX(t)/dt = 0$ and $dY(t)/dt = 0$, of the membrane potential of neurons X and Y, plotted on the base plane have three intersections (stationary points). (M) Bifurcation analysis for conservation DO with weak synaptic weights. The isolines plotted on the base plane have only one intersection (stationary point).

The correlation analysis of the DOs given in Fig. 5.2C-J, was obtained by subjecting the neurons to independent random-pulse generators, as in Fig. 5.2B, and collecting their outputs. In an excitation/inhibition DO, the neurons are phase-aligned with a positive/negative correlation at the main diagonal, (Fig. 5.2D, 5.2E). This means that excitation/inhibition does not introduce delay or memory. In a Coupling DO, the neurons are also phase-aligned Fig. 5.2F. In a selection DO, the dynamics are antagonistic, and result in a competition for being active, (Fig. 5.2G).

In a sequencing DO, a positive/negative correlation appears above/below the main diagonal (Fig. 5.2H). Finally, in a conservation DO, the activity is correlated, independently of phase differences. It thus introduces a memory element (Fig. 5.2I), which vanishes at low synaptic weights, (Fig. 5.2J).

To understand the dependencies of a DO to its parameters, let us take the memory effect realized by the conservation DO and perform a bifurcation analysis, to explore the number of fixed points, for different synaptic weights [Poincaré, 1885]. In Fig. 5.2K, The bifurcation plot represents how a self-excited neuron, as a special case of conservation, determines the dynamics. For large weights (purple line), the neuron has three fixed-points as follows: Stable (left), meta-stable (middle), and stable (right). The stable fixed-points are able to robustly preserve the membrane potential values, as being intuitively inactive (resting) at around $-70mV$ and active at around $-20mV$. This ability vanishes for a low synaptic weight (green line), as the only fixed-point corresponds to the resting potential. In Fig. 5.2L-M, we show the same analyses for two neurons. For large synaptic weights, the leftmost and rightmost fixed-points of the isolines plotted on the base plane, robustly preserve the potential. For low synaptic weights, this ability of the DO demolishes. In the next section, we describe how to design neural controllers based on DOs.

5.5 Design a DO-based Neural Network

In this section, we introduce our methodology for designing DO-based neural networks. For a sequential robot control task with identifiable finite action primitives, a network can be designed to fit the given behavior. Given a robotic environment with p sequential action primitives, we design a DO-based network by the principles described below:

Rule 1. *Add p command neurons for p motion primitive. - If two primitives have direct temporal dependencies, add a Sequencing DO between their underlying command neurons. - If two primitives function in parallel, add Conservation DO. - If two action primitives are mutually exclusive, add a Selection DO between their corresponding command neurons. - If a single action primitive should persist over a certain time, add a Self-Conservation DO.*

Rule 2. *Identify trigger conditions of primitives; add an Upper interneuron for each of the conditions and connect the interneuron to the command neurons by Excitation DOs.*

Rule 3. *For each of the Upper interneurons, identify the signals on which its condition triggers. These signals can come from other neural circuit modules or be defined based on the characteristics of the environment. Signals link their activity to their downstream interneurons by Coupling DOs.*

To clarify Rules 2 and 3, consider the neural circuit of Fig. 5.3 which is designed for parallel parking of a mobile robot. five motion primitives including (going backward, turn, backward, turn, stop) has to happen. Each action primitive has to be triggered and the decision has to be propagated to the command neurons. This is done by an upper interneurons.

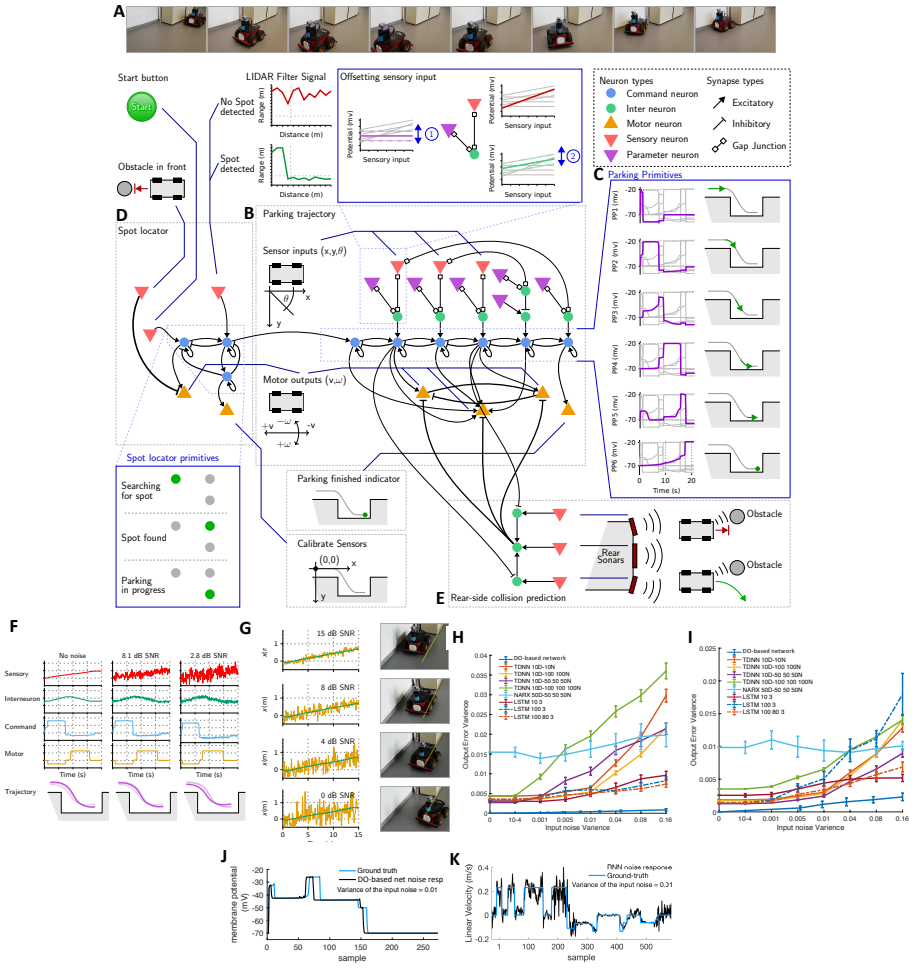


Figure 5.3: Npark. (A) Rover searches for a spot and performs the parking. (B-E) The architecture of the DO-based network designed parking. (B) Parking-trajectory circuit. (C) Six motion primitives for parking controlled by six command neurons. (D) The spot-locator circuit (E) The rear-side collision-avoidance circuit. (F-K) Input noise resilience. (F) Noise injection analysis. The increasing noise (from left to right) is directly applied to the sensory neurons. (G) The effect of noisy input data on the function of the network. (H) The variance of the linear-velocity output error while increasing the variance of the input noise. (I) The same analysis for the angular-velocity output. (J) The response of the network in the presence of noisy input. (K) The response of the TDNN 10D−100 100N RNN (10D = 10 delay elements and 100 100N = 2 layer each with 100 neurons), to the same noisy input. TDNN = time-delayed neural network [Waibel et al., 1989]. NARX = nonlinear autoregressive network with exogenous input [Billings, 2013]. LSTM = long short term memory [Hochreiter and Schmidhuber, 1997].

Rule 4. *k Sensory neurons are deployed for k observation variables. Sensory neurons are coupled with their downstream upper interneurons, by coupling DOs. Their connections are established*

such that particular pathways from the input to output, direct the attention [Vaswani et al., 2017] of the network towards specific actions.

Rule 5. *Devote n motor neurons corresponding to n control action. Command neurons positively correlating with the control action, synapse into the motor neurons by Excitation DO, and inhibit their negatively correlating downstream motor neurons, by inhibitory DOs.*

Described rules enable us to design highly sparse neural networks with auditable internal dynamics. By learning the parameters of these networks, we can guide the architecture to perform and generalize well in the control of robots.

5.6 Synaptic Parametrization

To optimize the parameters of a DO-based network, we adopted a Random-search memetic learning algorithm. Recently it has been shown that random search optimization strategies [Spall, 2005], can perform as good as gradient-based approaches, with additional advantages such as skipping the gradient issues [Salimans et al., 2017, Duan et al., 2016, Mania et al., 2018].

Our learning algorithm uses a population of (synaptic) parameter particles and repeats the following two steps until convergence: Generate a new population by randomly perturbing the current one. Resample the obtained population according to the cost of the particles (Network behavior deviation for this particle, from the desired behavior). Algorithm 7 outlines the working principles of the learning system.

5.7 Autonomous Parking of a Mobile Robot

In this section, we design DO-based neural networks to perform an autonomous parking procedure with a Pioneer P3-AT robot [OMRON ADEPT MOBILEROBOTS, 2011], by the design rules introduced in Sec. 5.5. The task includes three control procedures as finding a parking spot, performing a parking trajectory, and simultaneously avoiding possible obstacles. For each task, a DO-based network is designed and presented in Fig. 5.3D, 5.3B, and 5.3E, respectively. The core circuit (the parking trajectory), in Fig. 5.3B, follows Rule 1 to include six command neurons for six motion primitives shown in Fig. 5.3C second column and configures sequencing DO amongst them. Upper interneurons establish coupling DO with the Parameter neurons which condition their activation, based on Rule 2 and 3. (See Fig. 5.3B top side box). Based on Rule 4, sensory neurons only synapse into their downstream pathways on which they impose a high impact. For instance, the angular position (θ) sensor, connects to interneuron pathways that are involved in the control of the robot's turns. Three motor neurons are set to control right turning, left turning, and moving backward, then Rule 5 is applied for their connectivity.

The spot locator neural circuit, shown in Fig. 5.3D, designed to move the robot forward until a filtered Light Detection and Ranging (LIDAR) signal flags a proper parking location. We pre-processed the LIDAR signal with a non-linear Finite-Impulse-Response (FIR) filter before feeding it into the network [Moshchuk and Chen, 2009]. Once the spot is found, the circuit

Algorithm 7 Random-Search Memetic Algorithm

Input: A cost function f to minimize, Population size n
Output: Parameter θ such that θ is a minimum of f
$P \leftarrow$ random Population of size n
$\theta_{best} \leftarrow rand()$
while New θ_{best} found recently **do**
  for $i \leftarrow 1$ **to** n **do**
    Local-random-search(f,$P[i]$)
    if $f(P[i]) < f(\theta_{best})$ **then**
      $\theta_{best} \leftarrow P[i]$
    end if
  end for
  for $i \leftarrow 1$ **to** n **do**
    $q \leftarrow$ select best parameters from P
    $Q[i] \leftarrow q + rand()$
  end for
  $P \leftarrow Q$
end while
return θ_{best}

activates the Parking trajectory agent to initiate the parking process. While the parking is in action, a rear-side collision avoiding circuit (Fig. 5.3E), translates the Sonar sensory inputs to preventive signals to the motor neurons by the inhibition DO. A parking reference trajectory is provided as a set of points $T = \{(x_t, y_t, \theta_t) \mid t \in \{0, \ldots, M\}\}$. Correspondingly, we set a cost (objective) function as:

$$R = \sum_{i \in T} (i(t).x - s(t).x)^2 + (i(t).y - s(t).y)^2 + (i(t).\theta - s(t).\theta)^2, \tag{5.1}$$

where $s(t)$ is the state (x_t, y_t, θ_t) of the rover at time t. We then call the learning algorithm, to minimize this function with respect to the synaptic weights. A video of the resulting neural network's performance can be viewed at `https://youtu.be/zOnqVaSl9nM`.

5.8 Manipulating a Robotic Arm

In this section, we extend our experimentation to the design of a DO-based network to move a Cyton-Epsilon-300 robotic arm with eight degrees of freedom (seven joints and a gripper) [Corporation, 2015] in Fig. 5.4C, to a certain location, grab an object, move the object to a second location, and then release the object (Fig. 5.4A). Action primitives are divided into five tasks schematically explained in Fig. 5.4A right box, and based on Rule 1, 5 command neurons are wired together to control each action. Upper Interneurons in this setting, adopt a fully connected topology to translate 14 sensory observations to trigger commands for the command interneurons while respecting Rules 2 and 3. Each sensory neuron that corresponds to the angle of individual

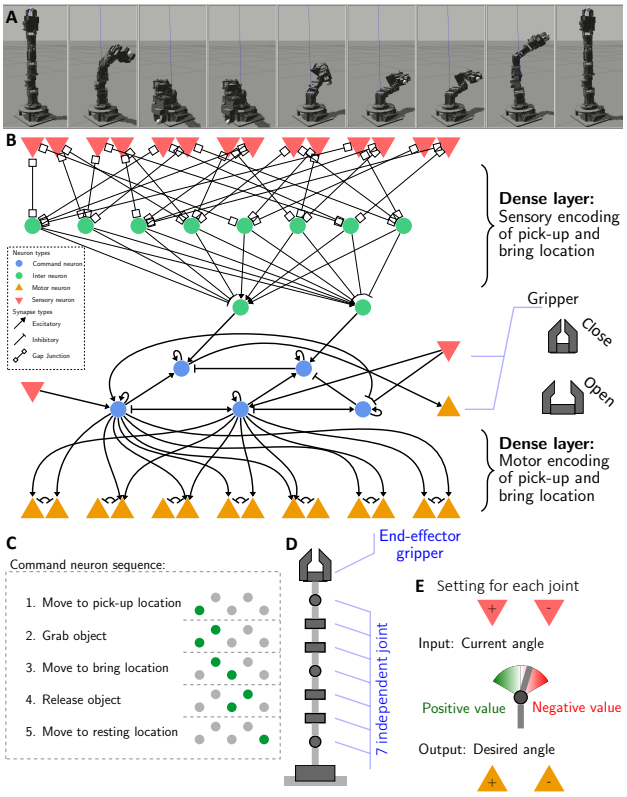


Figure 5.4: Manipulating a robotic arm. (A) Grasping and releasing an object at distinct positions. (B) Arm controller neural network (C) Command neurons' sequential activation (D) Illustration of the controlled end-effector composted of 7 joints and one gripper. (E) Indented cooperation of the command neurons.

joints (see the setting in Fig. 5.4D), forms coupling DO with two downstream interneurons randomly. This sparsity imposes an attention mechanism on the network and makes the interpretation of the decision pathways easier. Command neurons controlling the joints densely synapse into their 14 downstream motor neurons by Excitation DOs. The designed neural circuit is presented in Fig. 5.4B. The learning process of the network has been performed hierarchically. First, the sub-circuit, including sensory neurons to upper interneurons, was trained in a separate environment with the objective to activate when the arm reached the desired position. Then the motor neuron network is optimized in isolation with a supervised setting that with an objective of moving the arm to the desired position if the corresponding command neuron is activated. After stacking up the entire network, we fine-tuned all synaptic weights with Algorithm 7. A video demonstrating the performance of the neural controller on the arm in Gazebo can be viewed at `https://youtu.be/p8D3JTb8qLM`.

Table 5.1: Comparing parking performance of DO-based networks in terms of their network size with standard RNNs. RMSD = root mean squared deviation, $\mathbf{v}$ = linear velocity output, ω = angular velocity output and $\mathbf{E}$ = termination output

RNN Type	**Neurons Per-Layer**	**Params Total**	**v RMSD**	**ω RMSD**	**E RMSD**
TDNN-10D	10-3	943	$8.9{\cdot}10^{-3}$	$4.1{\cdot}10^{-3}$	$7.7{\cdot}10^{-3}$
TDNN-10D	100-100-3	19503	$7.0{\cdot}10^{-3}$	$4.2{\cdot}10^{-3}$	$9.4{\cdot}10^{-3}$
TDNN-10D	50-50-50-3	9803	$6.2{\cdot}10^{-3}$	$3.3{\cdot}10^{-3}$	$8.4{\cdot}10^{-3}$
NARX-50D	50-50-50-3	13153	$3.5{\cdot}10^{-3}$	$2.6{\cdot}10^{-3}$	$4.8{\cdot}10^{-3}$
LSTM	10-3	968	$3.5{\cdot}10^{-3}$	$3.5{\cdot}10^{-3}$	$3.5{\cdot}10^{-3}$
LSTM	100-3	45248	$2.0{\cdot}10^{-3}$	$2.0{\cdot}10^{-3}$	$2.0{\cdot}10^{-3}$
LSTM	100-80-3	102928	$1.6{\cdot}10^{-3}$	$1.6{\cdot}10^{-3}$	$1.6{\cdot}10^{-3}$
DO-based	**39**	**49**	Ground-truth		

5.9 Experimental evaluation

In this section, we point out the distinctions of DO-based networks compared to that of artificial recurrent neural networks and assess their performance. The parking network shown in Fig. 5.3B, comprises 39 neurons together with 49 trainable parameters. Compared to standard artificial neural networks generating similar dynamics, they are significantly smaller. We conducted an experiment to compare the parking performance of standard RNNs in generating the outputs of a DO-based network, given the sensory inputs. Table 5.1, summaries the performance of various RNN topologies.

5.9.1 Emergence of complex dynamics from compact networks

DO-based nets are 19 times smaller in terms of their trainable parameters than the smallest RNN (time-delayed neural network (TDNN) with ten delay elements). The reason for the capability of realizing complex dynamics with a fewer number of elements lies in their neuronal and synaptic model (Eq. 3.7). The model realizes liquid time constant dynamics, meaning that each neuron varies its time constant based on its presynaptic inputs. This is due to the synaptic model's nonlinearity, which becomes a rich resource for fitting complex dynamics with a fewer number of neurons.

5.9.2 DO-based networks are interpretable

Designing neural networks based on DOs, allows us to establish neuronal pathways with certain functionality, inside the network. For instance, in the parking network, the function of every node is known given their underlying design rules. In a more general case, such as the arm neural controller, the layer-wise design principles (Rules 1 to 5) increases the level of transparency of the network. In fact, the design principles realize an empirical attention mechanism, to govern

interpretable dynamics, where every input to the output pathway, contains interneurons with dedicated actions.

5.9.3 DO-based networks are highly resilient to noise

For the parking network, we performed two white Gaussian noise-injection experiments. The first exposes all sensory neurons to increasing internal noise and observes how the robot parks and how the noise propagates through the network to the output (Fig. 5.3F). The second watches how environmental input noise affects the performance of the network compared to other types of RNNs (Fig. 5.3G-I),

Fig. 5.3F-I, show the gradual worsening of the parking behavior. Fig. 5.3F and 5.3J, expose a remarkable property of the DO-based networks: The noise is filtered out as it propagates from the sensory-neurons layer to the motor-neurons layer. A phase shift occurs on the output. However, the network can still perform a decent parking trajectory even at a noise-level as large as the output signal itself. The capacitive nature of neurons in the neuronal model acting as a filter presumably explains this robustness. A video of the parking performance in the presence of input noise can be viewed at `https://youtu.be/tM9xBQFzBks`.

As illustrated in Fig. 5.3H and 5.3I, DO-based Networks considerably outperformed other RNN structures in terms of expressing noise-resilient output dynamics. The figures further demonstrate the sensitivity of the RNNs to noise attacks. While the input noise in DO-based networks causes a slight phase-shift in the output, as shown in Fig. 5.3J, the noise passes unhindered through all the layers of an RNN, and resulted in the distortion of the outputs, as shown in Fig. 5.3K. Hence, DO-based neural networks enhance their performance in terms of robustness to input noise, compared to other recurrent neural network topologies.

5.10 Conclusions and Discussions

We introduced a novel methodology for constructing compact, interpretable, and noise-resilient neural networks for controlling robots, inspired by the relational structures (Design Operators) of the nervous system of *C. elegans*. We experimentally illustrated the superiority of the performance of our compact DO-based neural networks in terms of robustness to noise, compared to standard RNNs.

DO-based networks construct a hierarchical network structure from sensory neurons, through an interleaved set of interneurons, to motor neurons. Their wiring structure is realized by a systematic set of rules, at multi-scale network resolutions. Furthermore, the synaptic and neuronal model constructs a sigmoidal nonlinearity on each synaptic link, resulting in the creation of varying time-constants of the network's nodes. This property enables DO-based networks to construct complex dynamics with a few number of elements.

Synaptic parameters of DO-based networks are then learned in a supervised fashion, utilizing a search-based optimization algorithm. The learning process enhances the scalability of the DO-based networks. The application of this type of circuits is broad in fitting the finite-time

horizon of n-dimensional continuous dynamical systems since their neuronal semantics realize universal approximation capabilities.

Many alternative approaches to the construction of DO-based networks can be taken. Model reduction methods on a densely connected network can be applied to obtain automatically generated neural networks while respecting Rules 1 to 5. DO-based neural networks are biophysically realistic artificial RNNs. We believe that their working principles presumably results in the development of better and safer AI systems, specifically in safety-critical robotic applications.

So far, we investigated robotic environments with rather small parameter spaces. How can we take advantage of the LTC model into the development of general-purpose intelligent agents with high-dimensional environmental characteristics? We will discuss this in the next chapter.

CHAPTER 6

Learning High-Fidelity Autonomous Driving agents by LTCs

6.1 Motivation

We set out to design an LTC-based intelligent agent that realizes a series of distinct competencies in learning to directly control an autonomous car from camera inputs. This domain raises important *representation learning [Bengio et al., 2013]* challenges; it is highly safety-critical [Knight, 2002], ergo demanding for intelligent controllers whose dynamics are immensely *interpretable*. Furthermore, although learned agents manifest *great performance* often in offline testing and in simulations, the performance degrades drastically during live driving. Additionally, the agents are desired to learn the true *causal structure* [Pearl, 2009, Peters et al., 2017] between the observed driving scenes and their corresponding optimal steering commands; ideally, we wish for the agent to implicitly learn to attend to the road's horizon when taking a current steering decision while maintaining an attractive performance. However, in practice, performant models have shown to learn a variety of *unfair* [Joseph et al., 2016] and sub-optimal [Peters et al., 2017] input-output causal structures [Fish et al., 2016, Vaswani et al., 2017]. Moreover, alongside a processing pipeline for the high-dimensional incoming input images, the agent has to incorporate a *short-term memory mechanism* to capture temporal dependencies.

Successful existing approaches [Bojarski et al., 2016, Xu et al., 2017, Amini et al., 2018a, Fridman et al., 2019], rely solely on deep convolutional neural networks architectures [LeCun et al., 1990a], that steer a car at a time t, based on the most recent camera frame [Amini et al., 2019] (Fig. 6.1a). While such feedforward models can drive the car on ideal input data, they fail to exploit the temporal nature of the task that would enable them to filter out transient disturbances. As a result, temporary corruptions of the input stream lead to unstable predictions.

Contrarily, recurrent neural networks (RNNs) [Hochreiter, 1991, Bengio et al., 1994], a class of artificial neural networks that take into account past observations at a current output decision

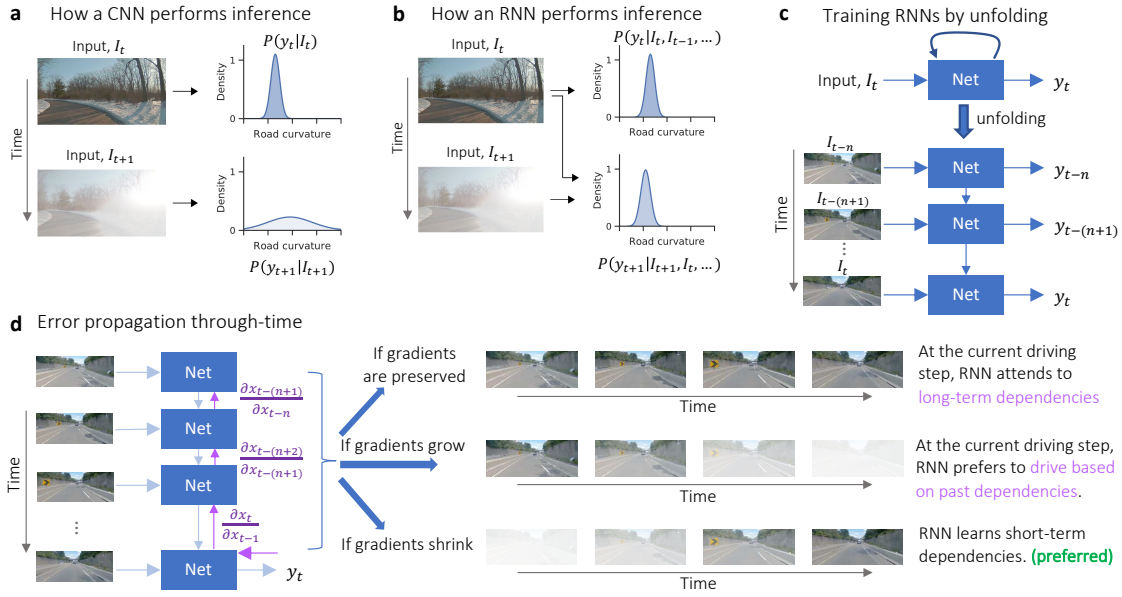


Figure 6.1: Recurrent neural networks' essence for lane-keeping tasks. **a.** A feedforward CNN network computes its output, $P(y_t|I_t)$ by relying solely on the current observation, I_t. Consequently, inputs that are corrupted by transient perturbations (bottom), will result in high output variance, and faulty decisions **b.** An RNN has access to past observations at a current driving step, enabling it to filter out transient corruptions that are present in the input stream **c.** Training RNNs by unrolling their state in time **d.** Then, applying back-propagation through time in an unfolded RNN. Purple derivatives indicate the dependency of the loss function's derivative in-respect-to an RNN's state-weights, to the evolution of the RNN's state, $x(t)$ in time. Blurred images depict weaker attention of the RNN, when computing a current decision.

through a feedback mechanism — principally would lead to a more robust end-to-end driving controller (Fig. 6.1b). RNNs are trained over finite-length labelled training sequences by the backpropagation algorithm [Rumelhart et al., 1986] applied to their unfolded feedforward representation [Bengio et al., 1994] (Figs. 6.1c, 6.1d). Historically, training RNNs has been challenging due to their elevated or diminished gradients during the learning phase [Hochreiter, 1991, Bengio et al., 1994]. Thanks to the development of the advanced gated RNNs, such as the long-short term memory (LSTM) [Hochreiter and Schmidhuber, 1997], the challenge is tackled, by enforcing a constant error flow through fixation of the recurrent weights to 1, and removing non-linearities within the feedback path [Hochreiter, 1991].

From a time-series modeling viewpoint, having a constant error flow is a desirable property as arbitrary data sequences may express long-term relations (Fig. 6.1d right). However, in the case of end-to-end autonomous driving, learning long-term dependencies can be detrimental due to the short-term causality of the underlying task. When driving a vehicle to follow the lane, humans do not recall images of the road from more than a few seconds ago to operate the steering wheel. Consequently, LSTM networks may capture spurious long-term dependencies present

Algorithm 8 Create NCP architecture

Require: Set of sensory neurons N_s, set of inter neurons N_i, set of command neuron N_c, set of motor neurons N_m, density parameters k_s, k_i, k_c and k_m.
Allocate graph (V,E) with $V = N_s \cup N_i \cup N_c \cup N_m$ and $E = \{\}$
call subroutine Connect sensory to inter neurons
call subroutine Connect intern to command neurons
call subroutine Recurrently connect command neurons
call subroutine Connect command to motor neurons
return (V,E)

Algorithm 9 Subroutine: Connect sensory to inter neurons

for all $s \in N_s$ **do**
  T is random permutation of the set N_i
  for $i = 1 \dots k_s$ **do**
    p is random variable of distribution $\{50\% \mapsto 1, 50\% \mapsto -1\}$
    Add synapse $(s \to T_i)$ to E with polarity p
  end for
end for
$\mu_{in} \leftarrow \frac{1}{|N_i|} \sum_{t \in N_i} |\{s|(s \to t) \in E\}|$ ∇ Compute mean fan-in of neurons N_i
for all $t \in N_i$ such that $\nexists s : (s \to t) \in E$ **do**
  S is random permutation of the set N_s
  for $i = 1 \dots \mu_{in}$ **do**
    p is random variable of distribution $\{50\% \mapsto 1, 50\% \mapsto -1\}$
    Add synapse $(S_i \to t)$ to E with polarity p
  end for
end for

in the training data and learn inadequate causal models [Pearl, 2009]. Besides, an exploding gradient phenomenon makes the learning process unstable and should be avoided (Fig. 6.1d right). Vanishing of the gradient prevents the RNN from learning correlations of events with long time-lags. This property enhances the real-world control performance of a learned RNN agent as it places a prior on the temporal attention span of the network to the most recent few input observations. We, therefore, design a novel model to carefully denoted this assumption.

6.2 Design and Learn Neural Circuit Policies

Conclusively, to address the representation learning challenges and the complexity of autonomous lane-keeping, we design a new end-to-end learning system that perceives the inputs by a sequel of convolutional layers, to capture image structures, and performs control by a novel RNN structure, termed a *neural circuit policy* (NCP). Neural dynamics in NCPs are represented by LTC neurons (Fig. 6.2a). An NCP network comprises four layers; sensory neurons, interneurons, command neurons, and motor neurons, similar to the Ordinary neural circuits described in Chapter 4. Command neurons recurrently synapse into each other, whereas the other layers deploy forward wiring diagram. This specific network topology is unique to the wiring diagram of the nematode

Algorithm 10 Subroutine: Connect inter to command neurons

for all $s \in N_i$ **do**
  T is random permutation of the set N_c
  for $i = 1 \dots k_i$ **do**
    p is random variable of distribution $\{50\% \mapsto 1, 50\% \mapsto -1\}$
    Add synapse $(s \to T_i)$ to E with polarity p
  end for
end for
$\mu_{in} \leftarrow \frac{1}{|N_c|} \sum_{t \in N_c} |\{s|(s \to t) \in E\}|$ ∇ Compute mean fan-in of neurons N_c
for all $t \in N_c$ such that $\nexists s : (s \to t) \in E$ **do**
  S is random permutation of the set N_i
  for $i = 1 \dots \mu_{in}$ **do**
    p is random variable of distribution $\{50\% \mapsto 1, 50\% \mapsto -1\}$
    Add synapse $(S_i \to t)$ to E with polarity p
  end for
end for

Algorithm 11 Subroutine: Recurrently connect command neurons

for $i = 1 \dots k_c$ **do**
  s is random element from N_c
  t is random element from N_c
  p is random variable of distribution $\{50\% \mapsto 1, 50\% \mapsto -1\}$
  Add synapse $(s \to t)$ to E with polarity p
end for

Algorithm 12 Subroutine: Connect command to motor neurons

for all $t \in N_m$ **do**
  S is random permutation of the set N_c
  for $i = 1 \dots k_m$ **do**
    p is random variable of distribution $\{50\% \mapsto 1, 50\% \mapsto -1\}$
    Add synapse $(S_i \to t)$ to E with polarity p
  end for
end for

C. elegans and was shown to bring attractive computational advantages [Yan et al., 2017, Kaplan et al., 2019, Lechner et al., 2019], as described in Chapter 4. At their representation core, they possess a nonlinear time-varying synaptic transmission mechanism that results in their significantly enhanced expressivity in time-series modeling, compared to their deep learning counterparts (Chapter 3). *Neural circuit policies'* main distinctions to ordinary neural circuits (Chapter 4) are three-folds; I) they are human-designed networks based on a novel generalized design principles, II) their wiring diagram is not taken directly from natural neural circuits. III), they are equipped with convolutional heads to be able to process high-dimensional input data (such as images), whereas ordinary neural circuits (Chapter 4) were limited in this regard.

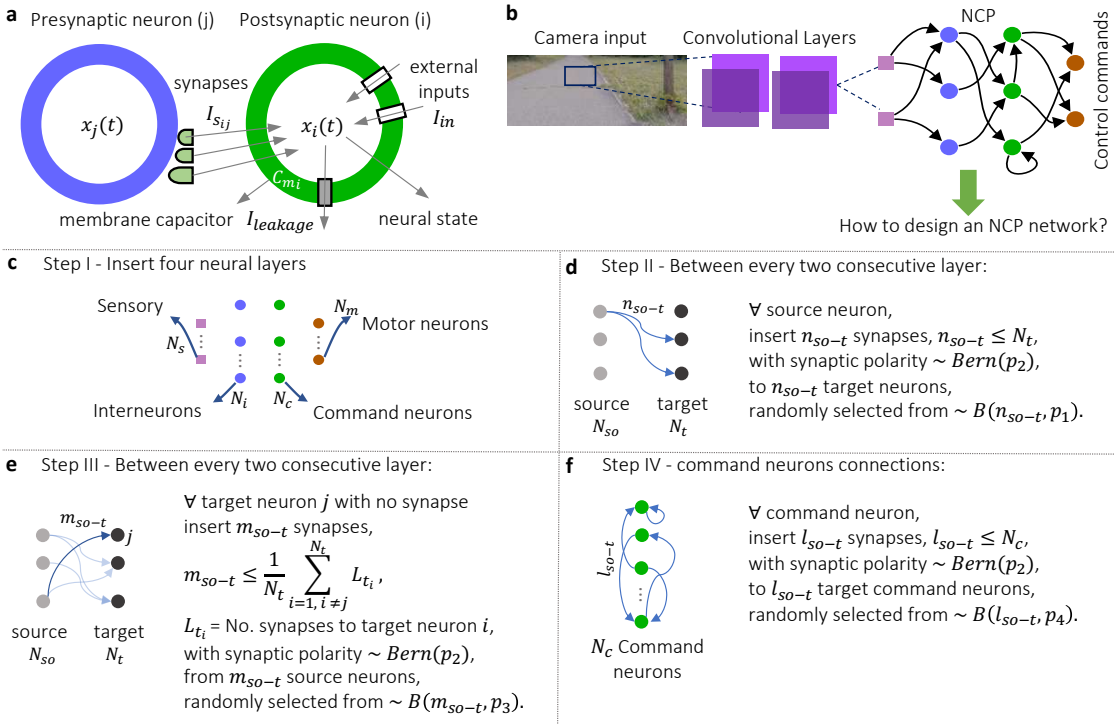


Figure 6.2: Designing NCP networks with LTC neural model. **a.** Representation of the neural state, $x_i(t)$, of a postsynaptic LTC neuron i receiving input currents from presynaptic neuron, j. **b.** Representation of an NCP end-to-end network; it perceives the camera inputs by a set of convolutional layers, then delivers a latent representation to the sensory neurons of a designed NCP (based on the steps described in c to f), to command control orders **c.** NCP design step I **d.** NCP design step II **e.** NCP design step III **f.** NCP design step IV.

6.2.1 Design NCPs

The architecture of an NCP network is determined by the design principles introduced in Figs. 6.2b to 6.2f. The design phase includes four steps to map inputs from the last convolutional layer to a steering control command. The architecture of an NCP is designed by procedurally calling subroutines given by Algorithm 8. The NCP design principles induce considerably compact and highly sparse networks of LTC neurons. The entire learning system consists of the convolutional head stacked with the NCP network (Fig. 6.2b) and is trained end-to-end by supervised learning. Given a designed NCP network, we apply a semi-implicit ODE-solver to obtain a numerically computable and stable representation of the system (Chapter 3). We then recursively fold the ODE-solver call into an RNN cell and prepare the system's training pipeline.

6.2.2 Numerical implementation of the NCP networks

To learn the parameters of an NCP circuit, we transform it into a differentiable representation. After modeling the circuit as a system of ordinary differential equations of LTC neurons, we

employ a numerical ODE solver to obtain a computable form of it. A solving method suitable for our purpose has to comply with the following three constraints: Firstly, the solver is applied to a real-time system that puts a hard limit on worst-case executing time. Hence, the solver uses a fixed step-size [Press et al., 2007a]. Secondly, the ODE model of an NCP is stiff [Press et al., 2007a, Hasani et al., 2020]. Consequently, to avoid numerical instabilities, we adopt a semi-implicit method [Press et al., 2007a]. Lastly, during the training phase, we compute partial derivatives by back-propagating through the solver. Similar to the stability arguments in the forward path, we need to monitor the error magnitude in the backward phase. In particular, a suitable solving method must not result in an exploding or a rapidly vanishing gradient. To comply with these constraints, we employed a simple Euler approach. As a result, in summary, for each neuron, we adopt a semi-implicit Euler approach with a fixed step-size of the form:

$$x_i(t+\Delta) := \frac{x_i(t)C_{m_i}/\Delta + g_{l_i}x_{leak_i}\sum_{j\in I_{in}} w_{ij}\ \sigma_i\Big(\gamma_{ij}(x_j(t)-\mu_i)\Big)E_{ij}}{C_{m_i}/\Delta + g_{l_i} + \sum_{j\in I_{in}} w_{ij}\ \sigma_i\Big(\gamma_{ij}(x_j(t)-\mu_{ij})\Big)}, \tag{6.1}$$

The set I_{in} represents the set of neurons that are presynaptic to neuron i. This equation was derived from the basic Euler formula [Press et al., 2007a]:

$$x(t+\Delta) := x(t) + \Delta \dot{f}\Big(x(t+\tau), u(t+1)\Big) \tag{6.2}$$

by setting $\tau = \Delta$ for all $x(t+\tau)$ that appear linear in $\dot{f}$, and setting $\tau = 0$ for all other occurrences.

Note that the well-known explicit (forward) Euler method can be obtained from equation 6.2 by setting $\tau = 0$. Likewise, the implicit (backward) Euler method is realized by equation 6.2 if setting $\tau = \Delta$ and solving the resulting non-linear equation for $x(t+\Delta)$. RNNs usually process their incoming input stream at a fixed sampling frequency (e.g., 30 Hz in the described end-to-end driving tasks). To achieve a decent precision - a computation complexity trade-off, we simulated the ODE at a frequency, six times higher than the input sampling rate; we packed 6 ODE solver steps into one RNN step. In both the training and testing phases, we initialized states of the ODE/RNN by zeros.

6.2.3 Training Procedure

Here we describe the training procedure of the models. If not stated otherwise, this description applies to the passive and active test scenarios. We formulated the end-to-end autonomous driving as a regression task. Hence, we adopted the square-error as the training loss function. As recordings of curves and turns are underrepresented in the training data, we multiplied a weighting factor to the loss value of each sample. This weighting factor $w(y) := \exp(|y|\alpha)$ depends on the target curvature y exponentially, thus assigns a higher priority to samples containing road-curves and turns. As the test track is located in a forest area where trees cast shadows with variable profiles on the road, we implemented a shadow augmentation data technique during training. In essence, we draw a semi-transparent black or white line over each training image. The location,

Algorithm 13 Training NCP networks with gradient descent

Require: Training set (X,Y), validation set $(\hat{X},\hat{Y})$, neural network $f(x,w) \mapsto y$, loss $L(y,\hat{y}) \mapsto \mathbb{R}$, minibatch size k

```
Initialise weights w
l_best := ∞
for e = 1 ... max_epochs do
    for i = 1 ... ⌊|X|/k⌋ do
        (x, y) is random batch of size k from (X, Y)
        w := w − α ∂L(y, f(x, w)))/∂w        ∇ stochastic gradient descent
    end for
    l_e := 1/|X̂| Σ_{(x,y)∈(X̂,Ŷ)} L(y, f(x, w))        ∇ validation loss
    if l_e < l_best then
        l_best := l_e
        w_best := w
    end if
end for
return w_best
```

orientation, and width of lines are randomly sampled from uniform distributions. We trained all models, except the feedforward CNN, on sub-sequences of 16 time-steps, which correspond to 0.53 real-time seconds. The neural state of standard CT-RNN and LSTM implementations are unbounded, which may lead to instabilities during closed-loop testing, as they are only trained on finite sequences. To avoid the internal states of the controller to grow indefinitely, i.e., a phenomenon known in control-theory as wind-up [Bohn and Atherton, 1995], we apply a clipping operation to the states of the CT-RNN and LSTM to keep the values within the range $[-5,5]$. We used *Adam* [Kingma and Ba, 2014] as the optimization algorithm with parameters shown in Table 6.2.

The convolutional layers' architecture for all RNN models are listed in Table 6.4. After the last convolutional layer, we applied four per-channel linear layers to obtain $8 \times 4 = 32$ latent features serving as sensory inputs to the RNN compartment. We empirically tuned the learning rates and the convolutional layers' hyperparameters and evaluated them on the passive dataset. We observed that NCP took advantage of a lower learning rate for the convolutional layers and a higher learning rate for the RNN compartment. We apply a per-image whitening filter to the images before feeding them into the networks.

From the gradient propagation perspective, our approach gave rise to a vanishing gradient phenomenon, which, as described in Fig. 6.1d is the preferable setting for learning a real-world autonomous vehicle control.".

6.3 Experimental setup

End-to-end driving is a feedback control problem, where control actuated by the agent proprioceptively affects future observations. However, during the supervised training phase, this feedback mechanism is utterly disregarded. We observed that such a train-test discrepancy led to situations where a trained neural network model that performs exceptionally well on the labeled sequences in an offline testing environment (See Table 6.1) fails to steer the car safely in a real testing case. Modern RNNs are particularly vulnerable in the described scenarios, as their decision-making process heavily relies on past observations. Hence, to properly assess the performance of the models, we chose the architectures that performed well during an offline test and evaluated them actively on a real car. We ran a 10-fold cross-testing [Bengio and Grandvalet, 2004] on ninety-four minutes of labeled sequences recorded in the Boston metropolitan area.

6.3.1 Vehicle setup

All data used to train networks was collected on a Toyota Prius 2015 V retrofitted with perception sensors (a forward-facing Leopard Imaging LI-AR0231-GMSL camera), inertial measurement unit (Xsens MTi 100-series IMU), GPS, and drive-by-wire steering [Naser et al., 2017]. All data logging was done directly on an NVIDIA Drive PX2, which is the in-car high-performance computing platform. The IMU was used to record rotation of the vehicle's rigid body frame and thus compute the curvature of the vehicle's traversed path. Specifically, given a yaw rate γ_t $(rads/sec)$, and the speed of the vehicle, v_t (m/sec), we compute the curvature of the path as:

$$y_t = \frac{1}{r_t} = \frac{\gamma_t}{v_t} \tag{6.3}$$

where r_t is the radius of the traversing circle. Ultimately, for the networks learned, we consider the problem of directly learning a control command from the human traversed road curvature (y_t) instead of the steering wheel angle (α_t). This is because α_t is a nonlinear function of both y_t and v_t and depends on the tire slip angle, road surface, weather conditions, and vehicle dynamics. This results in the notion that simply learning the steering wheel angle (i.e., *what* the human commanded) is not sufficient for autonomous navigation, and instead, we require knowledge of the traversed road curvature (i.e., *where* the human actually drove). For controlling the car at inference time, we can compute the steering wheel angle online using a bicycle model approximation:

$$\alpha_t = K \arctan(L y_t) \tag{6.4}$$

where K is the steering ratio (i.e., the ratio between steering and tire angle), and L is the length of the vehicle.

6.3.2 Passive test dataset

For the passive evaluation, we collected approximately five hours of driving data throughout diverse regions of the Boston metropolitan area during dry, wet, and snowy weather conditions on the highway, local, and residential roads. We processed the data by removing ambiguous segments such as lane switches, crossings, and congestion, from the recordings. We split the

Table 6.1: Results of the passive lane-keeping 10-fold cross-testing evaluation. As the squared errors on the test set are determined by large outliers, we reported squared and absolute error. Sparse LSTM models are trained with projected gradient descent to enforce a 95% sparsity level. All tested NCP architectures are composed of 19 neurons.

Model	Training square error	Test squared error
CNN	1.41 ± 0.30	4.28 ± 4.63
Vanilla RNN	0.14 ± 0.05	3.39 ± 4.39
CT-GRU	0.19 ± 0.05	3.63 ± 4.61
CT-RNN (19 units)	0.44 ± 0.14	3.62 ± 4.35
CT-RNN (64 units)	0.23 ± 0.09	3.43 ± 4.55
Sparse CT-RNN (19 units)	0.77 ± 0.35	4.03 ± 4.80
Sparse CT-RNN (64 units)	0.40 ± 0.43	3.72 ± 4.71
GRU	1.25 ± 1.02	5.06 ± 6.64
LSTM (64 units)	0.19 ± 0.05	$\mathbf{3.17} \pm 3.85$
LSTM (19 units)	0.16 ± 0.06	3.38 ± 4.48
Sparse LSTM (19 units)	1.05 ± 0.57	3.68 ± 5.21
Sparse LSTM (64 units)	0.29 ± 0.14	3.25 ± 3.93
NCP	0.43 ± 0.26	$\mathbf{3.22} \pm 3.92$
NCP (randomly wired)	2.12 ± 2.93	5.19 ± 5.43
NCP (fully-connected)	2.41 ± 3.44	5.18 ± 4.19

data into ten non-overlapping sets of equally sized chunks for the cross-testing procedure. For every ten sets, we trained a model on the union of the remaining nine sets, then evaluated the performance of the model on the withhold test set. The number of training epochs was optimized based on a validation set, which we separated from the union of the nine sets before training. In Table 6.1, we reported the mean and standard deviation over these ten test iterations.

6.3.3 Active test setup.

We conducted the active driving experiments on a private road system. To prepare the models, we collected approximately 94 minutes of data by maneuvering the vehicle through the test track. We split the data into a training and a validation set of ratio 3:1. The number of training epochs was selected based on the lowest error on the validation set achieved during training. See a list of full training parameters in Table 6.2. We tested each trained model 5 times around the test-track, without input perturbations, and two times while the input was disrupted by a zero-mean Gaussian distribution with variances 0.1, 0.2, and 0.3. Each evaluation consists of driving the car around one cycle of the outermost path of the track, in the counterclockwise direction. We started an evaluation by placing the vehicle at a designated initial location, accelerating the vehicle up to a constant speed of 4.47 m/s, and delegating the control of the steering system to the neural network. Every time the vehicle was maneuvered off the road, we manually steered the car back on track and reported a crash (Fig. 6.4a). For testing a model under noise, we connected a random number generator to the input stream, which added zero-mean Gaussian noise to the camera images. The variance of the Gaussian noise distribution was determined by the noise intensity level, i.e., level 1: 0.1, level 2, 0:2, and level 3: 0.3. The input images were scaled to the range [0,1] before the addition of the noise. To have the same noise pattern for all models, we fixed the initial seed of the random number generator to a constant value.

To perform a fair comparison, we equipped all RNN models with the same convolutional head that reduces the dimensionality of the input image to a more compact latent representation to be

Table 6.2: Models' training hyperparameters. The values of all the hyperparameters were selected informally through empirical evaluation over the passive training dataset. We did not search through the hyper-parameters space exhaustively, due to the computational cost. However, the use of a systematic meta-learning algorithm over these parameter spaces can presumably result in achieving better performances.

Variable	Value	Comment
Input resolution	200-by-78	Pixels
Input channels	3	8-bit RGB color space
Learning rate	$5 \cdot 10^{-4}$	CNN, LSTM, CT-RNN
Learning-rate	$1 \cdot 10^{-3}$	NCP (RNN compartment)
Learning-rate	$2.5 \cdot 10^{-5}$	NCP (convolutional head)
β_1	0.9	Parameter of Adam
β_2	0.999	Parameter of Adam
Minibatch-size	20	
Training sequences length	16	time-steps
α	0.1	Parameter in the weighting factor
Max. number of training epochs	100	Validation set determines actual epochs
δ_1	0.5	Dropout-rate of the CNN
δ_2	0.5	Dropout-rate of the CNN
δ_3	0.3	Dropout-rate of the CNN

Table 6.3: Layers of the feedforward CNN, adapted from [Bojarski et al., 2016]. Conv2D refers to a convolutional layer, F to the number of filters, K to the kernel size, S to the strides, U to the number of units in a fully-connected layer. The values of the dropout-rates δ_1,δ_2, and δ_3 were optimized on the passive benchmark and reported in Table 6.2

Layer	Type	Hyperparameters
1	Conv2D	F: 24, K: 5, S: 2
2	Conv2D	F: 36, K: 5, S: 2
3	Conv2D	F: 48, K: 3, S: 2
4	Conv2D	F: 64, K: 3, S: 1
5	Conv2D	F: 64, K: 3, S: 1
6	Flatten	
7	Dropout	δ_1
8	Fully-connected	U: 1000, ReLU
9	Dropout	δ_2
10	Fully-connected	U: 100, ReLU
11	Dropout	δ_3
12	Fully-connected	U: 1, Identity

fed into the RNN compartments (See the layer structures in Table 6.4).

We trained and evaluated the networks with the following RNN structure: 64-neuron LSTM and a 64-neuron CT-RNN, a 19-neuron NCP, and a feedforward CNN with 1100 neurons in its fully connected layers on a real driving scenario. Moreover, we compared these recurrent agents to the feedforward convolutional network tested in [Bojarski et al., 2016]. The learning curves and their

Table 6.4: Convolutional head

Layer	Filters	Kernel size	Strides
1	24	5	2
2	36	5	2
3	48	3	2
4	64	3	1
5	8	3	1

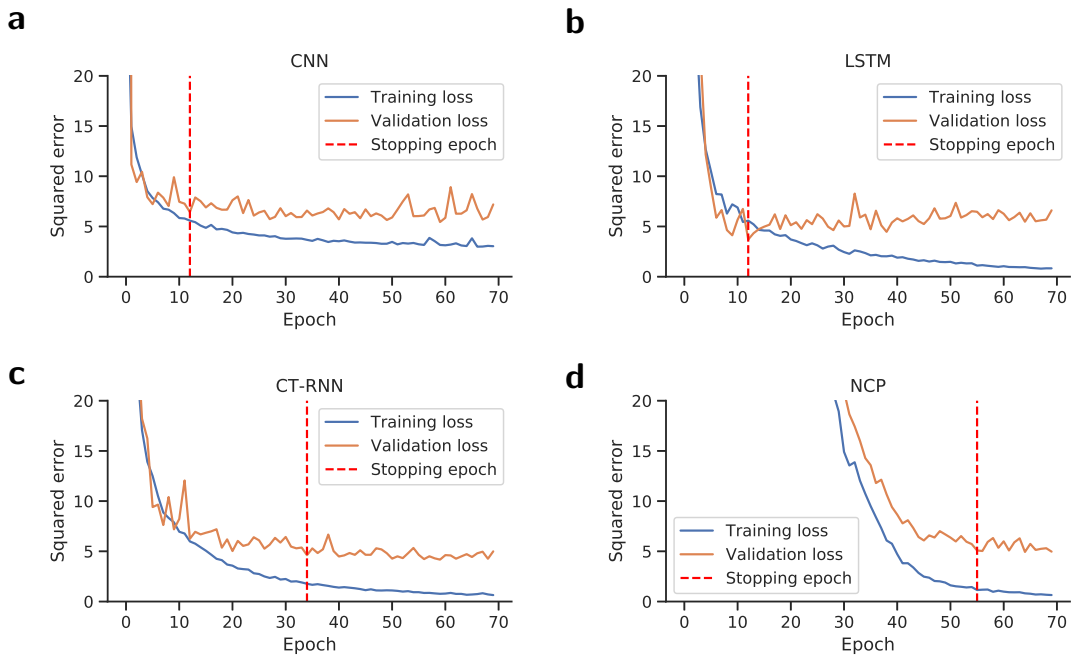


Figure 6.3: Learning curves of the models tested in the active driving experiments. Early stopping [Girosi et al., 1995, Smale and Zhou, 2007] is deployed as a regularization mechanism to obtain better generalization. The terminating epoch for each experiment, is reported in Table 6.5

corresponding termination phase are reported in Fig. 6.3 and Table 6.5

6.4 Results

6.4.1 Learning a compact neural representation"

Table 6.6 illustrates the compactness of the obtained NCP network compared to the other models. NCP is 1.8 orders of magnitude more compact compared to the CNN network that established the state-of-the-art of end-to-end driving [Bojarski et al., 2016]. Its control-network is 85 times sparser than that of LSTM and 24 times than CT-RNN networks, with 24 times smaller trainable parameter space compared to that of LSTMs.

Table 6.5: The learning termination shown in Fig. 6.3. Training and validation metrics of the models tested in the active driving experiment. As discussed thoroughly (Fig. 6.4), LSTM model achieves the best performance in the passive test but fails to express proper driving behaviour under environmental disturbances.

Model	Stopping epoch	Training loss	Validation loss
CNN	12	5.58	6.44
CT-RNN	34	1.77	4.62
LSTM	12	5.58	3.70
NCP	55	1.15	5.09

Table 6.6: Network size comparison

Model	Conv layers Param	RNN neurons	RNN synapses	RNN trainable param
CNN	5,068,900	-	-	-
CT-RNN	79,420	64	6112	6273
LSTM	79,420	64	24640	24897
NCP	79,420	**19**	**253**	**1065**

The results achieved by such a compact neural network is impressive in multiple aspects of an ideal autonomous mobile robot controller, described as follows:

6.4.2 Divergence from the road by the increasing input noise

Compared to all learning systems under-test, NCPs are significantly more resilient to the rising pixel-wise input perturbations, to go off-road — to crash (Fig. 6.4a).

6.4.3 Robustness of the output decisions in the presence of input noise

Figs. 6.4d and 6.4e depict examples of crash incidents, happened at the locations spotted on the map when the inputs to the networks were heavily perturbed by an input noise. Figs. 6.4d and 6.4e illustrate how the attention of each intact network is disrupted by the input noise and caused LSTM and CNN networks to drive the vehicle off-road. We quantified the influence of the input perturbations on the attention maps (the learned causal structure), by computing their structural similarity index (SSIM), represented in Fig. 6.4b. The figure indicates the performance domination of NCPs over other networks, in preserving the learned causal structure (SSIM closer to 1) as a result of input noise

Computing saliency maps of convolutional layers

Saliency maps are interpretation methods to visualize the inner workings of a trained neural network by highlighting parts of the input image that contributed most to the decision of a network. We employ saliency maps to analyze what our networks have learned to attend qualitatively. In particular, we are interested in how layers that are common to all tested architectures evolve differently during training. Consequently, we narrow our analysis to the convolutional layers at the beginning of the network. We adopted a technique named VisualBackProp [Bojarski

Algorithm 14 Compute saliency maps

Inputs: Convolutional feature maps $h_1, h_2, \ldots h_N$
$s :=$ average-over-channel-dimension(h_N)
for i **from** N-1 **to** 1 **do**
  $z :=$ average-over-channel-dimension(h_i)
  $s := z \odot$ deconvolution-to-size-of(s,z)
end for
return s.

et al., 2018] that has been developed deliberately for autonomous driving research, to compute the saliency maps presented. This method leverages the property of the ReLU activation that the value of each neuron in the feature map is either positive or zero. Where $\odot$ represents the element-wise multiplication and the deconvolution-to-size-of function scales the first argument to the dimension of the second argument by applying a de-convolution operation.

Structural Similarity Index

SSIM is a method to compare quality of given images [Wang et al., 2004]. It is composed of multiplication of three comparison criteria, the luminance l, contrast c and structure s for given images x and y as follows [Wang et al., 2004]:

$$SSIM(x,y) = [l(x,y)]^{\alpha}.[c(x,y)]^{\beta}.[s(x,y)]^{\gamma}, \tag{6.5}$$

where:

$$l = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1}, \quad c = \frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2}, \quad s = \frac{\sigma_{xy} + C_3}{\sigma_x\sigma_y + C_3}. \tag{6.6}$$

Here, C_1,C_2, and C_3 are the regularization constants, μ_x and μ_y are the means of x and y, σ_x and σ_y are standard deviations, and σ_{xy} is the cross-covariance of x and y. In the analysis provided in Fig. 6.4b, We computed the SSIM for pair saliency maps at each time-frame (overall 200 frames), between a noise-free version as reference and a perturbed one resulted from input noise injections. we set the exponents $\alpha = \beta = \gamma = 1$, the regularization components $C_1 = (0.01L)^2$, $C_2 = (0.03L)^2$, and $C_3 = C_2/2$, with $L = 255$ corresponding to the dynamics range of the input image values.

6.4.4 Driving with smooth neural activity

We also quantitatively measured the maximum steepness of the neural dynamics derivative (maximum local Lipschitz constant) for all neurons and reported the results in Fig. 6.4c top. We observed that the local decision-making process in NCPs is remarkably smoother than those of other network types (Fig. 6.4c bottom).

Lipschitz Continuity Computation

The limitation on the speed of change of a function can be computed by the Lipschitz continuity criteria ($|f(t_2) - f(t_1)| \leq L|t_2 - t_1|$). The smaller is the Lipschitz constant (L), transitions on

Algorithm 15 Compute Maximum Lipschitz Constants

Inputs: $X^{(N\times T)}$ n = number of neurons, T = length of the test episode
for n **from** 1 **to** N **do**
  for t **from** 1 **to** $T-1$ **do**
    $L(n,t) = \frac{dX}{dt} \approx X(n,t+1) - X(n,t)/\Delta t, \Delta t = 1$
  end for
end for
$L_{max}^{(N\times 1)} = \max(L^{(N,T)})$
Return $L_{max_{sorted}}$ =sort(L_{max}).

function f, are smoother. The following algorithm computes the maximum lipschitz constants for neural state activity of RNNs $X(t)$, for an episode of active testing for all RNN types reported in Fig. 6.4c:

6.4.5 Comparing learned causal structures

Figs. 6.5b to 6.5e represent instances of learned networks' attention maps during live testing. We observed that the desirable causal structure of autonomous lane-keeping — learning to drive by attending to a road's horizon) was concisely learned by the NCP network only. LSTM learned to attend to the roadsides in most scenarios; however, lighting conditions, as well as road profiles, can significantly affect the network's attention portfolio (Fig. 6.5b). CT-RNN's attention is inconsistent and is heavily influenced by the variations of the road's lighting conditions (Fig. 6.5c). CNN learned to drive by focusing on the side roads and ignoring the road itself (Fig. 6.5d). (See the entire saliency maps collected during live driving at `https://pub.ist.ac.at/~mlechner/driving/saliency_widget/saliency_maps.html`

6.4.6 Global network dynamics

To measure how concisely the networks learned the primitives of driving (straight roads, handling curves, and road jitters), we conducted a principle component analysis (PCA) and reported their variance explained in Figs. 6.5f to 6.5i.

PCA

For the analysis provided in Fig. 6.5f to 6.5n, we computed the principle components of the activity of individual neurons in every RNN network, collected from episodes of active driving tests. The output trajectory (Steering command) is not included in the PCA analysis, to investigate how the global purpose of the network is expressed by the main PCs of a network's neurons, standalone.

The PCA analysis demonstrated that a single principle component (PC) of NCP's neural dynamics could explain 92% (Fig. 6.5j) of the entire driving profile. To motivate this phenomenon further, we plotted the first and the second PC scores over the driving trajectory for all networks (Figs. 6.5k to 6.5n). NCP is the only model amongst the others that allocated different PC1 values to

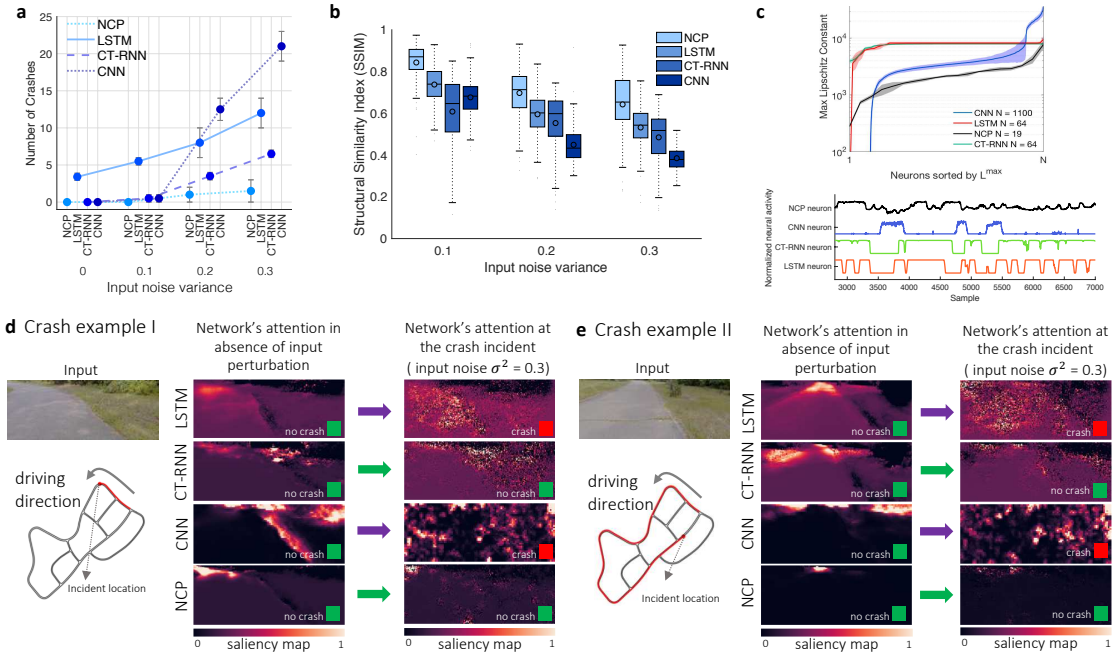


Figure 6.4: Robustness analysis. **a,** Number of crashes (steering commands with tendency to drive the vehicle off-road) for four RNN types, as the input noise variance increases, in active driving test (n=3) **b,** Variation of the structural sensitivity index of the saliency maps for four RNN topologies as the input noise variance increases, A higher value of SSIM is preferable (n=3) **c,** Maximum Lipschitz constant (an indicator of smoothness and stable dynamics) of the activity of every single neuron (sorted by the amplitude of the max Lipschitz constant on the horizontal axis) (n=5), Lower values are preferable. **d,** Example of the saliency maps before a crash event caused by LSTM and CNN, in the presence of input perturbations, and how it was handled by CT-RNN and NCP. **e,** A second example of a crash incident caused by LSTM and CNN. **Watch videos** of the driving performance with no input perturbations: NCP, LSTM, CT-RNN and CNN. And with a $\sigma^2 = 0.3$ input perturbation: NCP, LSTM, CT-RNN and CNN.

the curves and driving straight, while *fine-grained* control has been largely captured by its second PC. Other baseline networks require at least two to three PCs to capture the details of the driving profile up to 90% (6.5f to 6.5i).

Cell-level interpretability. Neural state (the amplitude of a neuron's output) and the coupling sensitivity (how a neuron adjusts its reaction speed when interacting with the environment) of LTC cells comprising an NCP network (Fig. 6.6a), can skillfully help understand how an LTC network's decision is made. Figs. 6.6b to 6.6d illustrates the activity of five selected neurons from the NCP driving agent, projected over the driving trajectory. The *motor neuron*'s activity illustrates how main motion primitives are delivered to various driving situations (Fig. 6.6b left).

Its coupling sensitivity demonstrates that the neuron tends to become more cautious during turns (setting smoother dynamics) while keeping its reaction speed at a relatively constant-rate during

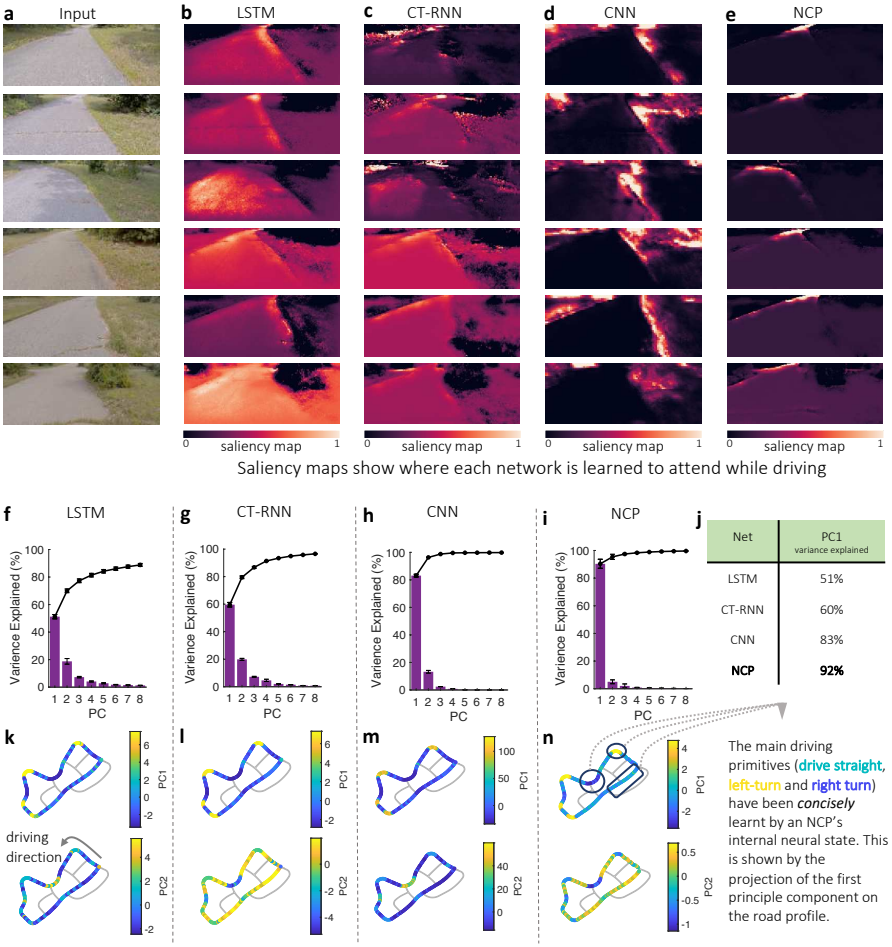


Figure 6.5: Global network dynamics. **a,** A sequence of input camera images during the active testing. b to e present a sequel of saliency maps computed to obtain the attention of the convolutional layers of the trained networks while driving. **b,** LSTM learned to attend to the roadsides in most scenarios; however, lighting conditions significantly affect its attention portfolio. **c,** CT-RNN's attention is inconsistent and is heavily influenced by the variations of the road's lighting conditions. **d,** CNN learned to drive by focusing on the side roads **e,** NCP learned to attend to the road's horizon when taking a driving decision. f to i represent the variance explained by first eight principle components of the activity of all neurons of **f,** LSTM, **g,** CT-RNN, **h,** CNN, and **i,** NCP, (n=5). The black line indicates the cumulative variance explained. **j,** First PC's variance-explained for all models. k to n shows the projection of the first (top) and the second (bottom) PC's score (The score of the n^{th} PC is computed by $PC^{(n)}_{score} = output\ vector \times weight_{PC^{(n)}}$), over the driving trajectory for **k,** LSTM, **l,** CT-RNN, **m,** CNN, and **n,** NCP.

straight motions. *Interneuron 1* learned to light up during left-turns (Fig. 6.6c top-left) while adjusting its dynamics to react faster at left-turning events Fig. 6.6c top-right). *Interneuron 2*, on

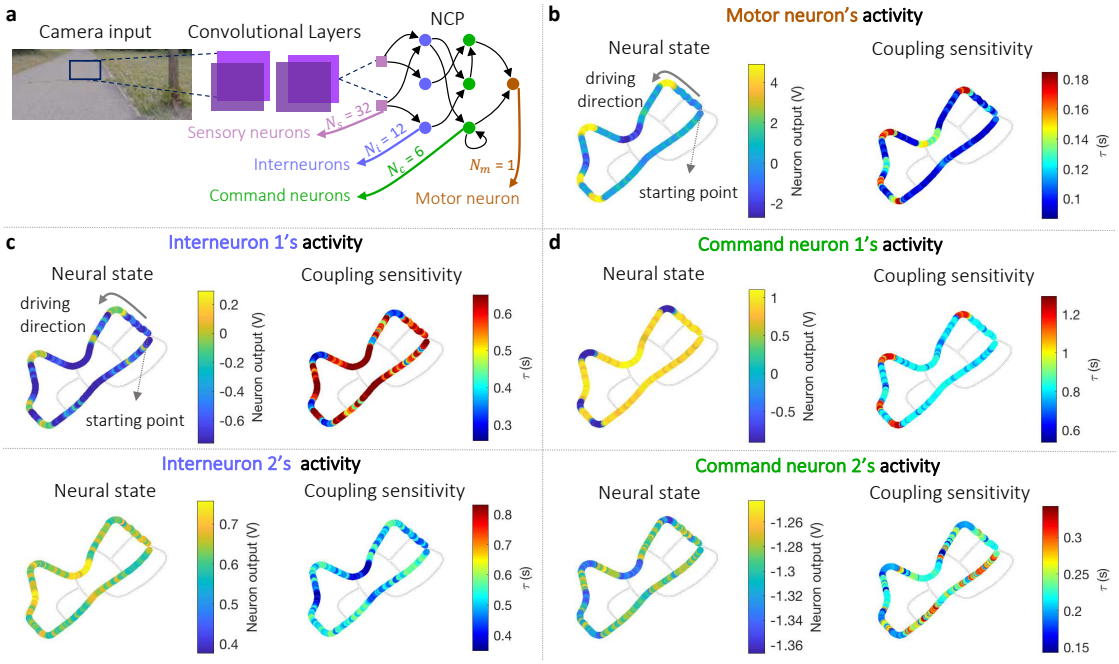


Figure 6.6: Intuitive comprehension of NCP's cells activity while driving. **a,** An NCP network trained end-to-end for autonomous lane-keeping. b to d depict examples of neural activities projected over the road-trajectory on which the car was driven. The neural state (representing the amplitude of a neuron's dynamics) and the coupling sensitivity (representing how a neuron adjusts its reaction speed) are plotted in each subsection. **b,** Neural activity of the motor neuron. **c,** Neural activity of two interneurons 1 and 2. **d,** Neural activity of two command neurons 1 and 2. An immediate explanation of the cell-level dynamics for the NCP network is achieved and is expandable to every internal element of the network. See more in Figs. 6.7 and 6.8.

the other hand, learned to rapidly get more activated during right-turns (Fig. 6.6c bottom).

Command neurons 1 is consistently activated during straight driving with a sensitive reaction speed while it is switched off on left-turns (Fig. 6.6d top). *Command neuron 2* is biased at lower membrane potentials and tunes the road jitters when the vehicle drives on a straight path (Fig. 6.6d bottom). This degree of immediate interpretation of dynamics is generalisable to every single cell within an NCP (Figs. 6.7 and 6.8). Such an attribute of NCPs, for the first time, opens the neural networks' black-box nature and significantly enhances our confidence to deploy them in safety-critical applications such as autonomous driving.

6.5 Conclusions

Neural Circuit policies are the smallest neural network agent that can proficiently control a vehicle on previously unseen environments, while demonstrating robustness to input artifacts, deploy causality, and realize interpretable dynamics. Real-world domains such as autonomous

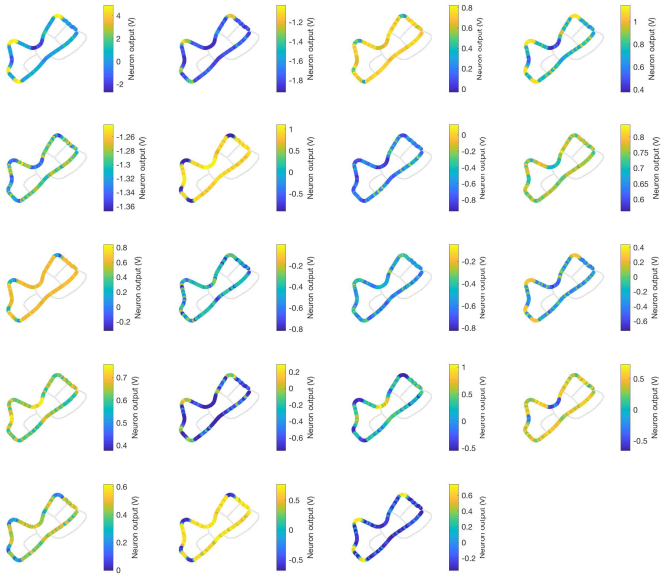


Figure 6.7: Neural activity of all NCP neurons presented in Fig. 6.6

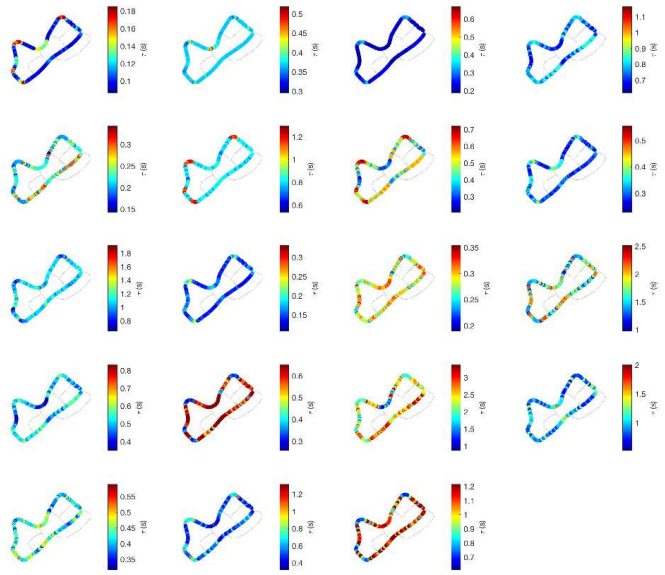


Figure 6.8: Coupling sensitivity of all NCP neurons presented in Fig. 6.6

driving, robotics, health, and medicine, are surrounded by environmental artifacts and uncertainty, and demands for robust real-time decision making. Moreover, similar to autonomous driving, many applications deal with complex high-dimensional input-output spaces that, when deployed in the real world, become safety-critical. The success of neural circuit policies in autonomous driving indicates that tackling the complexity of real-world problems does not necessarily require

complex deep neural network architectures.

Furthermore, we witnessed that in safety-critical domains, achieving great performance is not the only criteria to take into account when designing intelligent agents. It is of utmost importance to have a notion for the interpretation of the system. Such explanations of the network dynamics were achievable for networks consisting of LTC neurons. Videos of the driving performance of all learning algorithms can be viewed by the links provided in Table 6.7. Now the question is, how can we understand network dynamics of successful modern RNNs? In the next chapter, we introduce a generic method to obtain more understanding of other RNN types of network dynamics.

Table 6.7: Videos of the driving performance of all algorithms

Model	**Perturbation profile**	**Link**
CNN	no noise	https://youtu.be/5rwuF8dv3QM
CNN	noise with $\sigma^2 = 0.1$	https://youtu.be/bBgK2D34Ta8
CNN	noise with $\sigma^2 = 0.2$	https://youtu.be/UR-1w7kCbhU
CNN	noise with $\sigma^2 = 0.3$	https://youtu.be/en35asvYYtI
CT-RNN	no noise	https://youtu.be/5-1xpXQpJDg
CT-RNN	noise with $\sigma^2 = 0.1$	https://youtu.be/kMXXNmZJ1u8
CT-RNN	noise with $\sigma^2 = 0.2$	https://youtu.be/OD5O4u6JNh4
CT-RNN	noise with $\sigma^2 = 0.3$	https://youtu.be/_JdZpP0zx7k
LSTM	no noise	https://youtu.be/g_cosCKkHWc
LSTM	noise with $\sigma^2 = 0.1$	https://youtu.be/Z7Y3TXzT0yU
LSTM	noise with $\sigma^2 = 0.2$	https://youtu.be/JYVaU4c7iYI
LSTM	noise with $\sigma^2 = 0.3$	https://youtu.be/B6ITsPre2fU
NCP	no noise	https://youtu.be/_ejdrdU68os
NCP	noise with $\sigma^2 = 0.1$	https://youtu.be/KB8KWZQ6Iyg
NCP	noise with $\sigma^2 = 0.2$	https://youtu.be/lanjh-WnuR8
NCP	noise with $\sigma^2 = 0.3$	https://youtu.be/6ns9jteFfMQ

CHAPTER 7

Interpretability of Recurrent Neural Networks

7.1 Motivation

© [A7-Chapter7, 2019] — A key challenge for modern deep learning architectures is that of robust interpretation of the hidden dynamics and how they contribute to the system's decision-making ability as a whole. Many safety-critical applications of deep neural networks (NNs), such as robotic control and autonomous driving [Amini et al., 2018a, Bojarski et al., 2016, Levine et al., 2016, Mnih et al., 2015, Pomerleau, 1989], require metrics of explainability before they are deployed into the real world. In particular, interpreting the dynamics of recurrent neural networks (RNNs), which can process sequential data and are vastly used in such safety-critical domains, requires careful engineering of the network architecture [Karpathy et al., 2015]. This is because investigating their behavior enables us to reason about their hidden state-dynamics and thus design better learning models.

The hidden state representations of long short-term memory (LSTM) networks [Hochreiter and Schmidhuber, 1997], a subset of RNNs with explicit gating mechanisms, have been evaluated by gate-ablation analysis [Chung et al., 2014, Greff et al., 2017] and feature visualization techniques in linguistics [Karpathy et al., 2015, Strobelt et al., 2018]. While these studies provide criteria for networks with interpretable cells, they are empirically limited to feature visualization techniques, focus on hidden state dynamics in networks for text analysis, and thus suffer from poor generalizability to other domains. A robust, systematic method for assessing RNN dynamics across all sequential data modalities has yet to be developed.

In this chapter, we introduce a novel methodology to predict and interpret the hidden dynamics of LSTMs at the individual cell and global network level. We utilize response characterization techniques [Oppenheim and Young, 1983], wherein a dynamical system is exposed to a controlled set of input signals, and the associated outputs are systematically characterized. Concretely, we

present a systematic testbench to interpret the relative contributions, response speed, and even the phase-shifted nature of learned LSTM models. To analyze hidden state dynamics, we isolate individual LSTM cells from trained networks and expose them to defined input signals such as step and sinusoid functions. Through the evaluation of output attributes, such as response settling time, phase-shift, and amplitude, we demonstrate that it is possible to predict sub-regions of the network dynamics, rank cells based on their relative contribution to network output, and thus produce reproducible metrics of network interpretability.

For example, step response settling time delineates cells with fast and slow response dynamics. In addition, by considering the steady-state value of the cellular step response and the amplitude of the sinusoid response, we are able to identify cells that significantly contribute to a network's decision. We evaluate our methodology on a range of sequential datasets and demonstrate that our algorithms scale to large LSTM networks with millions of parameters.

The key contributions of this chapter can be summarized as follows:

1. Design and implementation of a novel and lightweight algorithm for systematic LSTM interpretation based on response characterization; and

2. Evaluation of our interpretation method on four sequential datasets including classification and regression tasks; and

3. Detailed interpretation of our trained LSTMs on the single-cell scale via distribution and ablation analysis as well as on the network scale via network capacity analysis.

First, we discuss related work in Sec. 7.2 and introduce the notion of RNNs as dynamic systems in Sec. 7.3. Sec. 7.4 presents our algorithm for response characterization and defines the extracted interpretable definitions. Finally, we discuss the interpretations enabled by this analysis in Sec. 7.5 through a series of experiments, and provide conclusions of our results in Sec. 7.6.

7.2 Related Works

Deep Neural Networks Interpretability - A number of impactful attempts have been proposed for interpreting deep networks through feature visualization [Erhan et al., 2009, Zeiler and Fergus, 2014, Yosinski et al., 2015, Karpathy et al., 2015, Strobelt et al., 2018, Bilal et al., 2018]. Feature maps can be empirically interpreted at various scales using neural activation analysis [Olah et al., 2018], where the activations of hidden neurons or the hidden-state of these neurons are computed and visualized. Additional approaches try to understand feature maps by evaluating attributions [Simonyan et al., 2013, Fong and Vedaldi, 2017, Kindermans et al., 2017, Sundararajan et al., 2017]. Feature attribution is commonly performed by computing saliency maps (a linear/nonlinear heatmap that quantifies the contribution of every input feature to the final output decision). The contributions of hidden neurons, depending on the desired level of interpretability, can be highlighted at various scales ranging from individual cell level to the channel and spatial filter space, or even to arbitrary groups of specific neurons [Olah et al., 2018].

A dimensionality reduction method can also be used to abstract from high dimensional feature maps into a low dimensional latent space representation to qualitatively interpret the most important underlying features that maximally describe the input [Bishop and Tipping, 1998, Gulrajani et al., 2016, Maaten and Hinton, 2008] or contribute towards the overall learning task [Amini et al., 2018b]. However, these methods often come with the cost of decreasing cell-level audibility.

Richer infrastructures have been recently developed to reason about the network's intrinsic kinetics. LSTMVis [Strobelt et al., 2018], relates the hidden state dynamics patterns of the LSTM networks to similar patterns observed in larger networks to explain an individual cell's functionality. A systematic framework has also been introduced that combines interpretability methodologies across multiple network scales [Olah et al., 2018]. This enables exploration over various levels of interpretability for deep NNs; however, there is still space to incorporate more techniques, such as robust statistics [Koh and Liang, 2017], information theory approaches [Shwartz-Ziv and Tishby, 2017], gradients in correlation-domain [Hasani et al., 2018c] and response characterization methods which we address here.

Recurrent Neural Networks Interpretability - Visualization of the hidden-state of a fixed-structure RNNs on text and linguistic datasets identifies interpretable cells that have learned to detect certain language syntaxes and semantics [Karpathy et al., 2015, Strobelt et al., 2018]. RNNs have also been shown to learn input-sensitive grammatical functions when their hidden activation patterns were visualized [Kádár et al., 2015, Kádár et al., 2017]. Moreover, gradient-based attribution evaluation methods were used to understand the RNN functionality in localizing keywords in the text. While these techniques provide rich insight into the dynamics of learned linguistics networks, the interpretation of the network often requires detailed prior knowledge about the data content and language. Therefore, such methods may face difficulties in terms of generalization to other forms of sequential data such as time-series forecasting as well as vision-based classification, which we focus on in our study.

Another way to build interpretability for RNNs is using the attention mechanism where the network architecture is constrained to attend to a particular part of the input space. RNNs equipped with an attention mechanism have been successfully applied in image captioning [You et al., 2016], the fine-alignment in machine translation [Luong et al., 2015], and text extraction from documents [Hermann et al., 2015]. Hidden-state visualization is a frequently shared property of all of these approaches in order to effectively understand the internals of the network. Hudson et al. [Hudson and Manning, 2018] also introduced Memory, Attention, and Composition (MAC) cells which can be used to design interpretable machine reasoning engines in an end-to-end fashion. MAC is able to perform highly accurate reasoning, directly from the data. However, application of these modifications to arbitrary network architectures is not always possible, and in the case of LSTM specifically, the extension is not possible in the current scope of MAC.

Recurrent Neural Networks Dynamics- Rigorous studies of the dynamical systems properties of RNNs, such as their activation function's independence property (IP) [Albertini and Sontag, 1994], state distinguishability [Albertini and Dai Pra, 1995], and observability [Garzon and Botelho, 1994, Garzon and Botelho, 1999] date back to more than two decades. Thorough analyses of how the long term dynamics are learned by the LSTM networks have been conducted

in [Hochreiter and Schmidhuber, 1997]. Gate ablation analysis on the LSTM networks has been performed to understand cell dynamics [Greff et al., 2017, Chung et al., 2014]. We introduce the response characterization method, as a novel, additional building block to understand and reason about LSTM hidden state dynamics.

7.3 Dynamics of Recurrent Neural Networks

RNNs described in Chapter 2, can be formulated as dynamical systems in the form of the following differential equation (For the sake of notation simplicity, we omit the time argument, t):

$$\dot{h} = \sigma(Rh + Wx), \quad y = Cx, \tag{7.1}$$

where h denotes its internal state ($'\cdot'$ illustrates time-shift or time derivative for the discrete and continuous-time systems, respectively), x stands for the input to the system, and $R^{[n\times n]}$, $W^{[n\times m]}$ and $C^{[p\times n]}$ are real-valued matrices representing recurrent weights, input weights and the output gains, respectively. $\sigma : \mathbb{R} \to \mathbb{R}$ indicates the non-linearactivation function. In the continuous setting, σ should be locally Lipschitz (see [Albertini and Sontag, 1993] for a more detailed discussion).

For the analytical interpretation of a dynamical system, the first necessary condition is to check its observability property. A dynamical system is observable if there is some input sequence that gives rise to distinct outputs for two different initial states at which the system is started [Sontag, 2013]. Observable systems realize unique internal parameter settings [Albertini and Sontag, 1994]. One can then reason about that parameter setting to interpret the network for a particular input profile. Information flow in LSTM networks carries on by the composition of static and time-varying dynamical behavior. This interleaving of building blocks makes a complex partially-dependent set of nonlinear dynamics that are hard to analytically formulate and to verify their observability properties As an alternative, here, we propose a novel technique for identifying and quantifying hidden sub-regions of internal dynamics within the network with a quantitative and systematic approach by using response characterization.

7.4 Methodology for Response Characterization of LSTM cells

In this section, we explore how response characterization techniques can be utilized to perform systematic, quantitative, and interpretable understanding of LSTM networks on both a macro-network and micro-cell scale. By observing the output of the system when fed various baseline inputs, we enable a computational pipeline for reasoning about the dynamics of these hidden units. Figure 7.1 provides a schematic for our response characterization pipeline. From a trained LSTM network, comprising of M LSTM units, we isolate individual LSTM cells and characterize their output responses based on a series of interpretable response metrics. We formalize the method as follows:

Definition 8. *Let G, be a trained LSTM network with M hidden LSTM units. Given the dynamics of the training dataset (number of input/output channels, the main frequency components, the*

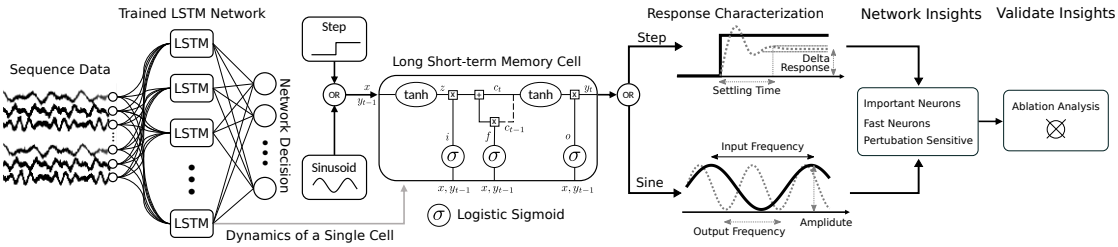


Figure 7.1: Response characterization method for LSTM cells. We take individual LSTM cells from a trained network and characterize their step and sinusoidal response. These responses predict quantitative and interpretable measures for the dynamics of the single units within the network. We then validate the predictions by performing a neuronal ablation analysis.

amplitude range of the inputs), we design specific step and sinusoidal inputs to the network, and get the following insights about the dynamics of the network at multi-scale resolutions:

- *the relative strength or contribution of components within the network;*
- *the reactiveness of components to sudden changes in input; and*
- *the phase alignment of the hidden outputs with respect to the input.*

Specifically, we analyze the responses of (1) the step input and (2) the sinusoidal input. We use the classic formulations for each of these input signals wherein (1) step: $x_t = \left[\left[t > \frac{T}{2}\right]\right]$; and (2) sinusoid: $x_t = \sin(2\pi f t)$; where $[[\cdot]]$ represents the mathematical indicator function.

Across a network of LSTM units, we can approximate sub-regions of the dynamics of a single cell, u, by extracting the input and recurrent weights corresponding to that individual cell. We then define a subsystem consisting of just that single cell and subsequently feed one of our baseline input signals, $x_t \ \forall_{t \in \{1..T\}}$ to observe the corresponding output response, y_t. In the following, we define the interpretable response metrics for the given basis input used in this study:

Definition 9. *The **initial** and **final response** of the step response signal is the starting and steady-state responses of the system respectively, while the **response output change** represents their relative difference.*

Response output change or the delta response for short determines the strength of the LSTM unit with a particular parameter setting, in terms of output amplitude. This metric can presumably detect significant contributor units to the network's decision.

Definition 10. *The **settling time** of the step response is elapsed time from the instantaneous input change to when the output lies within a 90% threshold window of its final response.*

Computing the *settling time* for individual LSTM units enables us to discover "fast units" and "slow units". This leads to the prediction of active cells when responding to a particular input profile.

Definition 11. *The **amplitude** and **frequency** of a cyclic response signal is the difference in output and rate at which the response output periodically cycles. The response frequency, $\hat{f}$, is computed by evaluating the highest energy component of the power spectral density: $\hat{f} = Argmax\ S_{yy}(f)$.*

The *amplitude* metric enables us to rank LSTM cells in terms of significant contributions to the output. This criterion is specifically effective in the case of trained RNNs on datasets with a cyclic nature. Given a sinusoidal input phase-shifts and phase variations expressed at the unit's output can be captured by evaluating the *frequency* attribute.

Definition 12. *The **correlation** of the output response with respect to the input signal is the dot product between the unbiased signals: $\sum_{t=1}^{T}(x_t - E[x]) \cdot (y_t - E[y])$.*

The *correlation* metric corresponds to the phase-alignment between input and output of the LSTM unit.

Systematic computation of each of the above response metrics for a given LSTM enables reasoning on the internal kinetics of that system. Specifically, a given LSTM network can be decomposed into its individual cell components, thus creating many smaller dynamical systems, which can be analyzed according to their individual response characterization metrics. Repeating this process for each of the cells in the entire network creates two scales of dynamic interpretability. Firstly, on the individual cell level within the network to identify those which are inherently exhibiting *fast* vs. *slow* responses to their input, quantify their relative contribution towards the system as a whole and even interpret their underlying phase-shift and alignment properties. Secondly, in addition to characterizing responses on the cell level, we also analyze the effect of network capacity on the dynamics of the network as a whole. Interpreting hidden model dynamics is not only interesting as a deployment tool but also as a debugging tool to pinpoint possible sources of undesired dynamics within the network.

While one can use these response characterization techniques to interpret individual cell dynamics, this analysis can also be performed on the aggregate network scale. After computing our response metrics for all decoupled cells independently, we then build full distributions over the set of all individual pieces of the network to gain an understanding of the dynamics of the network as a whole. This study of the response metric distributions presents another rich representation for reasoning about the dynamics, no longer at a local cellular scale, but now, on the global network scale.

7.5 Experimental Results

In the following section, we provide concrete results of our system in practice to interpret the dynamics of trained LSTMs for various sequence modeling tasks. We present our computed metric response characteristics both on the decoupled cellular level as well as the network scale and provide detailed and interpretable reasoning for these observed dynamics. We chose four benchmark sequential datasets and trained on various sized LSTM networks ranging from 32 to 320 LSTM cell networks. The results and analysis presented in this section demonstrate the

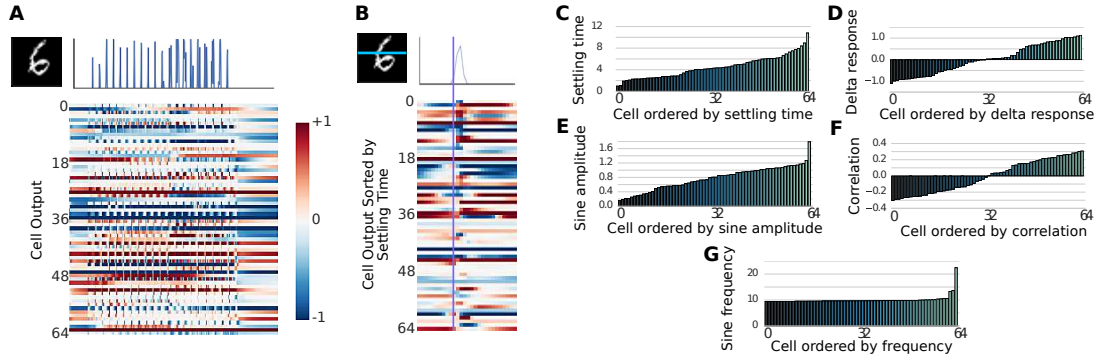


Figure 7.2: Cell-level interpretation of sequential MNIST. A) An example sequence for digit 6 together with the network dynamics for a 64-neuron LSTM network. B) One slice sequence from digit 6 and its underlying network dynamics sorted for the settling time attribute. (C-G) The distribution of hidden cells according to various response characterized distributions including (C) settling time distribution; (D) delta response distribution; (E) sine-wave amplitude distribution; (F) correlation distribution; and (G) sine-frequency distribution.

applicability of our algorithms to a wide range of temporal sequence problems and scalability towards deeper network structures.

We start by reasoning how our response characterization method can explain the hidden-state dynamics of learned LSTM networks for a sequential MNIST dataset and extend our findings to three additional datasets. The sequential MNIST dataset is a derivative of the classical MNIST dataset [LeCun et al., 2010] but instead where input images are flattened and treated as sequential time-series of brightness values. In this experiment, we perform an ablation analysis and demonstrate how some of our metrics find cells with significant contributions to the network's decision, across all datasets.

7.5.1 Response characterization metrics predict insightful dynamics for individual cells

Table 7.1: Hidden dynamic distributions by dataset. Systematic interpretation of internal dynamics distributions (mean and variance) of 128 cell LSTMs trained on various different benchmark datasets. The table shows the global speed and amplitude of the activity of the network in terms of dynamical properties of the response characterization metrics.

	Step Response		Sinusoidal Response		
Dataset	**Settle Time**	**Output Change**	**Amplitude**	**Correlation**	**Frequency**
Sequential-MNIST	6.96 ± 4.08	-0.04 ± 0.58	0.73 ± 0.30	0.17 ± 0.08	9.83 ± 0.46
S&P 500 Stock	5.62 ± 1.73	0.02 ± 0.16	0.31 ± 0.05	0.03 ± 0.02	2.86 ± 2.19
CO_2 Concentrations	5.65 ± 1.64	0.01 ± 0.12	0.27 ± 0.04	0.03 ± 0.01	9.83 ± 0.08
Protein Sequencing	7.96 ± 6.65	0.08 ± 0.54	0.68 ± 0.22	2.07 ± 1.21	10.36 ± 1.65

We start by training an LSTM network with 64 hidden LSTM cells to classify a sequential MNIST

dataset. Inputs to the cells are sequences of length 784 generated by stacking the pixels of the 28×28 hand-writing digits, row-wise (cf. Fig. 7.2A) and the output is the digit classification.

Individual LSTM cells were then isolated, and their step and sine-response were computed for the attributes defined formerly (cf. Fig. 7.4). Fig. 7.2C-G represent the distribution of cell activities, ranked by the specific metrics. The distribution of the settling time of the individual LSTM cells from a trained network, predicts low time-constant, (fast) cells (the first 20 neurons), and high-time constant (slow) cells (neurons 55-64) (Fig. 7.2C).

This interpretation allows us to indicate fast-activated/deactivated neurons at fast and slow phases of a particular input sequence. This is validated in Fig. 7.2B, where the output state of individual LSTM cells are visually demonstrated when the network receives a sequence of the digit 6. The figure is sorted with respect to the predicted settling time distribution. We observe that *fast-cells* react to fast-input dynamics almost immediately while *slow-cells* act in a slightly later phase. This effect becomes clear as you move down the heatmap in Fig. 7.2B and observe the time difference from the original activation.

The distribution of the delta-response indicates inhibitory and excitatory dynamics expressed by a 50% ratio (see Fig. 7.2D). This is confirmed by the input-output correlation criteria, where almost half of the neurons express antagonistic behavior to their respective sine-wave input (Fig. 7.2F). The sine-frequency distribution depicts that almost 90% of the LSTM cells kept the phase, nearly aligned to their respective sine-input, which indicates the existence of a linear transformation. A few cells learned to establish faster frequencies than their inputs, thereby realizing phase-shifting dynamics (Fig. 7.2G). The sine-amplitude distribution in Fig. 7.2E demonstrates that the learned LSTM cells realized various amplitudes that are almost linearly increasing. The ones with high amplitude can be interpreted as those maximally contributing to the network's decision. In the following sections, we investigate the generalization of these effects on other datasets.

7.5.2 Generalization of response metrics to other sequential datasets

We trained LSTM networks with 128 hidden cells, for four different temporal datasets: sequential MNIST [LeCun et al., 1998], S&P 500 stock prices [finance yahoo, 4 13] and CO_2 concentration for the Mauna Laua volcano [R. F. Keeling and S. J. Walker, 3 17] forecasting, and classification of protein secondary structure [Qian and Sejnowski, 1988]. Learned networks for each dataset are denoted seq-MNIST, Stock-Net, CO_2-Net, and Protein-Net. Table 7.1 summarizes the statistics for all five metrics with the network size of 128. The results represent the average cell response metric attributes for various datasets and demonstrate the global speed and amplitude of the activity of the network in terms of dynamical properties of the response characterization metrics.

Fig 7.3A-E, represents the distributions for the metrics sorted by the value of their specific attribute across all datasets. Cells in Protein-Net realized the fastest dynamics (i.e., smallest settling time) compared to the other networks while realizing a similar trend to the seq-MNIST (Fig. 7.3A). The settling time distribution for the LSTM units of CO_2 and Stock-Net depicts cell-groups with similar speed profiles. For instance, neurons 52 to 70 in Stock-Net, share the same settling time (Fig. 7.3A). Sine frequency stays constant for all networks except some outliers, which tend to modify their input-frequency (Fig. 7.3D). The delta response and the

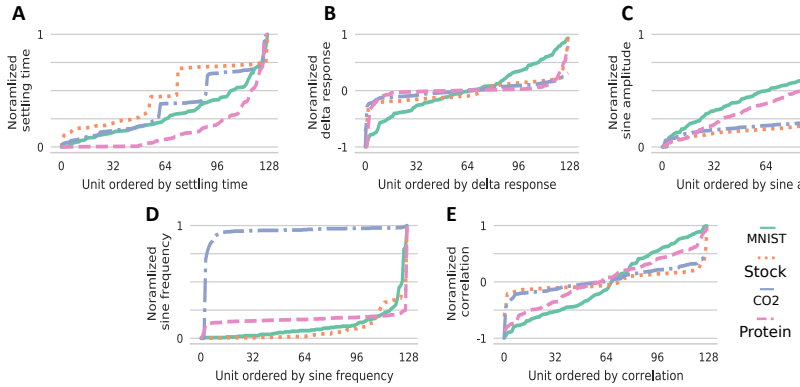


Figure 7.3: Cell level response distributions. (A-E) Response characterization metrics for networks with 128 individually ranked LSTM cells. The analyses predict A) cells with fast-dynamics and slow dynamic, (B and C) cells that are significantly contributing to the network decision, D) cells that realize phase shifting dynamics, and E) cells that are excitatory or inhibitory. The various line styles correspond to the different datasets used.

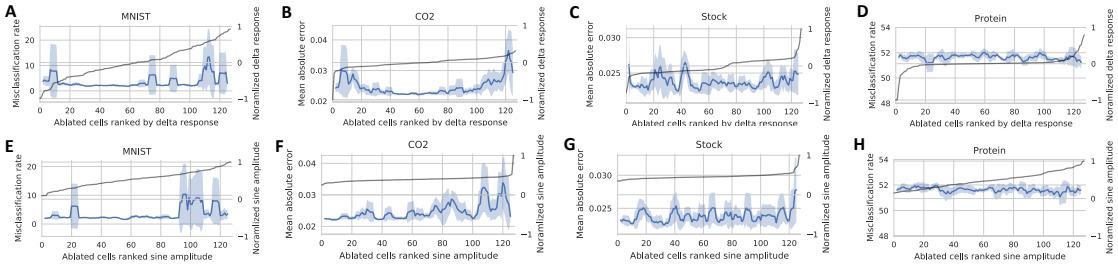


Figure 7.4: Cell level ablation analysis. Ablation of individual cells inside trained 128 cell LSTM networks across all four datasets (left to right). Changes in the predictive error are visualized against the ranked delta response (top) and sine amplitude (bottom) of the ablated cell. The Gray solid line represents the predictions of our method (right side vertical axis) as a function of the particular response metric. The solid blue line shows the mean, and the shadows represent the standard deviation of a moving average filter on the 23 ablated impact of individual neurons. This is done to highlight the trend of the ablation impact with respect to the sorted particular metric.

correlation metrics (Fig. 7.3B and Fig. 7.3E) both indicate the distribution of the inhibitory and excitatory behavior of individual cells within the network. Except for the Seq-MNIST net, neurons in all networks approximately keep a rate of 44% excitatory and 56% inhibitory dynamics. The high absolute amplitude neurons (at the two tails of Fig. 7.3C) are foreseen as the significant contributors to the output's decision. We validate this with an ablation analysis subsequently. Moreover, most neurons realize a low absolute delta-response value, for all datasets except for MNIST (Fig. 7.3B). This is an indication for cells with an equivalent influence on the output accuracy. Sine-amplitude stays invariant for most neurons in Stock and CO_2-Nets (Fig. 7.3C). For the seq-MNIST net and Protein-net, this distribution has a gradually increasing trend with weak

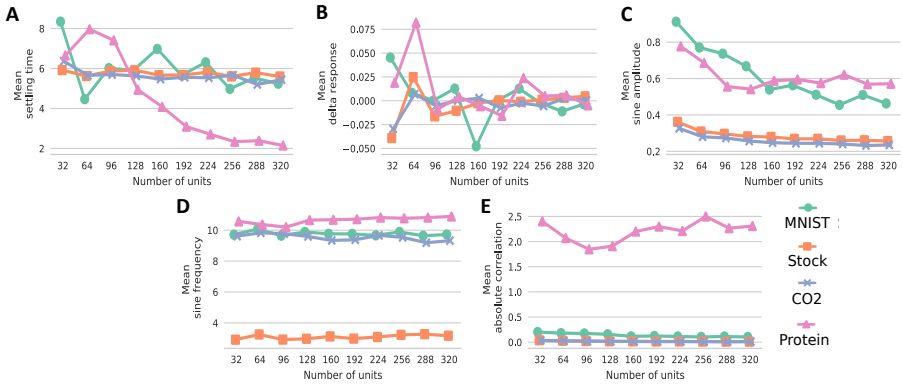


Figure 7.5: Network capacity analysis. (A-E) Response metrics as a function of the network's capacity. The analyses illustrate how response metrics provide insights on the global network scale in terms of settling time (A), delta response (B), sine amplitude (C), sine frequency (D), and correlation (E).

values. This predicts that individual cells are globally equivalently, contributing to the output.

7.5.3 Prediction of cell importance using response metrics

To assess the quality of the predictions and interpretations of the provided response characterization metrics, we performed individual cell-ablation analysis on LSTM networks and evaluated the cell-impact on the output accuracy (misclassification rate), for the classification problems and on the output performance (mean absolute error), for the regression problems. We knocked out neurons from trained LSTM networks with 128 neurons. Fig. 7.4A-H illustrate the performance of the network for individual cell ablations for all four datasets. The solid gray line in each subplot denotes for the predictions of the response metrics. For CO_2-Net, this confirms that neurons with higher sine amplitude tend to disrupt the output more (Fig 7.4D). For the same network, the delta response predicted that neurons with a high negative or positive value, are more significant in output's prediction. This is clearly illustrated in Fig. 7.4C. For seq-MNIST-Net, the same conclusions held true; neurons with a high absolute value of delta response or sine-amplitude reduce the accuracy at the output dramatically (Fig. 7.4A-B). By analyzing the sine-amplitude and delta-response of Protein-Net, we observe that neurons are equivalently valued and tend to contribute equivalently to the output accuracy. This is verified in the ablation analysis, shown in Fig. 7.4G and 7.4H, where the mean-misclassification error rate stays constant for all neural ablations. The absolute value for Stock-Net was also weak in terms of these two metrics, though there were some outliers at the tails of their distribution that predicted dominant neurons. This is clearly notable when comparing the neurons 120 to 128 of Fig. 7.4F to their prediction (gray line) where the amplitude of the response is maximal. In Fig. 7.4E ablation experiments for neurons 1 to 40 and 100 to 128 impose a higher impact on the overall output. This was also observed in the delta response prediction shown in 7.4B, since neurons with stronger output response was present at the two tails of the distribution.

7.5.4 Network-level Interpretability for Trained LSTMs

While we analyzed the response characterization distributions on a cellular level above, in this subsection, we focus on the effect of network capacity on observed hidden dynamics of the system on a global scale. Reasoning on this scale allows us to draw conclusions on how increasing the expressive capacity of LSTM networks trained on the same dataset can result in vastly different learned dynamics.

We experimentally vary the capacity by simply adding hidden LSTM cells to our network and retraining on the respective dataset from scratch. The relationship between each response characteristic metric and the network capacity is visualized in Fig. 7.5A-E. The trends across datasets are visualized in a single subplot to compare respective trends. One especially interesting result of this analysis is the capacity relationship with response amplitude (cf. Fig. 7.5C). Here we can see that the amplitude response decays roughly proportionally to $\frac{1}{N}$, for all datasets, where N is the number of LSTM cells. In other words, we get the intuitive finding that as we increase the number of LSTM cells, the magnitude of each cell's relative contribution needed to make a prediction will subsequently decrease.

Another key finding of this analysis is that the distribution of settling time is relatively constant across network capacity (cf. Fig. 7.5A). Intuitively, this means that the network is able to learn the underlying time delay constants represented in the dataset irrespective of the network capacity. One particularly interesting point comes for Protein-Net which exhibits vastly different behavior for both settling time (Fig. 7.5A) and correlation (Fig. 7.5E) than the remainder of the datasets. Upon closer inspection, we found that Protein-Net was heavily overfitting with increased capacity. This can be seen as an explanation for the rapid decay in its settling time as the addition of LSTM cells would increase the specificity of particular cells and exhibit dynamical properties aligning with effectively memorizing pieces of the training set.

7.6 Conclusion

In this chapter, we proposed a method of response characterization for LSTM networks to predict cell-contributions to the overall decision of a learned network on both the cell and network-level resolution. We further verified and validated our predictions by performing an ablation analysis to identify cell's, which contributed heavily to the network's output decision with our simple response characterization method. The resulting method establishes a novel building block for interpreting LSTM networks. The LSTM network's dynamic-space is broad and cannot be fully captured by fundamental input sequences. However, our methodology demonstrates that practical sub-regions of dynamics are reachable by response metrics, which we use to build a systematic testbench for LSTM interpretability. We have open-sourced our algorithm to encourage other researchers to further explore the dynamics of LSTM cells and interpret the kinetics of their sequential models. In the future, we aim to extend our approach to even more data modalities and analyze the training phase of LSTMs to interpret the learning of the converged dynamics presented in this work.

CHAPTER 8

Designing Interpretable RNNs for modeling Analog Integrated Circuits

8.1 Motivation

© [A8-Chapter8, 2017] — One challenging issue in the pre-silicon verification process of recently produced analog integrated circuits (IC)s is the development of high-performance models for carrying out time-efficient simulations. Transistor-level fault simulations of a single analog IC can take up to one or two weeks to be completed. As a result, over the past years, several attempts to develop fast behavioral models of the analog ICs have been investigated. Examples include SystemC, Verilog HDL, Verilog AMS, and Verilog-A models, which in principle, can realize very accurate models [Narayanan et al., 2008, Shokrolah-Shirazi and Miremadi, 2008, Pêcheux et al., 2005, Zhao and Cao, 2006]. However, the development of such models is not automated, and the associated human effort is considerable [Narayanan et al., 2008]. Moreover, this approach is unlikely to scale up to large libraries of existing analog components. Another example is the real number of modeling (RNM). In this method, analog parts of a mixed-signal IC are functionally modeled by real values, and they are used in the top-level system on chip verification [Balasubramanian and Hardee, 2013]. RNMs are fast and cover a large range of circuits. However, for analog circuits, including continuous-time feedbacks or detailed RC filter effects, it is not recommended [Balasubramanian and Hardee, 2013]. Moreover, RNM is not appropriate to be employed for circuits that are sensitive to nonlinear input-output (I/O) impedance interaction.

Here we propose an alternative machine-learning approach for automatically deriving neural network (NN) abstractions of integrated circuits, up to a prescribed tolerance of the behavioral features. NN modeling of the electronic circuits has been used in electromagnetic compatibility (EMC) testing, where the authors modeled a band-gap reference circuit (BGR) by utilizing an echo-state neural network [Magerl et al., 2015]. The developed NN model has shown a reasonable time performance in transient simulations; however, since the model is coded in Verilog-A, simulation speed-up is limited. In [Hasani et al., 2016], authors used a nonlinear

autoregressive neural network with exogenous input (NARX) for modeling the power-up behavior of a BGR. They demonstrated attractive improvements in the time performance of the transient simulations of the analog circuit within the Cadence AMS simulator by using this NARX model.

In this chapter, we take a compositional approach for learning the Overall time-domain behavior of a complex multiple-input multiple-output (MIMO) system, CompNN. CompNN learns in a first step, for each input i and each output j a small-sized nonlinear auto-regressive NNs with exogenous inputs (NARX) representing the transfer-function f_{iJ} from i to j. The learning dataset for h_{ij} is generated by varying only input i of the MIMO system and keeping all the other inputs constant. In a second step, for each output j, the transfer functions h_{ij} learned in Step 1, one for each input i, are combined by a (possibly nonlinear) function f_j, which is learned by employing another NN layer. The training dataset, in this case, is generated by applying all the inputs at the same time to the MIMO system. Once we constructed f_j for each output j, the overall output function is obtained as $f = (f_1, \ldots, f_n)$. We evaluate our approach by modeling the main time-domain behavioral features of a CMOS band-gap voltage reference circuit. We initially extract such features from the BGR circuit by using our I/O decomposition method. Consequently, we define trimming, load jump and line jump as the main behavioral features of the circuit to be modeled. Individual small-sized NARX networks are designed and trained in order to model the BGR output responses. We recompose the trained models by stacking a second layer network in a time-delayed neural network (TDNN) structure. The second layer is then trained in order to reproduce the output of the BGR. Such implementation provides us with an observable model where one can define a one-to-one mapping from specific behavioral features of the system to certain parts of the model. Finally, we employ our neural network model in transient simulation of the BGR and evaluate its performance. This is done by utilizing a co-simulation approach between MATLAB and Cadence AMS Designer environment [Cadence, 2017]. We demonstrate that by using such a 2-layer neural network structure, we can achieve one order of magnitude speed-up in the transient simulations.

The rest of the paper is organized as follows. In section II, we introduce our compositional approach to developing neural network models and define the case study. In Section III, we describe the NARX neural network architecture and identify the optimal quantity of components to be used in the neural network for each behavioral response. In Section IV, we explain the training process performed on the network and explore the performance of the designed models. Subsequently, in Section V, we train the second layer with the aim of merging the behavioral models into a single block. Finally, in Section VI, we employ a co-simulation approach for simulating our MATLAB/Simulink neural network model into Cadence Design environment and illustrate the performance of the network.

8.2 CompNN for MIMO system modeling

Let $I = \{i_1, i_2, ..., i_n\}$ be the vector of the defined inputs to a MIMO system, and $H = \{h_1, h_2, ..., h_m\}$ be the vector of nonlinear transfer functions delivering the corresponding output for each input exclusively, each output of the system is then constructed as $O = f(O_1, O_2, ..., O_m)$ where f, depending on the device under test (DUT), can be a linear or nonlinear function. As a result, we

train small-sized neural networks for modeling each component of vector H, and subsequently, we estimate the function f for merging such components by a second layer NN. We call such a compositional approach CompNN. CompNN provides us with the ability of mapping particular parts of the neural network model to specific behavioral features of the DUT and therefore having an observable model.

We demonstrate the performance of our method by developing a NN behavioral model of an analog integrated circuit: CMOS band-gap voltage reference circuit (BGR). A BGR outputs constant voltages (in our case 1V and 0.49V) regardless of possible variations caused by temperature change, power supply and load properties. Figure 8.1A depicts a symbolic representation of our BGR. The circuit is constructed from 50 transistors. We define the inputs to the system to be the power supply (V_{DD}) and three digital trimming inputs. A load-profile can be applied to the output-pin of the circuit (1V-Out). We, therefore, consider the load-profile as an input signal as well. Thus, the circuit realizes a multi-input single-output (MISO) dynamic system, which is a particular case of a MIMO system.

Figure 8.1B shows the BGR behavioral representation where the circuit comprises several behavioral features such as:

Power-up –, which is the activation of the power supply with several slopes and voltage levels.

Trimming inputs –, which enables the circuit to generate eight different stable outputs between 0.9 and 1.1 on its 1V-output pin. There are three digital trimming inputs.

Load jump – demonstrates the variations that occurred on the output voltage when a current load is applied.

Line jump – models the response of the BGR when there is a line jump on the power supply of the circuit.

Features are consequently recomposed by the function f and create the output of the circuit.

In [Hasani et al., 2016], authors employed a NARX NN for modeling the power-up behavior of the BGR. In this chapter, we model the rest of the decomposed features and thus complete the behavioral modeling of the circuit. We merge the behavioral features by approximating the function f using a second layer, time-delayed neural network.

8.3 Narx Neural-network architecture

Although the transfer function of a BGR is in principle constant, this is in practice highly nonlinear. As a consequence, modeling of the time-domain features requires powerful nonlinear system identification techniques and solutions. A nonlinear autoregressive neural network with exogenous input (NARX NN) appears to be a suitable framework for deriving approximations, up to a prescribed, maximum error, of the BGR. It has been previously demonstrated that a recurrent nature of the NARX NN topology consisting of only seven neurons and three three-time input-and-output delay components is able to precisely reproduce the turn-on behavior of the circuit [Hasani et al., 2016].

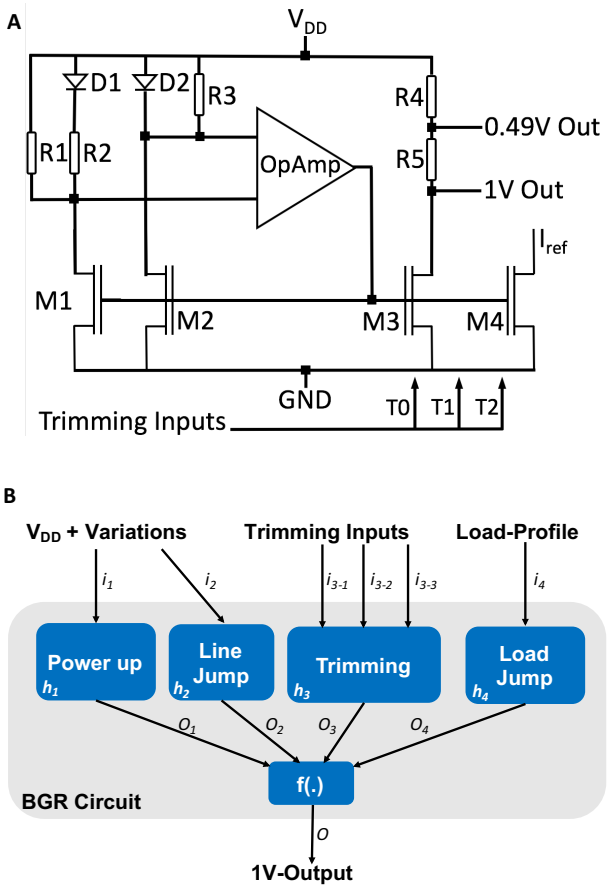


Figure 8.1: CMOS band-gap voltage reference circuit (BGR) A) Symbolic representation of the circuit schematic. B) Behavioral representation of the circuit.

In this chapter, we use the NARX architecture for modeling, in addition, the trimming, load jump and line jump behaviors of the BGR. The output of the network is constructed from the time-delayed components of the input signal $X(t)$ and output signal $Y(t)$, (see, for example, [Siegelmann et al., 1997]):

$$\begin{aligned} Y(t) = f(&X(t-1), X(t-2), ..., X(t-n_x), \\ &Y(t-1), Y(t-2), ..., Y(t-n_y)). \end{aligned} \tag{8.1}$$

The n_x and the n_y factors, define the input and output delays, that is, the number of discrete time steps within the input and the output histories that the component has to remember, in order to properly predict the next value of the output[Billings, 2013]. $n = n_x + n_y$ is the number of input nodes.

The size of the hidden layer is highly dependent on the number of input nodes. There are several ad-hoc approaches for defining the appropriate number of hidden neurons. For instance, one of the popular methods prescribes that the number of neurons within the hidden layer should be

Table 8.1: NARX network architecture for each of the BGR behavioral features

Features	♯ of delay components	♯ of hidden neurons
Trimming	3	10
Load Jump	3	7
Line Jump	3	7

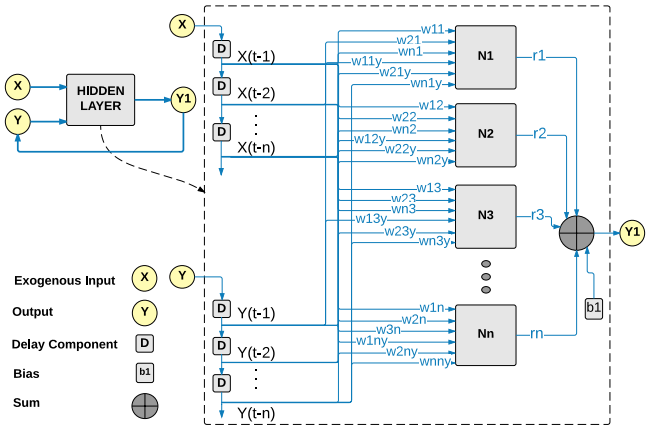


Figure 8.2: NARX neural network architecture. Note that the network realizes a recurrent topology where the output is fed-back into the input layer and causes further refinements on the predicted output signal $Y1$.

Table 8.2: Transient simulations performed for the training data collection purposes

Simulation	Simulation Time	CPU time	Input	Output	♯ of samples
Trimming	100 μs	1.4 s	Trimming inputs	$Vout_{1V}$	695
Load Jump	540 μs	1.3 s	Load Profile	$Vout_{1V}$	433
Line Jump	200 μs	1 s	V_{DD}	$Vout_{1V}$	501

between the number of input nodes [Heaton, 2008] and output nodes. We perform a grid search for choosing the optimal number of the delay components and hidden layer neurons [Hsu et al., 2003]. A hyperparameter space (d,h), consists of two parameters representing the quantity of delay components d, and the number of hidden-layer neurons h. Parameter d is chosen from a set $D = \{1,2,...,7\}$ and h from the set $H = \{1,2,...,15\}$. The Levenberg-Marquardt back propagation is performing a parameter optimization where the error of the validation dataset for each architecture pair (d,h), is calculated in the course of the training process. We ultimately select the architecture pair which results in the least validation error. Table 8.1 depicts the optimal number of delay components and hidden neurons chosen for realization of individual BGR features. As the output layer is a regressor, it comprises only one node.

The NARX architecture, therefore, is designed for each behavioral task, as shown in Figure 8.2. In this architecture, weighted input components synapse into the hidden layer nodes with

an all-to-all connection topology. We evaluated different activation functions such as (Elliot, logistic sigmoid, and tanh) and achieved the best performance by using a hyperbolic-tangent activation-function:

$$H = tanh(\sum_{i,j=1}^{N}(w_{ij}X_i) + b_j), \tag{8.2}$$

where H is the output of the hidden layer, w_{ij} represents the synaptic weight of the input X_i, from input node i to hidden node j and b_j depicts the bias weight applied to the hidden neuron j. The output of the NARX network is constructed as a linear sum of the weighted Hidden layer outputs. The network is designed in MATLAB[Demuth et al., 2015].

8.4 Training process and network performance

In order to collect adequate training datasets for teaching the NARX networks for the three behavioral features of the BGR that is, trimming, load jump, and line jump, we perform three transient simulations on the BGR by using the AMS simulator within the Cadence environment. Table 8.2 shows the details of the performed simulations and the collected datasets.

We aim to train a specific NARX network for each of the behavioral features where we use the input data as the exogenous input to the neural network and the output data as the target values to be learned. In order to gain high precision in the training process, we use the network in a feedforward topology in which the input of this topology consists of the original inputs and outputs, plus all the delayed inputs and outputs, up to their maximum input and output delays, respectively [Hasani et al., 2016]. A Levenberg-Marquardt (LM) back-propagation algorithm is employed for training each network [Marquardt, 1963]. The LM learning method, which is a modified version of the Gauss-Newton training algorithm, results in fast convergence of the gradient to its minimum since it is unaccompanied by calculation of the Hessian matrix. We initially define a cost function as follows:

$$E(w,b) = \frac{1}{2}\sum_{k\in K}(f(w,b)_k - t_k)^2, \tag{8.3}$$

where $E(w,b)$ stands for the error rate as a function of the weight w, and bias values b, $f(w,b)_k$ is the output generated by the neural network and t_k is the target outputs. We then try to minimize the error function for each training iteration with respect to the synaptic weights. $\triangle w$ which is calculated by the LM method and it is given by:

$$\triangle w = [J^T(w)J(w) + \eta I]^{-1}J^T(w)(f(w) - t), \tag{8.4}$$

Accordingly, the updated value of the weights is computed as:

$$w_{new} = w + \triangle w. \tag{8.5}$$

where $J(w)$ is the Jacobian matrix comprising the first-order derivatives of the error function with respect to weight values. Parameter η is the key to the fast convergence [Hagan and

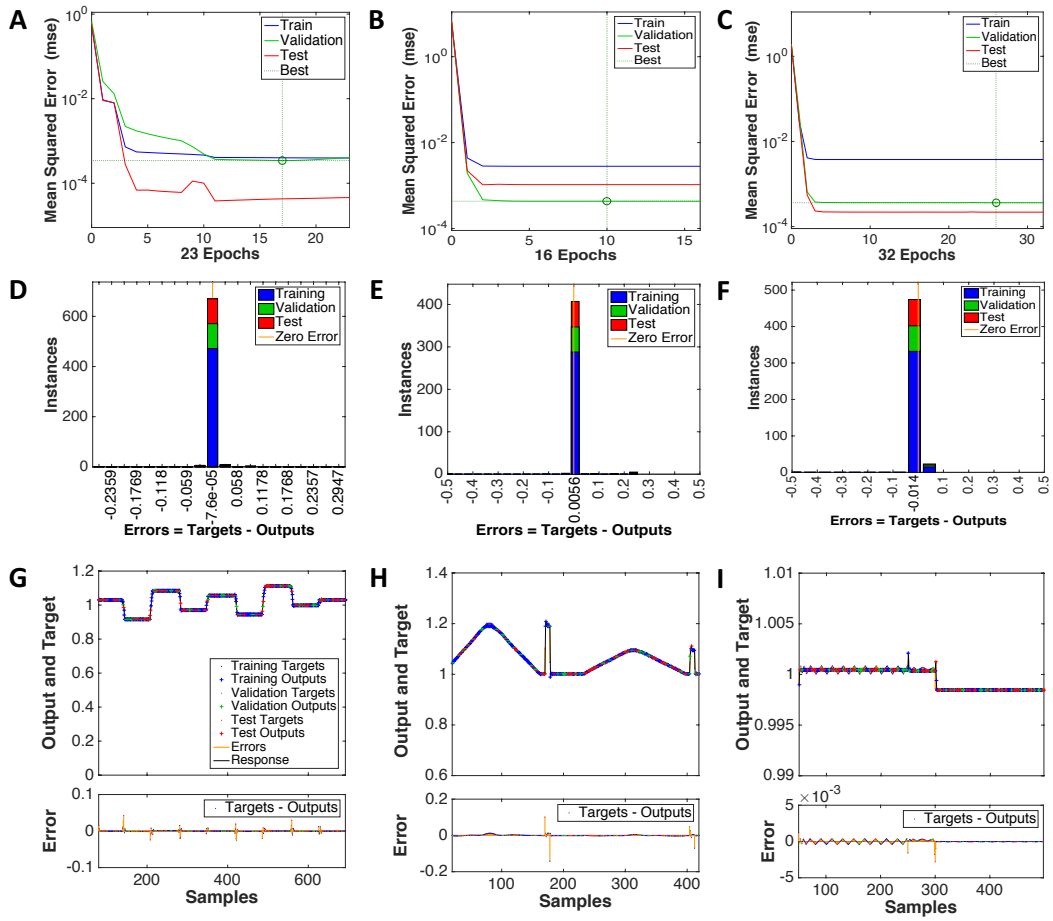


Figure 8.3: Network performance of the trimming, load jump, and line jump NARX behavioral models. A, B and C display the performance of the NARX neural network model of trimming, load jump and line jump, respectively, throughout the training process. The MSE is reduced drastically by each training step. In all three cases, the process terminated as soon as the validation dataset error stopped descending after six consequent epochs. D, E, and F show the error histogram of training samples for the NARX model of trimming, load jump, and line jump behavior, respectively. Note that most of the instances' error are close to the zero error line for each case. G, H, and I represent the output of the band-gap circuit together with its neural network response for trimming, load jump, and line jump behaviors, respectively. They also show the generated output error per sample.

Menhaj, 1994]. When this parameter is zero, the LM method realizes the common Gauss-Newton algorithm. If η increases throughout the training process, it is multiplied by an $\eta_{increase}$ value. On the contrary, when a training step results in a decrease of the value of η, its value gets reduced by a $\eta_{decrease}$ value. As a result, the cost function moves in a fast way towards the error reduction within each training epoch. The parameters' initial values and descriptions employed within the LM training algorithm are summarized in the Table 8.3.

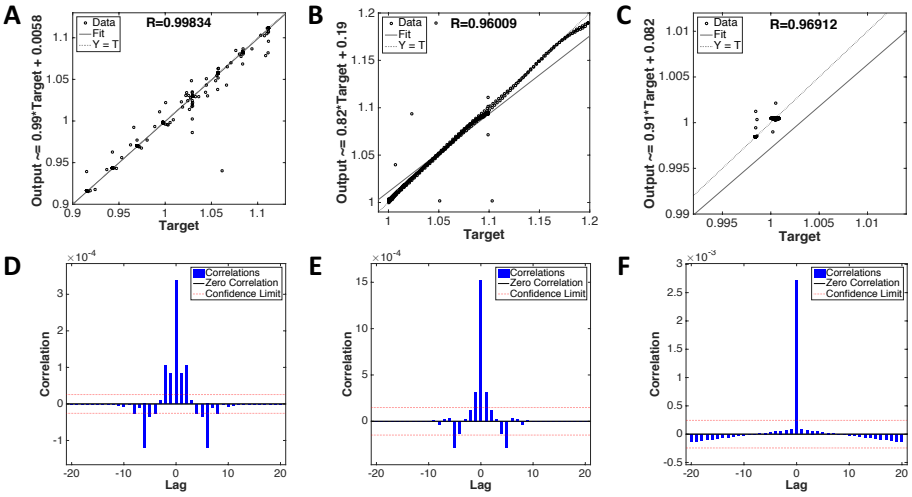


Figure 8.4: Linear regression and error auto-correlation function (ACF) representation of the NARX behavioral models. A, B, and C show the regression analysis, which is performed on the behavioral features, respectively, for the trimming, load jump, and the line jump. On the left-hand side axes of each regression plot, the fitting line function of the NARX output and the selected target values is computed. Note that R stands for the regression coefficient. D, E, and F demonstrate the error ACF calculated for our NARX models. Blue bars represent the correlation distribution of the lagged errors, and the red lines are the 95% confidence bounds (limit lines are located at an error correlation correspond to $\pm 2 \times$ *standard error* (SE)). For an ideal model, the error ACF will be a single bar at the lag zero while for a reliable model most of the lagged error components are located within the confidence boundaries.

Table 8.3: LM training algorithm parameters

Parameter Name	**Initial Value**	**Description**
max_epochs	1000	Maximum number of training iterations
ideal_error_value	0	Ideal error rate
max_refinement	6	Maximum validation error descending failure
min_cost_function	10^{-7}	Cost function minimum Value
eta	0.001	η initial value
eta_decrease	0.1	η decrease factor
eta_increase	10	η increase factor
max_eta	10^{10}	η Maximum η
max_time	inf	Maximum training time

For starting the training process, the collected samples are randomly divided into three data subsets consisting of:

- Training set (70%): This dataset is employed during the training process.
- Validation set (15%): This dataset is used for generalization and validation purposes. It

also plays a role in the termination of the training process.

- Test set (15%): This dataset provides an additional evaluation test after the training phase. It is not deployed during the learning process.

The training process terminates as soon as one of the conditions mentioned below occurs:

- No further refinement of the validation-dataset error-function is observed after $max\_refinement$ consequent training epochs.
- The cost function is minimized to $ideal\_error\_value$.
- η goes higher than max_η.
- The maximum number of training iterations is reached.
- The time of the training exceeds its maximum value.
- The Error function drops below $min\_cost\_function$.

Figure 8.3 illustrates the training performance of the three NARX networks, together with their corresponding error histogram. In all cases, the training process is concluded when no further reduction on the validation dataset error is noticed after six sequential training iteration. Moreover, it is observed that within trimming, load jump, and line jump, over 95% of the training samples have an average error of 7×10^{-5}, 5.6×10^{-3} and 1.4×10^{-3}, respectively. The time-series responses of the trimming, load jump, and line jump models, during the training process, are plotted in Figure 8.3 G, H and I, correspondingly. Note that the NARX networks precisely follow their target values.

In the case of the Trimming network, an input consisting of various trimming sets is applied as the training dataset network. The output varies around 1V whenever the trimming values toggle to a different configuration. Note that the 1V output of the BGR is modeled in this work. In the case of the load-jump network, two different current load profiles are separately applied to the 1V and 0.49V output of the BGR. Since the 0.49V output of the BGR is constructed from a resistor division on the 1V output pin, we expect to observe the voltage change caused by the load applied to the 0.49V output on the 1V pin. Therefore, as input to the load jump network, we take both load profiles into account. Finally, for the line jump, small variations on the power supply of the circuit are considered. In the ideal situation, we expect to see no change in the output. However, the output slightly varies, as it is shown in Figure 8.3I. The figure shows small fluctuations around 1V in the order of 10^{-3} due to a power supply variation of 10%. We notice that the line jump network imitates the behavior of the target values with decent accuracy.

Furthermore, linear regression is performed at the output layer of the neural network. The regression performance of the NARX network for each individual behavioral feature is shown in Figures 8.4A-C. The regression coefficients R are calculated to be close enough to $R = 1$, which

is the case of an ideal model. Moreover, the fitting-line function between the output of the NARX and target values are computed for each network.

In order to assess the efficiency of the network and the training process, we calculate the error auto-correlation function (ACF) in each case. The ACF explains how the output errors are correlated in time [Box et al., 2015]. Let the output error time-series, $e(t)$, be the difference between the generated output of the NARX network, $Y(t)$, and the target values, $T(t)$, $e(t) = Y(t) - T(t)$. The error correlation rate for the lag i, $\hat{\rho}_i$, is computed as follows:

$$\hat{\rho}_i = \frac{\sum_{t=i+1}^{T}(e_t - \hat{e})(e_{t-i} - \hat{e})}{\sum_{t=1}^{T}(e_t - \hat{e})^2}, \tag{8.6}$$

where T is the number of lags in time, which in our case is set to 20 and $\hat{e}$ stands for the average of the output error time-series. Ideally, the AFC comprises a single bar at the lag zero and the correlation rates of the other lagged-error components are zero. For a reliable model we set a 95% confidence limit equal to $\pm 2SE_\rho$, where SE is the standard error for checking the importance of the i^{th} lag for the autocorrelation, $\hat{\rho}_i$, and it is roughly calculated as follows:

$$SE_\rho = \sqrt{\frac{(1 + 2\sum_{j=1}^{i-j} \hat{\rho}_i{}^2)}{T}}. \tag{8.7}$$

Figures 8.4D-F show the error ACF plots for our trimming, load jump, and line jump networks, respectively. The horizontal red lines are the 95% confidence bounds. Note that in all cases, most of the error autocorrelation samples are within the confidence limits. This underlines the accuracy of the model.

Furthermore, in order to observe the behavior of the trained NARX models after the training process, we perform validation simulations by applying training datasets and datasets different from the training sets to the network. Figures 8.5A-F show the applied input profiles together with the time response of the networks, trimming, load jump, and the line jump. We observe that the neural network's output reasonably follows its target values in all cases. Based on the specification of our BGR, the acceptable error-rate at the 1V-output is 5%. Our neural network models generate a response in the case of different input datasets (Figures 8.5D-F), which satisfy such a condition.

Note that once the training process is terminated, the simulation time of the trained neural network is very fast. The CPU time recorded by MATLAB to perform our validation simulations is, on average, in the range of some milliseconds. Our learned models show improvements in the time performance by a factor of 17 when compared to their analog counterparts, during transient simulations. We experimentally verify such results in the following.

8.5 Recomposition function: A time-delayed neural network layer

In this section, we select a recomposition function f for combining behavioral models of the BGR, including the power-up behavior. By using the LM back-propagation algorithm, we train a time-delayed neural network (TDNN) comprised of three input delay elements and 200 hidden-layer neurons to be able to take the generated output of the four pre-trained NARX models and to

predict the correct 1V-output pin of the BGR. The structure is selected with the same approach as that of NARX models. Figure 8.6A represents the structure of the two-layer network. The network response to the training and test dataset is shown in Figure 8.6B and 8.6C, respectively. Matlab CPU time for executing the simulation of the network is approximately 50ms.

8.6 Co-simulation of Matlab/Simulink models and analog design environment

Here we utilize the Cadence AMS Designer/MATLAB co-simulation interface in order to evaluate the performance of the designed neural network model within the Analog Design Environment (ADE) of Cadence software, where we execute analog IC's fault simulations [Cadence, 2017]. Inside the co-simulation platform, a coupling module is provided in order to link Simulink and Cadence schematics environments. Figure 8.6A and 8.6B show the simulation environments in Simulink and Cadence schematics, respectively. We apply inputs to the neural network block in Simulink and simultaneously run a transient simulation in the Cadence ADE. Figure 8.6C and 8.6D depict the results of the co-simulation in case of training input dataset and test input dataset, correspondingly. The total CPU time for such transient simulations is calculated as 1.07s, while the same simulation of the transistor-level BGR takes 17.8s to be completed. As a result, we gain a simulation speed-up by a factor of 17.

8.7 Conclusions

We employed a new neural network modeling approach for complex MIMO systems (CompNN). We modeled individual I/O behavioral functions of the system by training NARX neural networks. We then merged the overall behavioral features by training a second layer TDNN. CompNN enabled us to define a one-to-one mapping from specific behavioral features of the system to certain parts of the model. We illustrated the performance of our modeling approach by designing behavioral NN models for a CMOS band-gap voltage reference circuit. Individual, small-sized NARX networks were designed and trained to imitate the trimming, load jump, and line jump responses of the BGR. Such pre-trained networks, together with the power-up behavior, were fed into a second time-delayed network in order to generate a single block representing the BGR.

The performance of the instructed networks was qualitatively and quantitatively analyzed by carrying out linear regression analysis, computing the error auto-correlation function, and calculating the error histogram for each model. We confirmed the level of generalization and the accuracy of such predictive neural networks by illustrating the output response of the models to various input patterns different from the training patterns. We subsequently created a single neural network block by adding the second layer for merging the behavioral features and training the network. Finally, we employed the designed network in a transient simulation and achieved sensible enhancement in the time performance of the simulation.

For future work, we intend to exploit our NARX models in the verification of analog integrated circuits, where the instantaneous response of the network, together with its high level of accuracy, results in significant improvements in the performance of the pre-silicon analog fault simulations.

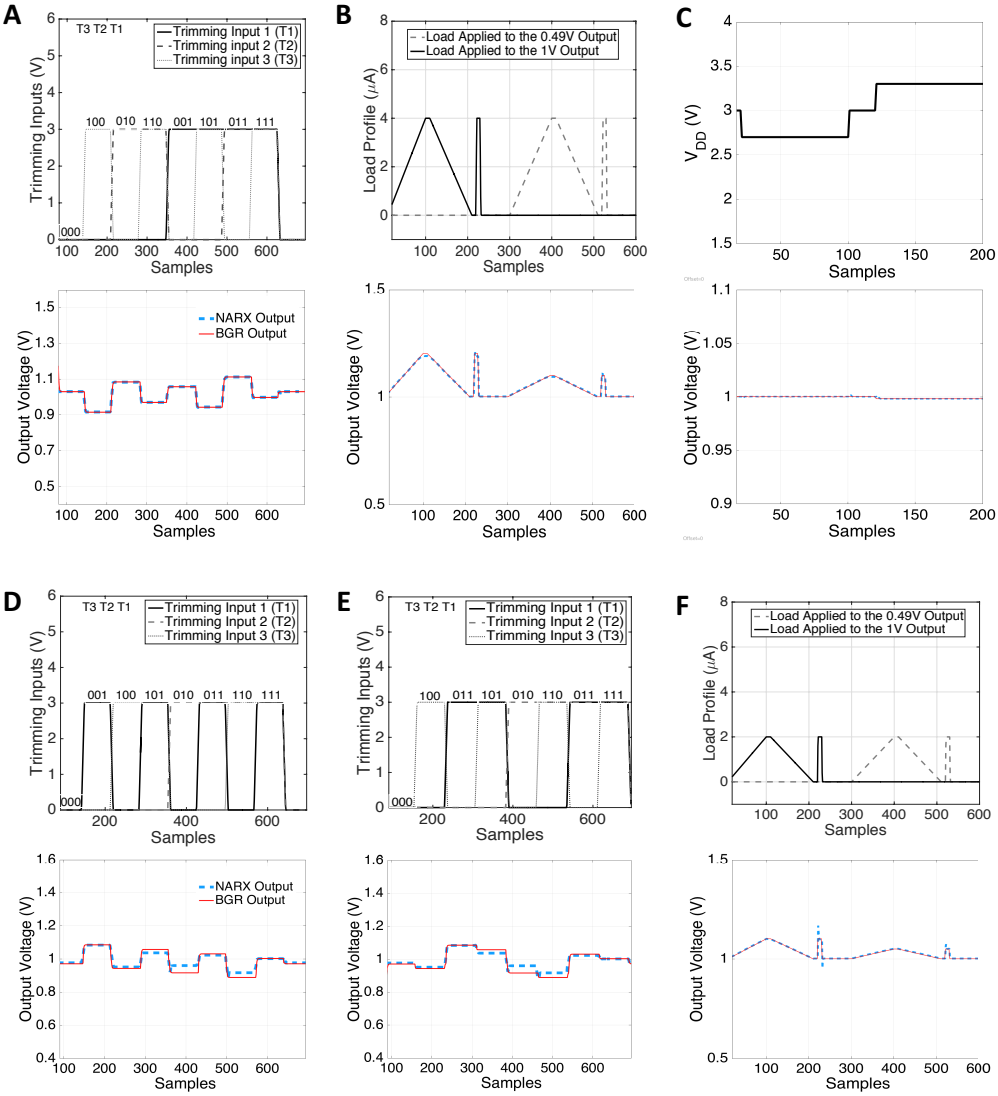


Figure 8.5: Time-response of the trained neural networks. A, B and C represent the input and output response of the NARX networks resembling trimming, load jump and line jump, after the training process where a simulink block of the network is generated. Training input data is applied to the network and its corresponding output is recorded. In B, we applied two load profiles, one to the 0.49V output and the other one to the 1V output. Since the 0.49V output of the BGR is created by using a resistor devision on the 1V output pin, at the output of the 1V pin we see the effect of the load connected to the 0.49V, as well. D and E depict two different input sets that are applied to the trained trimming neural network, in order to check the behavior of the NARX network in case of input patterns unalike the training input pattern. The same is checked for the load jump network in F. Note that the network generated a reasonable response in case of dissimilar input patterns in both cases.

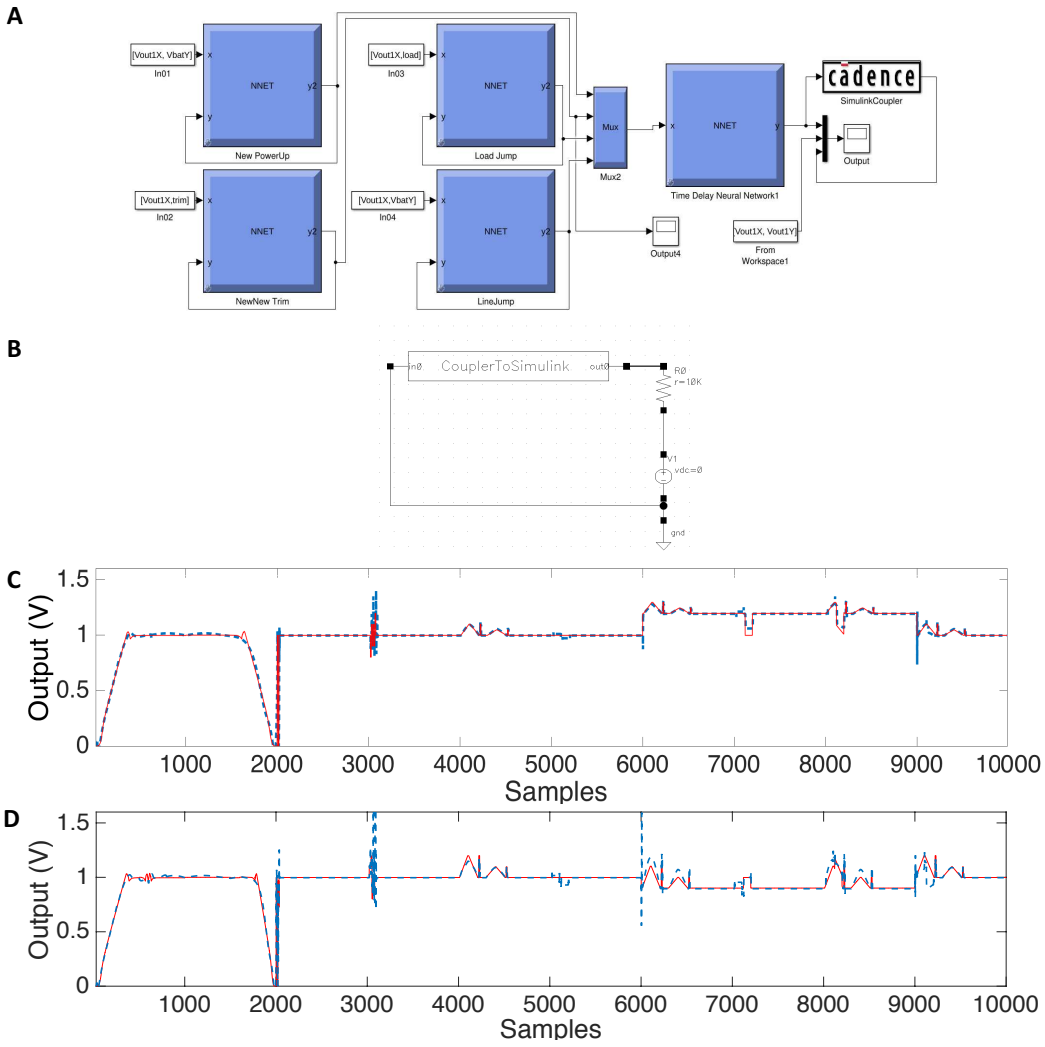


Figure 8.6: Two-layer neural network structure. A) Four NARX behavioral models are fed into the second layer network. B) Cadence schematic environment prepared to perform the co-simulation of Simulink model in Cadence AMS Designer C) Response of the BGR (solid red line) and its model (dashed blue line) to the training data. D) The response of the circuit and the model to the test pattern.

CHAPTER 9

Conclusions

The goal of this thesis was to delve into the properties of recurrent neural networks in continuous-time control spaces, to build more transparent RNN agents. In the following, we present our conclusions of the methods proposed in the thesis based on the order of chapters and pursue on discussing more formal critiques.

9.1 Summary Notes on Chapter 2

In Chapter 2, we motivated the advantages of using continuous-time (CT) neural network models and introduced a regularization method for learning state-stable CT-RNNs. In this regard, we determined a novel loss regularization method that, based on the Gershgorin circle theorem, forces the eigenvalues of the weight transition matrix of a linear CT-RNN to be strictly negative, thus ensuring its stability.

Note that this loss does not mathematically prove the stability of the closed control-loop system. We aimed to improve the stability of a deployed linear dynamical system (LDS) that is learned in an end-to-end fashion on fixed-length sequences. Rigorously proofing the stability of any closed control-loop system requires detailed knowledge and strong assumptions about the dynamics of the control environment. This prerequisite is antagonistic to the idea of end-to-end learning, which demands only minimal prior knowledge and assumptions.

An immediately related alternative to the Gershgorin regularization technique we proposed here, would be the use of the well-established Routh-Hurwitz stability criterion [Hurwitz, 1895] from control theory. This method suggests that an LDS is stable if all roots of the characteristic equation fall strictly on the right side of the s-plane. This is equivalent to the condition imposed by the Gershgorin loss on the weight matrix of an LDS to having negative eigenvalues.

Furthermore, we illustrated that our approach could be stacked with complex architectures such as multilayer perceptrons and convolutional neural networks. We showed that LDS could match and even surpass the generalization ability of the existing nonlinear RNN models. Studying the

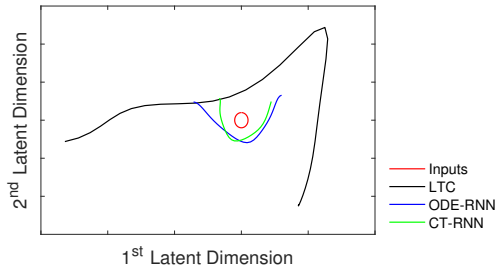


Figure 9.1: The latent representation of the Trajectory length of the output of a single layer LTC, ODE-RNN and CT-RNN exposed to a circular trajectory input.

stability properties of the resulting stacked nonlinear models and the stable-LDS is undoubtedly a possible research direction to take to develop end-to-end deep learning systems with safety guarantees.

9.2 Summary Notes on Chapter 3

In Chapter 3, we formulated a new brain-inspired instance of CT-RNNs, namely the liquid-time constant networks (LTC). We analyzed their properties and found activity bounds on their neural state dynamics. We illustrated how their dynamics are more expressive than standard CT-RNNs. We showed their superior performance compared to the other RNNs in time-series processing tasks. We experimentally demonstrated that LTCs possess compelling generalization capabilities, due to their biologically plausible dynamical representation, which resembles a dynamic causal model.

The universal approximation theory broadly explores the expressive power of a neural network model. A more rigorous measure of expressivity is demanding to compare models, specifically those networks specialized in spatiotemporal data processing, such as LTCs. Perhaps, the advances made on the expressive power of static deep learning models could be helpful to theoretically and quantitatively support our experimental evidence on the superiority of LTC models in terms of their expressivity to that of standard ODE-RNNs and CT-RNNs.

For instance, one can construct networks of the same size for LTCs, CT-RNNs, and ODE-RNNs, and compare the exponential growth in the number of linear regions of these models [Montufar et al., 2014]. Alternatively, another measure of expressivity, such as the trajectory length growth [Raghu et al., 2017], can be used to evaluate how a neural network model transforms a given input trajectory (e.g., a circular 2-dimensional input), by passing through its layers. Then, by measuring the length of the output trajectory, projected back in the input space, one can quantitatively define this method as a measure for the expressiveness of a given neural model.

More precisely, the trajectory length is defined as the *arc length* of a given trajectory $I(t)$, (e.g. a circle in 2D space), by the following equation [Raghu et al., 2017]:

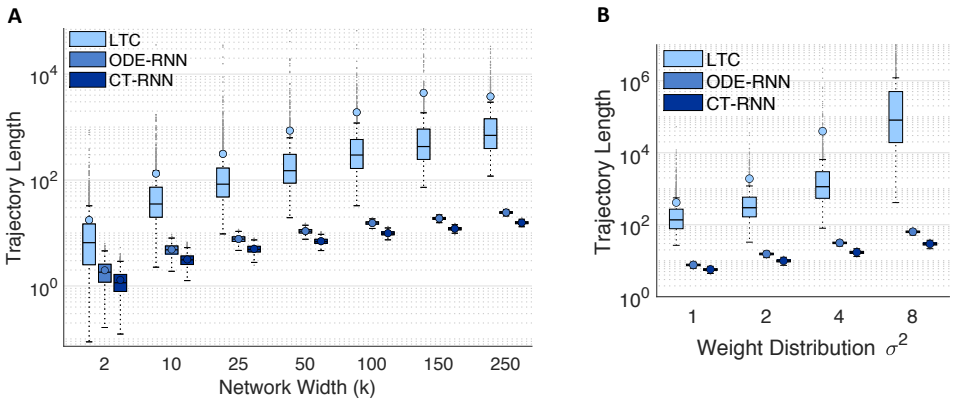


Figure 9.2: Trajectory length vs network size and weight distribution scaling factor. A) Trajectory length computed for 1000 networks of each class of models, for each network width size, randomly initialized. The exponential growth as a function of the network width is apparent. B) Trajectory length as a function of the weight distribution's variance.

$$l(I(t)) = \int_t \left\| \frac{dI(t)}{dt} \right\| dt.$$

To elaborate on this further, we have designed an experiment, to measure the trajectory length of a single hidden layer (layer width (N)=100 Neurons) LTC, CT-RNN and ODE-RNN. We initialized the weights by $\mathcal{N}(0, \sigma_w^2/N)$, and biases by $\mathcal{N}(0, \sigma_b^2)$. The weights corresponding to the similar computational compartments of each model set to be the same. We perform one forward-pass to these networks by exposing them to an input of the form $I(t) = \begin{bmatrix} I_1(t) = sin(t) \\ I_2(t) = cos(t) \end{bmatrix}$, for $t \in [0, 2\pi]$, which corresponds to a circular trajectory in 2D, shown by red in Fig. 9.1. Our preliminary observations suggest a much longer transformation of the length for the LTC model, when we projecting its 100-Dimensional output to a 2D latent space.

A simulation instance of a forward-pass of the network, with randomized weights, is shown in Fig. 9.1. We delved deeper into this to see how would a trajectory length grow for a continuous-time recurrent neural network by increasing its width (N). Fig. 9.2A represents an exponential growth in the trajectory length by increasing the network size and demonstrates a steady one to two orders of magnitude elevated growth for the LTC models compared to ODE-RNN and CT-RNN.

This is an interesting elementary observation as it suggests a fundamental difference between time-continuous models and static deep learning architectures in terms of the dependency of their trajectory length to their width. [Raghu et al., 2017] showed that the dependency of the network-width for a static deep model (by ReLU activation function) is only appearing in the base of the exponential bounds on the growth of the trajectory as: $\mathbb{E}\left[l(z^{(d)}(t))\right] \geq O\left(\frac{\sigma_w \sqrt{N}}{\sqrt{N+1}}\right)^d l(I(t))$, where $z^{(d)}(t))$ is the projection of the trajectory of layer d of the network in the input space, N is a layer's width, and σ_w is the standard deviation of the weights' associated distribution. We

observed, however, that a layer's size for LTC-like models contributes to the exponent of the trajectory length bound.

Moreover, the trajectory length bound proposed in [Raghu et al., 2017] for deep networks, demonstrates that the scale (σ_w^2) of the weight's distribution also contributes to the growth but linearly as a base. However, we witnessed a different pattern for time-continuous models in which the weights-scaling parameter, appears to contribute to the exponent, and in the case of LTCs, having even a faster impact $\propto e^{a\sqrt{a}}$, as shown in Fig. 9.2B. This paramilitary investigation confirms our experimental results, discussed in Chapter 3, by providing a quantitative measure for expressivity. Moreover, crafting such theoretical bounds for the continuous-time networks would be a great next step, and is part of our continued effort. Evidently, one can explore deeper into this; for instance, we can ask how the change of the solver would affect the trajectory length for models? What would be the outcome if we have more than a layer of ODEs as a model? What is the notion of depth in the analysis of more sophisticated solvers with adaptive time-step? To answer such questions, this thesis served as a base to start exploring the properties of time-continuous neural network models.

9.3 Summary Notes on Chapter 4 and 5

In Chapters 4 and 5, we designed networks of LTC neurons and explored their performance in many simulated and real-life robotic settings. The experiments proposed in these chapters are rather small, which is a direct result of the nature of the networks we approached. The field of connectome analysis is still in its infancy [Cook et al., 2019]. Up to a very recent discovery of the mapping of the nervous system of adult fruit fly [Xu et al., 2020], humans had only discovered the complete connectome of the adult *C. elegans* [White et al., 1986, Cook et al., 2019]. Moreover, the most published connectomes were that of larva stage animals. Besides the issue of obtaining a connectome, functional sub-circuits need to be identified by neuroscience researchers. Accordingly, we believe that at this current stage, performing RL with an entire *functionally ambiguous* connectome makes not much sense. Due to these two limitations, our experimental evaluation may appear limited in terms of network size.

Furthermore, instead of scaling our experiments to the state-of-the-art in terms of size, we aimed to diversify the nature of the tasks into standard RL tasks, Simulation to real-world applications, and higher dimensional action and observational spaces to the degree that a neural circuit naturally would allow.

As the field of connectome analysis is steadily growing, we are confident that our proposed approach emerges as a significant viewpoint casting on network-design paradigms in RL and Deep Learning, in more general application domains. Evidence of this analogy (though in a supervised learning setting) was later proposed in Chapter 6.

In the context of Chapter 4, we also discovered an intriguing property of small and sparse networks directly originated from nature, such as the Tap-withdrawal neural circuit of *C. elegans*. We hypothesized that the network could be a natural lottery ticket winner [Frankle and Carbin, 2018], as a network to be trained and deployed in control environments. It is worth noting that

a lottery ticket comprises a sub-circuit and its corresponding weight initialization. The TW circuit realizes such a sub-circuit with 77% sparsity level and governs many aspects of the weight initialization as well. For instance, in ordinary neural circuits (ONCs), the weight of a synapse is determined not only by its scalar strength value but also by its type and polarity (excitatory, inhibitory, or gap-junction). The type and polarity of all synapses were *initialized* to values observed in the neuroscience literature, thus stood as a base for our claim of having *lottery ticket winners*, provided by nature. This study offers a new perspective for many neurological-inspired future pieces of research for machine learning, especially network structure optimization.

9.4 Summary Notes on Chapter 6

We then scaled the application of LTC-based networks in Chapter 6, to enable high-dimensional information processing by a novel network design algorithm. LTC-based neural networks realize compact autonomous control agents, while being robust to input artifacts, deploy causality, and realize interpretable dynamics.

Black-box artificial intelligence algorithms are actively being used for sensitive and safety-critical applications throughout society, resulting in consequential challenges in autonomy, healthcare, and other domains. A large body of active research focuses on designing methodologies to explain the black-box, hoping for alleviating some of these limitations. These approaches, however, are likely to cause widespread confusion and bring about societal harms, such as fatal interactions of machines and humans [Rudin, 2019]. We believe that the key is to instead, design novel machine learning algorithms that possess interpretable skills inherently.

The work presented in Chapter 3 presented a novel methodology to design interpretable learning systems in high-stakes decision making, such as autonomous driving, while achieving superior performance compared to contemporary black-box machine learning models. To achieve this, we configured a combination of scalable deep learning topologies with significantly small-sized LTC networks to drive a car autonomously, with the following supremacy criteria. The control-network of the intelligent agent consists of only 19 neurons wired by 253 synapses. For the same task, the state-of-the-art black-box machine learning models are constructed from 1100 fully-connected neurons with 4.94M synapses. Besides, our NCPs demonstrated between $200\% - 1200\%$ more robustness to increasing input perturbations while driving, compared to state-of-the-art black-box learning models that crash (tend to steer the car off-road) often by the increase of the input noise. Nevertheless, we observed that our agents learned the actual causal structure of the task; they attended to a road horizon consistently, throughout live driving tests, even when their high-dimensional inputs are perturbed by noise, networks' attention showed significantly better robustness compared to other methods. More formally, the LTC compartment of an end-to-end driving network with convolutional heads enforced a causal prior during training, over the convolutional filters, so that their main attention be on the road's horizon while driving. This attribute became more sensible when we compare the attention maps of similarly designed end-to-end networks that were equipped with other RNN compartments. The reason for this phenomenon and the computational mechanism behind it is still an open question to be investigated.

The global network's activity and the individual cells' activity of our intelligent agents are

interpretable, meaning that we could identify the function of every control neuron. This process is practically infeasible for large-scale models, make their deployment in real-world decision-critical applications challenging.

Additionally, it is worth noting that the recent breakthrough lines of research in AI, which tackled the high-dimensional environments (e.g., Chess, Go, Atari, Starcraft II), took advantage of tremendously large-scale learning systems and algorithms [Mnih et al., 2015, Silver et al., 2016a, Silver et al., 2017, Silver et al., 2018, Vinyals et al., 2019, Dabney et al., 2020]. The real-world applications of equivalently large-scale environments, such as autonomous driving, inevitably require high degrees of explainability to ensure humans' safety. Achieving this property is hard if not practically infeasible for such large-scale networks. In this chapter, we identified outstandingly small neural networks that not only outperform the driving performance of large-scale black-box deep learning models but also come with superior explainability and robustness.

Real-world domains such as autonomous driving, robotics, health, and medicine, are surrounded by environmental uncertainties, and demand for robust real-time decision making. Moreover, similar to autonomous robot control, many applications deal with complex high-dimensional input-output spaces that, when deployed in the real world, become safety-critical. The success of LTC-based networks in autonomous control indicates that tackling the complexity of real-world problems does not necessarily require complex deep neural network architectures.

9.5 Summary Notes on Chapter 7 and 8

In Chapter 7, We further investigated methods for interpreting modern RNN architectures. We developed a novel dynamical system's approach to reason about the internal dynamics of a neural network. We showed the generalizability and scalability in a sequel of sequential data processing applications. Finally, in Chapter 8, we demonstrated a real-life application of recurrent networks and developed a compositional method to design them, to enhance interpretability while conducting effectively integrated circuit modeling.

9.6 Future Directions

The findings of this dissertation open up prospective research opportunities. The study of more detailed biophysical neural computation principles may lead to the development of better AI systems. Several attempts have been initiated in this direction; an example is the OpenWorm project [Sarma et al., 2018], in which we developed computational tools to investigate how the nervous system of the nematode *C. elegans* gives rise to its behavior.

Moreover, we observed that sparse and purposely designed networks of LTC neurons (similar to that of nervous systems) result in achieving better performances rather than densely wired networks. We have empirically quantified this phenomenon by the notion of the maximum flow rate. However, the fundamental reason for this phenomenon is yet to be studied.

Extrapolation with continuous-time neural network models. An attractive yet unsolved research question in the field of deep learning is how to encode extrapolation capabilities inside a neural network? Our observations interestingly indicated that LTC-based networks could perform extrapolation beyond the range over which they have been trained on. The foundation of the existence of such skills has to be thoroughly investigated.

Closed-form solutions for Neural ODEs. The time-continuous models introduced in this paper were all simulated and solved by numerical approximators (solvers). An attractive research question would be whether a closed-form solution for such neural ODEs is analytically achievable. A potential closed-form solution would arguably enhance the algorithmic complexity of neural ODEs.

Perceptron++ The LTC RNN model can inspire novel neural network architectures with nonlinear synaptic propagation schemes as well. For instance, a perceptron model could be updated to the following format:

$$y(t) = G(wf(x) + b(x)),$$

where G is a sigmoidal function, $f(x)$ and $b(x)$ are arbitrary-chosen continuous functions, to be investigated. Very recent evidence of existence of functional properties at the dendrites of neurons [Gidon et al., 2020], supports the idea of Perceptron++, to think about building richer neural representations.

List of Figures

1.1 Thesis topics and contributions map. 5

2.1 The feedforward neural network. 13
2.2 Standard RNN Architecture 14
2.3 Time-delayed neural network structure 17
2.4 NARX network architecture 17
2.5 Long short-term memory (LSTM) cell structure 18

3.1 Schematic view of a membrane neural model 32
3.2 CT-RNN vs LTC architecture 33
3.3 How a liquid coupling sensitivity results in a more expressive dynamics? 35
3.4 Compare LTC and CT-RNN 36
3.5 LTC vs DCM 38
3.6 LTC cell and Network with Hybrid solver 48
3.7 ODE to recurrent cell 49
3.8 Frequency extrapolation experiment 55

4.1 Ordinary Neural Circuits 59
4.2 Sparse and randomly wired network samples 60
4.3 Mapping the environments to the TW circuit 65
4.4 Learning curves for the TW circuit in standard RL tasks 67
4.5 Interpretability analysis of the parking task 70

5.1 Design Operator Networks 72
5.2 Design Operators 75
5.3 Npark 77
5.4 Manipulating a robotic arm 80

6.1 Recurrent neural networks' essence for lane-keeping tasks 86
6.2 Designing NCP networks with LTC neural model 89
6.3 Learning curves of the models tested in the active driving experiments 95
6.4 Robustness analysis 99
6.5 Global network dynamics 100
6.6 Intuitive comprehension of NCP's cells activity 101

6.7 Neural activity of all NCP neurons presented in Fig. 6.6 102
6.8 Coupling sensitivity of all NCP neurons presented in Fig. 6.6 102

7.1 Response characterization method for LSTM cells 109
7.2 Cell-level interpretation of sequential MNIST 111
7.3 Cell level response distributions 113
7.4 Cell level ablation analysis 113
7.5 Network capacity analysis 114

8.1 CMOS band-gap voltage reference circuit (BGR) 120
8.2 NARX neural network architecture 121
8.3 Network performance 123
8.4 Linear regression and ACF representation of the NARXs 124
8.5 Time-response of the trained neural networks 128
8.6 Two-layer neural network structure 129

9.1 Network's trajectory representation 132
9.2 Trajectory length vs network size and weight distribution scaling factor 133

List of Tables

3.1 Full list of parameter setting in the experiments of Fig. 3.3 and Fig. 3.4 37
3.2 Test performance of RNN models 51

4.1 Ratio of the average maximum flow rate of networks of Fig. 4.2 62
4.2 ONC versus random circuits 67
4.3 Comparison of ONC to DNNs with policy gradient algorithms 68
4.4 Compare ONC with deep learning models 68

5.1 Comparing parking performance 81

6.1 Results of the passive lane-keeping 10-fold cross-testing evaluation 93
6.2 Models' training hyperparameters 94
6.3 Layers of the feedforward CNN, adapted from [Bojarski et al., 2016] 94
6.4 Convolutional head 95
6.5 The learning termination shown in Fig. 6.3 96
6.6 Network size comparison 96
6.7 Videos of the driving performance of all algorithms 103

7.1 Hidden dynamic distributions by dataset 111

8.1 NARX network architecture for each of the BGR behavioral features 121
8.2 Transient simulations performed for the training data collection purposes 121
8.3 LM training algorithm parameters 124

Bibliography

[A5-Chapter5, 2019] A5-Chapter5 (2019). 2019 IEEE. Reprinted, with permission, from lechner, mathias and hasani, ramin and zimmer, manuel and henzinger, thomas a and grosu, radu, designing worm-inspired neural networks for interpretable robotic control. In *International Conference on Robotics and Automation (ICRA)*. IEEE.

[A7-Chapter7, 2019] A7-Chapter7 (2019). 2019 IEEE. Reprinted, with permission, from hasani, ramin and amini, alexander and lechner, mathias and naser, felix and grosu, radu and rus, daniela, response characterization for auditing cell dynamics in long short-term memory networks. In *International Joint Conference on Neural Networks (IJCNN)*. IEEE.

[A8-Chapter8, 2017] A8-Chapter8 (2017). 2017 IEEE. Reprinted, with permission, from hasani, ramin and haerle, dieter and baumgartner, christian and lomuscio, alessio and grosu, radu, compositional neural-network modeling of complex analog circuits. In *International Joint Conference on Neural Networks (IJCNN)*. IEEE.

[Abadi et al., 2016] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. (2016). Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283.

[Albertini and Dai Pra, 1995] Albertini, F. and Dai Pra, P. (1995). Recurrent neural networks: Identification and other system theoretic properties. *Neural Network Systems Techniques and Applications*, 3:1–41.

[Albertini and Sontag, 1993] Albertini, F. and Sontag, E. D. (1993). For neural networks, function determines form. *Neural Networks*, 6(7):975–990.

[Albertini and Sontag, 1994] Albertini, F. and Sontag, E. D. (1994). Uniqueness of weights for recurrent nets. *MATHEMATICAL RESEARCH*, 79:599–599.

[Alon, 2006] Alon, U. (2006). *An Introduction to Systems Biology: Design Principles of Biological Circuits*. Chapman & Hall/CRC Mathematical and Computational Biology. Taylor & Francis.

[Alon, 2007] Alon, U. (2007). Network motifs: theory and experimental approaches. *Nature Reviews Genetics*, 8(6):450–461.

[Amini et al., 2018a] Amini, A., Paull, L., Balch, T., Karaman, S., and Rus, D. (2018a). Learning steering bounds for parallel autonomous systems. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–8. IEEE.

[Amini et al., 2019] Amini, A., Rosman, G., Karaman, S., and Rus, D. (2019). Variational end-to-end navigation and localization. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8958–8964. IEEE.

[Amini et al., 2018b] Amini, A., Schwarting, W., Rosman, G., Araki, B., Karaman, S., and Rus, D. (2018b). Variational autoencoder for end-to-end control of autonomous driving with novelty detection and training de-biasing. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 568–575. IEEE.

[Anguita et al., 2013] Anguita, D., Ghio, A., Oneto, L., Parra, X., and Reyes-Ortiz, J. L. (2013). A public domain dataset for human activity recognition using smartphones. In *Esann*.

[Ardiel and Rankin, 2010] Ardiel, E. L. and Rankin, C. H. (2010). An elegant mind: learning and memory in *Caenorhabditis elegans*. *Learning & memory*, 17(4):191–201.

[Atkeson and Schaal, 1997] Atkeson, C. G. and Schaal, S. (1997). Robot learning from demonstration. In *ICML*, volume 97, pages 12–20. Citeseer.

[Balasubramanian and Hardee, 2013] Balasubramanian, S. and Hardee, P. (2013). Solutions for mixed-signal soc verification using real number models. *Cadence Design Systems*.

[Bargmann, 2006] Bargmann, C. I. (2006). Chemosensation in c. elegans. *WormBook*, pages 1–29.

[Barocas et al., 2017] Barocas, S., Hardt, M., and Narayanan, A. (2017). Fairness in machine learning. *NIPS Tutorial*.

[Beer, 1995] Beer, R. D. (1995). On the dynamics of small continuous-time recurrent neural networks. *Adaptive Behavior*, 3(4):469–509.

[Beer et al., 1992] Beer, R. D., Chiel, H. J., Quinn, R. D., Espenschied, K. S., and Larsson, P. (1992). A distributed neural network architecture for hexapod robot locomotion. *Neural Computation*, 4(3):356–365.

[Beetz et al., 2015] Beetz, M., Bálint-Benczédi, F., Blodow, N., Nyga, D., Wiedemeyer, T., and Márton, Z.-C. (2015). Robosherlock: Unstructured information processing for robot perception. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 1549–1556. IEEE.

[Bekey, 2005] Bekey, G. A. (2005). *Autonomous robots: from biological inspiration to implementation and control*. MIT press.

[Bengio et al., 2013] Bengio, Y., Courville, A., and Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828.

[Bengio and Grandvalet, 2004] Bengio, Y. and Grandvalet, Y. (2004). No unbiased estimator of the variance of k-fold cross-validation. *Journal of machine learning research*, 5(Sep):1089–1105.

[Bengio et al., 1994] Bengio, Y., Simard, P., Frasconi, P., et al. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166.

[Berkenkamp et al., 2017] Berkenkamp, F., Turchetta, M., Schoellig, A., and Krause, A. (2017). Safe model-based reinforcement learning with stability guarantees. In *Advances in Neural Information Processing Systems 30*, pages 908–919. Curran Associates, Inc.

[Biggs and MacDonald, 2003] Biggs, G. and MacDonald, B. (2003). A survey of robot programming systems. In *Proceedings of the Australasian conference on robotics and automation*, pages 1–3.

[Bilal et al., 2018] Bilal, A., Jourabloo, A., Ye, M., Liu, X., and Ren, L. (2018). Do convolutional neural networks learn class hierarchy? *IEEE transactions on visualization and computer graphics*, 24(1):152–162.

[Billings, 2013] Billings, S. A. (2013). *Nonlinear system identification: NARMAX methods in the time, frequency, and spatio-temporal domains*. John Wiley & Sons.

[Bishop and Tipping, 1998] Bishop, C. M. and Tipping, M. E. (1998). A hierarchical latent variable model for data visualization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(3):281–293.

[Blocher et al., 2017] Blocher, C., Saveriano, M., and Lee, D. (2017). Learning stable dynamical systems using contraction theory. In *2017 14th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*, pages 124–129. IEEE.

[Bohn and Atherton, 1995] Bohn, C. and Atherton, D. (1995). An analysis package comparing pid anti-windup strategies. *IEEE Control Systems Magazine*, 15(2):34–40.

[Bojarski et al., 2018] Bojarski, M., Choromanska, A., Choromanski, K., Firner, B., Ackel, L. J., Muller, U., Yeres, P., and Zieba, K. (2018). Visualbackprop: Efficient visualization of cnns for autonomous driving. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–8. IEEE.

[Bojarski et al., 2016] Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L. D., Monfort, M., Muller, U., Zhang, J., et al. (2016). End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*.

[Box et al., 2015] Box, G. E., Jenkins, G. M., Reinsel, G. C., and Ljung, G. M. (2015). *Time series analysis: forecasting and control*. John Wiley & Sons.

[Boykov and Kolmogorov, 2004] Boykov, Y. and Kolmogorov, V. (2004). An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE transactions on pattern analysis and machine intelligence*, 26(9):1124–1137.

[Brabazon et al., 2015] Brabazon, A., O'Neill, M., and McGarraghy, S. (2015). *Natural Computing Algorithms*. Springer Publishing Company, Incorporated, 1st edition.

[Brooks, 1986] Brooks, R. (1986). A robust layered control system for a mobile robot. *IEEE journal on robotics and automation*, 2(1):14–23.

[Cadence, 2017] Cadence (accessed in 2017). Cadence Virtuoso AMS Designer Simulator, cosimulation of mixed-signal systems with matlab and simulink. *Cadence*.

[Calinon et al., 2007] Calinon, S., Guenter, F., and Billard, A. (2007). On learning, representing, and generalizing a task in a humanoid robot. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 37(2):286–298.

[Camacho and Alba, 2013] Camacho, E. F. and Alba, C. B. (2013). *Model predictive control*. Springer Science & Business Media.

[Candanedo and Feldheim, 2016] Candanedo, L. M. and Feldheim, V. (2016). Accurate occupancy detection of an office room from light, temperature, humidity and co2 measurements using statistical learning models. *Energy and Buildings*, 112:28–39.

[Capuozzo and Livingston, 2011] Capuozzo, M. D. and Livingston, D. L. (2011). A compact evolutionary algorithm for integer spiking neural network robot controllers. In *2011 Proceedings of IEEE Southeastcon*, pages 237–242.

[Cardoso et al., 2017] Cardoso, V., Oliveira, J., Teixeira, T., Badue, C., Mutz, F., Oliveira-Santos, T., Veronese, L., and De Souza, A. F. (2017). A model-predictive motion planner for the iara autonomous car. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 225–230. IEEE.

[Chalfie et al., 1985a] Chalfie, M., Sulston, J., White, J., Southgate, E., Thomson, J., and Brenner, S. (1985a). The neural circuit for touch sensitivity in *Caenorhabditis elegans*. *Journal of Neuroscience*, 5(4):956–964.

[Chalfie et al., 1985b] Chalfie, M., Sulston, J. E., White, J. G., Southgate, E., Thomson, J. N., and Brenner, S. (1985b). The neural circuit for touch sensitivity in caenorhabditis elegans. *Journal of Neuroscience*, 5(4):956–964.

[Chen et al., 2006] Chen, B. L., Hall, D. H., and Chklovskii, D. B. (2006). Wiring optimization can relate neuronal structure and function. *Proceedings of the National Academy of Sciences of the United States of America*, 103(12):4723–4728.

[Chen et al., 2018] Chen, T. Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D. K. (2018). Neural ordinary differential equations. In *Advances in neural information processing systems*, pages 6571–6583.

[Chen et al., 2016] Chen, X., Duan, Y., Houthooft, R., Schulman, J., Sutskever, I., and Abbeel, P. (2016). Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2172–2180.

[Chua and Yang, 1988] Chua, L. O. and Yang, L. (1988). Cellular neural networks: Theory. *IEEE Transactions on circuits and systems*, 35(10):1257–1272.

[Chung et al., 2014] Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

[Cook et al., 2019] Cook, S. J., Jarrell, T. A., Brittin, C. A., Wang, Y., Bloniarz, A. E., Yakovlev, M. A., Nguyen, K. C., Tang, L. T.-H., Bayer, E. A., Duerr, J. S., et al. (2019). Whole-animal connectomes of both caenorhabditis elegans sexes. *Nature*, 571(7763):63–71.

[Corporation, 2015] Corporation, R. (2015). *Cyton Epsilon 300 Arm Specifications*. Robai Corporation.

[Cybenko, 1989] Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314.

[Dabney et al., 2020] Dabney, W., Kurth-Nelson, Z., Uchida, N., Starkweather, C. K., Hassabis, D., Munos, R., and Botvinick, M. (2020). A distributional code for value in dopamine-based reinforcement learning. *Nature*, pages 1–5.

[Dantam et al., 2016] Dantam, N. T., Kingston, Z. K., Chaudhuri, S., and Kavraki, L. E. (2016). Incremental task and motion planning: A constraint-based approach. In *Robotics: Science and systems*, volume 12, page 00052. Ann Arbor, MI, USA.

[Davis and Stretton, 1989] Davis, R. E. and Stretton, A. (1989). Signaling properties of ascaris motorneurons: graded active responses, graded synaptic transmission, and tonic transmitter release. *Journal of Neuroscience*, 9(2):415–425.

[Demuth et al., 2015] Demuth, H., Beale, M., and Hagan, M. (2015). Neural network toolbox. *User guide*.

[Deuflhard et al., 1987] Deuflhard, P., Hairer, E., and Zugck, J. (1987). One-step and extrapolation methods for differential-algebraic systems. *Numerische Mathematik*, 51(5):501–516.

[Dong et al., 2017] Dong, Y., Su, H., Zhu, J., and Zhang, B. (2017). Improving interpretability of deep neural networks with semantic information. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4306–4314.

[Doshi-Velez and Kim, 2017] Doshi-Velez, F. and Kim, B. (2017). Towards a rigorous science of interpretable machine learning. *arXiv preprint arXiv:1702.08608*.

[Dozat, 2016] Dozat, T. (2016). Incorporating nesterov momentum into adam. *Openreview*.

[Dua and Graff, 2017] Dua, D. and Graff, C. (2017). UCI machine learning repository.

[Duan et al., 2016] Duan, Y., Chen, X., Houthooft, R., Schulman, J., and Abbeel, P. (2016). Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*, pages 1329–1338.

[Duchi et al., 2011] Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159.

[Elfes, 1989] Elfes, A. (1989). Using occupancy grids for mobile robot perception and navigation. *Computer*, 22(6):46–57.

[Erhan et al., 2009] Erhan, D., Bengio, Y., Courville, A., and Vincent, P. (2009). Visualizing higher-layer features of a deep network. *University of Montreal*, 1341(3):1.

[Figueroa and Billard, 2018] Figueroa, N. and Billard, A. (2018). A physically-consistent bayesian non-parametric mixture model for dynamical system learning. In *Conference on Robot Learning*, pages 927–946.

[finance yahoo, 4 13] finance yahoo (Accessed: 2018-04-13). Yahoo Finance Website, S&P 500 Stock. `https://finance.yahoo.com/quote/%5EGSPC/`.

[Finn et al., 2017] Finn, C., Yu, T., Zhang, T., Abbeel, P., and Levine, S. (2017). One-shot visual imitation learning via meta-learning. *arXiv preprint arXiv:1709.04905*.

[Fish et al., 2016] Fish, B., Kun, J., and Lelkes, Á. D. (2016). A confidence-based approach for balancing fairness and accuracy. In *Proceedings of the 2016 SIAM International Conference on Data Mining*, pages 144–152. SIAM.

[Folgheraiter et al., 2006] Folgheraiter, M., Gini, G., Nava, A., and Mottola, N. (2006). A bioinspired neural controller for a mobile robot. In *2006 IEEE International Conference on Robotics and Biomimetics*, pages 1646–1651.

[Fong and Vedaldi, 2017] Fong, R. C. and Vedaldi, A. (2017). Interpretable explanations of black boxes by meaningful perturbation. *arXiv preprint arXiv:1704.03296*.

[Frankle and Carbin, 2018] Frankle, J. and Carbin, M. (2018). The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*.

[Fridman et al., 2019] Fridman, L., Brown, D. E., Glazer, M., Angell, W., Dodd, S., Jenik, B., Terwilliger, J., Patsekin, A., Kindelsberger, J., Ding, L., et al. (2019). Mit advanced vehicle technology study: Large-scale naturalistic driving study of driver behavior and interaction with automation. *IEEE Access*, 7:102021–102038.

[Friston et al., 2003] Friston, K. J., Harrison, L., and Penny, W. (2003). Dynamic causal modelling. *Neuroimage*, 19(4):1273–1302.

[Funahashi, 1989] Funahashi, K.-I. (1989). On the approximate realization of continuous mappings by neural networks. *Neural networks*, 2(3):183–192.

[Funahashi and Nakamura, 1993] Funahashi, K.-i. and Nakamura, Y. (1993). Approximation of dynamical systems by continuous time recurrent neural networks. *Neural networks*, 6(6):801–806.

[Garzon and Botelho, 1994] Garzon, M. and Botelho, F. (1994). Observability of neural network behavior. In *Advances in Neural Information Processing Systems*, pages 455–462.

[Garzon and Botelho, 1999] Garzon, M. and Botelho, F. (1999). Dynamical approximation by recurrent neural networks. *Neurocomputing*, 29(1-3):25–46.

[Georgeff and Lansky, 1987] Georgeff, M. P. and Lansky, A. L. (1987). Reactive reasoning and planning. In *AAAI*, volume 87, pages 677–682.

[Gerald, 2012] Gerald, T. (2012). *Ordinary Differential Equations and Dynamical Systems*, volume 140 of *Graduate Studies in Mathematics*. American Mathematical Society.

[GERSCHGORIN, 1931] GERSCHGORIN, S. (1931). Uber die abgrenzung der eigenwerte einer matrix. *Izv. Akad. Nauk. USSR. Otd. Fiz-Mat. Nauk*, 7:749–754.

[Gidon et al., 2020] Gidon, A., Zolnik, T. A., Fidzinski, P., Bolduan, F., Papoutsi, A., Poirazi, P., Holtkamp, M., Vida, I., and Larkum, M. E. (2020). Dendritic action potentials and computation in human layer 2/3 cortical neurons. *Science*, 367(6473):83–87.

[Girosi et al., 1995] Girosi, F., Jones, M., and Poggio, T. (1995). Regularization theory and neural networks architectures. *Neural computation*, 7(2):219–269.

[Gleeson et al., 2018] Gleeson, P., Lung, D., Grosu, R., Hasani, R., and Larson, S. D. (2018). c302: a multiscale framework for modelling the nervous system of caenorhabditis elegans. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 373(1758):20170379.

[Goh, 2017] Goh, G. (2017). Why momentum really works. *Distill*, 2(4):e6.

[Golub and Van Loan, 1996] Golub, G. H. and Van Loan, C. F. (1996). Matrix computations, johns hopkins u. *Math. Sci., Johns Hopkins University Press, Baltimore, MD*.

[Graves and Jaitly, 2014] Graves, A. and Jaitly, N. (2014). Towards end-to-end speech recognition with recurrent neural networks. In *International conference on machine learning*, pages 1764–1772.

[Graves et al., 2013] Graves, A., Mohamed, A.-r., and Hinton, G. (2013). Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. IEEE.

[Greff et al., 2017] Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., and Schmidhuber, J. (2017). LSTM: A search space odyssey. *IEEE transactions on neural networks and learning systems*, 28(10):2222–2232.

[Gu et al., 2016] Gu, S., Holly, E., Lillicrap, T. P., and Levine, S. (2016). Deep reinforcement learning for robotic manipulation. *CoRR*, abs/1610.00633.

[Gulrajani et al., 2016] Gulrajani, I., Kumar, K., Ahmed, F., Taiga, A. A., Visin, F., Vazquez, D., and Courville, A. (2016). Pixelvae: A latent variable model for natural images. *arXiv preprint arXiv:1611.05013*.

[Hagan and Menhaj, 1994] Hagan, M. T. and Menhaj, M. B. (1994). Training feedforward networks with the marquardt algorithm. *IEEE transactions on Neural Networks*, 5(6):989–993.

[Hagras et al., 2004] Hagras, H., Pounds-Cornish, A., Colley, M., Callaghan, V., and Clarke, G. (2004). Evolving spiking neural network controllers for autonomous robots. In *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*, volume 5, pages 4620–4626 Vol.5.

[Haley and Soloway, 1992] Haley, P. J. and Soloway, D. (1992). Extrapolation limitations of multilayer feedforward neural networks. In *[Proceedings 1992] IJCNN International Joint Conference on Neural Networks*, volume 4, pages 25–30. IEEE.

[Han et al., 2015] Han, S., Pool, J., Tran, J., and Dally, W. (2015). Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pages 1135–1143.

[Hardt, 2020] Hardt, M. (2020). Fairness notions in decision making. In *2020 Annual Meeting*. AAAS.

[Hardt et al., 2018] Hardt, M., Ma, T., and Recht, B. (2018). Gradient descent learns linear dynamical systems. *Journal of Machine Learning Research*, 19(29):1–44.

[Hardt et al., 2016] Hardt, M., Price, E., and Srebro, N. (2016). Equality of opportunity in supervised learning. In *Advances in neural information processing systems*, pages 3315–3323.

[Hasani et al., 2019] Hasani, R., Amini, A., Lechner, M., Naser, F., Grosu, R., and Rus, D. (2019). Response characterization for auditing cell dynamics in long short-term memory networks. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE.

[Hasani et al., 2018a] Hasani, R., Amini, A., Lechner, M., Naser, F., Rus, D., and Grosu, R. (2018a). Response characterization for auditing cell dynamics in long short-term memory networks. *arXiv preprint arXiv:1809.03864*.

[Hasani et al., 2017a] Hasani, R., Beneder, V., Fuchs, M., Lung, D., and Grosu, R. (2017a). Sim-ce: An advanced simulink platform for studying the brain of caenorhabditis elegans. *arXiv preprint arXiv:1703.06270*.

[Hasani et al., 2020] Hasani, R., Lechner, M., Amini, A., Rus, D., and Grosu, R. (2020). Liquid time-constant recurrent neural networks. *submitted to the Proceedings of the National Academy of Sciences*, –(–):–.

[Hasani et al., 2017b] Hasani, R. M., Beneder, V., Fuchs, M., Lung, D., and Grosu, R. (2017b). Sim-ce: An advanced simulink platform for studying the brain of caenorhabditis elegans. *arXiv preprint arXiv:1703.06270*.

[Hasani et al., 2017c] Hasani, R. M., Fuchs, M., Beneder, V., and Grosu, R. (2017c). Non-associative learning representation in the nervous system of the nematode caenorhabditis elegans. *arXiv preprint arXiv:1703.06264*.

[Hasani et al., 2016] Hasani, R. M., Haerle, D., and Grosu, R. (2016). Efficient modeling of complex analog integrated circuits using neural networks. In *2016 12th Conference on Ph. D. Research in Microelectronics and Electronics (PRIME)*, pages 1–4. IEEE.

[Hasani et al., 2018b] Hasani, R. M., Lechner, M., Amini, A., Rus, D., and Grosu, R. (2018b). Liquid time-constant recurrent neural networks as universal approximators. *arXiv preprint arXiv:1811.00321*.

[Hasani et al., 2018c] Hasani, R. M., Lechner, M., Amini, A., Rus, D., and Grosu, R. (2018c). Re-purposing compact neuronal circuit policies to govern reinforcement learning tasks. *arXiv preprint arXiv:1809.04423*.

[Hassabis et al., 2017] Hassabis, D., Kumaran, D., Summerfield, C., and Botvinick, M. (2017). Neuroscience-inspired artificial intelligence. *Neuron*, 95(2):245–258.

[Hassibi and Stork, 1993] Hassibi, B. and Stork, D. G. (1993). Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in neural information processing systems*, pages 164–171.

[He et al., 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

[Heaton, 2008] Heaton, J. (2008). *Introduction to neural networks with Java*. Heaton Research, Inc.

[Heess et al., 2017] Heess, N., Sriram, S., Lemmon, J., Merel, J., Wayne, G., Tassa, Y., Erez, T., Wang, Z., Eslami, A., Riedmiller, M., et al. (2017). Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286*.

[Heidrich-Meisner and Igel, 2008] Heidrich-Meisner, V. and Igel, C. (2008). Variable metric reinforcement learning methods applied to the noisy mountain car problem. In *European Workshop on Reinforcement Learning*, pages 136–150. Springer.

[Hermann et al., 2015] Hermann, K. M., Kocisky, T., Grefenstette, E., Espeholt, L., Kay, W., Suleyman, M., and Blunsom, P. (2015). Teaching machines to read and comprehend. In *Advances in Neural Information Processing Systems*, pages 1693–1701.

[Hinton et al., 2015] Hinton, G., Vinyals, O., and Dean, J. (2015). Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

[Hinton, 1977] Hinton, G. E. (1977). Relaxation and its role in vision. *Edinburgh Research Arxiv (ERA)*.

[Hinton et al., 2006] Hinton, G. E., Osindero, S., and Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554.

[Hinton and Salakhutdinov, 2006] Hinton, G. E. and Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507.

[Hirsch and Smale, 1973] Hirsch, M. W. and Smale, S. (1973). *Differential equations, dynamical systems and linear algebra*. Academic Press college division.

[Hochreiter, 1991] Hochreiter, S. (1991). Untersuchungen zu dynamischen neuronalen netzen. *Diploma, Technische Universität München*, 91(1).

[Hochreiter et al., 2001] Hochreiter, S., Bengio, Y., Frasconi, P., Schmidhuber, J., et al. (2001). Gradient flow in recurrent nets: the difficulty of learning long-term dependencies.

[Hochreiter and Schmidhuber, 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

[Hornik et al., 1989] Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366.

[Hsu et al., 2003] Hsu, C.-W., Chang, C.-C., Lin, C.-J., et al. (2003). A practical guide to support vector classification. *Researchgate.net*.

[Hu et al., 2018] Hu, Y., Huber, A., Anumula, J., and Liu, S.-C. (2018). Overcoming the vanishing gradient problem in plain recurrent networks. *arXiv preprint arXiv:1801.06105*.

[Huang et al., 2017] Huang, X., Kwiatkowska, M., Wang, S., and Wu, M. (2017). Safety verification of deep neural networks. In *International Conference on Computer Aided Verification*, pages 3–29. Springer.

[Hudson and Manning, 2018] Hudson, D. A. and Manning, C. D. (2018). Compositional attention networks for machine reasoning. *arXiv preprint arXiv:1803.03067*.

[Hung et al., 2014] Hung, W. N. N., Song, X., Tan, J., Li, X., Zhang, J., Wang, R., and Gao, P. (2014). Motion planning with satisfiability modulo theories. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 113–118.

[Hurwitz, 1895] Hurwitz, A. (1895). Ueber die bedingungen, unter welchen eine gleichung nur wurzeln mit negativen reellen theilen besitzt. *Mathematische Annalen*, 46(2):273–284.

[Ijspeert et al., 2013] Ijspeert, A. J., Nakanishi, J., Hoffmann, H., Pastor, P., and Schaal, S. (2013). Dynamical movement primitives: learning attractor models for motor behaviors. *Neural computation*, 25(2):328–373.

[Ioffe and Szegedy, 2015] Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.

[Islam et al., 2016] Islam, M. A., Wang, Q., Hasani, R. M., Balún, O., Clarke, E. M., Grosu, R., and Smolka, S. A. (2016). Probabilistic reachability analysis of the tap withdrawal circuit in caenorhabditis elegans. In *2016 IEEE International High Level Design Validation and Test Workshop (HLDVT)*, pages 170–177. IEEE.

[Joseph et al., 2016] Joseph, M., Kearns, M., Morgenstern, J. H., and Roth, A. (2016). Fairness in learning: Classic and contextual bandits. In *Advances in Neural Information Processing Systems*, pages 325–333.

[Kádár et al., 2015] Kádár, A., Chrupała, G., and Alishahi, A. (2015). Lingusitic analysis of multi-modal recurrent neural networks. In *Proceedings of the Fourth Workshop on Vision and Language*, pages 8–9.

[Kádár et al., 2017] Kádár, A., Chrupała, G., and Alishahi, A. (2017). Representation of linguistic form and function in recurrent neural networks. *Computational Linguistics*, 43(4):761–780.

[Kaplan et al., 2019] Kaplan, H. S., Thula, O. S., Khoss, N., and Zimmer, M. (2019). Nested neuronal dynamics orchestrate a behavioral hierarchy across timescales. *Neuron*.

[Karpathy et al., 2015] Karpathy, A., Johnson, J., and Fei-Fei, L. (2015). Visualizing and understanding recurrent networks. *arXiv preprint arXiv:1506.02078*.

[Kato et al., 2015a] Kato, S., Kaplan, H. S., Schrödel, T., Skora, S., Lindsay, T. H., Yemini, E., Lockery, S., and Zimmer, M. (2015a). Global brain dynamics embed the motor command sequence of caenorhabditis elegans. *Cell*, 163(3):656–669.

[Kato et al., 2015b] Kato, S., Kaplan, H. S., Schrödel, T., Skora, S., Lindsay, T. H., Yemini, E., Lockery, S., and Zimmer, M. (2015b). Global brain dynamics embed the motor command sequence of *Caenorhabditis elegans*. *Cell*, 163:656–669.

[Kavraki and LaValle, 2008] Kavraki, L. E. and LaValle, S. M. (2008). Motion planning. In Siciliano, B. and Khatib, O., editors, *Springer Handbook of Robotics*, pages 109–131. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Kawaguchi et al., 2017] Kawaguchi, K., Kaelbling, L. P., and Bengio, Y. (2017). Generalization in deep learning. *arXiv preprint arXiv:1710.05468*.

[Keskar et al., 2016] Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., and Tang, P. T. P. (2016). On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*.

[Khansari-Zadeh and Billard, 2011] Khansari-Zadeh, S. M. and Billard, A. (2011). Learning stable nonlinear dynamical systems with gaussian mixture models. *IEEE Transactions on Robotics*, 27(5):943–957.

[Khansari-Zadeh and Billard, 2014] Khansari-Zadeh, S. M. and Billard, A. (2014). Learning control lyapunov function to ensure stability of dynamical system-based robot reaching motions. *Robotics and Autonomous Systems*, 62(6):752–765.

[Kindermans et al., 2017] Kindermans, P.-J., Schütt, K. T., Alber, M., Müller, K.-R., and Dähne, S. (2017). Patternnet and patternlrp–improving the interpretability of neural networks. *arXiv preprint arXiv:1705.05598*.

[Kingma and Ba, 2014] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[Knight, 2002] Knight, J. C. (2002). Safety critical systems: challenges and directions. In *Proceedings of the 24th international conference on software engineering*, pages 547–550. ACM.

[Koch and Segev, 1998] Koch, C. and Segev, K. (1998). *Methods in Neuronal Modeling - From Ions to Networks*. MIT press, second edition.

[Koh and Liang, 2017] Koh, P. W. and Liang, P. (2017). Understanding black-box predictions via influence functions. *arXiv preprint arXiv:1703.04730*.

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

[Krueger and Memisevic, 2015] Krueger, D. and Memisevic, R. (2015). Regularizing rnns by stabilizing activations. *arXiv preprint arXiv:1511.08400*.

[Kutta, 1901] Kutta, W. (1901). Beitrag zur naherungsweisen integration totaler differentialgleichungen. *Z. Math. Phys.*, 46:435–453.

[Lage et al., 2018] Lage, I., Ross, A., Gershman, S. J., Kim, B., and Doshi-Velez, F. (2018). Human-in-the-loop interpretability prior. In *Advances in Neural Information Processing Systems*, pages 10159–10168.

[Lakshminarayanan et al., 2017] Lakshminarayanan, B., Pritzel, A., and Blundell, C. (2017). Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in Neural Information Processing Systems*, pages 6402–6413.

[Latombe, 2012a] Latombe, J.-C. (2012a). *Robot motion planning*, volume 124. Springer Science & Business Media.

[Latombe, 2012b] Latombe, J.-C. (2012b). *Robot motion planning*, volume 124. Springer Science & Business Media.

[LaValle, 1998] LaValle, S. (1998). Rapidly-exploring random trees: a new tool for path planning. *Research Report 9811*.

[Lechner et al., 2020] Lechner, M., Hasani, R., Rus, D., and Grosu, R. (2020). Gershgorin loss stabilizes the recurrent neural network compartment of an end-to-end robot learning scheme. In *Accepted to the International Conference on Robotics and Automation (ICRA)*. IEEE.

[Lechner et al., 2019] Lechner, M., Hasani, R., Zimmer, M., Henzinger, T. A., and Grosu, R. (2019). Designing worm-inspired neural networks for interpretable robotic control. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 87–94. IEEE.

[LeCun et al., 2015a] LeCun, Y., Bengio, Y., and Hinton, G. (2015a). Deep learning. *nature*, 521(7553):436–444.

[LeCun et al., 2015b] LeCun, Y., Bengio, Y., and Hinton, G. (2015b). Deep learning. *Nature*, pages 436–444.

[LeCun et al., 1990a] LeCun, Y., Boser, B. E., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. E., and Jackel, L. D. (1990a). Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*, pages 396–404.

[LeCun et al., 1998] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

[LeCun et al., 2010] LeCun, Y., Cortes, C., and Burges, C. (2010). Mnist handwritten digit database.

[LeCun et al., 1990b] LeCun, Y., Denker, J. S., and Solla, S. A. (1990b). Optimal brain damage. In *Advances in neural information processing systems*, pages 598–605.

[Lemme et al., 2014] Lemme, A., Neumann, K., Reinhart, R. F., and Steil, J. J. (2014). Neural learning of vector fields for encoding stable dynamical systems. *Neurocomputing*, 141:3–14.

[Levine et al., 2016] Levine, S., Finn, C., Darrell, T., and Abbeel, P. (2016). End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373.

[Levine and Koltun, 2013] Levine, S. and Koltun, V. (2013). Guided policy search. In *International Conference on Machine Learning*, pages 1–9.

[Li et al., 2012] Li, Z., Li, Y., Yi, Y., Huang, W., Yang, S., Niu, W., Zhang, L., Xu, Z., Qu, A., Wu, Z., et al. (2012). Dissecting a central flip-flop circuit that integrates contradictory sensory cues in c. elegans feeding regulation. *Nature communications*, 3:776.

[Lillicrap et al., 2015] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

[Lin et al., 1996] Lin, T., Horne, B. G., Tino, P., and Giles, C. L. (1996). Learning long-term dependencies in narx recurrent neural networks. *IEEE Transactions on Neural Networks*, 7(6):1329–1338.

[Liu et al., 2014] Liu, S., Yang, N., Li, M., and Zhou, M. (2014). A recursive recurrent neural network for statistical machine translation. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1491–1500.

[Lockery and Sejnowski, 1992] Lockery, S. and Sejnowski, T. (1992). Distributed processing of sensory information in the leech. iii. a dynamical neural network model of the local bending reflex. *Journal of Neuroscience*, 12(10):3877–3895.

[Long et al., 2018a] Long, Y., She, X., and Mukhopadhyay, S. (2018a). Hybridnet: integrating model-based and data-driven learning to predict evolution of dynamical systems. *arXiv preprint arXiv:1806.07439*.

[Long et al., 2018b] Long, Z., Lu, Y., and Dong, B. (2018b). Pde-net 2.0: Learning pdes from data with a numeric-symbolic hybrid deep network. *arXiv preprint arXiv:1812.04426*.

[Lu et al., 2017] Lu, Y., Zhong, A., Li, Q., and Dong, B. (2017). Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations. *arXiv preprint arXiv:1710.10121*.

[Luong et al., 2015] Luong, M.-T., Pham, H., and Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*.

[M. Hasani et al., 2018] M. Hasani, R., Lechner, M., Amini, A., Rus, D., and Grosu, R. (2018). Re-purposing compact neuronal circuit policies to govern reinforcement learning tasks. *arXiv preprint arXiv:1809.04423*.

[Maass, 1997] Maass, W. (1997). Networks of spiking neurons: the third generation of neural network models. *Neural networks*, 10(9):1659–1671.

[Maaten and Hinton, 2008] Maaten, L. v. d. and Hinton, G. (2008). Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605.

[Magerl et al., 2015] Magerl, M., Stockreiter, C., Eisenberger, O., Minixhofer, R., and Baric, A. (2015). Building interchangeable black-box models of integrated circuits for emc simulations. In *Electromagnetic Compatibility of Integrated Circuits (EMC Compo), 2015 10th International Workshop on the*, pages 258–263. IEEE.

[Mania et al., 2018] Mania, H., Guy, A., and Recht, B. (2018). Simple random search provides a competitive approach to reinforcement learning. *arXiv preprint arXiv:1803.07055*.

[Marquardt, 1963] Marquardt, D. W. (1963). An algorithm for least-squares estimation of nonlinear parameters. *Journal of the society for Industrial and Applied Mathematics*, 11(2):431–441.

[Martens and Sutskever, 2011] Martens, J. and Sutskever, I. (2011). Learning recurrent neural networks with hessian-free optimization. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1033–1040. Citeseer.

[Martius and Lampert, 2016] Martius, G. and Lampert, C. H. (2016). Extrapolation and learning equations. *arXiv preprint arXiv:1610.02995*.

[Masehian and Sedighizadeh, 2007] Masehian, E. and Sedighizadeh, D. (2007). Classic and heuristic approaches in robot motion planning-a chronological review. *World Academy of Science, Engineering and Technology*, 23:101–106.

[Mayer et al., 2008] Mayer, H., Gomez, F., Wierstra, D., Nagy, I., Knoll, A., and Schmidhuber, J. (2008). A system for robotic heart surgery that learns to tie knots using recurrent neural networks. *Advanced Robotics*, 22(13-14):1521–1537.

[Melis and Jaakkola, 2018] Melis, D. A. and Jaakkola, T. (2018). Towards robust interpretability with self-explaining neural networks. In *Advances in Neural Information Processing Systems*, pages 7775–7784.

[Mikolov et al., 2010] Mikolov, T., Karafiát, M., Burget, L., Černockỳ, J., and Khudanpur, S. (2010). Recurrent neural network based language model. In *Eleventh annual conference of the international speech communication association*.

[Mikolov et al., 2011] Mikolov, T., Kombrink, S., Burget, L., Černockỳ, J., and Khudanpur, S. (2011). Extensions of recurrent neural network language model. In *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5528–5531. IEEE.

[Milo et al., 2004] Milo, R., Itzkovitz, S., Kashtan, N., Levitt, R., Shen-Orr, S., Ayzenshtat, I., Sheffer, M., and Alon, U. (2004). Superfamilies of evolved and designed networks. *Science*, 303(5663):1538–1542.

[Milo et al., 2002] Milo, R., Shen-Orr, S., Itzkovitz, S., Kashtan, N., Chklovskii, D., and Alon, U. (2002). Network motifs: Simple building blocks of complex networks. *Science*, 298(5594):824–827.

[Mnih et al., 2016] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937.

[Mnih et al., 2015] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529.

[Mohamed et al., 2011] Mohamed, A.-r., Dahl, G. E., and Hinton, G. (2011). Acoustic modeling using deep belief networks. *IEEE transactions on audio, speech, and language processing*, 20(1):14–22.

[Molnar, 2019] Molnar, C. (2019). *Interpretable machine learning*. Lulu. com.

[Montufar et al., 2014] Montufar, G. F., Pascanu, R., Cho, K., and Bengio, Y. (2014). On the number of linear regions of deep neural networks. In *Advances in neural information processing systems*, pages 2924–2932.

[Morcos et al., 2019] Morcos, A., Yu, H., Paganini, M., and Tian, Y. (2019). One ticket to win them all: generalizing lottery ticket initializations across datasets and optimizers. In *Advances in Neural Information Processing Systems*, pages 4933–4943.

[Moshchuk and Chen, 2009] Moshchuk, N. and Chen, S.-K. (2009). Spot locator for autonomous parking. In *ASME 2009 International Mechanical Engineering Congress & Exposition*.

[Mozer et al., 2017] Mozer, M. C., Kazakov, D., and Lindsey, R. V. (2017). Discrete event, continuous time rnns. *arXiv preprint arXiv:1710.04110*.

[Nair and Hinton, 2010] Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814.

[Narayanan et al., 2008] Narayanan, R., Abbasi, N., Zaki, M., Al Sammane, G., and Tahar, S. (2008). On the simulation performance of contemporary ams hardware description languages. In *2008 International Conference on Microelectronics*, pages 361–364. IEEE.

[Naser et al., 2017] Naser, F., Dorhout, D., Proulx, S., Pendleton, S. D., Andersen, H., Schwarting, W., Paull, L., Alonso-Mora, J., Ang, M. H., Karaman, S., et al. (2017). A parallel autonomy research platform. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 933–940. IEEE.

[Nedunuri et al., 2014] Nedunuri, S., Prabhu, S., Moll, M., Chaudhuri, S., and Kavraki, L. E. (2014). Smt-based synthesis of integrated task and motion plans from plan outlines. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 655–662.

[Neelakantan et al., 2015] Neelakantan, A., Vilnis, L., Le, Q. V., Sutskever, I., Kaiser, L., Kurach, K., and Martens, J. (2015). Adding gradient noise improves learning for very deep networks. *arXiv preprint arXiv:1511.06807*.

[Nesterov, 1983] Nesterov, Y. (1983). A method of solving a convex programming problem with convergence rate o(1k2). In *Soviet Math. Dokl*, volume 27.

[Nichols et al., 2017] Nichols, A. L., Eichler, T., Latham, R., and Zimmer, M. (2017). A global brain state underlies c. elegans sleep behavior. *Science*, 356(6344):eaam6851.

[Novak et al., 2018] Novak, R., Bahri, Y., Abolafia, D. A., Pennington, J., and Sohl-Dickstein, J. (2018). Sensitivity and generalization in neural networks: an empirical study. *arXiv preprint arXiv:1802.08760*.

[Olah et al., 2018] Olah, C., Satyanarayan, A., Johnson, I., Carter, S., Schubert, L., Ye, K., and Mordvintsev, A. (2018). The building blocks of interpretability. *Distill*, 3(3):e10.

[OMRON ADEPT MOBILEROBOTS, 2011] OMRON ADEPT MOBILEROBOTS, L. (2011). *Datasheet: Pioneer 3-AT robot*. OMRON ADEPT MOBILEROBOTS, LLC.

[Oord et al., 2016] Oord, A. v. d., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., and Kavukcuoglu, K. (2016). Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*.

[OpenAI, 2018] OpenAI (2018). Openai five. https://blog.openai.com/openai-five/.

[Oppenheim and Young, 1983] Oppenheim, A. V., W. A. S. and Young, I. T. (1983). *Signals and systems*. Prentice-Hall.

[Pascanu et al., 2012] Pascanu, R., Mikolov, T., and Bengio, Y. (2012). Understanding the exploding gradient problem. *CoRR, abs/1211.5063*, 2.

[Pascanu et al., 2013] Pascanu, R., Mikolov, T., and Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318.

[Paszke et al., 2017] Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. (2017). Automatic differentiation in pytorch. *Openreview*.

[Pearce et al., 2018] Pearce, T., Zaki, M., Brintrup, A., and Neel, A. (2018). Uncertainty in neural networks: Bayesian ensembling. *arXiv preprint arXiv:1810.05546*.

[Pearl, 2009] Pearl, J. (2009). *Causality*. Cambridge university press.

[Pêcheux et al., 2005] Pêcheux, F., Lallement, C., and Vachoux, A. (2005). Vhdl-ams and verilog-ams as alternative hardware description languages for efficient modeling of multidiscipline systems. *IEEE transactions on Computer-Aided design of integrated Circuits and Systems*, 24(2):204–225.

[Peng et al., 2017] Peng, X. B., Andrychowicz, M., Zaremba, W., and Abbeel, P. (2017). Sim-to-real transfer of robotic control with dynamics randomization. *arXiv preprint arXiv:1710.06537*.

[Penny et al., 2005] Penny, W., Ghahramani, Z., and Friston, K. (2005). Bilinear dynamical systems. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 360(1457):983–993.

[Penny et al., 2004] Penny, W. D., Stephan, K. E., Mechelli, A., and Friston, K. J. (2004). Comparing dynamic causal models. *Neuroimage*, 22(3):1157–1172.

[Pérez-Escudero and de Polavieja, 2007] Pérez-Escudero, A. and de Polavieja, G. G. (2007). Optimally wired subnetwork determines neuroanatomy of caenorhabditis elegans. *Proceedings of the National Academy of Sciences*, 104(43):17180–17185.

[Peters et al., 2017] Peters, J., Janzing, D., and Schölkopf, B. (2017). *Elements of causal inference: foundations and learning algorithms*. MIT press.

[Poincaré, 1885] Poincaré, H. (1885). L'Équilibre d'une masse fluide animée d'un mouvement de rotation. *Acta Mathematica*, 7:259–380.

[Pokorny et al., 2016] Pokorny, F. T., Kragic, D., Kavraki, L. E., and Goldberg, K. (2016). High-dimensional winding-augmented motion planning with 2d topological task projections and persistent homology. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 24–31. IEEE.

[Pomerleau, 1989] Pomerleau, D. A. (1989). Alvinn: An autonomous land vehicle in a neural network. In *Advances in neural information processing systems*, pages 305–313.

[Press et al., 2007a] Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. (2007a). *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press.

[Press et al., 2007b] Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. (2007b). *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. Cambridge University Press, New York, NY, USA, 3 edition.

[Qian and Sejnowski, 1988] Qian, N. and Sejnowski, T. J. (1988). Predicting the secondary structure of globular proteins using neural network models. *Journal of molecular biology*, 202(4):865–884.

[R. F. Keeling and S. J. Walker, 3 17] R. F. Keeling, S. C. Piper, A. F. B. and S. J. Walker, M. L. (Accessed: 2018-03-17). Atmospheric carbon dioxide record. `http://cdiac.ess-dive.lbl.gov/ftp/trends/co2/maunaloa.co2`.

[Raghu et al., 2017] Raghu, M., Poole, B., Kleinberg, J., Ganguli, S., and Dickstein, J. S. (2017). On the expressive power of deep neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2847–2854. JMLR. org.

[Rankin et al., 1990] Rankin, C. H., Beck, C. D., and Chiba, C. M. (1990). Caenorhabditis elegans: a new model system for the study of learning and memory. *Behavioural brain research*, 37(1):89–92.

[Rastrigin, 1963] Rastrigin, L. A. (1963). About convergence of random search method in extremal control of multi-parameter systems. *Avtomat. i Telemekh.*, 24:1467–1473.

[Ravichandar et al., 2017] Ravichandar, H. C., Salehi, I., and Dani, A. P. (2017). Learning partially contracting dynamical systems from demonstrations. In *CoRL*, pages 369–378.

[Reddi et al., 2019] Reddi, S. J., Kale, S., and Kumar, S. (2019). On the convergence of adam and beyond. *arXiv preprint arXiv:1904.09237*.

[Rethage et al., 2018] Rethage, D., Pons, J., and Serra, X. (2018). A wavenet for speech denoising. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5069–5073. IEEE.

[Richards et al., 2018] Richards, S. M., Berkenkamp, F., and Krause, A. (2018). The lyapunov neural network: Adaptive stability certification for safe learning of dynamic systems. *arXiv preprint arXiv:1808.00924*.

[Robinson et al., 1996] Robinson, T., Hochberg, M., and Renals, S. (1996). The use of recurrent neural networks in continuous speech recognition. In *Automatic speech and speaker recognition*, pages 233–258. Springer.

[Rodgers, 1985] Rodgers, D. P. (1985). Improvements in multiprocessor system design. *ACM SIGARCH Computer Architecture News*, 13(3):225–231.

[Rubanova et al., 2019] Rubanova, Y., Chen, T. Q., and Duvenaud, D. K. (2019). Latent ordinary differential equations for irregularly-sampled time series. In *Advances in Neural Information Processing Systems*, pages 5321–5331.

[Rudin, 2019] Rudin, C. (2019). Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 1(5):206–215.

[Rumelhart et al., 1986] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *nature*, 323(6088):533–536.

[Rumelhart et al., 1988] Rumelhart, D. E., Hinton, G. E., Williams, R. J., et al. (1988). Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1.

[Runge, 1895] Runge, C. (1895). Über die numerische auflösung von differentialgleichungen. *Mathematische Annalen*, 46(2):167–178.

[Rusu et al., 2016] Rusu, A. A., Vecerik, M., Rothörl, T., Heess, N., Pascanu, R., and Hadsell, R. (2016). Sim-to-real robot learning from pixels with progressive nets. *arXiv preprint arXiv:1610.04286*.

[Sahoo et al., 2018] Sahoo, S. S., Lampert, C. H., and Martius, G. (2018). Learning equations for extrapolation and control. *arXiv preprint arXiv:1806.07259*.

[Sak et al., 2014] Sak, H., Senior, A., and Beaufays, F. (2014). Long short-term memory recurrent neural network architectures for large scale acoustic modeling. In *Fifteenth annual conference of the international speech communication association*.

[Salimans et al., 2017] Salimans, T., Ho, J., Chen, X., Sidor, S., and Sutskever, I. (2017). Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*.

[Sarma et al., 2018] Sarma, G. P., Lee, C. W., Portegys, T., Ghayoomie, V., Jacobs, T., Alicea, B., Cantarelli, M., Currie, M., Gerkin, R. C., Gingell, S., et al. (2018). Openworm: overview and recent advances in integrative biological simulation of caenorhabditis elegans. *Philosophical Transactions of the Royal Society B*, 373(1758):20170382.

[Schäfer and Zimmermann, 2006] Schäfer, A. M. and Zimmermann, H. G. (2006). Recurrent neural networks are universal approximators. In *International Conference on Artificial Neural Networks*, pages 632–640. Springer.

[Schrittwieser et al., 2019] Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., et al. (2019). Mastering atari, go, chess and shogi by planning with a learned model. *arXiv preprint arXiv:1911.08265*.

[Schulman et al., 2017] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

[Shiloach and Vishkin, 1982] Shiloach, Y. and Vishkin, U. (1982). An o (n2log n) parallel max-flow algorithm. *Journal of Algorithms*, 3(2):128–146.

[Shokrolah-Shirazi and Miremadi, 2008] Shokrolah-Shirazi, M. and Miremadi, S. G. (2008). Fpga-based fault injection into synthesizable verilog hdl models. In *Secure System Integration and Reliability Improvement, 2008. SSIRI'08. Second International Conference on*, pages 143–149. IEEE.

[Shon et al., 2005] Shon, A. P., Grochow, K., and Rao, R. P. (2005). Robotic imitation from human motion capture using gaussian processes. In *5th IEEE-RAS International Conference on Humanoid Robots, 2005.*, pages 129–134. IEEE.

[Shwartz-Ziv and Tishby, 2017] Shwartz-Ziv, R. and Tishby, N. (2017). Opening the black box of deep neural networks via information. *arXiv preprint arXiv:1703.00810*.

[Siegelmann et al., 1997] Siegelmann, H. T., Horne, B. G., and Giles, C. L. (1997). Computational capabilities of recurrent narx neural networks. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 27(2):208–215.

[Silver et al., 2016a] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016a). Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489. Article.

[Silver et al., 2016b] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016b). Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484.

[Silver et al., 2018] Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al. (2018). A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144.

[Silver et al., 2017] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. (2017). Mastering the game of go without human knowledge. *nature*, 550(7676):354–359.

[Simonyan et al., 2013] Simonyan, K., Vedaldi, A., and Zisserman, A. (2013). Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*.

[Simonyan and Zisserman, 2014] Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

[Singh et al., 2017] Singh, S., Majumdar, A., Slotine, J.-J., and Pavone, M. (2017). Robust online motion planning via contraction theory and convex optimization. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5883–5890. IEEE.

[Singh and Sutton, 1996] Singh, S. P. and Sutton, R. S. (1996). Reinforcement learning with replacing eligibility traces. *Recent Advances in Reinforcement Learning*, pages 123–158.

[Smale and Zhou, 2007] Smale, S. and Zhou, D.-X. (2007). Learning theory estimates via integral operators and their approximations. *Constructive approximation*, 26(2):153–172.

[Sontag, 2013] Sontag, E. D. (2013). *Mathematical control theory: deterministic finite dimensional systems*, volume 6. Springer Science & Business Media.

[Spall, 2005] Spall, J. C. (2005). *Introduction to stochastic search and optimization: estimation, simulation, and control*, volume 65. John Wiley & Sons.

[Srivastava et al., 2014] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958.

[Srivastava et al., 2015] Srivastava, N., Mansimov, E., and Salakhudinov, R. (2015). Unsupervised learning of video representations using lstms. In *International conference on machine learning*, pages 843–852.

[Stavridis et al., 2017] Stavridis, S., Papageorgiou, D., and Doulgeri, Z. (2017). Dynamical system based robotic motion generation with obstacle avoidance. *IEEE Robotics and Automation Letters*, 2(2):712–718.

[Strobelt et al., 2018] Strobelt, H., Gehrmann, S., Pfister, H., and Rush, A. M. (2018). Lstmvis: A tool for visual analysis of hidden state dynamics in recurrent neural networks. *IEEE transactions on visualization and computer graphics*, 24(1):667–676.

[Suissa-Peleg et al., 2016] Suissa-Peleg, A., Haehn, D., Knowles-Barley, S., Kaynig, V., Jones, T. R., Wilson, A., Schalek, R., Lichtman, J. W., and Pfister, H. (2016). Automatic neural reconstruction from petavoxel of electron microscopy data. *Microscopy and Microanalysis*, 22(S3):536–537.

[Sundararajan et al., 2017] Sundararajan, M., Taly, A., and Yan, Q. (2017). Axiomatic attribution for deep networks. *arXiv preprint arXiv:1703.01365*.

[Sutskever et al., 2011] Sutskever, I., Martens, J., and Hinton, G. E. (2011). Generating text with recurrent neural networks. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1017–1024.

[Sutskever et al., 2014] Sutskever, I., Vinyals, O., and Le, Q. (2014). Sequence to sequence learning with neural networks. *Advances in NIPS*.

[Szczecinski et al., 2015] Szczecinski, N. S., Chrzanowski, D. M., Cofer, D. W., Terrasi, A. S., Moore, D. R., Martin, J. P., Ritzmann, R. E., and Quinn, R. D. (2015). Introducing mantisbot: Hexapod robot controlled by a high-fidelity, real-time neural simulation. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3875–3881.

[Szczecinski et al., 2017] Szczecinski, N. S., Hunt, A. J., and Quinn, R. D. (2017). Design process and tools for dynamic neuromechanical models and robot controllers. *Biological Cybernetics*, 111(1):105–127.

[Szegedy et al., 2015] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9.

[Szigeti et al., 2014] Szigeti, B., Gleeson, P., Vella, M., Khayrulin, S., Palyanov, A., Hokanson, J., Currie, M., Cantarelli, M., Idili, G., and Larson, S. (2014). Openworm: an open-science approach to modeling caenorhabditis elegans. *Frontiers in computational neuroscience*, 8.

[Tan et al., 2018] Tan, S., Caruana, R., Hooker, G., Koch, P., and Gordo, A. (2018). Learning global additive explanations for neural nets using model distillation. *arXiv preprint arXiv:1801.08640*.

[Teschl, 2012] Teschl, G. (2012). *Ordinary differential equations and dynamical systems*, volume 140. American Mathematical Soc.

[Tieleman and Hinton, 2012] Tieleman, T. and Hinton, G. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31.

[Trask et al., 2018] Trask, A., Hill, F., Reed, S. E., Rae, J., Dyer, C., and Blunsom, P. (2018). Neural arithmetic logic units. In *Advances in Neural Information Processing Systems*, pages 8035–8044.

[Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

[Vidyasagar and Karandikar, 2006] Vidyasagar, M. and Karandikar, R. L. (2006). A learning theory approach to system identification and stochastic adaptive control. In *Probabilistic and randomized methods for design under uncertainty*, pages 265–302. Springer.

[Vinyals et al., 2019] Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., et al. (2019). Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354.

[Wagner et al., 2014] Wagner, P. K., Peres, S. M., Madeo, R. C. B., de Moraes Lima, C. A., and de Almeida Freitas, F. (2014). Gesture unit segmentation using spatial-temporal information and machine learning. In *The Twenty-Seventh International Flairs Conference*.

[Waibel et al., 1989] Waibel, A., Hanazawa, T., Hinton, G., Shikano, K., and Lang, K. J. (1989). Phoneme recognition using time-delay neural networks. *IEEE transactions on acoustics, speech, and signal processing*, 37(3):328–339.

[Wang et al., 2004] Wang, Z., Bovik, A. C., Sheikh, H. R., Simoncelli, E. P., et al. (2004). Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612.

[Wen et al., 2012] Wen, Q., Po, M. D., Hulme, E., Chen, S., Liu, X., Kwok, S. W., Gershow, M., Leifer, A. M., Butler, V., Fang-Yen, C., et al. (2012). Proprioceptive coupling within motor neurons drives c. elegans forward locomotion. *Neuron*, 76(4):750–761.

[Werbos et al., 1990] Werbos, P. J. et al. (1990). Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560.

[Westphal et al., 2013] Westphal, A., Blustein, D., and Ayers, J. (2013). A biomimetic neuronal network-based controller for guided helicopter flight. In Lepora, N. F., Mura, A., Krapp, H. G., Verschure, P. F. M. J., and Prescott, T. J., editors, *Biomimetic and Biohybrid Systems: Second International Conference, Living Machines 2013, London, UK, July 29 – August 2, 2013. Proceedings*, pages 299–310. Springer Berlin Heidelberg, Berlin, Heidelberg.

[White et al., 1986] White, J. G., Southgate, E., Thomson, J. N., and Brenner, S. (1986). The structure of the nervous system of the nematode caenorhabditis elegans. *Philos Trans R Soc Lond B Biol Sci*, 314(1165):1–340.

[Wicks and Rankin, 1995] Wicks, S. and Rankin, C. (1995). Integration of mechanosensory stimuli in *Caenorhabditis elegans*. *Journal of Neuroscience*, 15(3):2434–2444.

[Wicks et al., 1996] Wicks, S. R., Roehrig, C. J., and Rankin, C. H. (1996). A dynamic network simulation of the nematode tap withdrawal circuit: predictions concerning synaptic function using behavioral criteria. *Journal of Neuroscience*, 16(12):4017–4031.

[Williams et al., 2016] Williams, J. J., Kim, J., Rafferty, A., Maldonado, S., Gajos, K. Z., Lasecki, W. S., and Heffernan, N. (2016). Axis: Generating explanations at scale with learnersourcing and machine learning. In *Proceedings of the Third (2016) ACM Conference on Learning@ Scale*, pages 379–388.

[Williams and Zipser, 1989] Williams, R. J. and Zipser, D. (1989). A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280.

[Xingjian et al., 2015] Xingjian, S., Chen, Z., Wang, H., Yeung, D.-Y., Wong, W.-K., and Woo, W.-c. (2015). Convolutional lstm network: A machine learning approach for precipitation nowcasting. In *Advances in neural information processing systems*, pages 802–810.

[Xu et al., 2020] Xu, C. S., Januszewski, M., Lu, Z., Takemura, S.-y., Hayworth, K., Huang, G., Shinomiya, K., Maitin-Shepard, J., Ackerman, D., Berg, S., et al. (2020). A connectome of the adult drosophila central brain. *BioRxiv*.

[Xu et al., 2017] Xu, H., Gao, Y., Yu, F., and Darrell, T. (2017). End-to-end learning of driving models from large-scale video datasets. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2174–2182.

[Yan et al., 2017] Yan, G., Vértes, P. E., Towlson, E. K., Chew, Y. L., Walker, D. S., Schafer, W. R., and Barabási, A.-L. (2017). Network control principles predict neuron function in the caenorhabditis elegans connectome. *Nature*, 550(7677):519.

[Yosinski et al., 2015] Yosinski, J., Clune, J., Nguyen, A., Fuchs, T., and Lipson, H. (2015). Understanding neural networks through deep visualization. *arXiv preprint arXiv:1506.06579*.

[You et al., 2016] You, Q., Jin, H., Wang, Z., Fang, C., and Luo, J. (2016). Image captioning with semantic attention. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4651–4659.

[Zanzotto, 2019] Zanzotto, F. M. (2019). Human-in-the-loop artificial intelligence. *Journal of Artificial Intelligence Research*, 64:243–252.

[Zeiler, 2012] Zeiler, M. D. (2012). Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.

[Zeiler and Fergus, 2014] Zeiler, M. D. and Fergus, R. (2014). Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer.

[Zeng et al., 2017] Zeng, K.-H., Shen, W. B., Huang, D.-A., Sun, M., and Carlos Niebles, J. (2017). Visual forecasting by imitating dynamics in natural sequences. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2999–3008.

[Zhang et al., 2014] Zhang, H., Wang, Z., and Liu, D. (2014). A comprehensive review of stability analysis of continuous-time recurrent neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 25(7):1229–1262.

[Zhang and Fan, 2008] Zhang, K. and Fan, W. (2008). Forecasting skewed biased stochastic ozone days: analyses, solutions and beyond. *Knowledge and Information Systems*, 14(3):299–326.

[Zhang et al., 2017] Zhang, M., Geng, X., Bruce, J., Caluwaerts, K., Vespignani, M., SunSpiral, V., Abbeel, P., and Levine, S. (2017). Deep reinforcement learning for tensegrity robot locomotion. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 634–641. IEEE.

[Zhang et al., 2016] Zhang, M., McCarthy, Z., Finn, C., Levine, S., and Abbeel, P. (2016). Learning deep neural network policies with continuous memory states. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 520–527. IEEE.

[Zhao and Cao, 2006] Zhao, W. and Cao, Y. (2006). New generation of predictive technology model for sub-45 nm early design exploration. *IEEE Transactions on Electron Devices*, 53(11):2816–2823.

[Zhou et al., 2019] Zhou, H., Lan, J., Liu, R., and Yosinski, J. (2019). Deconstructing lottery tickets: Zeros, signs, and the supermask. In *Advances in Neural Information Processing Systems*, pages 3592–3602.