# Symbols, Dynamics, and Maps: A Neurosymbolic Approach to Spatial Cognition

by

Nicole Sandra-Yaffa Dumont

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2025

© Nicole Sandra-Yaffa Dumont 2025

**Examining Committee Membership**

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

| | |
|---|---|
| External Examiner | Ila Fiete<br>Professor,<br>Department of Brain and Cognitive Sciences,<br>Massachusetts Institute of Technology |
| Supervisor | Chris Eliasmith<br>Professor,<br>Department of Philosophy and<br>Department of Systems Design Engineering,<br>University of Waterloo |
| Supervisor | Jeff Orchard<br>Associate Professor,<br>David R. Cheriton School of Computer Science,<br>University of Waterloo |
| Internal Member | Pascal Poupart<br>Professor,<br>David R. Cheriton School of Computer Science,<br>University of Waterloo |
| Internal Member | Jesse Hoey<br>Professor,<br>David R. Cheriton School of Computer Science,<br>University of Waterloo |
| Internal-External Member | Sue Ann Campbell<br>Professor,<br>Department of Applied Mathematics,<br>University of Waterloo |

**Author's Declaration**

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

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

## Statement of Contributions

Nicole Sandra-Yaffa Dumont is the sole author of Chapters 1 and 6. These chapters were either not written for publication. The remaining chapters are, in heavily edited and extended form, based on the following published works:

[1] Nicole Sandra-Yaffa Dumont and Chris Eliasmith (2020). "Accurate representation for spatial cognition using grid cells". In: *42nd Annual Meeting of the Cognitive Science Society*. Toronto, ON: Cognitive Science Society, pp. 2367–2373

[2] Aaron R Voelker, Peter Blouw, Xuan Choo, Nicole Sandra-Yaffa Dumont, Terrence C Stewart, and Chris Eliasmith (2021). "Simulating and Predicting Dynamical Systems With Spatial Semantic Pointers". In: *Neural Computation* 33.8, pp. 2033–2067

[3] Nicole Sandra-Yaffa Dumont, Jeff Orchard, and Chris Eliasmith (2022). "A model of path integration that connects neural and symbolic representation". In: *Proceedings of the Annual Meeting of the Cognitive Science Society*. Vol. 44. 44. Toronto, ON: Cognitive Science Society

[4] Nicole Sandra-Yaffa Dumont, Andreas Stöckel, P Michael Furlong, Madeleine Bartlett, Chris Eliasmith, and Terrence C Stewart (2023b). "Biologically-Based Computation: How Neural Details and Dynamics Are Suited for Implementing a Variety of Algorithms". In: *Brain Sciences* 13.2, p. 245

[5] Nicole Sandra-Yaffa Dumont, P Michael Furlong, Jeff Orchard, and Chris Eliasmith (2023a). "Exploiting semantic information in a spiking neural SLAM system". In: *Frontiers in Neuroscience* 17, p. 1190515

[6] Madeleine Bartlett, Nicole Sandra-Yaffa Dumont, Michael P Furlong, and Terrence C Stewart (2022). "Biologically-Plausible Memory for Continuous-Time Reinforcement Learning". In

[7] P Michael Furlong, Katherine Simone, Nicole Dumont, Madeline Bartlett, Terrence C. Stewart, Jeff Orchard, and Chris Eliasmith (2024). "Biologically-plausible Markov Chain Monte Carlo Sampling from Vector Symbolic Algebra-encoded Distributions". In: *Artificial Neural Networks and Machine Learning – ICANN 2024*. Springer Nature Switzerland

The use of material from these publications in this thesis is with permission of the copyright holders. All figures in this thesis were created by the author. Annotations such as "inspired by" or "adapted from" merely indicate the source of the presented information.

**Contributions to the individual publications**

[1] Nicole Dumont developed the methods, conducted the experiments, created the figures and code, and wrote the manuscript. Dr. Chris Eliasmith supervised this work, providing feedback on the methods and edits to the manuscript. Discussions with Dr. Terrence C. Stewart and Dr. Peter Duggins, mentioned in the acknowledgements, helped improve this paper.

[2] Dr. Aaron R. Voelker and Dr. Peter Blouw are the primary authors of this work, contributing equally to the theoretical development, experimental design, and writing of this paper, with contributions from Dr. Xuan Choo and Dr. Terrence C. Stewart as well. Dr. Chris Eliasmith supervised this work. The dynamics of SSPs are presented in this publication and were independently derived by both Nicole Dumont and Dr. Aaron R. Voelker. Nicole Dumont edited the manuscript, wrote Sections 2.2.2, 2.2.3, parts of 3.3.3, performed the experiments presented in Figures 8 and 9, and made those figures along with Figure 3.

[3] Nicole Dumont developed the methods, conducted the experiments, created the figures and code, and wrote the manuscript. Dr. Chris Eliasmith and Dr. Jeff Orchard supervised this work, providing feedback on the methods and edits to the manuscript.

[4] Nicole Dumont, Dr. Andreas Stöckel, and Dr. P. Michael Furlong contributed equally to this paper. The work (writing, experiments, and figures) in Section 4 (with the expectation of Section 4.3) and Section 5 was done by Nicole Dumont. Dr. Andreas Stöckel and Dr. Terrence C. Stewart wrote Sections 1 and 2. The figures in Section 2 and all work in Section 3 was done by Dr. Andreas Stöckel. The work in Section 6 was done by Dr. P. Michael Furlong. The work in Section 4.3 was done by Dr. Madeleine Bartlett. Dr. Terrence C. Stewart and Dr. Chris Eliasmith supervised this work. All authors contributed to editing the manuscript.

[5] Nicole Dumont developed the methods, conducted the experiments, created the figures and code, and wrote the manuscript. Dr. Chris Eliasmith and Dr. Jeff Orchard supervised this work, providing feedback on the methods and edits to the manuscript. Dr. P. Michael Furlong edited the manuscript and contributed to the development of and discussion on the probabilistic interpretation of the model included in this work.

[6] Dr. Madeleine Bartlett and Nicole Dumont contributed equally to this work. Both contributed to the writing and codebase. The method presented was developed by Nicole Dumont, and the experiments were performed by Dr. Madeleine Bartlett. Dr. P. Michael Furlong provided feedback and edited the manuscript. Dr. Terrence C. Stewart supervised this work.

[7] Most of the work in this paper – the theoretical development of the algorithm presented, the codebase, and most experiments and writing – was done by Dr. P. Michael Furlong and Dr. Kathryn Simone. Nicole Dumont edited the manuscript, wrote Sections 2.3 and 3.5, created the spiking neural network implementation, and performed the spiking

network experiments (Figure 4, and "MCMC-SNN-HRR" in Table 1). Dr. Madeleine Bartlett provided feedback and edited the manuscript. Dr. Chris Eliasmith, Dr. Jeff Orchard, and Dr. Terrence C. Stewart all supervised this work.

All of this work was supported by CFI and OIT infrastructure funding, as well as the Canada Research Chairs program and NSERC Discovery grant 261453, and AFOSR grant FA9550-17-1-0026.

**Use of these publications in this thesis**

*Chapter 2:* Parts of this chapter were adapted form the background and methods sections of [1] and [5].

*Chapter 3:* Parts of the introduction and background are taken from [1] and [3]. The construction of HexSSPs, originally presented in [1], is included in Section 3.3, however, the experiments and figures in Section 3.3 are original to this work and the writing has been extended and modified. Section 3.4 is adapted from [3], with some modifications. Section 3.5 is adapted from Section 5 of [4] and Fig. 3.5 was taken from Section 4 of [4].

*Chapter 4:* This chapter is adapted, in an extended form, from [5].

*Chapter 5:* The LDN-TD($\theta$) learning rule, described in Section 5.3.2, was originally presented in [6]. The spiking SSP sampling model, described in Section 5.5.2, was presented in [7] and Figure 5.15 is adapted from Figure 4 of [7]. All text, experiments, figures (with the expectation of Figure 5.15), and other concepts presented in this chapter are orginal.

**Abstract**

The discovery of various spatially sensitive neurons in the hippocampal formation, such as place, grid, and boundary cells, has provided valuable insights into the neural mechanisms underlying spatial representation and navigation. However, neural activity and connectivity data alone cannot fully reveal the brain's algorithms. Bridging this gap requires computational models that not only explain the low-level activity of spatially sensitive cells but also link it to higher-level symbolic representations manipulable within a cognitive framework – models capable of binding spatial representations to discrete abstractions, while also supporting hierarchical and probabilistic structures that enable reasoning and decision-making.

The Semantic Pointer Architecture (SPA; Eliasmith, 2013), in combination with the Neural Engineering Framework (NEF; Eliasmith et al., 2003), provides a mathematical and computational framework to represent symbols and implement dynamical systems in spiking neural networks. Spatial Semantic Pointers (SSPs; Komer et al., 2019), an extension to the SPA, encode continuous variables, such as spatial locations, while supporting the binding of spatial information with other features – continuous or discrete – into compressed, multi-domain representations. This flexibility allows SSPs to model diverse cognitive processes, ranging from spatial memory to abstract reasoning, offering a unified theory for how continuous variables might be represented and manipulated in the brain.

In this thesis, we leverage these tools to model key components of spatial cognition, including path integration, cognitive map creation, and reinforcement learning. Our contributions include the development of SSP-PI, a SSP-based path integration model that combines velocity-controlled oscillators with attractor dynamics to integrate continuous spatial variables. We also introduce SSP-SLAM, a biologically inspired spiking neural SLAM system capable of constructing semantic cognitive maps that bind and associate spatial and non-spatial features. Furthermore, we propose spiking RL models that demonstrate how SSP embeddings can effectively represent successor features, reward distributions, and stochastic policies. Finally, we use the SPA and SSPs to construct state embeddings for deep RL networks, demonstrating their utility in tasks requiring mixed semantic-spatial representations.

Our findings underscore the potential of SSPs to act as a unifying framework for understanding spatial representation in the brain while advancing biologically inspired approaches to navigation and learning in artificial systems. This work bridges theoretical neuroscience and artificial intelligence, laying the groundwork for future explorations of shared principles across spatial and abstract cognition.

**Acknowledgements**

This dissertation would not have been possible without the support and encouragement of many people to whom I am deeply grateful.

First and foremost, I could not have undertaken this journey without the help of my supervisors, Dr. Chris Eliasmith and Dr. Jeff Orchard. I would like to thank Chris for all of his guidance and advice, especially at the early stages of my degree, which helped me shape my ideas and goals into concrete projects. I would like to thank Jeff for jumping in while Chris was on sabbatical, quickly getting up to speed, and offering valuable support. Chris and Jeff have given me the academic freedom to explore my interests, develop my own ideas, and trust my instincts as a researcher. Their advice and encouragement have been instrumental in helping me grow into a more confident and independent scholar. I have greatly enjoyed our many discussions over the years and will always be grateful for their guidance.

I would also like to thank my entire thesis committee: Dr. Ila Fiete, Dr. Pascal Poupart, Dr. Jesse Hoey, and Dr. Sue Ann Campbell. Thank you for taking the time out of your busy schedules to read my dissertation and provide feedback.

I am fortunate to have been part of two vibrant research groups: the Computational Neuroscience Research Group (CNRG) and the NeuroCognitive Computing Lab. All of the members of these labs have been instrumental in creating a collaborative and friendly environment. I would like to especially thank the colleagues with whom I have had the privilege of collaborating with – Michael Furlong, Terry Stewart, Madeleine Bartlett, Kathryn Simone, and Andreas Stöckel. Lastly, I would like to thank Ivana Kajić, who has provided me opportunities and advice that have helped me grow and forge meaningful connections within our field.

Finally, I would like to acknowledge the support of my friends and family, especially my parents, Randy Dumont and EJ. Thank you for being there for me, even when I was stressed, busy, or absent.

*—Nicole Dumont*

**Dedication**

*In loving memory of my brother, Aaron Dumont, whose confidence and creativity continues to inspire me every day.*

# Table of Contents

**List of Figures** **xiii**

**List of Tables** **xv**

**List of Abbreviations** **xvi**

**1 Introduction** **1**
1.1 Functions involved in navigation . . . 2
1.2 Why study navigation? . . . 5
1.2.1 Neuroscience . . . 6
1.2.2 Applications to AI . . . 6
1.3 Research landscape . . . 7
1.4 Thesis structure and contributions . . . 10

**2 Methods for Neurosymbolic Modelling** **13**
2.1 Spiking Neural Networks . . . 14
2.1.1 Neuron Models . . . 14
2.1.2 Plasticity . . . 16
2.1.3 Neural Coding . . . 17
2.2 The Neural Engineering Framework . . . 19
2.2.1 Representation . . . 20
2.2.2 Transformation . . . 23
2.2.3 Dynamics . . . 24
2.2.4 Learning rules . . . 25
2.2.5 Summary . . . 27
2.3 Vector Symbolic Algebras . . . 27
2.3.1 Representing discrete symbols and structures . . . 30
2.3.2 Representing integer values . . . 31
2.4 Spatial Semantic Pointers . . . 31
2.4.1 Visualizing SSPs with similarity maps . . . 35
2.4.2 Decoding SSPs . . . 37
2.4.3 Connection to Random Fourier Features . . . 38
2.4.4 Probability representations . . . 39
2.5 Conclusion . . . 41

**3 Grid Cells & Path Integration** **42**
3.1 Introduction . . . 43

3.2 Background . . . 44
3.2.1 Neuroscience . . . 44
3.2.2 Models . . . 47
3.3 Hexagonal SSPs . . . 49
3.3.1 Grid cells with SSPs . . . 50
3.3.2 Multi-scale grid cells . . . 51
3.3.3 Decoding SSP neural representation . . . 53
3.3.4 Other spatial cells . . . 57
3.3.5 Discussion . . . 58
3.4 Path integration . . . 61
3.4.1 SSP dynamics . . . 62
3.4.2 SSP dynamics in the Fourier domain . . . 63
3.4.3 Attractor dynamics for PI . . . 63
3.4.4 SSP-PI network . . . 64
3.4.5 Grid cells and PI . . . 66
3.4.6 Results . . . 68
3.5 Temporal integration . . . 75
3.6 Discussion . . . 76
3.6.1 Performance of SSP-PI . . . 76
3.6.2 Biological realism of SSP-PI . . . 77
3.6.3 Relation to prior PI models . . . 78
3.7 Conclusion . . . 79

**4 Simultaneous Localization & Mapping 81**
4.1 Introduction . . . 82
4.2 Background . . . 84
4.2.1 Neuroscience . . . 84
4.2.2 Models of neural mapping and SLAM . . . 85
4.3 SSP-SLAM model . . . 87
4.3.1 Localization Module . . . 88
4.3.2 Landmark Perception Module . . . 89
4.3.3 Environment Map Module . . . 89
4.3.4 Loop Closure Module . . . 91
4.4 Results . . . 91
4.4.1 Querying maps . . . 91
4.4.2 Replicating neural activity patterns . . . 94
4.4.3 Localization . . . 94
4.4.4 Neuromorphic simulation . . . 97
4.5 Discussion . . . 99
4.5.1 Prior Research . . . 99
4.5.2 Performance . . . 101
4.5.3 Limitations . . . 103

4.5.4 Future work . . . 103
4.6 Conclusion . . . 104

**5 Biologically-Constrained Reinforcement Learning 105**
5.1 Introduction . . . 106
5.2 Background . . . 107
5.2.1 Reinforcement Learning Methods . . . 107
5.2.2 Prior Work . . . 113
5.2.3 Memory . . . 116
5.3 Continuous Time Temporal Difference Learning . . . 121
5.3.1 TD(0) . . . 121
5.3.2 TD($\theta$) . . . 125
5.3.3 TD($\lambda$) . . . 126
5.3.4 Examples . . . 130
5.3.5 Discussion . . . 132
5.4 Temporal Difference Learning of Distributions . . . 135
5.4.1 Successor features . . . 136
5.4.2 Reward distributions . . . 140
5.4.3 Discussion . . . 141
5.5 Action selection . . . 144
5.5.1 Learning policies . . . 144
5.5.2 Sampling policies . . . 147
5.5.3 Examples . . . 149
5.5.4 Discussion . . . 149
5.6 Conclusion . . . 153

**6 Neurosymbolic Representations for Reinforcement Learning 155**
6.1 Introduction . . . 156
6.2 Related prior work . . . 157
6.2.1 Grid cells in RL . . . 157
6.2.2 Neurosymbolic RL . . . 158
6.3 SSPs for control . . . 159
6.3.1 Methods . . . 159
6.3.2 Results . . . 161
6.4 VSAs for structured scene representation . . . 163
6.4.1 Methods . . . 163
6.4.2 Results . . . 166
6.5 Discussion . . . 170
6.5.1 Findings . . . 170
6.5.2 Limitations . . . 170
6.5.3 Future work . . . 170
6.6 Conclusion . . . 175

**7 Conclusion** **176**
7.1 Contributions . . . 177
7.2 Key findings . . . 178
7.3 Future work . . . 181
7.4 Final remarks . . . 184

**Bibliography** **186**

**A Supplementary detail** **223**
A.1 SSP dynamics . . . 223
A.1.1 Stability analysis of attractor dynamics . . . 223
A.1.2 Higher-Order Spatial Semantic Pointer (SSP) dynamics . . . 223

**B Extended methods and experiments** **225**
B.1 Non-stationary SSPs and Recursive SSP Dynamics . . . 225
B.2 MiniGrid experiments . . . 226

**C Experiment parameters** **231**

# List of Figures


1.1 Overview of the thesis . . . 10

2.1 Encoding and decoding in the NEF . . . 20
2.2 Dynamics in the NEF . . . 24
2.3 Online learning in the NEF . . . 26
2.4 Overview of the principles of the Neural Engineering Framework (NEF) . . . 28
2.5 SSP mapping from low to high dimensional spaces via phasors . . . 32
2.6 Spatial map encoding with SSPs . . . 33
2.7 SSPs as the collective pattern of a set of phasors . . . 34
2.8 SSP similarity maps . . . 36
2.9 Probability density estimation with a bundle of SSPs . . . 39

3.1 The hippocampal formation and spatially sensitive neuron types. . . . 45
3.2 SSP Kernel Functions . . . 51
3.3 Grid cell firing patterns . . . 53
3.4 Decoding error for different representations. . . . 55
3.5 Firing patterns of LIF neurons representing SSPs. . . . 57
3.6 Example correlation matrices and their top eigenvectors . . . 60
3.7 Rotation of SSP Fourier components representing a dynamic variable . . . 62
3.8 VCO dynamics with and without a limit cycle attractor . . . 63
3.9 The SSP path integration model . . . 65
3.10 SSP path integrator errors over band-limited white noise trajectories . . . 67
3.11 Errors in the kernel evolution of the path integrator . . . 69
3.12 Multiple hypothesis PI with SSPs . . . 70
3.13 SSP path integrator parameter sweep . . . 71
3.14 SSP path integrator errors over random walks . . . 72
3.15 Path integration results using band cells versus grid cells . . . 73
3.16 Firing patterns of neurons in SSP-PI. . . . 73
3.17 Rate maps and grid scores of neurons in SSP-PI and real neurons. . . . 74
3.18 Trajectory representation with SSPs . . . 76

4.1 A diagram illustrating simultaneous localization and mapping (SLAM) . . . 83
4.2 Anatomy of the medial temporal lobe . . . 85
4.3 SSP-simultaneous localization and mapping (SLAM) model . . . 87
4.4 Diagram of the environment mapping module of SSP-SLAM . . . 90
4.5 Diagram of the loop closure module of SSP-SLAM . . . 91
4.6 Querying for landmarks with SSP-SLAM . . . 92
4.7 Querying for vectors to landmarks and querying over areas with SSP-SLAM . . . 93

4.8 Firing patterns of neurons in the SSP-SLAM network . . . 95
4.9 Error in SSP-SLAM versus the path integration model . . . 96
4.10 Path estimates from SSP-SLAM . . . 98
4.11 SSP-SLAM run on the NenogLoihi emulator . . . 99

5.1 A Markov decision process . . . 108
5.2 State space model view . . . 118
5.3 The shifted Legendre polynomials . . . 120
5.4 Diagram of value network that uses LDN memories for TD learning . . . 122
5.5 Learning via replay phases with an LDN . . . 123
5.6 Learning a value function with replay phases . . . 124
5.7 Learning continuously with an LDN . . . 125
5.8 Learned value functions for T-maze task . . . 131
5.9 Learned value functions for trace conditioning task . . . 133
5.10 Diagram of a successor feature network that uses LDN memories for TD learning. 137
5.11 Successor features in a hairpin maze trained with LDN-TD($\lambda$) . . . 138
5.12 Successor features in two connected rooms trained with LDN-TD($\lambda$) . . . 139
5.13 Distributional reward embedding . . . 141
5.14 Diagram of the actor network . . . 145
5.15 Output of the Langevin dynamics model of SSP bundle sampling. . . . 148
5.16 10-arm bandit problem . . . 150
5.17 Water maze problem . . . 151
5.18 Mixed discrete-continuous policy and action representations using SSPs . . . 152

6.1 Actor-critic network with SSP feature extractor layer . . . 160
6.2 Sample efficiency curves plotting the IQM of return on the control benchmarks . . 161
6.3 Probability of improvement on the control benchmarks . . . 162
6.4 MiniGrid environments . . . 163
6.5 MiniGrid feature extraction methods . . . 165
6.6 Return IQM on MiniGrid and BabyAI tasks . . . 168
6.7 Probability of improvement and performance profiles for the MiniGrid tasks . . . 169
6.8 Return IQM on MiniGrid tasks using HexSSP versus RandSSP initialization . . . 169
6.9 Successor feature network and feature learning methods . . . 172
6.10 Results using SSPs as the features in a AC successor feature model . . . 174

B.1 Return IQM on MiniGrid and BabyAI tasks with lower-dimensional embeddings . 228
B.2 Return IQM with and without a trainable phase matrix on two MiniGrid tasks . . 229
B.3 Results from HRR view embeddings, with and without pose information . . . 229
B.4 Return IQM on MiniGrid and BabyAI tasks with Tanh activation functions . . . 230

# List of Tables


3.1 Spatial cell types in the NEF . . . . . 58

4.1 Localization errors for SSP-SLAM and SSP-PI in 2D and 3D environments. . . . . 97
4.2 Comparison of bio-inspired SLAM models . . . . . 102

C.1 Default spiking neural network parameters . . . . . 231
C.3 Hyperparameters for the SSP-PI accuracy results . . . . . 231
C.2 Hyperparameters for the SSP decoding error results . . . . . 232
C.4 Hyperparameters for the SSP-PI error versus model parameters experiments . . . 232
C.5 Hyperparameters for the SSP-SLAM model . . . . . 233
C.6 Hyperparameters for the Loihi SSP-SLAM model . . . . . 233
C.7 Hyperparameters for the mapping example with SSP-SLAM . . . . . 234
C.8 Hyperparameters for value learning in a T-maze . . . . . 235
C.9 Hyperparameters for value learning during trace conditioning . . . . . 235
C.10 Hyperparameters for successor feature learning . . . . . 236
C.11 Hyperparameters for successor feature learning . . . . . 236
C.12 Hyperparameters for SSP sampling example . . . . . 236
C.13 Hyperparameters for bandit example . . . . . 237
C.14 Hyperparameters for water maze example . . . . . 237
C.15 Hyperparameters for control tasks . . . . . 238
C.16 Hyperparameters for the MiniGrid tasks . . . . . 238

# List of Abbreviations

**A2C** Advantage Actor-Critic.
**AC** Actor-Critic.
**AI** artificial intelligence.
**ANN** artificial neural network.

**BCM** Bienenstock-Cooper-Munro.

**CANN** continuous attractor neural network.
**CNN** convolutional neural network.

**DFT** discrete Fourier transform.
**DQN** Deep Q-Network.

**EC** entorhinal cortex.

**FFT** fast Fourier transform.
**FHRR** Fourier Holographic Reduced Representation.

**GAE** Generalized Advantage Estimation.

**HRL** hierarchical reinforcement learning.
**HRR** Holographic Reduced Representation.

**IQM** interquartile mean.

**KDE** kernel density estimation.
**KME** kernel mean embedding.

**LDN** Legendre Delay Network.
**LEC** lateral entorhinal cortex.
**LIF** leaky integrate-and-fire.
**LSTM** Long Short-Term Memory.
**LTI** linear time invariant.

**MDP** Markov decision process.
**MEC** medial entorhinal cortex.
**ML** machine learning.
**MLE** maximum likelihood estimate.
**MMD** maximum mean discrepancy.

**NEF** Neural Engineering Framework.

**OI** oscillator-interference.
**OVC** object vector cell.

**PCA** principal component analysis.
**PDF** probability density function.
**PES** prescribed error sensitivity.
**PI** path integration.
**PPO** Proximal Policy Optimization.

**RFF** Random Fourier Feature.
**RKHS** reproducing kernel Hilbert space.
**RL** reinforcement learning.
**RMSE** root mean square error.
**RNN** recurrent neural network.
**rSSP** Recursive Spatial Semantic Pointer.

**SAC** Soft Actor-Critic.
**SF** successor feature.
**SLAM** simultaneous localization and mapping.
**SNN** spiking neural network.
**SP** Semantic Pointer.
**SPA** Semantic Pointer Architecture.
**SR** successor representation.
**SSM** state space model.

**SSP** Spatial Semantic Pointer.

**STDP** spike-timing-dependent plasticity.

**TD** temporal difference.

**VCO** velocity controlled oscillator.

**VSA** Vector Symbolic Algebra.

**WTA** winner-take-all.

# Chapter 1
# Introduction

> Space is not only a critical sense but a fascinating one because unlike other senses space is not analyzed by a specialized sensory organ. How, then, is space represented in the brain? Because we do not have a sensory organ dedicated to space, the representation of space is a quintessentially cognitive sensibility: it is the binding problem writ large. The brain must combine inputs from several different sensory modalities and then generate a complete internal representation that does not depend exclusively on any one input.
>
> — Eric R. Kandel
> *In Search of Memory: The Emergence of a New Science of Mind*

Navigation is an essential cognitive function, relying on the brain's ability to represent space and time. Furthermore, the challenges of navigation highlight the brain's ability to adapt such representations quickly, and use them as a scaffolding for episodic memory – the type of memory critical to an agent's identity. Throughout this thesis, we touch on each of these themes in an effort to develop new, general methods for enhancing our understanding of cognition. To do so, this thesis explores three core questions: How does the brain encode spatial information? How does this encoding support navigation and related cognitive tasks? And, more practically, how can these mechanisms inspire computational models in artificial intelligence (AI) and machine learning (ML)? We explore these questions using biologically plausible computational models that reflect hypothesized neural processes. Building on prior work, we use neurosymbolic modelling methods – in particular, Vector Symbolic Algebras (VSAs) – to construct feature representations that reproduce neural activity observed in specific brain areas during navigation while also enabling symbolic manipulation of semantic representations of objects.

This chapter presents our motivation for studying spatial navigation, as well as our central hypothesis: The latent structures encoded by neural populations representing spatial information can be interpreted through a VSA framework, which includes an algebra for continuous and discrete symbolic manipulation. This framework, grounded in the dynamics of these neural populations, addresses critical gaps in understanding how the brain represents and manipulates spatial information. Chapters 1 and 2 lay the groundwork of this thesis, with Chapter 2 introducing the main modelling methods. Building on this groundwork, Chapters 3–5 detail spiking neural models of key navigation-related functions: path integration, cognitive mapping, simultaneous localization and mapping (SLAM), and biologically plausible RL. These models,

grounded in neuroscience and deployable on neuromorphic hardware, collectively demonstrate a novel and effective approach to building biologically inspired navigation systems.

## 1.1 Functions involved in navigation

For all intelligent biological entities, navigation is critical for survival, playing a central role in tasks such as locating food, finding shelter, and migration. Consider an animal navigating to a new food source and returning home. This process involves:

- exploring an environment to locate the new food source;
- maintaining an estimate of self-location in unfamiliar areas;
- storing the location of the food source within a broader environmental context for future retrieval;
- encoding details about the source (*e.g.*, its reward value) to guide future actions;
- using memory of the home location to plan a (possibly novel) path to home; and
- navigating home, potentially switching to a metric map-based strategy or route-following as familiar landmarks or the destination come into view.

This seemingly simple scenario engages multiple functions: spatial representation, dynamic updating of self-location (*e.g.*, spatial updating or path integration), mapping (a type of memory), and learning. These overlapping functions and their neural underpinnings, are briefly expanded on in the following sections. For more detailed explanations and a more comprehensive overview of the relevant neural anatomy, please refer to the background sections in Chapters 3 and 4.

**Spatial Representation**. To navigate, an animal must represent variables such as its position within the environment. In the brain, such spatial representation relies on specialized classes of neurons found in the hippocampal formation and surrounding regions, which encode an animal's position and orientation in both egocentric (self-centred) and allocentric (environment-centred) frames of reference. For example, *place cells* in the hippocampus activate when an animal occupies a specific location (O'Keefe et al., 1971), while *grid cells* in the medial entorhinal cortex (MEC) generate a hexagonal pattern of firing fields that tile the environment (Hafting et al., 2005). Together, these neurons form a neural basis for the representation of space, which is essential for navigation and memory.

Many types of specialized neuron for spatial representation are found in the hippocampus. The hippocampus is a three-layer structure, with surrounding areas including the perirhinal, parahippocampal, entorhinal, and retrosplenial cortices. The hippocampus connects to other

regions primarily via parahippocampal areas to sensory and association areas of the neocortex, as well as through the fornix to subcortical structures. While this thesis focuses on mammalian brains, structures in birds, reptiles, amphibians, and some fish (*e.g.*, the medial pallium) are considered homologous to the mammalian hippocampus, and perform similar functional roles in spatial cognition (Striedter, 2016; Bingman et al., 2003; Rodrıguez et al., 2002; Salas et al., 2003).

Interestingly, the brain appears to use similar coding principles as those found in hippocampal formation to represent other continuous variables that are not spatial. non-spatial. Experiments suggest that, in some contexts, time (Peters-Founshtein et al., 2024), odour (Bao et al., 2019), social space (Park et al., 2021), and other abstract quantities (Constantinescu et al., 2016) are encoded through neural activity patterns exhibiting periodic firing, akin to grid cells in spatial navigation. This suggests that spatial coding principles may be general purpose, and that there are shared mechanisms for representing, and potentially updating, continuous variables in the brain.

**Spatial updating**. Spatial updating is the process of tracking one's position relative to the surrounding environment over time. It involves the continual updating of the brain's internal representation of location as one moves through space, and is used in both vector-based and map-based navigational strategies. For example, maintaining homing vectors to important locations (*e.g.*, shelter or food) requires the brain to update these vectors using self-motion information. Similarly, building an internal map of the environment relies on tracking one's position relative to landmarks.

Path integration (PI), also known as dead-reckoning, is a type of spatial updating in which self-position estimates are derived solely from self-motion cues. Animals, including rodents (Mittelstaedt et al., 1982; Etienne, 1987; Benhamou, 1997), bats (Aharon et al., 2017), and humans (Mittelstaedt et al., 2001), can perform path integration while navigating environments in the absence of visual cues. This computation is often attributed to the MEC (McNaughton et al., 2006; Gil et al., 2018; Fukawa et al., 2020) and is critical for maintaining spatial orientation over time.

**Mapping**. Spatial updating plays a role in more complex navigational processes, such as *cognitive mapping*. Tolman (1948) first proposed that animals construct "cognitive maps," mental allocentric models of the environment that encode spatial relationships between landmarks and other features. Early maze studies with rodents demonstrated that they could find shortcuts or novel routes to a goal, indicating the use of cognitive maps rather than basic stimulus-response learning (Tolman, 1948). Cognitive mapping enables the brain to generalize across environments and make spatial inferences beyond immediate sensory input.

PI is integral to many cognitive mapping strategies, and mapping, in turn, is essential for accurate spatial updating – and spatial cognition more generally (Burgess et al., 2002). While PI relies on self-motion cues, animals also integrate external sensory information, such as visual landmarks and odour trails, to correct the drift that naturally accumulates in PI computations.

By combining multi-sensory inputs with PI, animals form cognitive spatial maps – binding spatial and non-spatial features into a coherent representation. This computation resembles simultaneous localization and mapping (SLAM) algorithms in robotics, which involve path integration while simultaneously constructing environment maps for error correction.

The cognitive map theory (O'Keefe et al., 1978) positions the hippocampus as the primary site of cognitive map formation. Supporting this hypothesis is the existence of place cells and evidence of spatial memory deficits in individuals with hippocampal damage (O'Keefe et al., 1975; Girardeau et al., 2009). The formation of these maps likely involves both associative learning, where synchronous activation of neural assemblies binds different features into an associative code (El-Gaby et al., 2021), and predictive learning, wherein temporal patterns in neural populations encode relational structures between features (Liu et al., 2023). Beyond spatial representations, the hippocampus is also thought to support non-spatial cognitive maps, encoding associations between diverse continuous and discrete features (Tavares et al., 2015; Aronov et al., 2017; Courellis et al., 2024).

**Memory**. Memory supports navigation by enabling the retention and recall of spatial information, such as environment maps, routes, and landmarks. Episodic and spatial memory thus support context-dependent decision-making and adaptive strategies based on environmental familiarity. The hippocampus is thought to play an important role in the formation of declarative memories – those that can be consciously recalled – including semantic memories (facts and concepts) and episodic memories (Eichenbaum, 1992; Andersen, 2007). Damage to the hippocampus can result in anterograde amnesia – new memories cannot be formed, though older memories often remain intact (Parkin, 1996; Squire et al., 1998). Thus, as part of a broader memory system, the hippocampus is believed to store sequences of events and places in the short to medium term, with these memories later consolidated in the neocortex over time (Eichenbaum, 2000; Vargha-Khadem et al., 1997; Squire et al., 1998; Rempel-Clower et al., 1996). Additonally, rather than storing the detailed sensory information of experiences, the hippocampus is hypothesized to function as an 'index' or 'pointer' system, encoding references to sensory and contextual details that are distributed across other areas of the brain (Teyler et al., 1986; Teyler et al., 2007).

The memory theory of hippocampal function does not contradict the cognitive map theory, as spatial maps can be viewed as a specific type of memory – information embedded within a spatial, rather than spatial-temporal, framework. More generally, the relational theory of hippocampal function posits that the hippocampus encodes information within broader relational frameworks, not limited to spatial or temporal domains (Cohen, 1993; Eichenbaum, 2001; Monti et al., 2014; Olsen et al., 2012). These frameworks go beyond simple stimulus-response associations, capturing more nuanced relationships such as distances between locations, sequences or timing of events, hierarchical structures, and relative salience or importance between features.

**Learning**. Mapping and memory are products of learning processes. Navigation involves various forms of learning, including associative learning and reinforcement learning (RL), which

involve understanding an environment's transition structure for planning and associating states and actions with outcomes. Associative learning and RL are not confined to a single brain region; they are general mechanisms used and supported by various areas. For example, RL is facilitated by regions such as the striatum, amygdala, basal ganglia, prefrontal cortex, and hippocampus (Pennartz et al., 2011). The hippocampus plays a multifaceted role in RL: cognitive maps aid in planning, episodic memory contributes to RL through recall and replay, and relational learning enables generalization across related experiences (*e.g.*, though transitive inference).

**Planning**. Planning in navigation involves predicting future states and making decisions based on internal representations of the environment. This process integrates spatial memory, cognitive maps, and current sensory input to simulate potential routes, evaluate their costs, and optimize actions to achieve specific goals. The prefrontal cortex, in collaboration with hippocampal and parietal networks, plays a central role in constructing these simulations (Calton et al., 2009; Javadi et al., 2017; Javadi et al., 2017; Patai et al., 2021). The hippocampus contributes to planning in navigation by encoding maps and experiences, which can be used for replay and route simulation (Ólafsdóttir et al., 2018).

The features we have discussed – spatial representation, updating, mapping, memory, learning, and planning – are all engaged to varying degrees in the earlier example of a critter locating food and returning home. Each function contributes to solving specific challenges: maintaining an internal sense of location, linking destinations to their broader context, and finding efficient paths. While navigation is a natural context to study these processes, their utility is broader. The same principles can be applied to abstract domains, such as reasoning about time, social interactions, or other continuous features. Thus, studying navigation is a means of uncovering the fundamental mechanisms by which the brain infers, encodes, and manipulates information about the world.

## 1.2 Why study navigation?

The ability to create internal maps by binding features to structured representations has applications beyond navigation. Many forms of knowledge representation in machine learning and AI – such as graphs, trees, lists, dictionaries, and metric spaces – can be described in precisely this form. Additionally, dynamic feature encoding and temporal updating are relevant to both biological and artificial systems. Consequently, while we focus on navigation in animals in this thesis, the computational principles and models we explore have broader applicability. In neuroscience, studying navigation reveals general principles of cognition that extend to non-spatial domains. In AI and robotics, navigation is fundamental for mobile autonomy, offering inspiration for algorithms that integrate efficient encoding, updating, and planning.

### 1.2.1 Neuroscience

Although much has been uncovered about the sensory cortices – their environmental responses, tuning curves, and how their activity relates to different features in specific modalities – the higher-level associative cortices remain far more enigmatic. Unlike sensory cortices, these areas process complex, abstract, and internally generated information. There may not be direct external reference points to guide interpretations of their neural activity, and the information they encode is often highly multiplexed, integrating input from multiple sources.

As argued in Rowland et al. (2016), cortices involved in spatial cognition, such as the hippocampus and entorhinal cortex (EC), stand out as an exception. While spatial representations are internally generated and synthesized from sensory inputs, they correspond to an external, observable phenomenon: the position and movement of an animal in physical space. This grounding in an external reference makes spatial representations more accessible for study. Moreover, a wide range of species are capable of navigation and spatial cognition, likely reflecting an evolutionarily ancient brain function (Salas et al., 2003; Murray et al., 2017).

There is growing evidence that the neural mechanisms behind spatial representation and navigation may be general mechanisms for encoding other continuous features, including more abstract and conceptual ones, potentially as a result of evolutionary exaptation (Aronov et al., 2017; Constantinescu et al., 2016; Buzsáki et al., 2023). Again, consider the computations associated with navigating to and from a food source described earlier (Section 1.1). It is interesting to note, as an example, how those same functions map to the seemingly very different task of social reasoning. Maintaining an estimate of one's position in a social group over the course of social interactions is analogous to maintaining an estimate of self-location (Liang et al., 2024). While evaluating one's social standing relative to particular individuals along multiple axes is, in some ways, analogous to the computation of a vector to home or a landmark (Parkinson et al., 2014). Or perhaps a cognitive map representing a social landscape could used for decision making (Park et al., 2021). Thus, research into spatial cognition opens the door to understanding the brain's basic computational strategies that underlie many aspects of thought and behaviour. Computational modelling plays an important role in this endeavour by enabling us to reverse-engineer, instantiate, and thoroughly test these algorithms.

### 1.2.2 Applications to AI

Collaborative efforts between neuroscience and AI have driven significant advances in understanding and creating intelligent systems. Artificial neural networks, for instance, originated from the collaboration between a neurophysiologist and a logician (McCulloch et al., 1943), while RL emerged from combining insights from animal learning psychology and optimal control theory (Sutton et al., 2018). Despite this history of synergy, there is still much that AI can learn from biological systems.

Mobile robots and autonomous vehicles, for example, must process spatial information and navigate complex, dynamic environments. While animals excel at these tasks – adapting

flexibly to environmental changes and transferring knowledge to novel contexts – creating truly autonomous agents with comparable capabilities remains a challenge. Additionally, biological systems do all of this with remarkable energy efficiency. This has inspired research into neuromorphic hardware, platforms that aim to mimic the brain's architecture, leveraging sparse activity and event-based computation to achieve efficient processing. Thus, the study of biological navigation could inform advancements in autonomous systems, particularly in navigation and decision-making in power-constrained contexts.

## 1.3 Research landscape

In this section, we briefly outline the landscape of research related to spatial navigation. Detailed literature reviews are provided on a topic-by-topic basis in subsequent chapters. Our purpose here is to identify gaps in past work and situate our contributions at a high level. In particular, we posit that modelling navigation requires a unified framework for integrating discrete and continuous data that includes dynamics, uncertainty, and methods for association and composition.

**Models of localization and spatial updating**. Spatially sensitive neurons in the hippocampal formation, such as place cells and grid cells, have been extensively studied and modelled. Research in this area began with ring attractors—recurrent neural networks designed to sustain activity bumps — originally developed to model the head direction system (Zhang, 1996). The concept of ring attractors was later extended to plane attractors for modelling path integration in two-dimensional spaces, incorporating place cells (Samsonovich et al., 1997; Conklin et al., 2005). By introducing specific periodic boundary conditions, grid cell-like representations emerge in these models (Conklin et al., 2005; Guanella et al., 2007; Burak et al., 2009). Note that grid cells can also arise in other types of PI models that utilize oscillatory dynamics, either as a complement to or independent of attractor mechanisms. Models of path integration have also been combined with associative learning mechanisms, enabling corrections via cognitive maps (Samsonovich et al., 1997; Hardcastle et al., 2015).

The activity patterns of grid cell populations occupy a latent representational space characterized by toroidal geometry (Gardner et al., 2022). This geometry has been an active area of research, along with the efficient coding properties of grid cells, which exhibit redundancy and robustness, enabling precise spatial representation (Sreenivasan et al., 2011; Mathis et al., 2015; Vágó et al., 2018). However, the relationship between these continuous representational spaces and symbolic reasoning remains underexplored. Specifically, many models cannot bind continuous spatial variables (*e.g.*, metric coordinates) to discrete, symbolic information (*e.g.*, landmarks, objects, or contextual labels). It is also unclear how to construct hierarchical or structured representations that integrate fine-grained continuous metrics with coarse-grained symbolic abstractions. Secondly, PI models do not often incorporate probabilistic interpretations of spatial and non-spatial variables, limiting their ability to represent uncertainty or probability distributions over space. Such capabilities are essential for connecting neural

navigation models to robotics methods like SLAM.

There are promising but partial exceptions that attempt to address some of these gaps. For example, George et al. (2021) introduced graph-based cognitive maps for spatial navigation or navigation over general, even discrete spaces. However, these graphs discretize space and do not capture neural dynamics. Another example, from Whittington et al. (2020), generalizes across spatial and non-spatial tasks, but its framework lacks uncertainty or a principled algebra for compositionality and interaction between representations.

In this thesis, we extend past work by presenting a neurosymbolic modelling framework to represent continuous variables probabilistically. We incorporate dynamics to model path integration in Chapter 3 and explore learning cognitive maps that bind features into structured representations. These approaches are applied to SLAM in Chapter 4, bridging neural navigation frameworks with probabilistic robotics methods.

**Reinforcement learning (RL).** RL is a computational framework in which an agent learns to make decisions through trial and error, receiving rewards or penalties based on its actions. This framework is highly relevant to biological navigation, where animals may navigate toward goals or avoid harmful stimuli, learning from past experiences.

In the brain, the fronto-striatal-basal ganglia circuit plays a key role in motor control and action selection. It is particularly important for stimulus-response learning, where actions are associated with outcomes, and for reward prediction, where expected future rewards influence current decisions (Chakravarthy et al., 2010; Humphries, 2022). While decision-making in navigation involves a distributed network of brain areas – including the hippocampus and EC for spatial memory and mapping, the retrosplenial and posterior parietal cortices for reference frame transformations (Mitchell et al., 2018; Calton et al., 2009), and the prefrontal cortex for planning (Patai et al., 2021)[1] – the fronto-striatal-basal ganglia circuit is particularly associated with reinforcement learning.

This circuit is often linked to model-free learning, a form of RL where the animal learns action policies directly from experience, rather than constructing an internal model of the environment. RL-based models have been instrumental in explaining how animals use reward signals to guide navigation decisions. By modelling RL mechanisms, researchers have gained deeper insights into neuromodulatory signals and their role in navigation-related behaviour (Schultz et al., 1997; Stoianov et al., 2018; Neftci et al., 2019).

Significant progress has been made in developing biologically plausible RL models—for example, models based on local learning rules, spiking neural networks, and continuous-time dynamics (Izhikevich, 2007; Potjans et al., 2009; Rasmussen et al., 2017; Yan et al., 2019). Such models have been used to simulate path-finding and goal-finding behaviour (Niv, 2009; Chalmers et al., 2016; Ashwood et al., 2022), and grid-cell-like representations have been

[1]This list of areas involved in navigation and decision-making is incomplete and the descriptions of the roles of these areas are quite simplified. In reality, whole chapters could be written about each of these areas' involvement in navigation.

incorporated into RL systems to represent spatial states in a biologically inspired manner (Banino et al., 2018; Dang et al., 2021).

Despite these advances, key gaps remain. Many biologically plausible models neglect neural episodic memory systems for learning from past experiences. Additionally, there is limited exploration of how grid-cell-like representations could integrate uncertainty, a natural fit with RL's probabilistic framework. Furthermore, while some research has extended grid cell-like representations to encode non-spatial variables (Yu et al., 2021; Bartlett et al., 2023b), these efforts have been restricted to simple toy problems, leaving their broader applicability underexplored.

The work in this thesis directly addresses these challenges. In Chapter 5, we use a neural model of episodic memory, implemented via a state space model (SSM), to learn in the model-free setting and leverage grid-cell-like representations to encode probability distributions in spiking RL systems. In Chapter 6, we apply grid-cell state encoding to more difficult, high-dimensional non-spatial problems and develop a compositional state embedding that combines grid-cell-like representations with discrete landmark features for navigation. By bridging these gaps, this research aims to advance our understanding of how neural systems support learning, memory, and decision making in both spatial and abstract domains.

**Neurosymbolic modelling**. Neurosymbolic AI integrates symbolic representations and reasoning with neural processing. This area encompasses a wide range of techniques, many of which are not concerned with biological plausibility and are rarely used in computational neuroscience. For example, neurosymbolic AI might involve symbolic reasoning engines that work alongside deep artificial neural networks (Bhuyan et al., 2024).

In contrast, Vector Symbolic Algebras (VSAs) are a neurosymbolic modelling approach that originated in cognitive science to address the challenge of integrating symbolic representation and manipulation within connectionist models (Gayler, 2004). In VSAs, concepts and symbols are represented by high-dimensional vectors, and symbolic operations are performed through vector operations. A distinctive feature of VSAs is their ability to support "computation in superposition," where multiple concepts are encoded simultaneously in a distributed manner. Additionally, VSAs offer a solution to the binding problem, allowing compositional and structured symbolic information – such as sequences, relations, and sets – to be represented and manipulated. These operations have been applied to model many cognitive functions and have even been integrated with Large Language Models for question-answering (Laube et al., 2024) and mathematical reasoning (Dhanraj et al., 2025). Furthermore, by performing symbolic manipulation and reasoning within a vector space using distributed, high-dimensional representations, VSAs mirror key principles of neural computation, aligning them closely with biologically inspired models.

Specifically, the Semantic Pointer Architecture (SPA) demonstrates how such representations and operations can be performed in spiking neural networks (Eliasmith, 2013, Section 7). Continuous variables such as location and time can be represented in the SPA using Spatial

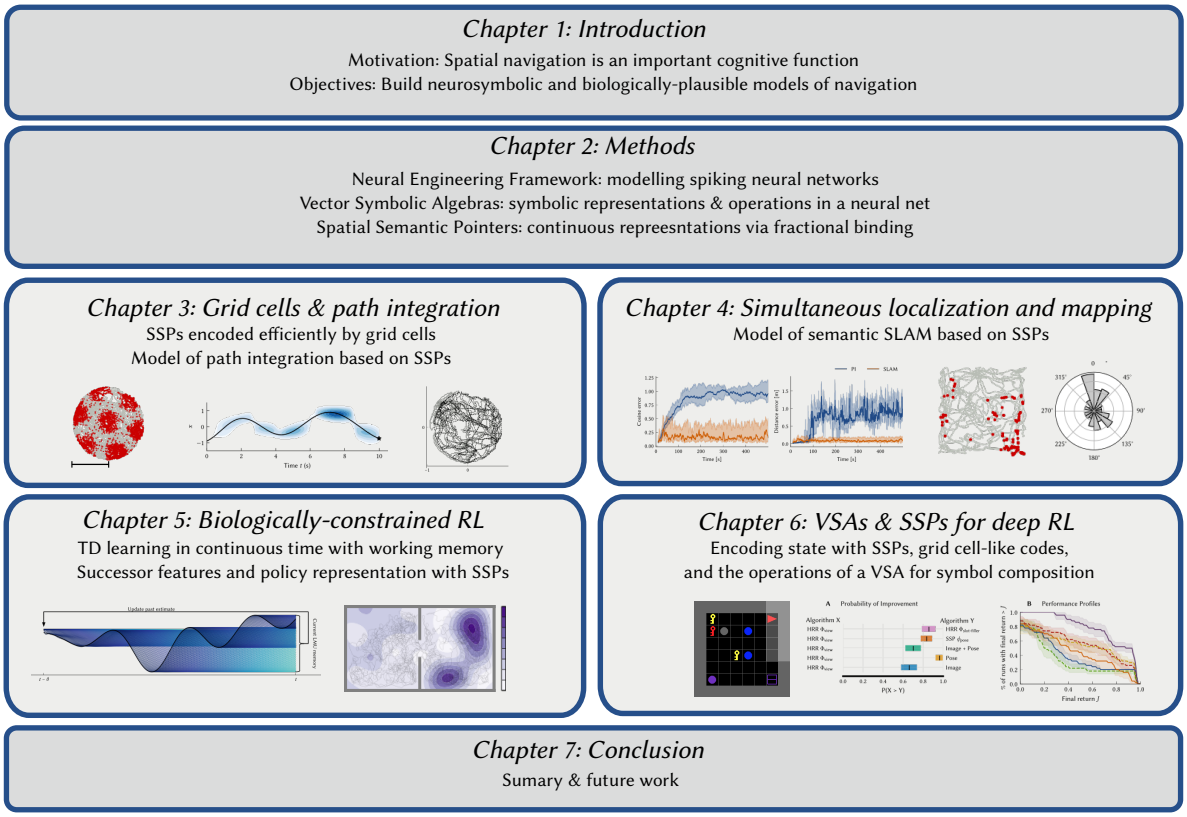



**Figure 1.1:** An overview of the thesis.

Semantic Pointers (SSPs; see Section 2.4), which are closely related to Random Fourier Features and kernel methods (see Section 2.4.3). Spatial Semantic Pointers (SSPs) and the Semantic Pointer Architecture (SPA) provide a framework for combining spatial and temporal encodings with symbolic or discrete representations in a biologically plausible way. Although these methods have not yet been widely applied to model computations in the hippocampal formation and EC, they are particularly well-suited for this. These methods offer new insights into the distributed representations used in spatial navigation and the computations they enable. This thesis employs these techniques extensively, highlighting their utility in bridging symbolic and neural representations in biologically inspired systems.

## 1.4 Thesis structure and contributions

As shown in Figure 1.1, the structure of this thesis consists of this introduction, which outlines the motivation and objectives, followed by a methods and background chapter, four chapters that present different models and methods related to spatial navigation, and finally a concluding summary chapter. Next we briefly summarize the key purpose and contributions of each of these chapters in turn.

Chapter 2 introduces methods for biologically plausible neurosymbolic modelling. It provides an overview of the foundational methods and tools employed throughout the thesis, including the Semantic Pointer Architecture (SPA) and the Neural Engineering Framework (NEF). This chapter provides a review of computational neuroscience methods, focusing on how spiking neural populations are modelled using the NEF and how population coding can be interpreted as representing latent variables. Additionally, it covers local learning rules and the control-theoretic analysis of recurrent networks. Vector Symbolic Algebras (VSAs) are also described, with particular attention given to Spatial Semantic Pointers (SSPs) for continuous representations and their relationship to kernel methods, thereby offering a probabilistic interpretation of VSA operations. The methods introduced in this chapter form the basis for all subsequent work, while methods specific to individual chapters are presented later as needed.

Chapter 3 focuses on spatial representation and path integration using SSPs. This chapter demonstrates how SSPs serve as an effective encoding scheme with low decoding error and shows how SSPs with particular parameterizations can model grid cells. It also explains how SSPs can model arbitrary spatial tuning curves. The chapter derives the dynamics of SSPs and introduces a spiking model of path integration, SSP-PI, which shares key features with existing path integration models. Interestingly we find this model derived from a VSA shares key features with existing PI models while offering the unique interpretation of the PI output as an SSP, allowing the output to be incorporated into compositional structures, a topic explored further in Chapter 4.

Chapter 4 covers cognitive mapping and SLAM. The output from the path integration model can be used to build a cognitive map – represented as a heteroassociative memory network that associates spatial and non-spatial information. This cognitive map can correct drift in the path integration model. This chapter introduces a novel SLAM model, the first biologically inspired spiking semantic SLAM system we are aware of. Additionally, we demonstrate that SSP-SLAM is amenable to neuromorphic implementation by testing it on a Loihi chip emulator.

Chapter 5 explores biologically plausible model-free RL and episodic memory, focusing on decision-making in spatial contexts. While previous chapters addressed the dynamics of spatial representations and map building, this chapter deals with how an agent can learn to act. It introduces methods for temporal-difference learning in spiking neural networks that use a time-cell model of episodic memory to compute errors over time and learn value functions and successor features based on SSP state encodings. The probabilistic interpretation of SSPs also facilitates policy learning and sampling, both of which are discussed here.

Chapter 6 shifts focus from biological considerations to the application of SSPs and VSA features in deep RL. While the previous chapter centred on spiking neural networks and local learning rules, this chapter investigates neural and cognitively inspired representations in deep RL models. We show that SSPs provide an effective state representation in both spatial and non-spatial tasks. Moreover, cognitive maps and mixed continuous-symbolic representations are shown to be effective tools within deep RL models. This chapter highlights the utility of neuroscience- and cognitive science-inspired representations in AI.

Finally, a conclusion summarizing key findings and directions of future work is provided in Chapter 7. Overall, this thesis addresses key topics in spatial navigation and biologically plausible reinforcement learning. We explore how biologically inspired models of neural populations can represent spatial and cognitive processes, integrating symbolic reasoning with neural computation. Our main contributions include:

- the development of a biologically plausible path integration model based on SSPs;
- the introduction of a spiking semantic SLAM system for cognitive mapping using SSPs;
- proposing methods for temporal-difference learning in spiking neural networks with an Legendre Delay Network (LDN) working memory; and
- the application of spatial-semantic representations in deep reinforcement learning tasks.

# Chapter 2

# Methods for Neurosymbolic Modelling

> The brain, with its billions of cells connected by synapses, is the ultimate example of a complex system that cannot be understood through the study of individual components alone. In order to unveil the neural basis of complex behaviours and functions, such as perception, movement, cognition, memory, and emotion, we must acknowledge and catalog the interactions between the neurons, allowing us to integrate multiple levels of observations and apply diverse approaches, including computational and mathematical modelling.
>
> — Dániel L Barabási, et al.
> *Neuroscience Needs Network Science*

Before narrowing our focus to spatial representations and navigation in the brain, we take a step back and review the basics of neural signalling and computation. This review begins with single-neuron models and ends with a neurosymbolic modelling framework for building large-scale cognitive models with neural networks. A key concept we discuss is how a neural population's collective activity effectively encodes lower-dimensional latent representations. Additionally, we explore how neural connections between and within populations facilitate transformations and computations within this latent vector space.

A significant portion of this chapter focuses on the Neural Engineering Framework (NEF), a set of principles for modelling neural circuits using latent spaces. Following this, we discuss the Semantic Pointer Architecture (SPA), which, among other things, integrates the NEF with a Vector Symbolic Algebra (VSA) – a set of mathematical operations to describe symbolic manipulation and composition in a vector space. The SPA is an architecture for biologically plausible' cognitive modelling that links higher-level, symbolic descriptions of cognitive systems to a low-level description of neural activity and connectivity. The SPA is the basis of Spaun (Eliasmith et al., 2012; Choo, 2018), the largest functioning brain model to date. The computational methods of the SPA are the foundation of all work presented in this thesis. Lastly, we examine how continuous variables are represented within the SPA with Spatial Semantic Pointers (SSPs). These representations are crucial components of the navigation models presented in subsequent chapters of this thesis.

## 2.1 Spiking Neural Networks

### *2.1.1 Neuron Models*

First, we review the basic units of a neural network – neurons – and how we might characterize their behaviour. When a neuron receives an injected current $J(t) > 0$ (*e.g.*, via a microelectrode inserted in its soma or from dendritic currents), its membrane potential rises. In the absence of input (*i.e.*, $J(t) = 0$), this membrane potential decays back to a resting value, $V_{\text{rest}}$. If the membrane potential reaches some threshold value, $V_{\text{thresh}}$, the neuron generates an *action potential*, also known as a *spike*. During a spike, the membrane potential suddenly rises and then falls below the resting potential, making the neuron hyperpolarized. The short hyperpolarization time is known as the *refractory period*.

Due to a cascade of chemical differentials, spikes travel down the neuron's axon to the axon terminals. The arrival of the spike in the terminals triggers the release of neurotransmitters at chemical synapses. The neurotransmitters bind to receptors on the dendrites of postsynaptic neurons. Depending on the type of neurotransmitter, this may excite the postsynaptic neuron (*i.e.*, induce a positive current), inhibit the postsynaptic neuron (*i.e.*, induce a negative current), or have some other modulatory effect. The strength of such effects varies amongst synapses – due to the amount of neurotransmitter and number of receptors – and is often summarized by a single parameter called the *synaptic weight*, which can change over short and long time scales. In actuality, the complete story is much more complex than what is described here (see, for example, Levitan et al., 2015), but all models of neurons involve some degree of simplification.

Many mathematical models characterizing a neuron's behaviour have been proposed, and they exist in a broad spectrum of complexity. One of the simplest is the McCulloch-Pitts neuron (McCulloch et al., 1943), in which a neuron is viewed as a unit that receives a sequence of binary inputs (from other neurons) and outputs a one if the weighted sum of inputs extends a threshold and a zero otherwise. This model captures the fact that the generation of spikes involves a summation or integration of input and thresholding. However, this model is not dynamic – it does not describe neural input or output as continuous-time signals, nor does it capture the fact that the integration of inputs over time triggers spikes or include a refractory period.

A slightly more complex and realistic model is the Hodgkin-Huxley model[1] (Hodgkin et al., 1952). This set of continuous non-linear differential equations characterizes the soma membrane as a capacitor and a neuron's ion channels as variable resistors. The system has no analytic solution, and instead, numerical methods must be used to compute membrane potentials over time. However, this is computationally prohibitive for large-scale simulations of neural networks, so building cognitive neural networks using Hodgkin-Huxley neurons is

[1]By today's standards, this model is still fairly simple, and more complex models have since been proposed. For example, multi-compartment models view the neuron as a structure with spatial extension, rather than a point object, and model voltage along dendrites (Ermentrout et al., 2010). However, these models involve coupled non-linear differential equations making them computationally prohibitive to simulate.

infeasible.

Because we are interested in building such large-scale models, we adopt a model between these poles of complexity – the leaky integrate-and-fire (LIF) model. The LIF is one of the earliest neuron models. It was invented by Lapicque in 1907 (Abbott, 1999) when recordings of action potentials had been made, but the electrophysiology of the neuron was not yet understood. Models in the integrate-and-fire family describe the dynamics of the subthreshold membrane voltage, $V(t)$, with a simple differential equation. The LIF model is given by

$$\frac{dV(t)}{dt} = \frac{1}{RC}\left(V_{\text{rest}} - V(t)\right) + \frac{1}{C}J(t), \qquad \text{while } V(t) < V_{\text{thresh}}, \tag{2.1}$$

where $J(t)$ is incoming current, $R$ is membrane resistance, $C$ is membrane capacitance, $V_{\text{rest}}$ is the resting potential, and $V_{\text{thresh}}$ is the threshold potential that determines when spikes are generated. The membrane voltage behaves as a leaky integrator until the threshold is reached, at which point a spike occurs, and the voltage is held at a reset potential, $V_{\text{reset}}$, over the refractory period lasting $\tau_{\text{ref}}$ seconds:

$$\begin{aligned} V(t) &= V_{\text{spike}}, \qquad \text{when } t = t_{\text{thresh}}, \\ V(t) &= V_{\text{reset}}, \qquad \text{when } t_{\text{thresh}} < t \leq t_{\text{thresh}} + \tau_{\text{ref}} \end{aligned} \tag{2.2}$$

where $t_{\text{thresh}}$ is the time at which a spike occurs, and $V_{\text{spike}}$ is some large value we use to indicate a spike event.

The LIF differential system is linear while $V(t) < V_{\text{thresh}}$, and, thus, we can write a closed-form solution for both $V(t)$ in the sub-threshold regime and for the average spiking rate, $G_{\text{rate}}[J]$, given a constant input current $J$:

$$V(t) = \left(1 - e^{-\frac{t}{RC}}\right)\left(V_{\text{rest}} + RJ(t)\right) + e^{-\frac{t}{RC}}V(0), \tag{2.3}$$

$$G_{\text{rate}}[J] = \begin{cases} 0 & \text{if } J \leq J_{\text{thresh}} \\ \left(\tau_{\text{ref}} - RC\log\left(1 - \frac{V_{\text{thresh}} - V_{\text{reset}}}{V_{\text{rest}} - V_{\text{reset}} + RJ}\right)\right)^{-1} & \text{if } J > J_{\text{thresh}} \end{cases} \tag{2.4}$$

Note that, without loss of generality, we may scale currents and voltages such that $V_{\text{thresh}} = 1$ and $V_{\text{rest}} = V_{\text{reset}} = 0$ to simplify the above equations.

The LIF neuron captures, in broad strokes, the dynamics of biological neurons and can be well-fitted to empirical data. While the Hodgkin-Huxley model, and other, more complex neuron models, are more accurate, the LIF neuron is computationally cheap to simulate. This makes modelling large-scale neural networks feasible and is why LIF models are widely used in computational neuroscience.

In machine learning (ML), the units of neural networks are artificial neurons, which characterize 'neuron' output via a non-linear activation function (often a sigmoid, tanh, or ReLU function). These models do not attempt to emulate biological neurons and are not dynamic. Dynamics on the network level are also limited. Non-recurrent networks built from

such units are run via single pass-throughs per input rather than being simulated over time periods – as is necessary in spiking neural networks for information to percolate through the system. Nonetheless, the simple 'neuron' units assumed by ML facilitate the training of incredibly large networks via stochastic gradient descent-based methods and have been highly successful across many applications.

While artificial neural networks (ANNs) are generally preferable to spiking neural networks (SNNs) for industrial applications and practical deployment, SNNs offer distinct advantages for brain modelling due to their biological realism. This realism allows modellers to make predictions and more direct comparisons to real neural data. The continuous-time dynamics of SNNs better represent temporal information processing, which is crucial for real-time applications. Additionally, adhering to the constraints imposed by biological plausibility can lead to a deeper understanding of neural processes and the underlying algorithms of the brain. SNNs also demonstrate robustness to noise and improved energy efficiency through event-driven computation, making them suitable for deployment on neuromorphic hardware.

### *2.1.2 Plasticity*

To model cognitive functions, we require populations of neurons with inter-population connections. Generally speaking, when a postsynaptic neuron is connected to many presynaptic neurons, their signals combine through spatial and temporal summation of the postsynaptic potentials, which may be excitatory or inhibitory. These potentials are filtered or smoothed spike trains. The exact form of temporal filtering is heterogeneous. There is a diversity of postsynaptic potential shapes across the brain, and neuromodulators can also affect postsynaptic impacts (Matsumura et al., 1996; Fon et al., 2001).

In our models, the incoming current to a postsynaptic neuron is a weighted sum over the postsynaptic potentials received from all presynaptic neurons that transmit information to that cell. By "weights," we mean parameters that summarize the strength, or efficacy, of the connection between a set of presynaptic and postsynaptic neurons. Several factors affect synaptic strength, including the number of available synaptic vesicles, the concentration of calcium ions entering the presynaptic terminal, the density of receptors on the postsynaptic membrane (Atwood et al., 2002; Citri et al., 2008), and even the spatial structure of the synapse (Freche et al., 2011).

These factors are not constant, and hence synapse strength can be modulated – this is synaptic plasticity, and is the physiological basis for some forms of learning and memory. For example, synaptic potentiation is the process by which synaptic strength is increased. Potentiation can occur on short timescales (milliseconds to minutes) or longer time scales (minutes to hours or longer). Short term plasticity can occur due to a temporary increase in the release of neurotransmitters or an increase in postsynaptic receptor sensitivity. The mechanisms of long-term potentiation are often described as Hebbian plasticity, summarized as "cells that fire together, wire together" (Shatz, 1992).

In models, the many factors affecting synaptic strength are often summarized with a single parameter, a synaptic weight $w_{ij}$ that corresponds to the connection from presynaptic neuron $i$ to postsynaptic neuron $j$. Standard Hebbian learning can be expressed as $\Delta w_{ij} = \eta a_i b_j$, where $\eta$ is the learning rate, $a_i$ is the presynaptic activity, and $b_j$ is the postsynaptic activity. Three-factor Hebbian learning adds a third factor, $E$, often representing a neuromodulatory signal, to modify synaptic changes based on other information (*e.g.*, error, reward, feedback, or attention). The rule can be expressed as

$$\Delta w_{ij} = \eta E a_i b_j \tag{2.5}$$

We describe a variant of three-factor Hebbian learning later in Section 2.2.4, which we use in subsequent models.

Note that biological learning and neural plasticity differ from typical learning in ANNs. Synaptic plasticity in the brain is a dynamic, activity-dependent process, while in deep ANNs, learning happens through the optimization of weights in response to a defined loss function using backpropagation. When modelling a fixed system, it may be reasonable to optimize weights offline to create a network that models a desired function or behaviour rather than modelling the whole process of development and life-long learning via biological mechanisms. However, for describing the process of online learning in a biological system, standard practices in deep learning are problematic.

For one, neurons use local information, such as presynaptic and postsynaptic activity (and potentially other signals like neuromodulators), to adjust synaptic strength. In ANNs, learning is global: the weight updates are driven by a global error signal propagated across the entire network through backpropagation. Backpropagation adjusts the weights of every neuron in the network based on the contribution to the overall error, without local learning rules. The brain is not able to instantaneously calculate and propagate gradients across many layers of neurons, making backpropagation biologically implausible.

In addition, gradient computation and backpropagation in deep learning models require significant computational resources and energy, especially for large-scale models. Biological systems, by contrast, are highly energy-efficient, using sparse, asynchronous spiking and local synaptic adjustments to optimize their learning processes. Thus, while gradient computations and global updates are practical for training deep neural networks, they pose limitations in terms of biological plausibility and adaptability.[2]

### *2.1.3 Neural Coding*

Understanding the response of neurons to injected current is a far cry from understanding how neurons encode and communicate *information* through spike trains. This brings us to the concepts of tuning curves and population codes.

[2]There are learning algorithms that attempt to mimic aspects of backpropagation with realistic neural mechanisms, such as feedback alignment and predictive coding. Such methods are not used in this work. Instead, we primarily model learning processes via neuromodulated plasticity and three-factor learning rules.

Finding relationships between neural activity and stimuli is easiest in sensorimotor areas, as the stimuli are external measurable features rather than internal state variables. In a classic experiment by Hubel et al. (1962), neural activity was recorded in response to visual stimuli, specifically a rectangular bar at different rotations. They identified neurons with a preferential sensitivity to specific orientations. The response of a neuron defined in this way is often described as its *tuning curve*, a function mapping a stimulus property (*e.g.*, bar orientation) to activity or firing rate. In the aforementioned study, this response resembled a bell curve peaked at a specific orientation angle, known as the neuron's *preferred stimulus*. Similar tuning curves are observed in motor cortical cells that respond to the angle of arm movements (Georgopoulos et al., 1982). A related concept is the neuron's *receptive field*, which is the area of sensory or feature space that elicits some response.

Neurons with more abstract and complex preferred stimuli than orientation have been observed. For instance, neurons in the fusiform face area exhibit strong selectivity for images of faces as opposed to objects or other visual stimuli (Kanwisher et al., 2006). The mapping between represented information and collective activity is called a *neural code* or *population code*. Inferring the neural code – identifying what information a population of neurons encodes and the characteristics of the encoding and decoding scheme – can be challenging, especially in brain areas where encoded information reflects internal states or is more integrated and abstract.

Considering the collective activity of a population of $N$ neurons, we can imagine their instantaneous activity as a point in the *neuron space*, an $N$-dimensional space of all possible neural states. This activity often has structure, residing on a lower-dimensional latent manifold (Yu et al., 2008; Cunningham et al., 2014; Gallego et al., 2018; Pandarinath et al., 2018; Mitchell-Heggs et al., 2023). The dimensionality and geometry of these so-called *neural manifolds* reveals information about a population's latent variables and their dynamics.

Jazayeri et al. (2021) make the distinction between a population's intrinsic and embedding dimensionality. They define embedding dimensionality as the size of Euclidean space in which the latent manifold resides – what we shall call the *state space* of the population, typically identified through linear dimensionality reduction methods like principal component analysis (PCA). Intrinsic dimensionality represents the number of independent variables required to describe the neural activity, estimated via non-linear dimensionality reduction methods, and is always less than or equal to the embedding dimensionality. By distinguishing between intrinsic and embedding dimensionality, we can understand both *what* information is represented within a population and *how* it is organized. As stated by Jazayeri et al. (2021), "[d]ifferent hypotheses about the nature of information processing in the brain can be cast as different forms of embedding."

An array of past research has estimated the dimensional properties of latent spaces in neural populations across various brain areas. Chaudhuri et al. (2019) examined the neural manifold of head direction cells in the anterodorsal thalamic nucleus of mice, finding it to form a convoluted ring within a three-dimensional space. Similarly, Low et al. (2018) and Nieh et al.

(2021) investigated the CA1 region of the hippocampus during navigation tasks, identifying intrinsic dimensionalities ranging from four to six and embedding dimensionalities between 29 and 47. Zhang et al. (2023) proposed hyperbolic geometry within CA1 neural manifolds, while Gardner et al. (2022) found a toroidal neural manifold in the medial entorhinal cortex's grid cells. Critically, manifold geometry is not inconsequential – in fact, it can be directly related to performance on cognitive tasks. For example, Sohn et al. (2019) demonstrated that the manifold curvature in the dorsomedial frontal cortex during a timing task reflects Bayesian computation, influenced by varying prior beliefs to support Bayesian estimates.

Although these empirical studies illuminate aspects of the neural code, they do not fully describe it. They assist researchers in forming hypotheses, but computational modelling is essential for testing these hypotheses and generating testable predictions. One prominent hypothesis about the neural code posits that information can be deciphered by other brain regions through a linear readout or decoding of neural activity (Eliasmith et al., 2003; Salinas et al., 1994; Pouget et al., 2000). This hypothesis places particular constraints on neural coding schemes. Additionally, the principle of linear decodability is crucial for cognitive models that make use of separating linear hyperplanes – be it for classification or disentangling distributed neural representations (DiCarlo et al., 2007; Pan et al., 2019; Bernardi et al., 2020). In this thesis, we use the NEF and SPA to model neural manifolds, allowing us to specify what information is embedded in neural activities and how this process may be implemented in SNNs.

## 2.2 The Neural Engineering Framework

The Neural Engineering Framework (NEF; Eliasmith et al. (2003)) is a well-established method for building neural simulations constrained by low-level neurobiological details. Over the last two decades, the NEF has been used to construct models of brain areas such as the thalamus (Stewart et al., 2010b), basal ganglia (Stewart et al., 2010b), primary motor cortex (DeWolf et al., 2011), and amygdala (Duggins et al., 2024). Additionally, it has been used to build models of specific cognitive functions such as working memory (Singh et al., 2006; Duggins et al., 2017), long-term memory (Gosmann et al., 2021), attention (Bobier et al., 2014), motor control (DeWolf et al., 2016; Vaidyanathan et al., 2019), and social decision-making (Duggins et al., 2022). This framework is implemented in Nengo (Bekolay et al., 2014), an open-source software package.

The NEF is based on the ideas of population codes, latent variables, and linear readout. Variables are non-linearly encoded and linearly decoded by a population's collective neural activity. In particular, the NEF consists of three core principles: *representation*, *transformation*, and *dynamics*. Together, these present a well-formulated approach to neural modelling based on information theory and control theory that can be used to test hypotheses and validate proposed neural algorithms.

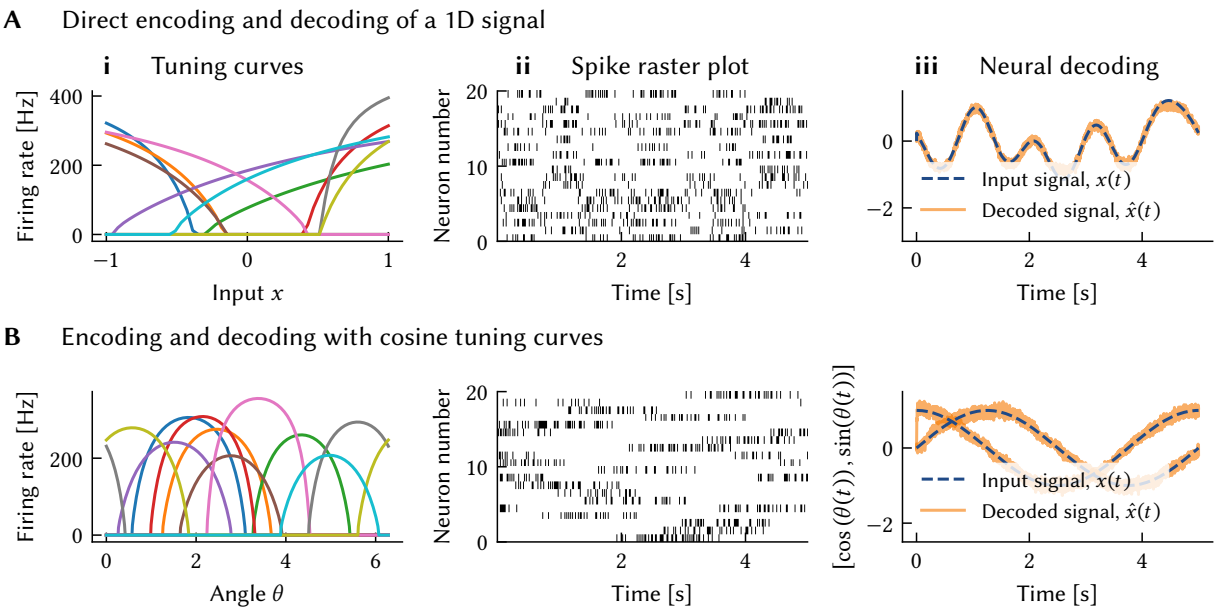



**Figure 2.1: Encoding and decoding in the NEF. (A)** An example of encoding and decoding a 1D signal directly. **i)** The tuning curves of a population of 10 LIF neurons. The functions are the rate approximation LIF neurons – see Eq. (2.4). **ii)** The simulated spike trains of the neurons given an input $x(t)$. This type of plot is called a *spike raster plot*. **iii)** The input signal $x(t)$, along with the linearly decoded output of the population. **(B)** Cosine tuning curves in the NEF. **i** Ten examples of tuning curves from a population of 20 LIF neurons encoding $[\cos(\theta(t)), \sin(\theta(t))]$. **ii)** The population's spiking activity. **iii)** The input to the population, $[\cos(\theta(t)), \sin(\theta(t))]$, along with the linearly decoded output of the population.

### 2.2.1 *Representation*

**Neural encoding.** The first principle of the NEF addresses neural *representation*. It proposes that we can encode a vector signal $\boldsymbol{x} \in \mathbb{R}^d$ in the collective activity of a population of $N$ neurons. A single neuron's activity is given by

$$a_i(\boldsymbol{x}) = G_i\left[\alpha_i \langle \boldsymbol{e}_i, \boldsymbol{x} \rangle + \beta_i\right], \tag{2.6}$$

where $a_i$ is the activity of neuron $i$,[3] $\langle \cdot, \cdot \rangle$ is the vector inner product,[4] and $\alpha_i$, $\beta_i$, and $\boldsymbol{e}_i$ are parameters that shape the neuron's tuning curve. $G_i$ is some neuron model, a non-linear function that defines the relationship between the neuron's activity and the incoming current. The standard choices of non-linear activation functions in artificial neural networks may be used here (*e.g.*, ReLu or sigmoid), although how to define their temporal properties may be

[3] Given a signal input, $\boldsymbol{x}(t)$, the activity $a_i(\boldsymbol{x}(t))$ may be a firing rate over time or a spike train, depending on the neuron model $G$ that is used.

[4] Extensions to the NEF have been proposed that generalize this to inner products over continuous-time signals and functions (Stöckel, 2022).

unclear. However, we primarily use the LIF model given in Eq. (2.1) and Eq. (2.2), and so neural activity over time, $a_i(\boldsymbol{x}(t))$, is a spike train.

The neural tuning curves are set by parameters $\alpha_i$, the gain factor; $\beta_i$, the bias; and $\boldsymbol{e}_i$, the neuron's *encoder*. Encoders define what sort of input a particular neuron is sensitive to, hence capturing the tuning curve of a neuron. Input orthogonal to a neuron's encoder does not elicit a response (above its baseline), while input aligned with the encoder does elicit a strong response – *i.e.*, the encoder is the neuron's *preferred direction vector*. We always assume that $\boldsymbol{e}_i$ is unit length.

Note from Eq. (2.1) that the LIF neuron fires when $\langle \boldsymbol{e}_i, \boldsymbol{x} \rangle > {}^{1-\beta_i}/_{\alpha_i}$, and that the neuron's maximum firing rate (obtained when $\langle \boldsymbol{e}_i, \boldsymbol{x} \rangle = 1$) is given by $G_{\text{rate}}[\alpha_i + \beta_i]$. Thus, the bias and gain determine the sensitivity of the neuron – how similar input must be to the neuron's preferred stimulus for it to fire at all and how frequently the neuron fires at its preferred stimulus. Typically, we draw random maximum firing rates and intercepts from distributions to obtain heterogeneous populations with good representational properties. These variables may also be set to match neurobiological data. These values are, in turn, used to compute the gains and biases.

Given 1D input, the tuning curves of LIF neurons look like Figure 2.1A(i). Examples of actual output spike trains given a random input signal are shown in Figure 2.1A(ii). The firing rate of the LIF neuron is zero until $\langle \boldsymbol{e}_i, \boldsymbol{x} \rangle$ (which is $\pm x$ in the 1D case) exceeds a threshold, which is defined by the neuron's intercept $\beta_i$. Beyond the intercept, the firing rate is monotonically increasing.

This differs from the example given in Section 2.1.3, in which we described neurons with cosine tuning curves with respect to the visual stimuli of a bar at a particular angle. To model these responses, we must first map bar orientation, $\theta$, to a higher-dimensional variable, $\phi(\theta) = [\cos(\theta), \sin(\theta)]$. This results in tuning curves like those shown in Figure 2.1B(ii). This is a straightforward example of the difference between the intrinsic, state, and activity space of a population. In the model, 1D information, $\theta$, (the intrinsic variable) is represented as a point on a circle in 2D space, $\phi(\theta)$, (the embedding/state variable) that is encoded in the $N$-dimensional neural activity space of the population.

In general, suppose we want to model neural responses so that they depend on the relationship between a presented stimulus and each neuron's preferred stimulus. We can achieve this by defining a function that captures how a neuron's response varies based on the similarity between these two stimuli. We denote this function as $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, where $k(\boldsymbol{x}, \boldsymbol{x}_{\text{preferred}})$ represents the firing rate of a neuron given an input stimulus $\boldsymbol{x}$. This function could be designed to emulate the tuning curves of biological neurons found in a particular dataset.

According to Mercer's theorem, if $k$ is a positive-definite kernel,[5] then there exists a feature mapping, $\phi : \mathcal{X} \to \mathcal{H}$, such that $\langle \phi(\boldsymbol{x}), \phi(\boldsymbol{x}_{\text{preferred}}) \rangle = k(\boldsymbol{x}, \boldsymbol{x}_{\text{preferred}})$. This mapping projects

[5]That is to say, $K$ is symmetric and positive semi-definite, where $K$ is a matrix such that $K_{i,j} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ for any sets of $\boldsymbol{x}_i, \boldsymbol{x}_j \in \mathcal{X}$.

from the stimulus space into a Hilbert space such that the kernel can be expressed as the inner product of two features. Essentially, for every positive definite kernel, there exists a corresponding feature mapping – this is the basis of the "kernel trick." Thus, by leveraging feature mappings, we can effectively model a wide array of neural responses using the simple encoding equation given in Eq. (2.6). This is relied on in later chapters of this thesis.

It is important to note that this is a descriptive account of neural encoding. Generally, in the NEF, we set neural parameters and feature maps to match biological tuning curves rather than to propose explanations for the origin, formation, or learning of neural tuning.

**Neural decoding**. The representation principle also explains how to linearly decode the vector encoded by the activity of a population of $N$ neurons:

$$\hat{\boldsymbol{x}} = \sum_{i=1}^{N} a_i(t) * h_{\text{syn}}(t)\boldsymbol{d}_i = D\boldsymbol{a}(t) \tag{2.7}$$

where $*$ is convolution, $\boldsymbol{d}_i \in \mathbb{R}^d$ are called *decoders*, $D \in \mathbb{R}^{d\times N}$ is a matrix of decoders, and $\boldsymbol{a}(t) = [a_1(t) * h_{\text{syn}}(t), \dots, a_N(t) * h_{\text{syn}}(t)]^T \in \mathbb{R}^N$ is a vector of filtered activities. The function $h_{\text{syn}}(t)$ is called the *postsynaptic filter*, and it has the effect of 'smoothing' incoming spikes over time (see Figure 2.4). We shall use a first-order low-pass filter parameterized by $\tau$, the *postsynaptic time constant*, as the synaptic filter. The filter can be written as a dynamical system:

$$\tau \frac{dh_{\text{syn}}(t)}{dt} = -h_{\text{syn}}(t) + a(t), \tag{2.8}$$

which, given an impulse input (*i.e.*, a spike) at time $t_{\text{spike}}$, has the solution

$$h_{\text{syn}}(t) = \begin{cases} \frac{1}{\tau} e^{-(t-t_{\text{spike}})/\tau} & \text{if } t > t_{\text{spike}} \\ 0 & t \leq t_{\text{spike}}. \end{cases} \tag{2.9}$$

The decoders are solved for via regularized least-squares optimization, *i.e.*, ridge regression:

$$\min_D \frac{1}{n_s} \sum_{j=1}^{n_s} \left\| \boldsymbol{x}_j - D\boldsymbol{a}(\boldsymbol{x}_j) \right\|^2 + \lambda \sum_{i=1}^{N} \left\| \boldsymbol{d}_i \right\|^2, \tag{2.10}$$

where $D \in \mathbb{R}^{d\times N}$ is a matrix whose columns are the decoders, $\{\boldsymbol{x}_j\}_{j=0}^{n_s}$ is a set of $n_s$ sample inputs, and $\boldsymbol{a}(\boldsymbol{x}) \in \mathbb{R}^N$ is the vector of the firing rates (*i.e.*, the rate approximation for LIF neurons given in equation Eq. (2.4)) of all neurons given input $\boldsymbol{x}$. The second term in the objective function is the $L_2$-regularization weight penalty – this produces small, non-zero decoders. Other types of regularization may also be used.

The optimal decoders are given by

$$D^T = A^+ X, \quad \text{where } A^+ = (A^T A + 2n_s\lambda I)^{-1} A. \tag{2.11}$$

The term $A^+$ is the regularized Moore-Penrose pseudo-inverse of $A$, $A \in \mathbb{R}^{N \times n_s}$ is a matrix of sampled activity rates with $A_{ij} = \boldsymbol{a}_i(\boldsymbol{x}_j)$, and $X \in \mathbb{R}^{n_s \times d}$ is a matrix whose rows are the input samples. Figure 2.1A-B(iii) shows examples of input signals and approximations of these signals obtained by linearly decoding spike trains with optimal decoders $D$.

### 2.2.2 Transformation

The *transformation* principle of the NEF provides the method for setting weights between two neural populations to compute a target function. Assume a population of $N$ neurons representing a vector, $\boldsymbol{x} \in \mathbb{R}^d$, is fully connected to a different population of $N'$ neurons. The second population should represent some function of the input vector, that is, $f(\boldsymbol{x}) \in \mathbb{R}^{d'}$. This function can be decoded out of the first population's activity,

$$\hat{f}(\boldsymbol{x}) = \sum_{i=1}^{n} a_i(t) * h(t) \boldsymbol{d}_i^{(f)}. \tag{2.12}$$

The second population's neurons receive this as current input. These function-specific decoders, $\boldsymbol{d}_i^{(f)} \in \mathbb{R}^{d'}$, can be solved for using samples from the desired function and least-squares optimization. In other words, we use Eq. (2.11) with $X$ replaced by a matrix whose columns are target output samples, $f(X) \in \mathbb{R}^{n_s \times d'}$:

$$D^{(f)^T} = A^+ f(X). \tag{2.13}$$

If sample outputs are unavailable, decoders can be learned online in response to error signals (see Section 2.2.4).

Decoding the output of the first population and encoding it in the activity of the second population is equivalent to multiplying the filtered activities of the first population with a weight matrix and feeding that current into the second population, which has activities given by

$$b_j(t) = G_j \left[ \sum_{i=1}^{N} w_{ji} a_i(t) + \beta_j \right], \tag{2.14}$$

$$\text{where } w_{ji} = \alpha_j \boldsymbol{e}_j \cdot \boldsymbol{d}_i^{(f)}. \tag{2.15}$$

Thus, models built with the NEF approach of non-linear encoding and linear decoding are just standard (spiking) neural networks with weight matrices, $W \in \mathbb{R}^{N' \times N}$, defining connections. However, the NEF approach has a particular constraint: the weight matrices are, typically,[6] set as the product between the encoders of the post-population (which are fixed, often to match

[6] Full-rank neuron-to-neuron connection weight matrices can be used in NEF models, but low-rank outer product matrices are more commonly used.

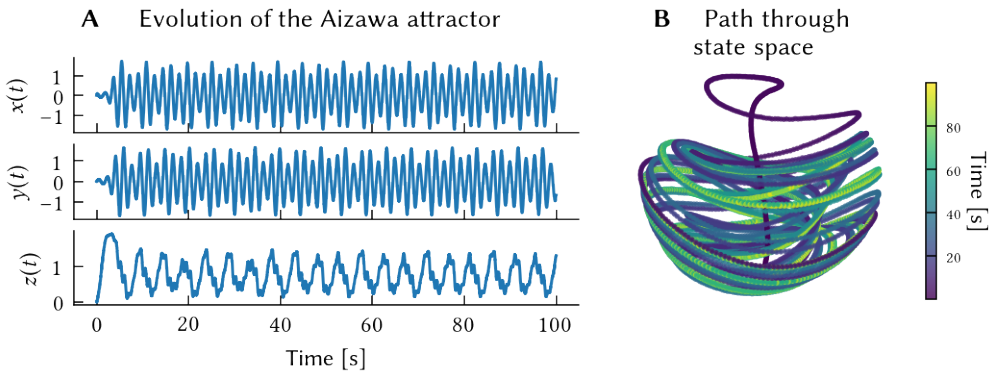



**Figure 2.2: The Aizawa attractor implemented in an SNN using the NEF.** The Aizawa attractor is defined by the following dynamical system: $[\dot{x}, \dot{y}, \dot{z}] = [(z-b)x - dy, dx + (z-b)y, c + az - \frac{1}{3}z^3 - x^2 + fzx^3]$ where $a = 0.95$, $b = 0.7, c = 0.6$, $d = 3.5$, $e = 0.25$, $f = 0.1$. The state variables $[x(t), y(t), z(t)]$ were represented by a population of 2000 recurrently connected LIF neurons. Recurrent connection weights were set to approximate the Aizawa system. **(A)** The state variables decoded by the neural population over the entire simulation. **(B)** The path traced out by the state variables in 3D space.

biological tuning curves) and the decoders of the pre-population (which are optimized). Thus, the weight matrices are not full-rank, reducing the computational overhead during simulation.[7]

This is similar to low-rank factorization in deep learning (Hu et al., 2021), where one approximates a weight matrix as the product of two smaller matrices to reduces the number of parameters – though usually both matrices are optimized, unlike here.

### 2.2.3 *Dynamics*

The last principle of the NEF is *dynamics*. Dynamical systems can be encoded in a recurrently connected population of spiking neurons. Consider a linear time invariant (LTI) system,

$$\dot{\boldsymbol{x}}(t) = A\boldsymbol{x}(t) + B\boldsymbol{u}(t), \tag{2.16}$$

where $\boldsymbol{u}(t)$ is an input signal, $\boldsymbol{x}(t)$ is the state variable we would like to encode in a population, $A \in \mathbb{R}^{d \times d}$ is the *state matrix*, and $B \in \mathbb{R}^{d \times d_u}$ is the *input matrix*.

To realize the dynamics specified by $A$ and $B$, we might think to set incoming weights to approximate the transform $B$ and the recurrent connection to approximate $A$ (using the same method as in Section 2.2.2). However, this does not work well in practice due to the fact that the LIF neuron model coupled with our synaptic model is a leaky integrator and Eq. (2.16) assumes perfect temporal integration. Filtering 'distorts' the state dynamics but in a predictable way.

[7] In particular, the cost of computing $W\boldsymbol{a}$ with a full-rank weight matrix is $\mathcal{O}(N'N)$, while computing $ED\boldsymbol{a}$ is $\mathcal{O}(d'N + d'N')$ with $d' \ll N, N'$.

To account for postsynaptic filtering, the NEF states that we must modify the target feedback and input matrices:

$$A \to \tau A + I, \quad B \to \tau B. \tag{2.17}$$

To implement a non-linear dynamical system,

$$\dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}(t)) + g(\boldsymbol{u}(t)), \tag{2.18}$$

we apply a similar correction,

$$f(\boldsymbol{x}(t)) \to \tau f(\boldsymbol{x}(t)) + \boldsymbol{x}(t), \quad g(\boldsymbol{u}(t)) \to \tau g(\boldsymbol{u}(t)). \tag{2.19}$$

Thus, to approximate the dynamics in Eq. (2.18), we set incoming weights (or, rather, the decoders) to approximate the transform $\tau g(\boldsymbol{u}(t))$ and the recurrent connection to approximate $\tau f(\boldsymbol{x}(t)) + \boldsymbol{x}(t)$. This is done by solving for the optimal decoders using Eq. (2.13). The NEF dynamics principle can be used to model a wide range of dynamical systems in neural populations.[8] To illustrate the application of the dynamics principle, we provide an example in Figure 2.2 – here, an SNN is used to model the Aizawa attractor, a chaotic attractor system with a three-dimensional state space.

The dynamics of a population's latent state may shape the geometry and characteristics of the manifold on which the state variables reside. Consider attractor dynamics, which constrain the state in specific ways, as we see in our example. The state, while varying over time (Figure 2.2A), lies on a spherical-shaped attractor (Figure 2.2B). Attractor dynamics have frequently been used to model cognitive functions, such as memory (*e.g.*, Hopfield, 1982; Amit et al., 1989), while chaotic attractors have been used to model memory (Pereira-Obilinovic et al., 2023), perception (Skarda et al., 1987), and switching between neural states and dynamical regimes (Inoue et al., 2020). Here, we use the Aizawa attractor as an example of the types of dynamics that can be modelled with the NEF. However, many other systems – such as oscillators, integrators, and memory traces – can also be modelled.

Note that, in the case of both transformations and dynamics, we optimize a single set of (factorized) connection weights to approximate the target transformation/dynamical system. This does not guarantee that the approximation is good. Sufficient neural resources (*i.e.*, number of neurons, appropriate tuning curves) are required. Furthermore, some computations may require multi-layer networks (which must be trained via other means) to be accurately realized. Other approaches to designing and training neural networks can be used in conjunction with the NEF methods.

### *2.2.4 Learning rules*

When the desired transform function or example input-output data is known in advance, feedforward and recurrent weights can be optimized by least-squares as described above.

[8] Note however that there is no $h(x(t), u(t))$ term in Eq. (2.18). If we wish to model a dynamical system that includes non-linear interactions between the state and input, we must include the input signal in the latent space of the population, *i.e.*, $x'(t) = [x(t), u(t)]$.

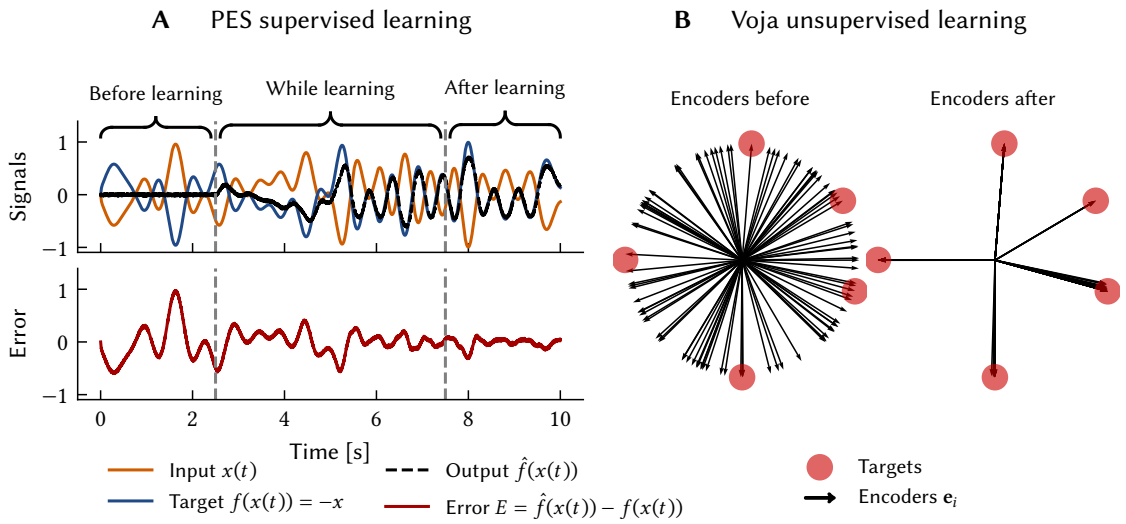



**Figure 2.3: Different types of online learning in the NEF. (A)** The prescribed error sensitivity (PES) learning rule is used to learn a target function, $f(x) = -x$, given a white noise signal as input, $x(t)$. In the 'before learning' phase, the output decoded from the population, $\hat{f}(x(t))$, does not match the target (upper panel), and the error $E = \hat{f}(x(t)) - f(x(t))$ is high (bottom panel). During the 'while learning' phase the PES rule is used to modify decoders according to Eq. (2.20). At $t = 7.5$s the PES rule is turned off ($\eta = 0$). We see that the output aligns closely, but not perfectly, with the target function during this 'after learning' phase. **(B)** The Voja rule is used to modify the encoders of a population. A population of 100 LIF neurons, representing a two-dimensional state, is initialized with random unit-length encoders $\boldsymbol{e}_i \sim \text{Unif}(\mathbb{S}^1)$ for $i \in 1, 2, \ldots, 100$ (these are shown in the 'encoders before' panel). The population receives input $\boldsymbol{x}(t)$, which cycles across five possible states (the 'targets' shown in red). The Voja learning rule moves the encoders toward the targets, so after learning, the population has neurons that selectively fire at only one of the targets

However, there are circumstances in which these transformations are not known in advance, and instead, one would like the network to adapt and learn online.

Biologically plausible learning rules that only use local information are used in the NEF for modifying synaptic weights online. One example is the prescribed error sensitivity (PES; MacNeil et al., 2011) rule, a biologically plausible supervised learning rule that modifies decoders. This rule shifts the pre-population's decoders in response to an error signal:

$$\Delta \boldsymbol{d}_i = \eta \boldsymbol{E} a_i \quad \text{or, equivalently,} \quad \Delta w_{ji} = -\eta \alpha_j \boldsymbol{e}_j \cdot \boldsymbol{E} a_i, \tag{2.20}$$

where $\eta$ is a learning rate, $a_j$ are pre-population neural activities (filtered spikes), $\alpha_j$ are post-population gains, $\boldsymbol{e}_j$ are the post-population encoders, and $E$ is an error signal we seek to minimize. Note that we have pre- and post-neural activities and an extra term, $\boldsymbol{e}_j \cdot \boldsymbol{E}$ (an error signal projected onto the neural encoders), and so this is a type of three-factor Hebbian learning rule.

An example of this rule in action is given in Figure 2.3A. A white noise signal, $x(t)$, is given as input to a population, and the goal is to learn to decode the target function $f(x) = -x$ from

this population. Initially, decoders are all zero, but once an error signal is supplied ('While learning' Figure 2.3A), the PES rule modifies these decoders until learning is turned off ('After learning' Figure 2.3A). We see that initially, the error $E$ is high, but over the course of learning, the error decreases as the output algins with the target function.

PES has been used to train memory networks (Trujillo et al., 2014), reinforcement learning models (Rasmussen et al., 2017), binding networks (Bekolay et al., 2013), and models of inverse kinematics (Zaidel et al., 2021). Additionally, data from spike timing-dependent plasticity experiments is matched by PES when used in combination with the unsupervised Bienenstock-Cooper-Munro (BCM) learning rule, which sparsifies weights (Bekolay et al., 2013).

There are also unsupervised learning rules commonly used in NEF models to learn encoders – and, thus, tuning curves. For example, the Oja learning rule is a modification of the Hebbian learning rule that improves stability (Oja, 1982). The vector version of this, the Voja learning rule, shifts encoders so that neurons fire selectively at particular inputs and activity is sparsified:

$$\Delta \boldsymbol{e}_i = \eta a_i (\boldsymbol{x} - \boldsymbol{e}_i) \tag{2.21}$$

where $\eta$ is a learning rate, $\boldsymbol{e}_i$ is the encoders of neuron $i$, $a_i$ is filtered activity of neuron $i$, and $\boldsymbol{x}$ is the input vector encoded by the neurons. This has the effect of shifting the encoders towards input (Figure 2.3B), resulting in a sparser rather than distributed encoding of typical input. This rule has been previously used to train hetero-associative memory networks (Voelker et al., 2014). Sparse activity is advantageous in memory networks as it minimizes overlap between patterns, reducing interference when multiple memories are stored.

### *2.2.5 Summary*

Figure 2.4 summarizes the encoding, transformation, and dynamics principles of the NEF. A neuron receives spike inputs from other neurons, and these incoming spikes are filtered through $h_{\text{syn}}(t)$. The filtered spikes are then decoded using the decoders $D$, followed by re-encoding through $\alpha_i \langle \boldsymbol{e}_i, \boldsymbol{y}(t) \rangle$. In practice, this decoding-encoding process can be viewed as a weight matrix applied to the filtered inputs, but in simulation, it is represented as separate encoding and decoding to reduce computational complexity and give the modeller control over encoders, which influence the tuning curves.

These operations are conceptualized as occurring in the dendrites, while the soma receives the resulting voltage. The incoming voltage is integrated (following Eq. (2.1)) until a threshold is reached, triggering a spike. Outgoing spikes $b_j(t)$ from a population may supply input to other populations or to the population itself via recurrent connections.

## 2.3 Vector Symbolic Algebras

While the NEF describes how to build a neural system that realizes a latent variable-level description of an algorithm, it says nothing about what kinds of algorithms may be instantiated

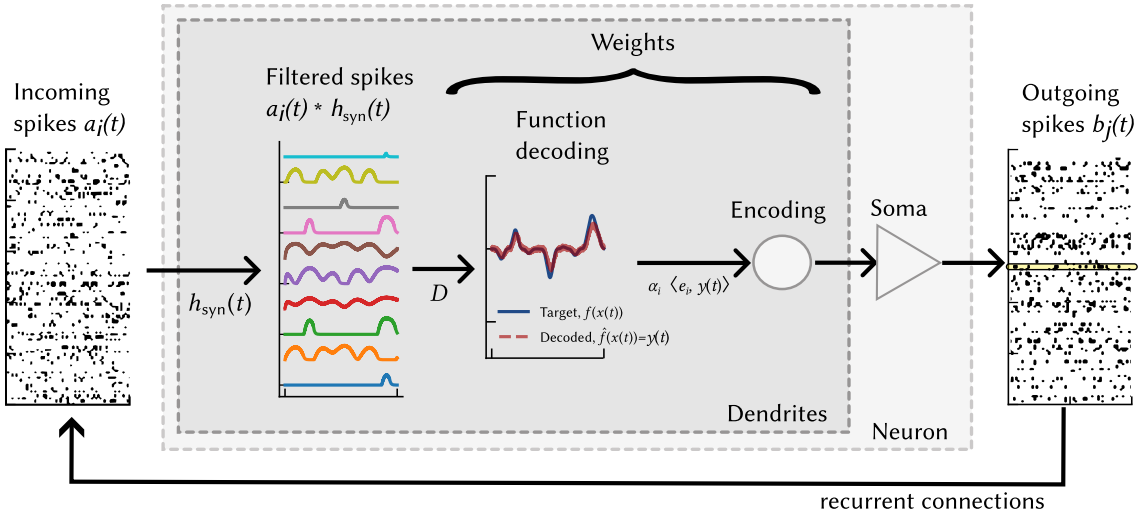



**Figure 2.4: Overview of neural encoding, transformation, and dynamics in the NEF.** Incoming spikes are filtered, decoded, and re-encoded, with operations in the dendrites. The soma integrates the resulting voltage, producing output spikes upon reaching a threshold (output from a single neuron is highlighted in yellow on the right). Outgoing spikes may loop back as feedback via a recurrent connection. Adapted from Figure 2.15 in Eliasmith (2013).

in the brain. The SPA extends the NEF by, among other things, combining it with a Vector Symbolic Algebra (VSA).[9] The VSA approach to neurosymbolic modelling describes how neural representations may be algebraically combined and manipulated to realize certain cognitive operations.

In VSAs, symbols and structured compositions of symbols are represented as high-dimensional vectors – *i.e.*, a distributed representation of discrete symbols. We call these Semantic Pointers (SPs). Algebraic operations defined over a vector space correspond to operations on such symbols. These operations include a similarity measure, a bundling operation, a binding operation, and an inverse operation.

VSAs are a family of related methods. Different variants define these operations in different ways and place different constraints on the symbol-like vectors. Examples of VSAs include Holographic Reduced Representation (HRR) and its realization in the Fourier domain (FHRR; Plate, 1995), Binary Spatter Codes (Kanerva, 1996), Multiply-Add-Permute (Gayler et al., 2009), Vector-Derived Transformation Binding (Gosmann et al., 2019), and more. We use HRRs in this thesis and now briefly describe each of the four operations used in this particular VSA.

*Similarity.* A similarity measure between two vectors indicates the semantic similarity of the symbols they represent. For example, the vector representation of a 'rose' might have a higher similarity measure with a 'lily' vector compared to a 'toad' vector. In HRRs, similarity is given

[9]Vector Symbolic Algebras (VSAs) are more commonly referred to as Vector Symbolic Architectures (VSAs). The term VSA is sometimes used interchangeably with hyperdimensional computing.

by the cosine measure (or dot product), which is also the measure for semantic similarity used in word2vec and many other machine learning representations (Mikolov et al., 2013).

*Bundling.* A bundling or collecting operation maps a set of vectors to a single vector, such that the resulting vector is similar to all from the set. In HRRs, vector addition serves as the bundling operation. This is compositional representation through *superposition*, mirroring mechanisms in the brain where neural populations encode overlapping representations within the same region. By storing multiple items in a single vector, bundling optimizes storage efficiency but increases the risk of interference as more items are added. To mitigate this, high-dimensional (and often sparse) vectors are employed, as their properties reduce the likelihood of interference.

*Binding.* A binding operation maps a pair of vectors to a single vector that is dissimilar to both of the original input vectors. If bundling is compositional representation via superposition, then binding is compositional representation via association or transformation. It is a way to bind symbols together to create a novel symbol (*e.g.*, combining a slot and filler to have a single slot-filler representation). For example, 'red' and 'ball' could be bound together to represent a red ball. In HRRs, binding is done by circular convolution, which can be computed in several ways,

$$A \circledast B = \sum_{m=0}^{d-1} A_m B_{(k-m) \bmod d}, \quad \text{for } k = 0, 1, \ldots, d-1 \tag{2.22}$$

$$= \mathcal{F}^{-1}\left\{\mathcal{F}\left\{A\right\} \odot \mathcal{F}\left\{B\right\}\right\}, \tag{2.23}$$

$$= \mathrm{Circ}(A)B, \tag{2.24}$$

where $\mathcal{F}\{\cdot\}$ is the discrete Fourier transform (DFT), $\odot$ is the Hadamard product, and $\mathrm{Circ}(A) \in \mathbb{R}^{d \times d}$ is a circulant matrix obtained by stacking circularly shifted versions of the vector $A$. In other words, binding is equivalent to element-wise multiplication in the Fourier domain or matrix multiplication with a circulant matrix. Spiking neural circuits have been built to accurately perform this operation (Eliasmith, 2005b) and learn it online (Bekolay et al., 2013), so this binding operation is neurally feasible.

*Inverse.* The inverse operation takes a single input vector and produces a single output vector that reverses the effect of binding with the input vector, $(A \circledast B) \circledast B^{-1} = A$. In HRRs, an easy-to-compute and numerically stable approximate inverse (involution) is used.

$$(A \circledast B) \circledast B^{-1} \approx A, \quad B^{-1} = [B_1, B_d, B_{d-1}, \ldots, B_2] \tag{2.25}$$

*Example.* To illustrate how these operations can encode and manipulate symbolic representations, consider the example of an ordered list. Vector representations of each object (fillers) are bound with vector representations of list positions (slots), and the results are summed to represent the collection in a vector:

$$M = \sum_i A_i \circledast P_i, \tag{2.26}$$

where $A_i$ is the vector representation of the object seen at position $i$, and $P_i$ is a vector encoding list position $i$. The final representation, $M$, can be queried to retrieve, for example, the position of a given object in the list by binding the total vector with the inverse of that object's vector. The result of this binding approximates the vector representing that object's index: $M \circledast A_j^{-1} \approx P_j$. This sort of memory trace was used by Choo et al. (2010) and Gosmann et al. (2021) to model long and short-term memory and serial recall. In Chapter 4, we adapt this approach to continuous mappings, associating objects with continuous locations instead of discrete list positions, which can later be queried for recall.

VSAs have also been used to represent various data structures in past work, including binary trees (Frady et al., 2020), graphs (Rachkovskij et al., 2001), stacks (Stewart et al., 2014), n-gram statistics (Joshi et al., 2017), images (Penzkofer et al., 2024), and knowledge graphs (Laube et al., 2024; see Kleyko et al., 2022 for more examples). The SPA in particular have been applied to many other tasks and used to model a wide range of phenomena. For example, Kajić et al. (2019a) used SPs to represent emotions, constructed by binding information about external stimuli, linguistic constructs, physiological responses, and stored concepts. Blouw et al. (2016) used the SPA to model categorization, replicating human experimental data. In the context of language processing, Plebe et al. (2020) modelled sentence disambiguation. The algebra of HRRs was used to construct hierarchical embeddings of ambiguous sentences, which, when combined with SPs generated by an object recognition network, could be queried to produce correct interpretations.

### *2.3.1 Representing discrete symbols and structures*

We have prescribed the HRR algebra for manipulating and composing SPs but have yet to describe how we map from symbols to vectors. First, let us consider discrete concepts, categories, or objects. For HRRs, discrete symbols are represented by vectors on the unit hyper-sphere. The vectors should, in some way, contain the semantic content of the symbol they represent, and the similarity measure between vectors should capture semantic similarity.

We might imagine such a vector as the combined output from different neural systems – a normalized, distributed representation combining perhaps visual, memory, and emotional content. In practice, statistical techniques can be applied to real-world data to construct useful embedding vectors. For instance, word2vec and GloVe are popular models for creating word embeddings by leveraging large corpora and the contextual relationships between words (Mikolov et al., 2013; Pennington et al., 2014). Other ML techniques can be used in conjunction with VSAs (Mitrokhin et al., 2020). In biological modelling contexts, these embeddings are sometimes derived from representations coming from models of sensory systems (Eliasmith et al., 2012). For some cognitive tasks, a meaningful distribution of vectors is not essential, and a corpus of completely distinct symbols may suffice. In such instances, high dimensional random vectors – which ensures low similarity between individual items – are sufficient.

In addition to discrete objects, these types of vectors can be used to create structure in VSA

representations by playing the role of the slot in slot-filler relationships. For example, sentences can be represented in a slot-filler manner with slots such as 'subject' or 'verb' specifying the functional roles of particular discrete data such as 'Bob' or 'throw,'

$$\text{subject} \circledast \text{Bob} + \text{verb} \circledast \text{throw} + \text{object} \circledast \text{ball}. \tag{2.27}$$

Alternatively, an object may have properties such as 'colour,' 'open' or 'closed,' and 'type.' Discrete symbol vectors can be used to represent not only the values of these properties but also the functional role of these discrete values,

$$\text{object} = \text{colour} \circledast \text{red} + \text{is\_open} \circledast \text{False} + \text{type} \circledast \text{door}. \tag{2.28}$$

### *2.3.2 Representing integer values*

Consider the task of representing integer values in a VSA model. Such representations are necessary in cognitive models designed to count, perform simple arithmetic, or maintain an ordered list. Iterative self-binding is a natural way to construct such representations. An integer, $n \in \mathbb{N}$, can be encoded by repeatedly binding a 'base' or starting vector to itself $n$ times:

$$\phi(n) = \underbrace{\boldsymbol{\phi} \circledast \boldsymbol{\phi} \circledast \cdots \circledast \boldsymbol{\phi}}_{\text{n times}}. \tag{2.29}$$

With this, we can bind objects from an ordered list with vectors encoding their indexes and sum across to generate a distributed representation of the complete list (Choo et al., 2010). Notably, any such structure has discrete 'slots'.

Specifically, we generate the base vectors by uniformly sampling their phases in the Fourier domain:[10]

$$\boldsymbol{\phi} = \mathcal{F}^{-1}\left\{e^{i\boldsymbol{a}}\right\}, \quad \boldsymbol{a} \sim \text{Unif}[-\pi, \pi], \tag{2.30}$$

where $\mathcal{F}^{-1}\{\cdot\}$ is the inverse discrete Fourier transform and the exponential is applied element-wise to $\boldsymbol{a}$. This construction ensures that when such vectors are bound together via circular convolution, the resulting vector's magnitude remains on the unit hyper-sphere. We refer to vectors with this property – vectors whose Fourier components all have a magnitude of one – as *unitary*.

## 2.4 Spatial Semantic Pointers

Iterative self-binding can be easily extended to *fractional binding* (Plate, 1992) to represent real-valued slots by exploiting the fact that convolution in the time domain is multiplication in the Fourier domain. Such representations are called Spatial Semantic Pointer (SSPs; Komer et al., 2019), or *fractional power encodings* (Frady et al., 2022), and are closely related to the method of Random Fourier Features (Rahimi et al., 2007) – see Section 2.4.3.

[10]While we most commonly use uniformly distributed phases, other distributions, *e.g.*, Gaussian, may be used instead.

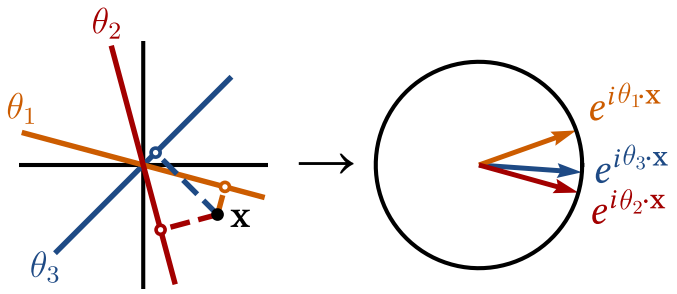



**Figure 2.5:** The Spatial Semantic Pointer (SSP) projection from a low dimensional space ($\boldsymbol{x} \in \mathbb{R}^2$) to higher dimensional space (the set of phasors, $[e^{i\boldsymbol{\theta}_1 \cdot \boldsymbol{x}}, e^{i\boldsymbol{\theta}_2 \cdot \boldsymbol{x}}, e^{i\boldsymbol{\theta}_2 \cdot \boldsymbol{x}}] \in \mathbb{C}^3$). Here only three phasors are used, but in practical applications typically there are dozens to hundreds.

**Definition 2.1** (Spatial Semantic Pointer). *A Spatial Semantic Pointer (SSP) represents a value $\boldsymbol{x} \in \mathbb{R}^n$ in the HRR VSA and is given by the output from a feature map $\phi : \mathbb{R}^n \to \mathbb{R}^d$ (with $d \gg n$),*

$$\phi(\boldsymbol{x}) = \mathcal{F}^{-1}\left\{e^{i\Theta\frac{\boldsymbol{x}}{\ell}}\right\},$$

*where $\Theta \in \mathbb{R}^{d \times n}$ is the* phase matrix *of the representation $\ell \in \mathbb{R}^n$ is the* length scale *or* bandwidth *of the representation, and the exponential is applied element-wise.*

The SSP mapping is a projection of $\boldsymbol{x}$ to a higher-dimensional space parameterized by $\Theta$ and $\ell$. The low-dimensional $\boldsymbol{x}$ (with components scaled by the vector $\ell$) is projected onto a set of $d$ vectors in $\mathbb{R}^n$ space, given by the rows of the phase matrix. These scalars are then stored in the phases of complex exponentials (Figure 2.5).

SSPs can be viewed as a type of combinatorial coding scheme, as $\boldsymbol{x}$ is encoded by the combination of many periodic elements, the phasors, $e^{i\boldsymbol{\theta}_j \cdot \boldsymbol{x}}$. Or, this can be viewed as a continuous-valued version of a residue number system (**kymn2023computing**; Burak et al., 2006). In residue number systems, an integer is represented by its residues modulo several coprime integers. Instead of discrete residues, the SSP phases are continuous and correspond to the alignment of $\boldsymbol{x}$ with the periodic elements defined by $\Theta$ (Figure 2.7A).

As a result of this periodic encoding, SSPs inhabit a vector subspace with a particular structure and geometry. Specifically, SSPs lie on a low-dimensional manifold known as a hyper Clifford torus (Komer, 2020). Each component of an SSP is a phasor that lies on the unit circle in the complex plane. As a result, the product of these $d$ unit circles forms an $n$-torus. This manifold is smooth and locally flat, meaning that in small neighbourhoods, the geometry closely resembles Euclidean space, which supports approximate preservation of local distances.

When SSPs are encoded in the activity of a neural population using the NEF encoding equation (Eq. (2.6)), this geometric structure is mirrored in the state space of the population. However, the firing patterns of individual neurons encoding an SSP depend on the neuron

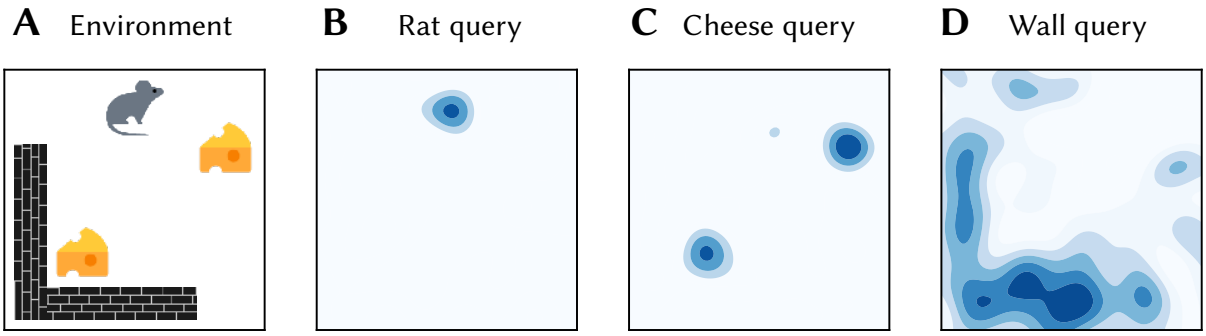



**Figure 2.6: Map encoding using SSPs. (A)** A 2D cartoon environment consisting of a rat, walls, and cheese. Information about the objects and their locations was encoded in a single vector $E$, as per Eq. (2.31). **(B)** The vector $E$ was queried for the location of the rat by approximate unbinding: $E \circledast R^{-1} \approx \phi(\boldsymbol{x}_1)+$ noise. The cosine similarity between the query output and SSP representations of points gridded over 2D space was computed and plotted to produce the above heat map. **(C)** The similarity map obtained from querying the map $E$ for the location of cheese, $E \circledast C^{-1}$. **(D)** The similarity map obtained from querying for the wall area, $E \circledast W^{-1}$.

model parameters (*e.g.*, gain, bias, encoder) as well as the SSP mapping parameters (the phase matrix and length scale).

The parameters of the SSP projection may be sampled randomly, set with heuristics, designed for constructing tuning curves matching those of biological neurons, or learned when SSPs are used as a part of a larger system. There are, however, some constraints on the phase matrix. For $\phi(\boldsymbol{x})$ to be real-valued, $\mathcal{F}\{\phi(\boldsymbol{x})\}$ must have conjugate symmetry, *i.e.*, $\Theta_{m,:} = -\Theta_{d-m,:}$. Additionally, the DC term $\mathcal{F}\{\phi(\boldsymbol{x})\}_0$ (and the Nyquist term $\mathcal{F}\{\phi(\boldsymbol{x})\}_{d/2}$ if $d$ is even) must be real-valued, and, also, all the magnitude of each phasor is one. Thus, the mapping is completely defined by $n\lfloor {}^{d+1}/_{2} \rfloor$ parameters.

SSPs are representations in the HRR VSA, and thus, all HRR operations can be applied to SSPs to manipulate them in meaningful ways. Furthermore, SSP can be used in tandem with the symbol-like representations described in Section 2.3.1. To give a concrete example, consider a simple 2D environment consisting of different objects and landmarks: a rat, two pieces of cheese, and a wall (see Figure 2.6A). The position of the rat, $\boldsymbol{x}_1$, can be encoded as an SSP, $\phi(\boldsymbol{x}_1)$. This, in turn, can be bound (*i.e.*, circularly convolved) with the vector representation of the *concept* of a rat, $R$, to obtain $R \circledast \phi(\boldsymbol{x}_1)$ – this represents a rat at a particular location. Likewise, the vector representation of the concept of cheese can be bound with SSPs encoding their locations, $\boldsymbol{x}_2$ and $\boldsymbol{x}_3$, to obtain $C \circledast (\phi(\boldsymbol{x}_2) + \phi(\boldsymbol{x}_3))$. In this case, the bundle/sum of SSPs is used to represent a set of locations. The wall in the environment covers an area, which can be represented by integrating the SSP encoding over the area, $\iint_A \phi(\boldsymbol{x}) dA$. This can be bound with the vector representation of the concept of walls, $W$. All together, the complete environment can be represented by adding all these object-location vector encodings:

$$E = R \circledast \phi(\boldsymbol{x}_1) + C \circledast \big(\phi(\boldsymbol{x}_2) + \phi(\boldsymbol{x}_3)\big) + W \circledast \iint_A \phi(\boldsymbol{x}) dA \tag{2.31}$$

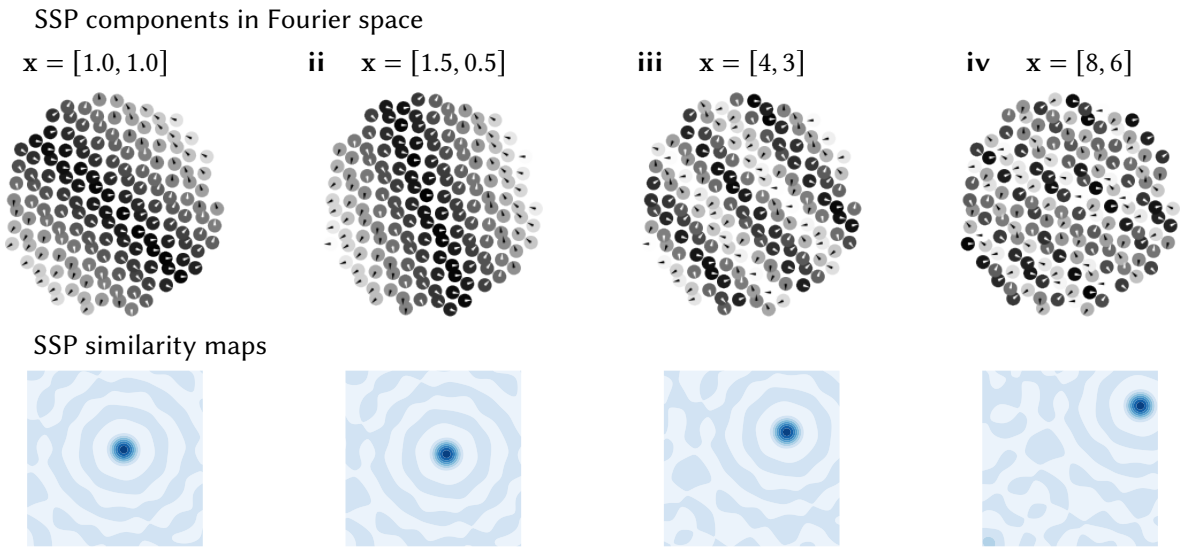



**Figure 2.7: SSPs represent variables via the relative pattern of a set of phasors.** SSPs were constructed to represent four 2-dimensional variables: **i)** $\boldsymbol{x}_0 = [1, 1]$, **ii)** $\boldsymbol{x}_1 = [1.5, 0.5]$, **iii)** $\boldsymbol{x}_2 = [4, 3]$, **iv)** $\boldsymbol{x}_3 = [8, 6]$. **(A)** In the Fourier domain, the SSP $\mathcal{F}\left\{\phi(\boldsymbol{x}_j)\right\} = e^{i\Theta \boldsymbol{x}_j}$ is a vector of phasors. Each phasor is visualized as an arrow within a circle, where the arrow represents the real and imaginary components of $e^{i\boldsymbol{\theta}_k \cdot \boldsymbol{x}_j}$ as a 2D vector. These arrows are positioned at $\boldsymbol{\theta}_k$, the $k^{\text{th}}$ row of the phase matrix $\Theta$. The circle's shading indicates the phase of the corresponding phasor. The variable $\boldsymbol{x}$ is encoded by the collective pattern of these periodic components. Comparing the phasor patterns of nearby points, such as (i) and (ii), reveals small shifts in the collective structure. For more distant points, like (iii) and (iv), phase wrapping (when phases exceed $2\pi$) results in significantly different patterns. **(B)** The similarity plot of each SSP illustrates how closely related the representations are for different $\boldsymbol{x}_j$ values.

This vector was constructed and "queried" for different locations with approximate unbinding. The results of this are shown in Figure 2.6B-D. The high-dimensional SSPs are visualized in this figure via their similarity to neighbouring points, as discussed in the next section.

A useful property of SSPs is that binding in the SSP space is equivalent to addition in the underlying variable space:

$$\phi(\boldsymbol{x}) \circledast \phi(\boldsymbol{x}') = \phi(\boldsymbol{x} + \boldsymbol{x}')\,. \tag{2.32}$$

Thus, it is easy to 'update' SSP representations without re-encoding. For example, the wall encoding used in Figure 2.6 can be shifted with a single binding operation, $\phi(\Delta\boldsymbol{x}) \circledast \int\int_A \phi(\boldsymbol{x}) dA = \int\int_A \phi(\boldsymbol{x} + \Delta\boldsymbol{x}) dA$.

Additionally, binding SSP representations of variables, $\boldsymbol{x}$ and $\boldsymbol{y}$, encoded with different encoding matrices, results in a new SSP representation of $[\boldsymbol{x}, \boldsymbol{y}]^T$:

$$\phi_X(\boldsymbol{x}) \circledast \phi_Y(\boldsymbol{y}) = \mathcal{F}^{-1}\left\{e^{i\Theta_X \frac{x}{\ell_x}} \odot e^{i\Theta_Y \frac{y}{\ell_y}}\right\} = \mathcal{F}^{-1}\left\{e^{i[\Theta_X\ \Theta_Y]\left[\begin{smallmatrix} x/\ell_x \\ y/\ell_y \end{smallmatrix}\right]}\right\}. \tag{2.33}$$

### 2.4.1 Visualizing SSPs with similarity maps

Generally, the SSP representation of a number is similar to the SSP representations of numbers close in value, and dissimilar to numbers far away – with some rippling (Figure 2.7B and Figure 2.8A). Thus, similarity is a tool for visualizing high-dimensional SSP vectors in the original input space, as done in Figure 2.6 and Figure 2.7B. By calculating and plotting the similarities between a specific SSP and a set of SSPs encoding points gridded across $n$-dimensional space, we generate what we refer to as *similarity maps*. For example, an SSP, $\phi'$, representing a 1D variable can be visualized by plotting $k(x) = \phi' \cdot \phi(x)$ over $x$ (see Figure 2.8A). The similarity map of an SSP representing a 2D variables can be plotted as a surface plot or heat map (see Figure 2.8B).

As shown in Figure 2.8A, the dot product between SSPs is a kernel function, $k(\boldsymbol{x}, \boldsymbol{y}) = k(\boldsymbol{x} - \boldsymbol{y}) = \phi(\boldsymbol{x}) \cdot \phi(\boldsymbol{y})$. This kernel's shape is dependent on the sampling distribution of the phase matrix components – in particular, it resembles the inverse Fourier transform of the distribution. Suppose the rows of the phase matrix are independent and identically distributed, sampled from a distribution $P$ with probability density function $p(\boldsymbol{\theta})$, then the expected kernel is

$$\mathbb{E}_{\Theta \sim P}[\phi(\boldsymbol{x}) \cdot \phi(\boldsymbol{x} + \Delta\boldsymbol{x})] = \mathbb{E}_{\Theta \sim P}\left[\frac{1}{d} \overline{e^{i\Theta \boldsymbol{x}/\ell}} \cdot e^{i\Theta(\boldsymbol{x}+\Delta\boldsymbol{x})/\ell}\right], \tag{2.34}$$

$$= \mathbb{E}_{\Theta \sim P}\left[\frac{1}{d} \sum_{j=1}^{d} e^{i\boldsymbol{\theta}_j^T \Delta\boldsymbol{x}/\ell}\right], \tag{2.35}$$

$$= \int_{\mathbb{R}^n} e^{i\boldsymbol{\theta}^T \Delta\boldsymbol{x}/\ell} p(\boldsymbol{\theta}) d\boldsymbol{\theta}, \tag{2.36}$$

obtained using Parseval's theorem and the law of the unconscious statistician. This is the inverse Fourier transform of the distribution $p(\boldsymbol{\theta})$ – *i.e.*, its characteristic function $\varphi_{\boldsymbol{\theta}}(\Delta x)$.[11]

For example, consider an SSP representing a 1D variable with phase matrix elements sampled from a uniform distribution, Unif$[-\pi, \pi]$. In this case, the expectation of the kernel is a sinc function, as shown in Figure 2.8A(i), or a product of sinc functions in the 2D case (Voelker, 2020),

$$\mathbb{E}[k(\boldsymbol{x}, \boldsymbol{x} + \Delta\boldsymbol{x})] \propto \operatorname{sinc}(\Delta x_1) \operatorname{sinc}(\Delta x_2). \tag{2.37}$$

Normally distributed phase matrix rows produce a Gaussian-like kernel (Figure 2.8A(ii)). In practice, given a finite SSP size $d$, the similarity function is a finite sum of cosines that roughly approximates the Gaussian kernel. Other examples of similarity functions given different phase matrix distributions are given in Figure 2.8.

[11] Here we define the characteristic function as $\varphi_{\boldsymbol{\theta}}(\boldsymbol{x}) = \mathbb{E}\left[e^{-2\pi i \boldsymbol{\theta}^T \boldsymbol{x}}\right]$, while some authors use $\varphi_{\boldsymbol{\theta}}(\boldsymbol{x}) = \mathbb{E}\left[e^{i\boldsymbol{\theta}^T \boldsymbol{x}}\right]$. This is just a change of variables.

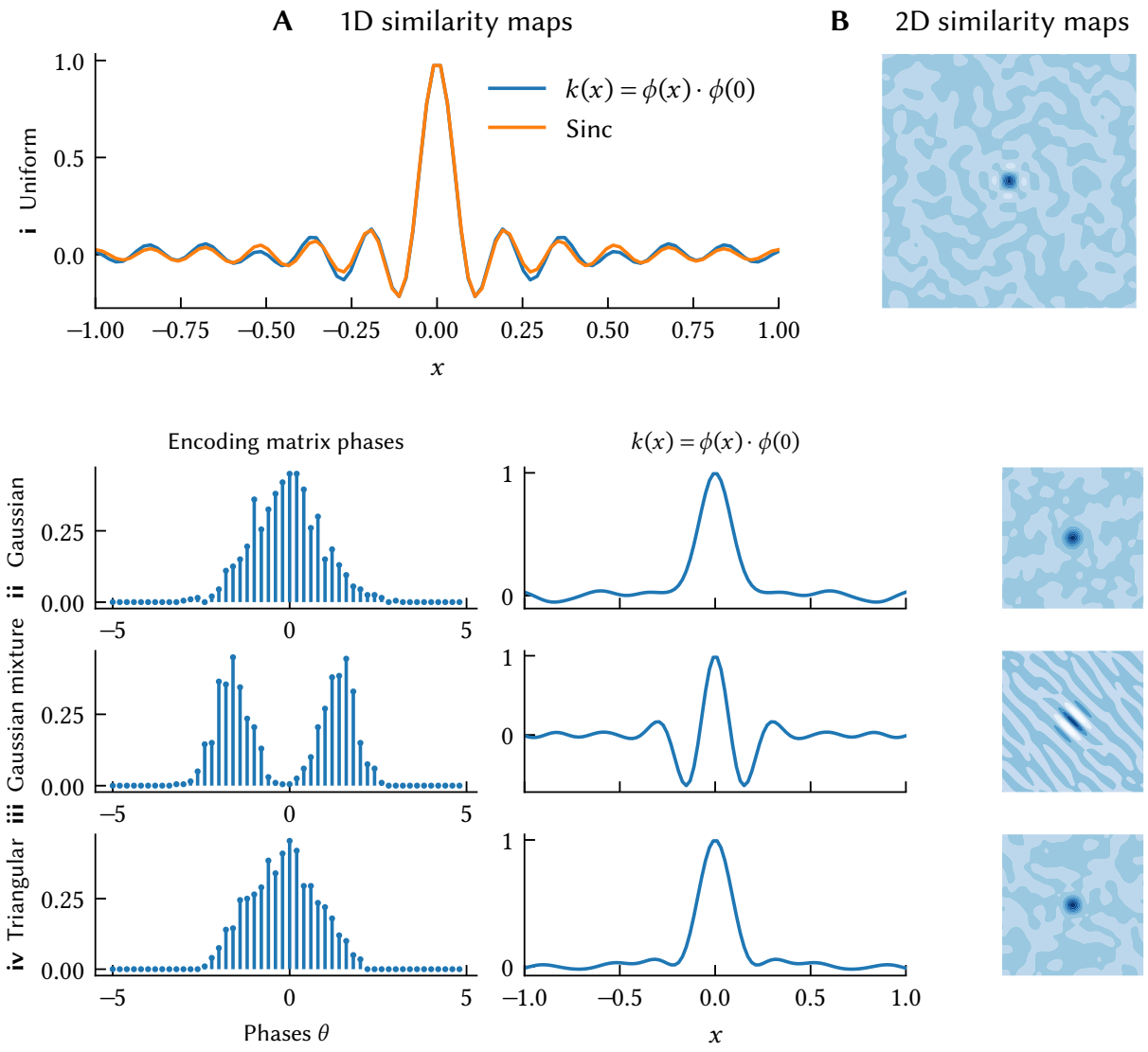



**Figure 2.8:** SSP similarity maps ($k(\boldsymbol{x}) = \phi(\boldsymbol{x}) \cdot \phi(\boldsymbol{0})$) for one and two-dimensional encoding. Each row demonstrates the effect of sampling the components of the phase matrix $A$ from different distributions – a uniform, Gaussian, a mixture of two Gaussisn, and triangular, respectively. The parameters of these distributions were chosen so that the scale of the kernels would be roughly comparable. In particular, they were selected so that $\mathbb{E}[|\boldsymbol{\theta}|]$ would be equal for $\boldsymbol{\theta}$ sampled from each distribution. **(A)** The 1D similarity maps. The first row compares the kernel induced when $M$ is uniformly sampled to the sinc function. The other rows show the distribution of the sampled $\Theta$ components and the corresponding kernel. Also, shown, in the first column, are histograms of the sampled phases. **(B)** The 2D kernels are given the same sampling distributions.

Note that we have assumed i.i.d. phase matrix elements. If instead, the rows of the phase matrix are sampled from different distributions, then the expectation of the kernel is a normalized sum of characteristic functions, $\mathbb{E}_\Theta[k(\Delta\boldsymbol{x})] = \frac{1}{d}\sum_j \varphi_{\theta_j}(\Delta x)$, *i.e.*, the characteristic

function associated with the mixture of phase matrix rows.

### 2.4.2 Decoding SSPs

Mapping to the SSP space is a well-defined process, but direct inversion is not due to the multiple branches of the complex logarithm. Decoding may be formulated as an optimization problem. Given a vector, $\phi \in \mathbb{R}^d$, interpreted as an SSP, the variable it encodes is estimated by

$$\arg\max_{\boldsymbol{x}} \phi(\boldsymbol{x}) \cdot \phi. \tag{2.38}$$

In other words, the decoded $\boldsymbol{x}$ is the location of the maximum value of the similarity function. Later, in Section 2.4.4, we describe the probabilistic interpretation of the similarity function and it becomes clear that the solution is a maximum likelihood estimate (MLE) of $\boldsymbol{x}$ given $\phi(\boldsymbol{x})$. If we attempt to solve for this directly, we get

$$\phi \cdot \phi(\boldsymbol{x}) \propto e^{i\Theta \boldsymbol{x}} \cdot \overline{\mathcal{F}\{\phi\}} = \sum_{k=1}^{d} \mathrm{Re}\left(\mathcal{F}\{\phi\}_k\right) \cos(\boldsymbol{\theta}_k \cdot \boldsymbol{x}) + \mathrm{Im}\left(\mathcal{F}\{\phi\}_k\right) \sin(\boldsymbol{\theta}_k \cdot \boldsymbol{x}), \tag{2.39}$$

which is maximized when $\boldsymbol{\theta}_k \cdot \boldsymbol{x} = \arg\left(\mathcal{F}\{\phi\}_k\right)$. Thus, the solution is given by

$$(\Theta^T\Theta)^{-1}\Theta^T \arg(\mathcal{F}\{\phi\}). \tag{2.40}$$

Decoding is essential in practical applications where the encoded variable must be recovered for interpretation, decision-making, or communication with other systems. However, there is a problem with solving Eq. (2.40) – the angle of a complex number is a multi-valued function, resulting in a non-linear integer optimization problem.

There are several alternative strategies used to decode SSPs effectively. A straightforward approach involves solving Eq. (2.38) by computing similarity with a finite set of candidate SSPs and selecting the value with the highest similarity as the decoded output (Komer, 2020) – what we call the *candidate set method*. This can be neurally realized with an autoassociative network (Stewart et al., 2011). However, this method is computationally expensive and scales poorly. Alternatives include global optimization methods, which can be computationally prohibitive (Bremer et al., 2024). A more efficient solution is the use of a trained neural network, which can decode SSPs rapidly and with fair accuracy, even with a relatively small network size (Komer et al., 2019). However, such networks are specific to the SSP parameters and do not generalize.

Thus, there is a trade-off between speed and accuracy, and the choice of decoding method will depend on the demands of the particular use case. For example, in real-time applications involving communication with external systems, such as sending motion commands to a robot, rapid decoding may be essential. In contrast, for offline analysis or probing the outputs of neural models, accuracy may take precedence over speed.

In our work, decoding is primarily used to interpret the output of neural models trained to encode and manipulate SSPs. Since decoding in our experiments is not typically used in a feedback loop or real-time setting, computational speed is less critical. Thus, we often employ the candidate set method, which is computationally expensive, but accurate and easy to implement.

A related problem to decoding is that of *clean-up*. Clean-up of SSPs becomes necessary when the representations are noisy or corrupted, which can arise from neural representational noise or imprecise dynamics. This is especially relevant in settings where SSPs are manipulated repeatedly, as noise can accumulate over successive binding and unbinding operations. Hopfield networks have been previously used for cleaning up the vector output of VSA operations (Steinberg et al., 2022; Clarkson et al., 2023) and have been applied to clean-up of integer representation in HRRs (**kymn2023computing**).

### *2.4.3 Connection to Random Fourier Features*

Readers familiar with kernel approximation methods may notice a similarity between SSPs and Random Fourier Features (RFFs; Rahimi et al., 2007). Traditional kernel methods leverage the 'kernel trick' to apply kernel functions to data pairs without explicitly mapping the data into the potentially infinite-dimensional reproducing kernel Hilbert space (RKHS). In contrast, RFFs project data into a finite-dimensional feature space that approximates the desired kernel function – using essentially the same mapping as used in SSPs.

**Definition 2.2** (Random Fourier Feature (RFF)). *A Random Fourier Feature is the output of the following random feature mapping:*

$$\varphi(\boldsymbol{x}) = \frac{1}{\sqrt{s}} \left[ a_1 e^{-i\boldsymbol{\omega}_1^T \boldsymbol{x}}, \dots, a_d e^{-i\boldsymbol{\omega}_d^T \boldsymbol{x}} \right]^T = \frac{1}{\sqrt{s}} \boldsymbol{a} \odot e^{-iW\boldsymbol{x}},$$

*where* $W = [\boldsymbol{\omega}_1 \mid \cdots \mid \boldsymbol{\omega}_d]^T \in \mathbb{R}^{d \times n}$ *is called the feature matrix, and* $\boldsymbol{a}$ *are the feature weights.*

The insight behind the RFF mapping comes from Bochner's Theorem (Bochner, 2005), which states that a continuous and shift-invariant function, $k(\boldsymbol{x}, \boldsymbol{x} + \Delta\boldsymbol{x}) = k(\Delta\boldsymbol{x})$, is positive definite if and only if it is the Fourier transform of a non-negative measure $p(\boldsymbol{\omega})$,

$$k(\boldsymbol{x} - \boldsymbol{y}) = \int p(\boldsymbol{\omega}) e^{i\boldsymbol{\omega}^T \Delta\boldsymbol{x}} d\boldsymbol{\omega} = \mathbb{E}_{\boldsymbol{\omega} \sim P} \left[ e^{i\boldsymbol{\omega}^T \Delta\boldsymbol{x}} \overline{e^{i\boldsymbol{\omega}^T \Delta\boldsymbol{x}}} \right]. \tag{2.41}$$

Thus, the dot product of RFFs, $\varphi(\boldsymbol{x}) \cdot \varphi(\boldsymbol{x} + \Delta\boldsymbol{x})$, approximates the kernel $k$ when the feature vectors are sampled from the inverse Fourier transform of $k$. These may be preset by assuming the form of the desired kernel or learned from data.

Although developed independently, with different motivations (finite feature maps for kernel approximations versus continuous feature representations for cognitive systems), RFFs and SSPs use fundamentally the same feature mapping. However, there are differences between

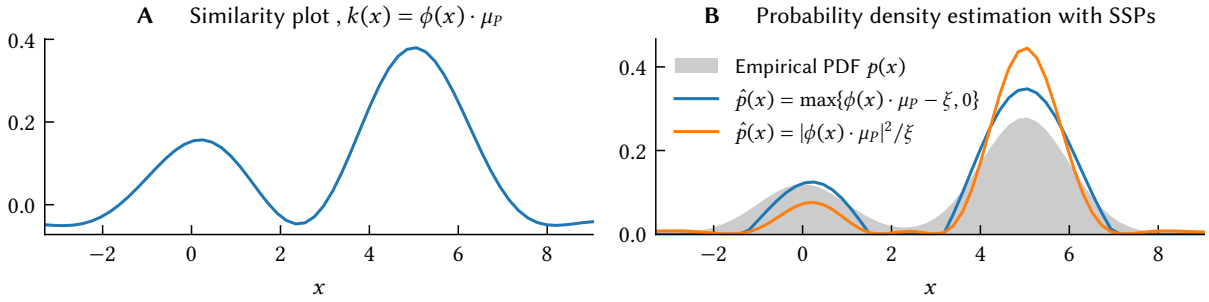



**Figure 2.9: Probability density estimation with a bundle of SSPs (A)** The similarity plot, $k(x) = \phi(x) \cdot \boldsymbol{\mu}_P$, of a normalized SSP bundle, $\boldsymbol{\mu}_P$, of 500 samples points from a distribution $P$, as given in Eq. (2.43). The size of the SSPs is $d = 901$. **(B)** The similarity measure is passed through a function to approximate the probability density function (PDF), $\hat{p}(x)$, of $P$. The output from two transformations is shown here. The first is a shift and rectification of the similarity, and the second is squaring and normalizing of the similarity.

these methods. SSPs are representations in the HRR VSA, and so the algebra of HRRs can be used to compose these representations, as was done on Eq. (2.31). This means that HRRs support feature maps for continuous, discrete, mixed discrete-continuous variables, as well as data structures such as sets (Kommers et al., 2015), sequences (Rasmussen et al., 2011), hash tables (Heddes et al., 2022), words (Jones et al., 2007), and graphs (Laube et al., 2024). This dimensional-preserving language for composition and structure is absent in the RFF literature.

### *2.4.4 Probability representations*

The algebra of cognition defined by VSAs and SSPs has a probabilistic interpretation (Furlong et al., 2022; Furlong et al., 2023). Using the NEF to implement and transform SSPs, spiking neural networks can be constructed to encode probability distributions and perform computations such as entropy and mutual information estimation (Furlong et al., 2021; Furlong et al., 2023).

RFFs are frequently used for kernel density estimation (KDE), a non-parametric method for estimating a PDF, and kernel mean embedding (KME), in which a probability distribution is represented as an element of an RKHS. Empirically, this embedding is computed as the mean of data samples in the RKHS. The KME of a probability distribution $P$ is defined as:

$$\mu_p = \mathbb{E}_{x \sim P}\left[k(\boldsymbol{x}, \cdot)\right] \approx \frac{1}{N} \sum_{i=1}^{N} k(\boldsymbol{x}_i, \cdot), \tag{2.42}$$

where $k(\cdot, \cdot)$ is a kernel function, and $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$ are a set of a set of samples, $\boldsymbol{x}_j \sim P$. The PDF, $p(\cdot)$, of $P$ is estimated as $p(\boldsymbol{x}) \approx \frac{1}{h}\mu_p(\boldsymbol{x})$, where $h$ is the bandwidth parameter of the kernel. KDE has the advantage of being non-parametric and flexible, allowing the estimation of complex and multi-modal distributions.

The KME can be approximated by replacing the exact kernel function, $k(\boldsymbol{x}_i, \cdot)$, with the dot product of RFFs or SSPs. In short, we can average over SSPs encoding data samples

(Figure 2.9A),

$$\boldsymbol{\mu}_P = \frac{1}{N}\sum_{i=1}^{N}\phi(\boldsymbol{x}_i), \tag{2.43}$$

and estimate the PDF of the underlying distribution as

$$\hat{p}(\boldsymbol{x}) \approx f(\boldsymbol{\mu}_P \cdot \phi(\boldsymbol{x})), \tag{2.44}$$

where $f : \mathbb{R} \to [0, 1]$ is a function that maps similarity to a probability density (Figure 2.9B). The SSP (or RFF) approximation can reduce the computational complexity of kernel density estimation – instead of maintaining the $N$ samples points $\boldsymbol{x}_i$ and requiring $N$ kernel evaluations, only a single $d$ vector must be maintained, and the evaluation cost is $\mathcal{O}(d)$. When $N >> d$, the SSP approach to KDE is a more efficient and scalable solution.

Note that, unlike the kernels typically used in KDE, the SSP similarity function, $\boldsymbol{\mu}_P \cdot \phi(\boldsymbol{x})$, can take on negative values. The function $f$ is used to normalize similarity, $\int_{\boldsymbol{x}\in\mathbb{R}^n} f(\boldsymbol{\mu}_P \cdot \phi(\boldsymbol{x}))d\boldsymbol{x} = 1$, and map to non-negative values. Several classes of functions can be used for this.

- *Shifting and rectification (Glad's rule):* Let $f(\boldsymbol{\mu}_P \cdot \phi(\boldsymbol{x})) = \max\{\boldsymbol{\mu}_P \cdot \phi(\boldsymbol{x}) - \xi, 0\}$, where $\xi$ selected to normalize the function. This transformation has previously been used to obtain density estimates from sinc kernels (Glad et al., 2003; Glad et al., 2007). Note that this is simply a ReLU neuron with bias $\xi$, and either weights $\boldsymbol{\mu}_P$ and input $\phi(\boldsymbol{x})$, or vice versa – weights $\phi(\boldsymbol{x})$ and input $\boldsymbol{\mu}$. In the former case, a population of ReLU neurons (with encoders $\boldsymbol{\mu}_{P_j}$) can be interpreted as estimating the probability of input $\phi(\boldsymbol{x})$ under different distributions $\{P_j\}$. In the latter case, we have a population of neurons with encoders $\phi(\boldsymbol{x}_j)$, and activities that represent the probabilities of different sample points $\boldsymbol{x}_j$ under an input distribution represented by $\boldsymbol{\mu}_P$. This transformation was used in past work (Furlong et al., 2021; Furlong et al., 2023; Furlong et al., 2024).
- *Squaring and normalization (Born's rule):* Another possible transform is $f(\boldsymbol{\mu}_P \cdot \phi(\boldsymbol{x})) = (\boldsymbol{\mu}_P \cdot \phi(\boldsymbol{x}))^2/\xi$, where $\xi$ is set to normalize the similarity (Furlong et al., 2021). Note that in quantum mechanics, the squared modulus is used to map complex probability amplitudes into probability densities, which is why we refer to this as the Born rule. We shall primarily use this transformation due to its simplicity.

Regardless of the choice of transformation, using SSPs for distribution approximation leads to new interpretations of standard VSA operations: bundling corresponds to adding new data points to a mean embedding, functioning as a form of belief update; binding enables multivariate KDE and the construction of joint distributions; and unbinding, its inverse, is analogous to conditioning (refer to Furlong and Eliasmith (2022) for more details on the probabilistic interpretation of these operations).

The ability to model probability distributions and probabilistic statements with VSAs and SSPs supports biologically relevant computations while maintaining a compact and structured

representation. This approach has been applied to tasks such as novelty detection in a model of the insect olfactory system (Stewart et al., 2023). Furthermore, the compositionality of VSAs and SSPs allows the integration of probabilistic information with structured data representations. This capability is leveraged in simultaneous localization and mapping (SLAM) and reinforcement learning (RL) models presented in later chapters.

## 2.5 Conclusion

In this chapter, we reviewed the foundational methods of the NEF and VSAs, particularly HRRs, and introduced an extension of HRRs to continuous variable encoding through SSPs. In this thesis, we employ the NEF to construct functional SNNs, where algorithms are defined by transformations and dynamics within a latent space, and realized in SNNs using the three NEF principles. SSPs play a central role as the basis for spatial representation within our navigation models, and we leverage the cognitive algebra of HRRs to construct models that manipulate and bind spatial information.

In the next chapter, we explore methods for setting the parameters of the SSP mapping, going beyond random variable assignments (as often done with RFFs) by incorporating the findings from research on grid cell coding efficiency. We derive the dynamics of SSPs, demonstrate that this spatial representation inherently aligns with neural oscillatory behaviour, and provide a foundation for further model development in spatial navigation.

# Chapter 3

# Grid Cells & Path Integration

> The space circuit of the mammalian hippocampus and entorhinal cortex is one of the first nonsensory "cognitive" functions of the cortex that may be understood in mechanistic detail within a not too distant future. Understanding how space is created in this circuit may provide important clues about general principles for cortical computation, extending well beyond the domain of space into the realm of thinking, planning, reflection, and imagination.
>
> — May-Britt Moser and Edvard I. Moser
> *Understanding the Cortex through Grid Cells*

**Summary**

This chapter focuses on spatial representation and path integration using SSPs. We introduce HexSSPs, a structured encoding scheme inspired by grid cell activity, which efficiently represents spatial variables while preserving geometric properties such as local conformal isometry. We demonstrate the scalable coding properties of HexSSPs. Additionally, we derive the dynamics of SSPs and present a path integration model, SSP-PI, which incorporates velocity-controlled oscillators stabilized with attractor dynamics, thus linking SSPs to neural oscillator activity. The model is tested in 2D, 3D, and higher-dimensional spaces, and its applicability extends to temporal integration as well.

**Material in this chapter from prior publications**

Material in this chapter has been adapted from several prior conference and journal publications. The generation of grid cells and place cells using Spatial Semantic Pointers (SSPs) was first presented at the *Cognitive Science Society Conference* (Dumont et al., 2020). The dynamics of SSP were derived and used to construct a model of path integration in Dumont et al. (2022), presented at the *Cognitive Science Society Conference*, and Voelker et al. (2021), a paper published in *Neural Computation*. Finally, a model of spatial-temporal integration was included in Dumont et al. (2023b), published in *Brain Sciences*.

## 3.1 Introduction

Self-localization is an essential component of any complex navigation system – be it biological or artificial. Path integration (PI) – also known as 'dead reckoning' – is one of the simplest forms of self-localization. This is the process of integrating idiothetic cues[1], obtained from a variety of sensory systems, to maintain an estimate of one's location in space (relative to some starting location). PI is critical to many navigation strategies and is an ability observed in humans and many other animals. It is the foundation of vector-based and map-based navigational strategies. With PI, animals can self-localize without allocentric spatial cues, though this does incur significant error over time and path length (Mittelstaedt et al., 1982; Etienne et al., 1996; Kearns et al., 2002; Stangl et al., 2020).

Beyond navigation, PI is an important component of larger systems capable of advanced reasoning, planning, and cognition. The results of PI are used in downstream tasks, like creating cognitive maps, associating memories with locations, and planning trajectories for exploration (Savelli et al., 2019; Buzsáki, 2005). Maintaining homing vectors to important locations (*e.g.*, shelter or food) while moving through an environment requires updating those vectors using self-motion information. Building a cognitive map of an environment also requires tracking one's position relative to landmarks. Furthermore, there is evidence that the same neural mechanisms underlying spatial representation and PI may be used in the integration and mapping of non-spatial features and concepts – this includes odours, sounds, visual features, and even position in a social hierarchy (Constantinescu et al., 2016; Bao et al., 2019; Garvert et al., 2017). This has led many to suggest that the methods by which the brain encodes and integrates spatial information are, in fact, instantiations of fundamental mechanisms underlying perception and cognition (Chen et al., 2022; Mok et al., 2019).

Although computational modelling of PI has been well-studied, there is still a gap in our understanding of how the output of a neural PI model may be used for a variety of downstream functions – such as cognitive mapping, which may involve integrating PI output with symbol-like representations of objects, landmarks, and other discrete features. In this chapter, we approach the problem of PI using the Semantic Pointer Architecture (SPA) and, more precisely, the SSP encoding of continuous variables. The algebra of the SPA provides a set of neurally feasible operations, which can be used to perform vector computations on self-location estimates and to combine such estimates with symbol-like representations (to create landmark maps) and temporal representations (to encode trajectories and spatial memories).

The contributions of this chapter are as follows:

- We show that SSPs can be modified to generate representations at the neuron level that resemble those found in the brain (both grid cells and place cells). We show that these

[1]This includes vestibular cues, proprioceptive cues, and motor commands. Essentially, this is information about body motion and orientation and does not include external environmental cues.

grid-cell-ike SSPs are significantly more accurate for encoding cognitive maps than the randomly generated SSPs used in prior work.

- We derive the dynamics of SSPs and show that this naturally produces oscillators. In fact, we recover the mechanisms underlying a popular class of PI models.
- With the addition of attractor dynamics, we present a novel neural PI model called SSP-PI. This is a hybrid oscillator-interference and continuous attractor model and can be used to integrate continuous variables of any dimensionality. We demonstrate its function in 2D and 3D spaces.
- A description and example of temporal integration and trajectory encoding is presented.

The structure of this chapter is as follows. In Section 3.2 we review the neuroscience of spatial representation and path integration along with past models of path integration and grid cells. In Sections 3.3-3.3.3, we propose a novel method for modelling grid and place cells using SSPs. In Sections 3.4-3.4.4, we derive the dynamics of SSPs and present a spiking neural implementation. Furthermore, in Section 3.5, we demonstrate that time can be included as another dimension of integration in SSPs and thus used to model trajectory memory. Finally, the chapter ends with a discussion and a conclusion (Sections 3.6 and 3.7).

## 3.2 Background

### *3.2.1 Neuroscience*

Tolman (1948) observed that rats learning to navigate through mazes discovered shortcuts and proposed that they use a "cognitive map". Evidence for the idea that animals have an internal allocentric representation of space has accumulated, specifically from electrophysiological recordings in the hippocampal-entorhinal circuit, where several families of spatially sensitive neurons have been discovered. The first such discovery was that of place cells in the hippocampus (O'Keefe et al., 1971; O'Keefe et al., 1978). These neurons fire selectively at specific locations in an environment (Figure 3.1B). Some place cells will fire at consistent locations, even with limited external sensory information (Quirk et al., 1990), but most have activity dependent on landmarks, environment features, task demands, and even rewards (Geiller et al., 2017; Xiao et al., 2020).

The hippocampus is known to play an important role in episodic memory (Eichenbaum, 2017; Howard, 2017). This is not incongruent with the idea of the hippocampus as a cognitive map – episodic memory can be thought of as mapping events in space and time (Schiller et al., 2015). The specificity of context-dependency in place fields supports both the differentiation and generalization between contexts, a critical feature for organizing memory and supporting learning from past experiences. This has been described as a 'hierarchical coding scheme' for memory, allowing the hippocampus to encode information across a spectrum – from broad

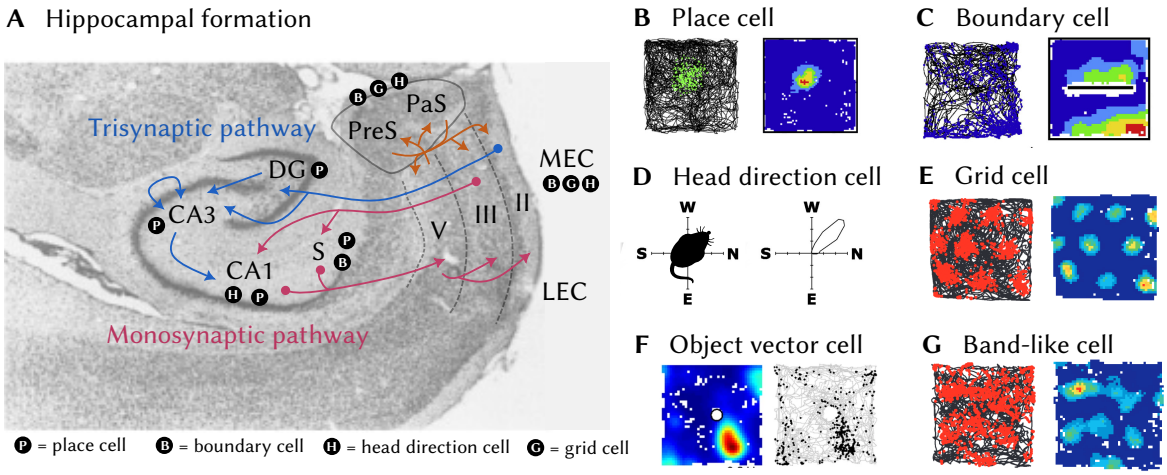



**Figure 3.1: The hippocampal formation and spatially sensitive neuron types. (A)** The background image is a horizontal section through the hippocampal formation (Figure 1a in Coutureau et al., 2009), which includes the dentate gyrus (DG), hippocampal fields CA1 and CA3, and the subiculum (S). Also shown is the parahippocampal region – presubiculum (PreS) and parasubiculum (PaS) – and the medial and lateral entorhinal cortex (MEC and LEC). The entorhinal cortex (EC) has six layers, though only layers II, III, and V are labelled here. The locations particular cell types have been found are denoted by labelled black circles. Major projections between and within these regions are also depicted, inducing the trisynaptic and monosynaptic pathways. In the trisynaptic pathway, information flows from the EC to the DG and CA3, and then onto CA1. While the monosynaptic pathway refers to the direct projections from the EC to CA1. The deep layers of the EC project to superficial layers. The PreS and PaS project to all layers of the MEC (Canto et al., 2012). Diagram inspired by Figure 1c in Witter et al. (2006). **(B)** An example of the firing pattern and receptive field of a CA1 place cell. **(C)** A boundary/border cell. **(D)** A head direction cell – such cells can be found in the parahippocampus, all layers of the MEC (Sargolini et al., 2006), and in several regions outside the hippocampal formation. **(B-D)** are adapted from Nyberg et al. (2022). **(E)** A grid cell found in the superficial layers of the MEC, though grid cells have been found in the parahippocampus as well (Boccara et al., 2010). **(F)** An object vector cell in the MEC (Høydal et al., 2019). **(G)** A spatially periodic cell with a band-like pattern; such patterns have recorded in the MEC, PreS, and PaS. **(E,G)** are adapted from Krupic et al. (2012).

cognitive maps to context-specific representations and individual episodic memories (Smith et al., 2006).

Other spatially sensitive neurons have since been discovered. Head direction cells (Figure 3.1D), sensitive to an animal's heading direction, are found in many areas of the brain (Ranck, 1985; Taube et al., 1990; Peyrache et al., 2015), including the medial entorhinal cortex (MEC; Sargolini et al., 2006). Boundary cells, found in the medial entorhinal cortex (MEC) and parasubiculum, fire when an animal is near the boundaries of an environment (Figure 3.1C; Solstad et al., 2008). Also in the MEC are object vector cells (OVCs), with firing fields dependent on distance and direction to any objects in an environment (Figure 3.1F; Høydal et al., 2019). These are different than the landmark-vector place cells found in the hippocampus, which fire at a certain displacement from objects of a particular identity or with particular features (Deshmukh et al., 2013).

Central to this discussion are grid cells, neurons found in the MEC that fire at hexagonally tiled points in an environment (Figure 3.1EE). These neurons are found in many mammals, including rats (Hafting et al., 2005), bats (Yartsev et al., 2011), monkeys (Killian et al., 2012), and humans (Doeller et al., 2010). The phase, scale, and orientation of the hexagonal lattice varies among grid cells, though there are topographical trends. The spacing of the tiled receptive fields is organized along the dorso-ventral axis of the MEC, from small to large-scale modules (Brun et al., 2008), while patterns of different phases are found scattered among each other. There are also trends across layers of the MEC; neurons with more 'pure' grid firing patterns are typically found in the superficial layers (II/III), while in deeper layers (V), grid-pattern neurons that only fire when an animal has a particular head direction (called conjunctive cells) are more prevalent (Sargolini et al., 2006).

Interestingly, the characteristics of the receptive fields of the grid cell – phase, scale, and orientation – are generally consistent (relative between neurons) in different environments (Fyhn et al., 2007). This is quite unlike place cells. When moving to a new environment or after large environment changes, global remapping takes place in place cells: distinct subsets of place cells are active in different environments, and neurons found in both subsets fire at locations not correlated with their previous place fields (Muller et al., 1987; Leutgeb et al., 2004). Grid cell firing patterns will translate and/or rotate, but in a consistent manner across the population (Fyhn et al., 2007). When environment changes are minor, partial remapping occurs. Only a fraction of place cell activity is altered (Anderson et al., 2003) and grid cells may not remap at all (Fyhn et al., 2007). The entorhinal cortex thus seems to represent information about the structure of space in a more general, less-environment dependent manner[2] compared to the hippocampus, which seems to use space to contextualization information and memory.

Spatial representation in the entorhinal cortex and hippocampus is not limited to just 2D

[2]Firing fields of grid cells can reflect environment geometry – this is believed to be caused by interactions of grid and boundary cells. Additionally, grid cell patterns can reset or re-anchor in a task-dependent fashion (Peng et al., 2023). So to say they are completely environment agnostic would be incorrect (Keinath et al., 2018). However, they are certainly less environment-dependent than representations in the hippocampus.

planar space, but also 3D volumetric space. Place cells (Yartsev et al., 2013), boundary cells (Ginosar et al., 2021), head direction cells (Finkelstein et al., 2015), and even grid cells (Ginosar et al., 2021; Grieves et al., 2021) generalize to three dimensions. However, 3D 'grid cells' do not strictly follow a regular lattice (*e.g.*, a face-centred cubic or hexagonal close-packed structure) but instead have more irregularly gridded receptive fields.

Grid cells are generally considered to play a vital role in PI. In the MEC, they live alongside neurons sensitive to head-direction and speed (Sargolini et al., 2006), which encode information required for PI. In Gil et al. (2018), genetic modifications disrupting grid cells resulted in impaired path integration performance, supporting this hypothesis. It has been suggested that the function of grid cells may be to encode a representation of space robust to noise and to help error-correct integration (Sreenivasan et al., 2011). We return to this hypothesis in more detail in Section 3.4.5

Recent experiments have found hexadirectional activity (*i.e.*, 'grid cell-like') during non-spatial tasks, indicating that the neural mechanisms behind self-location representation may be a general mechanism for encoding continuous variables (Constantinescu et al., 2016; Staudigl et al., 2018; Bao et al., 2019).[3] This evidence has been used to support the idea of conceptual spaces – a generalization of cognitive maps applied to concepts (Bellmund et al., 2018). Cognitive representation in a geometric fashion is, of course, central to the SPA; in this framework, semantic pointers (*i.e.*, compressed vectors) are used to represent anything from low level visual features to high level concepts. Cognitive semantic pointers can be bound together to build syntactic structures which live in a continuous vector space where distance represents concept similarity.

### *3.2.2 Models*

Many computational models of grid cells and their possible role in PI have been proposed since their discovery. Models generally to fall into one of three categories: continuous attractor neural network (CANN) models (Samsonovich et al., 1997; Tsodyks, 1999; Conklin et al., 2005; Burak et al., 2009; Agmon et al., 2020), oscillator-interference (OI) models (O'Keefe et al., 2005; Burgess et al., 2007; Hasselmo et al., 2007; Orchard et al., 2013), and deep learning models of grid cells (Banino et al., 2018; Cueva et al., 2018; Sorscher et al., 2019; Xu et al., 2022).

*Continuous Attractor Networks.* In CANN models, PI is performed by a recurrently connected neural sheet whose dynamics sustains a single Gaussian-like activity bump that represents self-location (Samsonovich et al., 1997; Conklin et al., 2005; Fuhs et al., 2006; McNaughton et al., 2006). Particular connectivity is used to achieve this – excitation from nearby neurons to maintain the bump and inhibition from distant neurons to prevent multiple bumps forming. The activity bump moves across the sheet in response to velocity signals. Typically, periodic

[3]These neurons are found outside the MEC, in areas including the orbitofrontal cortex (OFC), anterior medial temporal lobe (ATL), ventromedial prefrontal cortex (vmPFC) and anterior and posterior cingulate cortex (ACC and PCC).

boundary conditions are added to address the question of how the bump should be shifted when input drives it off the edge of the neural sheet. When these boundary conditions are set to produce a twisted torus topology, hexagonally tiled activity patterns emerge (Guanella et al., 2007). Additionally, CANN models are quite accurate, capable of integrating for up to 10 minutes when neural noise is sufficiently limited (Burak et al., 2009).

*Oscillatory Interference Networks.* In OI models, grid cell firing patterns are the result of interference between velocity controlled oscillators (VCOs) – these are oscillators whose frequency is modulated by a velocity signal. In this setup, a location estimate is encoded in the phase differences between oscillators. In first-generation OI models, it was theorized that the intrinsic oscillation of membrane potentials in grid cell dendrites increases or decreases from a baseline somatic oscillation (corresponding to theta rhythm) in response to velocity input (Burgess et al., 2007; Hasselmo et al., 2007). One problem with such models is the high degree of noise in membrane oscillations, which would result in hexagonal firing patterns deteriorating in just seconds (Zilli et al., 2009). In second generation OI models, more stable VCOs are modelled with networks of recurrently connected neurons (Blair et al., 2008; Zilli et al., 2010; Orchard et al., 2013).

Models have also been proposed that use a CANN to correct VCOs, reducing phase drift and error accumulation (Bush et al., 2014). Given that OI networks model many oscillators, their collective activity lies on a manifold with a toroidal topology, much like CANNs. However, these classes are not without differences. In OI models, neurons with striped or banded firing patterns are used to model VCOs and grid cells are generated from their output – adding an intermediate step that CANNs lack.

*Optimality of Grid Cells.* CANN and OI models provide explanations of how grid cell receptive fields may emerge as a result of network dynamics and their possible role in a PI circuit. Additionally, such models have made predictions that have subsequently been confirmed or supported experimentally (Conklin et al., 2005; Fyhn et al., 2007; Krupic et al., 2012; Gardner et al., 2022). For example, Conklin et al. (2005) "was the first modelling work to link [toroidal CANNs] to the firing patterns of cells that would later be called grid cells" (Zilli, 2012). However, there is still the question of *why*, out of all possible spatially receptive fields, brains learn a hexagonally latticed representation. Grid cells seem to be ubiquitous across mammals and may also be involved in representing non-spatial continuous variables, suggesting that there must be some compelling reason for their existence. It is theorized that a grid cell coding of space is especially robust to noise and has an exponential coding capacity, making them superior to classical population codes (Sreenivasan et al., 2011). In particular, Mathis et al. (2015) argue that the Fisher information of a population encoding of space is maximized (and decoding error is minimized) by latticed activity with the highest packing ratio – *i.e.*, a hexagonal lattice for 2D and a face-centred cubic lattice for 3D spatial representations. The multi-scale nature of the grid code is also a key aspect of its coding efficiency (Wei et al., 2015; Vágó et al., 2018).

*Learning Grid Cells.* A recent line of research has explored this question of optimality by using deep learning to train, rather than engineer, recurrent path integration networks and

have reported the emergence of grid cell-like activity (Banino et al., 2018; Cueva et al., 2018; Sorscher et al., 2019; Xu et al., 2022). In Dordek et al. (2016), hexagonal grid activity emerges from applying non-negative pca (NN-PCA) to a matrix of place cell activations. Inspired by this, Sorscher et al. (2019) trained a network to perform path integration with a non-negativity constraint and recovered grid cells. They also proved that the optimal receptive fields for decoding place cells[4] are approximately combinations of Fourier modes. Furthermore, they assert that any non-linear constraint or regularization (such as non-negativity) promotes the dominance of Fourier mode combinations that result in hexagonal interference patterns during optimization. Xu et al. (2022) and Schaeffer et al. (2024) demonstrated the emergence of grid cells with different path integrator constraints, notably through the inclusion of a term that promotes learned vector embeddings to adhere to conformal isometry.

Gridded activity also emerges in predictive learning as an optimal basis for constructing successor representations – a function that predicts future state visitations (discussed more in Chapter 5) – over a random walk in a 2D environment (Stachenfeld et al., 2017). More generally, Fourier modes are an optimal basis for successor representations over any environment with a translation-invariant state transition function (Yu et al., 2021).

Deep learning models of grid cells have faced criticism, as the grid cells identified are often not truly hexagonally latticed and results depend significantly on the particulars of the problem set-up and hyper-parameters. The learned 'grid cells' from Banino et al. (2018) have since been reported to have grid scores indistinguishable from low-pass filtered noise (Sorscher et al., 2023). Furthermore, results from Sorscher et al. (2019) crucially depend on place cells receptive fields being modelled by difference-of-softmaxes with fixed widths (Schaeffer et al., 2022).

## 3.3 Hexagonal SSPs

The SPA posits that Semantic Pointers (SPs) play a crucial role in cognition. These pointers are composable, compressed neural representations that can function as symbols while maintaining underlying semantic information. The SPA relates high-level symbolic descriptions of cognition to vector spaces to low-level neural activity. Its strength lies in the ability to switch between these levels of abstraction while also distinguishing them from one another. While this is appealing theoretically, to truly test these ideas we must consider their practical implementation – in particular, the construction, manipulation, and application of SPs.

Holographic Reduced Representations (HRRs) are a choice of Vector Symbolic Algebra (VSA) with desirable properties that may be used in the SPA. Beyond supporting high-level symbolic, this algebra also provides a natural way to represent continuous variables through fractional binding as Spatial Semantic Pointers (SSPs). We have already introduced SSPs, but we have not discussed how to set the phase matrix that parameterizes the SSP embedding. In this section, we address this question and demonstrate that SSPs can be used to model

[4]Given several assumptions: periodic spatial boundary conditions, and homogeneous and isotropic place cell receptive fields. Note that these assumptions are not consistent with recordings from real place cells.

grid cells and place cells – linking symbolic descriptions and continuous variables to neural representations.

### 3.3.1 Grid cells with SSPs

Grid cell activity patterns have been shown to be an efficient coding basis robust to noise (Sreenivasan et al., 2011). The activity of a grid cell is proportional to a sum of three plane waves whose wave vectors are 120° apart (Figure 3.2). Specifically, activity over space $\boldsymbol{x}$ of grid cell $j$ in a population is given by

$$a_j(\boldsymbol{x}) \propto \sum_{k=1}^{3} e^{i\boldsymbol{v}_k^{(j)}\cdot\boldsymbol{x}} + e^{-i\boldsymbol{v}_k^{(j)}\cdot\boldsymbol{x}} \tag{3.1}$$

$$\text{where} \quad \sum_{k=1}^{3} \boldsymbol{v}_k^{(j)} = 0, \quad |\boldsymbol{v}_k^{(j)}| = |\boldsymbol{v}_{k'}^{(j)}| \quad \forall j, k, k'. \tag{3.2}$$

where $\boldsymbol{v}_k^{(j)}$ for $k = 1, 2, 3$ are the three vectors that are 120° apart. The magnitude of the vectors, $|\boldsymbol{v}_k^{(j)}|$, will determine the resolution of the grid: the distance between nearest peaks in the grid pattern will be $4\pi/\sqrt{3}|\boldsymbol{v}_k^{(j)}|$.

Given the existence of grid cells in nature and their efficient coding properties, we would like to model such cells with SSPs. This desiderata will constrain the phase matrix that parameterizes the SSP mapping. To obtain grid cell responses in a population representing an SSP, the activity matrix $A$ must have rows proportional to linear combinations of plane waves, as in Eq. (3.1). First, consider the activity of a single neuron with an encoder $\boldsymbol{e}_j$ set as an SSP representing a fixed location $\boldsymbol{x}_j$ (the preferred firing location of the neuron). The $\langle \boldsymbol{e}_j, \phi(\boldsymbol{x}) \rangle$ part of the activity of neuron $j$ will be

$$\boldsymbol{e}_j \cdot \phi(\boldsymbol{x}) = \frac{1}{d} e^{i\Theta\boldsymbol{x}_j/\ell} \cdot e^{i\Theta\boldsymbol{x}/\ell} = \frac{1}{d} \sum_{k=0}^{d-1} e^{i\boldsymbol{\theta}_k^T(\boldsymbol{x}_j+\boldsymbol{x})/\ell}. \tag{3.3}$$

where $\Theta$ is the phase matrix and $\ell$ the length scale of the SSP representation. The neuron's tuning curve will be proportional to the SSP similarity function and thus resemble the interference pattern of $d$ plane waves, each of whose direction of propagation is given by a row of the phase matrix, $\Theta$. We want to set these parameters so that this equation is equal to Eq. (3.1). This occurs when the SSP dimensionality is $d = 7$, to capture the 3 plane waves and a zero-frequency term, and when

$$\Theta = \begin{bmatrix} \boldsymbol{V} \\ \boldsymbol{0} \\ -\text{flip}(\boldsymbol{V}) \end{bmatrix} = \begin{bmatrix} \boldsymbol{v}_1^T \\ \boldsymbol{v}_2^T \\ \boldsymbol{v}_3^T \\ \boldsymbol{0}^T \\ -\boldsymbol{v}_3^T \\ -\boldsymbol{v}_2^T \\ -\boldsymbol{v}_1^T \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ -\frac{1}{2} & \frac{\sqrt{3}}{2} \\ -\frac{1}{2} & -\frac{\sqrt{3}}{2} \\ 0 & 0 \\ \frac{1}{2} & \frac{\sqrt{3}}{2} \\ \frac{1}{2} & -\frac{\sqrt{3}}{2} \\ -1 & 0 \end{bmatrix}. \tag{3.4}$$

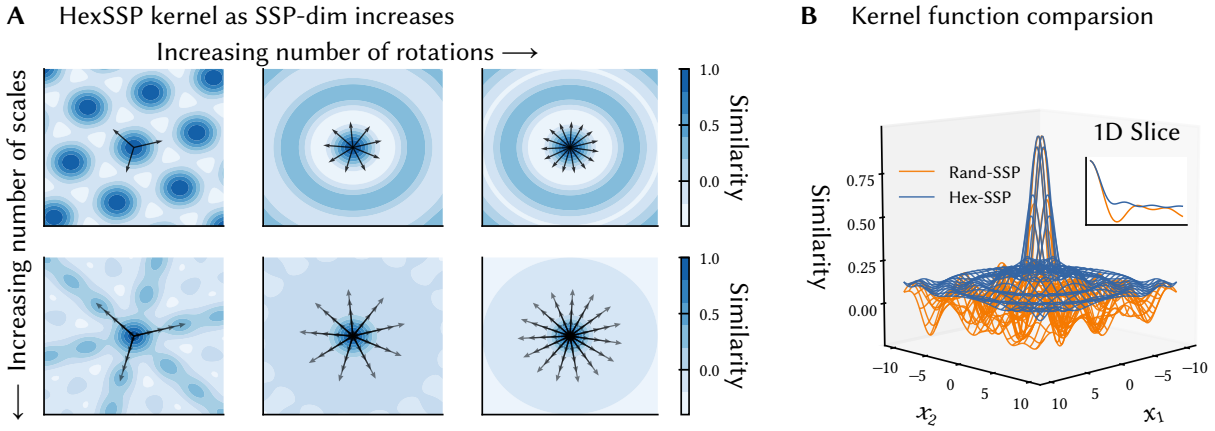



**Figure 3.2: SSP Kernel Functions.** A kernel function can be approximated by the dot product (*i.e.*, similarity) between SSP vectors. **(A)** Kernel function induced by HexSSPs representing points in a 2D space as a heat map. The initial 3-component simplex (top left) produces hexagonal interference patterns like those found in grid cell neurons. Adding rotated simplices (top right) produces a surface with a central peak surrounded by rings. Increasing the number of rotated simplices (bottom left) results in a starburst pattern. Adding many rotated and scaled simplices (bottom right) reduces the magnitude of the surrounding rings and increases smoothness and robustness to noise. **(B)** RandSSPs result in a sinc kernel function, while HexSSPs produce a sum of sinc functions, making local maxima and minima more shallow.

The first three rows of the phase matrix are the vertices of an equilateral triangle, $\boldsymbol{V}$, followed by the zero-frequency and symmetric components. The interference of the three plane waves, each 120° apart, will be a hexagonal pattern. Scaling or rotating the rows of $\Theta$ will affect the orientation and resolution of the hexagonal activity patterns exhibited by the neurons representing $\phi(\boldsymbol{x}) = \mathcal{F}^{-1}\left\{e^{i\Theta \boldsymbol{x}}\right\}$.

We can generalize this to produce gridded activity in a population encoding an SSP representation of some higher $n$-dimensional space by setting the rows of $\Theta \in \mathbb{R}^{2(n+1)+1 \times n}$ with the vertices of a regular $n$-simplex. The simplex vertices are determined by minimizing the relation $\sum_{i=1}^{m+1} \sum_{j=1, i \neq j}^{m+1} \boldsymbol{v}_i \cdot \boldsymbol{v}_j$. The resulting gridded activity will follow a hexagonal lattice when $n = 2$ and a face-centered cubic lattice when $n = 3$, which are the densest possible lattice packings of circles and spheres respectively. In higher dimensional spaces, lattices based on regular simplices do not yield the highest packing densities and solutions to the close packing problem are only known for certain values of $n$. However, this still provides a simple method for constructing gridded neural tuning curves in higher-dimensional spaces.

### *3.3.2 Multi-scale grid cells*

A gridded similarity map means that the SSP cannot represent variables uniquely. We would like the similarity map of the SSP to have a single bump, while still being able to produce gridded activity patterns in neurons. This can be done by using a higher dimensional SSP

**Algorithm 1: HexSSP phase matrix construction.** $n$ is the dimensionality of the variables being encoded; $\mathscr{H} \subseteq \mathbb{R}$ is a set of scaling values; and $\boldsymbol{V}$ is the matrix whose rows are the vertices of a regular $n$-simplex.

**procedure** HEX-SSP($n, \mathscr{H}$)
  $\boldsymbol{V} \leftarrow \begin{bmatrix} - & \boldsymbol{v}_1^T & - \\ & \vdots & \\ - & \boldsymbol{v}_{n+1}^T & - \end{bmatrix}$ ▷ The vertices of a regular $n$-simplex
  $\Theta_{\text{half}} \leftarrow \text{stack}\left[\left\{h\boldsymbol{V}\boldsymbol{R} \,|\, h \in \mathscr{H}, \boldsymbol{R} \sim \text{SO}(n)\right\}\right]$
  $\Theta \leftarrow \begin{bmatrix} \Theta_{\text{half}} \\ 0 \;\ldots\; 0 \\ -\text{flip}(\Theta_{\text{half}},\ \text{axis} = 0) \end{bmatrix}$
  **return** $\Theta$
**end procedure**

projection obtained by stacking scaled and rotated versions of the phase matrix in Eq. (3.4). An SSP constructed in this manner is referred to as a HexSSP, while SSPs constructed by uniformly sampling phases are called RandSSPs. The procedure for constructing a HexSSP phase matrix is given by Algorithm 1.

Composing multiple rotations and scalings of $\boldsymbol{V}$ changes the kernel function. Figure 3.2 shows the effect of increasing the dimensionality of a HexSSP, by increasing the number of scales (moving downward in Figure 3.2AA) and rotations (moving rightward in Figure 3.2A) of the starting simplex matrix $\boldsymbol{V}$. The resulting interference pattern has shallower ripples compared to RandSSPs (Figure 3.2B).

**HexSSP Encoders**. A neural population can be used to encode a HexSSP. However, if SSP encoders are used the neurons will not have grid cell-like activity. Instead, neurons will resemble place cells. To recover grid cells, we must select neural encoders that 'pick-out' single simplices contained in the phase matrix.

We begin by defining the $i^{th}$ simplex SSP

$$\phi^{(i)}(\boldsymbol{x}) = \mathscr{F}^{-1}\left\{ e^{is_i \begin{bmatrix} \boldsymbol{V} \\ \boldsymbol{0} \\ -\text{flip}(\boldsymbol{V}) \end{bmatrix} R_i \boldsymbol{x}} \right\} \in \mathbb{R}^{2(n+1)+1} \tag{3.5}$$

We can construct a matrix $\bar{B}_i$ that, when applied to $\phi^{(i)}(\boldsymbol{x})$ in the Fourier domain, will project it to a $d$-dimensional vector that contains $\mathscr{F}\left\{\phi^{(i)}(\boldsymbol{x})\right\}$ in the same position as it is in $\mathscr{F}\{\phi(\boldsymbol{x})\}$ and zeros elsewhere. The sum of all such projections will produce the complete HexSSP vector,

$$\phi(\boldsymbol{x}) = \mathscr{F}^{-1}\left\{ \sum_i \bar{B}_i \mathscr{F}\left\{\phi^{(i)}(\boldsymbol{x})\right\} \right\}. \tag{3.6}$$

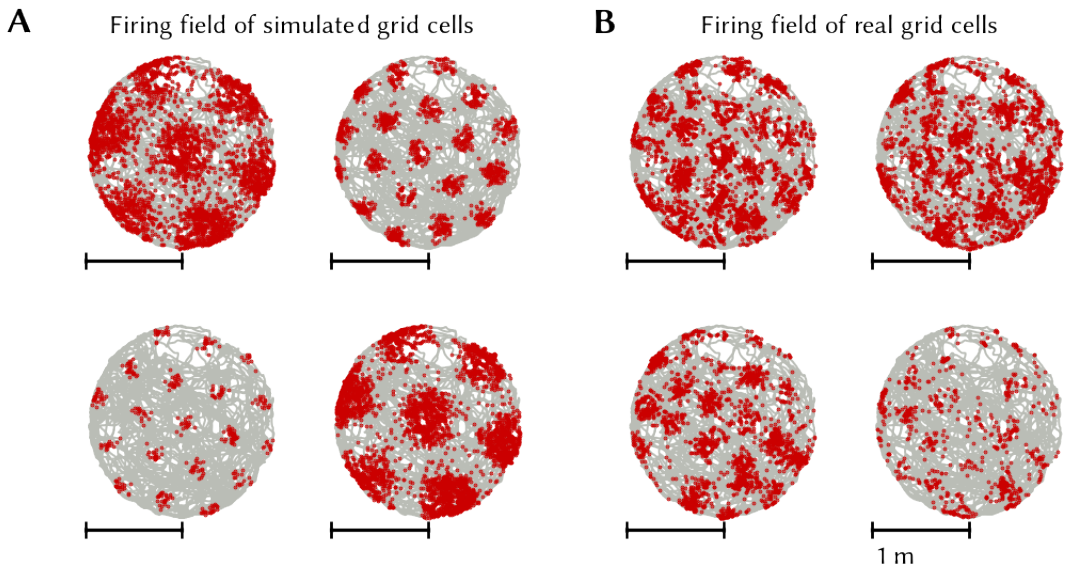



**Figure 3.3: (A)** Grid cells from a population encoding the HexSSP $\phi(\boldsymbol{x}(t))$, where $\boldsymbol{x}(t)$ is the path shown in grey (obtained from Hafting et al. (2005)). Red dots indicate the positions at which the cells fired. **(B)** Firing patterns from real grid cells (Hafting et al., 2005).

If a neural population encoding the SSP $\phi(\boldsymbol{x})$ has neurons with encoders

$$\boldsymbol{e}_j = \mathcal{F}^{-1}\left\{\bar{B}_i \mathcal{F}\left\{\phi^{(i)}(\boldsymbol{x}_j)\right\}\right\}, \tag{3.7}$$

then neuron $j$ will have a hexagonally gridded tuning curve. The scale of the grid corresponds to the $i^{th}$ simplex scale, $s_i$, and the orientation of the grid is determined by the $i^{th}$ simplex rotation, $R_i$. The shifts and receptive field widths of the lattice will also vary within the population due to heterogeneous threshold current and bias parameters. Examples of the firing patterns of select neurons within a grid cell population are shown in Figure 3.3.

### *3.3.3 Decoding SSP neural representation*

Replicating grid cell patterns is useful for biological fidelity, but does not address the question of why such patterns are useful. Many researchers have posited that the grid cell code has particularly high representational accuracy and robustness to noise compared to other neural coding schemes (*e.g.*, Sreenivasan et al., 2011). In this section, we empirically test this hypothesis within a VSA framework, using SSP-based grid cell representations.

Recall the problem of optimal linear decoding of a variable from a population of $N$ neurons (with regularization) given in Eq. (2.10). The root mean square error (RMSE) of decoding $n_s$ samples is

$$E = \sqrt{\frac{1}{n_s}\sum_{j=1}^{n_s}\left\|\boldsymbol{x}_j - D^*\boldsymbol{a}(\boldsymbol{x}_j)\right\|^2}. \tag{3.8}$$

We are linearly decoding a variable from a finite set of non-linear tuning curves – this reconstruction will have error. For many standard choices of tuning curves, $\boldsymbol{a}_j(\boldsymbol{x})$, this error scales as a power law, typically $1/\sqrt{N}$ (Seung et al., 1993; Zhang et al., 1999; Shamir et al., 2004). This is true for the direct Neural Engineering Framework (NEF) encoding scheme and any population with identical tuning curves up to translations (*e.g.*, place cells, head direction cells). In Sreenivasan et al. (2011), they showed that, in contrast to these 'classical codes', the decoding RMSE of a grid code scales exponentially. This is because grid cells employ a modular multi-periodic structure and the combined activity of these modules forms a high-dimensional representation that can encode spatial locations with exponentially increasing precision. Essentially, Sreenivasan et al. argue that neural redundancy is used more efficiently in grid coding schemes.

*Encoding methods.* We evaluate the decoding error for neural encoding of SSP representations – in particular, for 2D location encoded using HexSSP generated grid cells. Locations in 2D space were encoded via the activity of a population of $N$ leaky integrate-and-fire (LIF) neurons in several different manners. The representations and encoders tested are:

1. Direct encoding of the 2D variable $\boldsymbol{x}$: $a_i(t) = G_i\left[\alpha_i\langle \boldsymbol{e}_i, \boldsymbol{x}\rangle + \beta_i\right]$
    a) Neural encoders are randomly sampled from the unit circle (the standard choice in the NEF), $\boldsymbol{e}_i \sim \mathbb{S}^1$.
2. A $d$-dimensional RandSSP[5] representation of the 2D variable $\boldsymbol{x}$: $a_i(t) = G_i\left[\alpha_i\langle \boldsymbol{e}_i, \phi_{\text{rand}}(\boldsymbol{x})\rangle + \beta_i\right]$ with $\boldsymbol{e}_i = \phi_{\text{rand}}(\boldsymbol{x}_i)$, $\boldsymbol{x}_i \sim \mathcal{U}(-1, 1)$
    a) 'Sparse' tuning curves: encoders are set by SSPs representing random points scheme and we set gains and biases such that neurons are highly selective, with tuning curves resembling bumps or place cells (Figure 3.4A(i)).
    b) 'Dense' tuning curves: encoders are set by SSPs representing random points scheme and we set gains and biases such that there is significant variability in neuron sensitivity – this means that some neurons will have tuning curves resembling the SSP similarity map – *i.e.*, noisy place cells (Figure 3.4A(ii)).
3. A $d$-dimensional HexSSPs representation: $a_i(t) = G_i\left[\alpha_i\langle \boldsymbol{e}_i, \phi_{\text{hex}}(\boldsymbol{x})\rangle + \beta_i\right]$.
    a) The encoders given in Eq. (3.7) are used. These 'pick out' triplet Fourier components of the representation, producing grid cell-like responses (Figure 3.4A(iii)).

*Decoding methods.* To evaluate these representations, the firing rates of the neurons, given by Eq. (2.4), encoding these different representations are recorded over $n_x = 10000$ sample points over $\boldsymbol{x} \in [-r, r]^2$. These rates are used to construct the activity matrix, $A$. Gaussian noise is added to the activity, $A \leftarrow A + \varepsilon$ with $\varepsilon \sim \mathcal{N}(0, \sigma \max(A))$ and $\sigma = 0.01$. From this matrix, output is decoded.

[5]The phase matrix is sampled from $\mathcal{U}[-\pi, \pi]$.

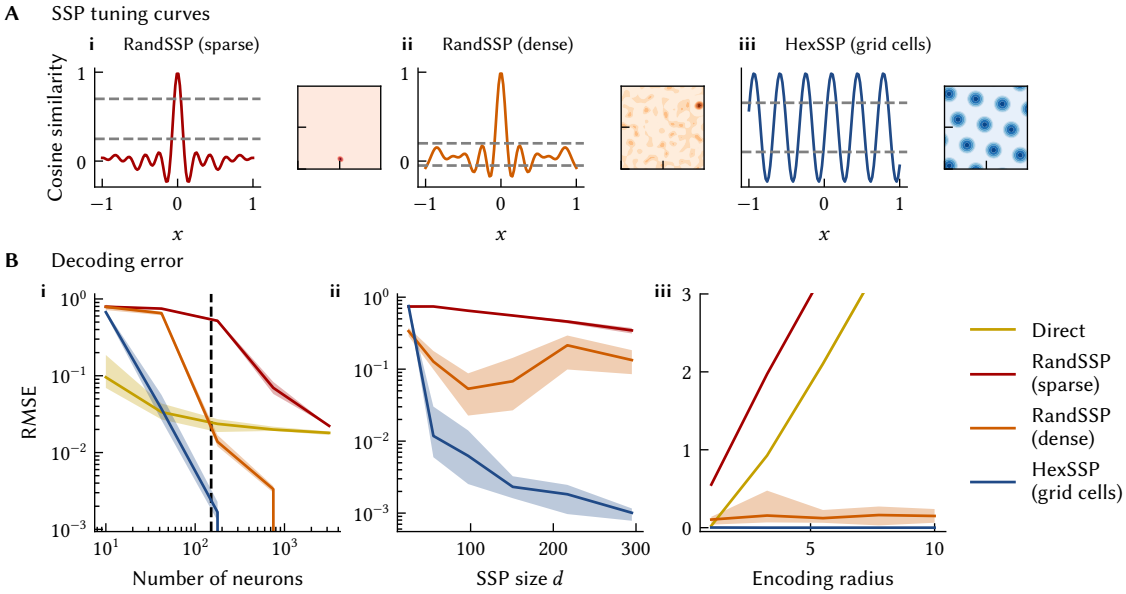



**Figure 3.4: Decoding error for different SSP and non-SSP representations and encoding schemes.** Values are non-linearly encoded by and then decoded from neural activity. **(A)** The cosine similarity between 1D values and an example encoder, with dashed lines marking the range of intercepts used. Also shown are example 2D tuning curves plotted as a heat map. The values are encoded either directly (with random encoders), **i-ii)** with RandSSP, or **iii)** with HexSSP. **(B)** The RMSE between the decoded $\hat{\boldsymbol{x}}$ and actual variables $\boldsymbol{x}$ versus **i)** the number of neurons $N$ (with $d = 97$ and $\sigma = 0$), **i)** the SSP size $d$ (with $N = d$ and $\sigma = 0$), and **iii)** the range from which the encoded variables are drawn, $\boldsymbol{x} \in [-r, r]^2$. Note the the error in the last plot is not on a log scale. The vertical dashed black line in **i)** indicates $N = d$ – *i.e.*, when the number of neurons used is equal to the size of the SSPs.

For the direct representation case, $\hat{\boldsymbol{x}}$ is linearly decoded from the neural activities. The decoding matrix is given by Eq. (2.11), $D^T = A^+ X$ where $A^+$ is the regularized Moore-Penrose pseudo-inverse of a non-noisy activity matrix and $X$ is a matrix of data samples from $[-1, 1]^2$. For SSPs, $\hat{\phi}$ is first linearly decoded. The decoding matrix is again given by Eq. (2.11); however, in this case, the data samples are from $[-1, 1]^d$ since the population is used to represent $d$-dimensional vectors. Next, the maximum likelihood estimate (MLE) $\hat{\boldsymbol{x}}$ is obtained from $\hat{\phi}$ by solving Eq. (2.38) via the candidate set method.

*Results.* The key results and their implications are summarized below:

- **Scaling with the number of neurons** $N$: Figure 3.4B plots the RMSE of decoding the variable $\boldsymbol{x}$ versus the number of neurons $N$ (Figure 3.4B(i)). We observe that the decoding error for direct neural encoding scales as a sub-power law – this is worse than the expected power law scaling due to the noise added to the encoding matrix.

  For the 'sparse' RandSSPs,error scaling exhibits two distinct regimes. When $N < d$, the SSP size, the system is underparameterized, and decoding errors decrease slowly

due to insufficient representation capacity. Once $N > d$, error scaling transitions to a power-law decay, as the population can fully encode the $d$-dimensional space.

For dense RandSSPs and HexSSPs, the scaling transitions to exponential decay when $N$ is high enough. There is a steep drop in error, with the curves appearing nearly vertical. For HexSSPs, this drop to zero error occurs when $N$ is slightly higher than $d$, and for the 'dense' RandSSPs, this drop occurs at a higher $N$. While the complete SSP vector may not be accurately encoded by these populations, the correct variable can still be accurately decoded.

- **Scaling with SSP size** $d$: In this experiment, the number of neurons also increases with $d$ – in particular, $N =$, a regime in which errors are small but non-zero. As $d$ increases, we observe the fastest error decay with HexSSPs (Figure 3.4B(ii)).

  For dense RandSSPs, the error initially decreases but then begins to increase. This behaviour arises because the dense RandSSPs become less distributed and more sparse at higher $d$. As the SSP similarity profiles sharpen, neurons become more selective, transitioning from noisy place-cell-like tuning to more localized place-cell responses.

- **Effect of Variable Range** $r$: In Figure 3.4B(iii), the decoding error given different ranges of variables is plotted. In the direct encoding case, the decoders are optimized over $[-1, 1]^2$, but are then applied to activity encoding variables over $[-r, r]^2$. For RandSSPs, the place cell centres are sampled over $[-1, 1]^2$. The high decoding errors for direct encoding and sparse RandSSPs when $r > 1$ is expected, as both of these methods rely on local codes optimized for a limited range of inputs, making them unable to generalize beyond this range.

  Dense RandSSPs have non-zero decoding error, but it does not seem to scale significantly with $r$. Although neurons in this case have place centres in $[-1, 1]^2$, their tuning is not purely local and somewhat distributed, preserving some capacity for generalization.

  HexSSPs maintain nearly zero decoding error across all tested ranges, highlighting the robustness of their distributed, multi-periodic phase coding. This property makes HexSSPs especially suitable for tasks requiring extrapolation or representation of large variable ranges.

The results of this experiment suggest that phase coding, as employed by SSPs, generally has more favourable scaling properties than classical codes, with grid cells being an especially good special case. The reason for this advantage is that the error in the grid code is distributed and corrected through the redundancy in the multi-periodic structure. In contrast, when RandSSPs are used to encode variables with place cells, (*i.e.*, the sparse encoding), they effectively turn the distributed code into a sparse, local code.

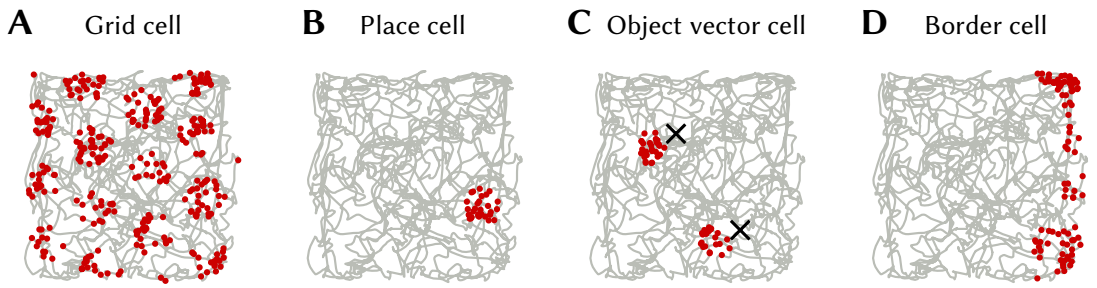



**Figure 3.5: Firing patterns of LIF neurons representing SSPs. (A)** A grid cell from a population encoding $\phi(\boldsymbol{x}(t))$, where $\boldsymbol{x}(t)$ is the path shown in grey (obtained from (Sargolini et al., 2006)). Red dots indicate the positions at which the cell fired. **(B)** A place cell from a population encoding $\phi(\boldsymbol{x}(t))$. **(C)** An object vector cell from a population encoding the SSP representation of the vector between $\boldsymbol{x}(t)$ and any objects in view. Object locations are marked with an 'x'. **(D)** A boundary cell from a population encoding the SSP representation of the vector between $\boldsymbol{x}(t)$ and a wall along the right side of the environment.

### *3.3.4 Other spatial cells*

We have now seen that SSPs can be used to model grid cells. In Section 3.2.1, we also discussed other spatially sensitive cells in the hippocampal and parahippocampal regions, such as boundary cells and object vector cells. These cell types can be modelled using SSPs, as shown in Figure 3.5.

The grid cell in Figure 3.5A is obtained from a population representing location in 2D space, $\phi(\boldsymbol{x})$, with an encoder given by Eq. (3.7), as done earlier in Section 3.3. The place cell in Figure 3.5B is also from a population representing $\phi(\boldsymbol{x})$ with encoder equal to an SSP representation of a fixed location, $\phi(\boldsymbol{x}_j)$. This is similar to RandSSPs with sparse tuning described in Section 3.3.3, but using a HexSSP instead. Place cells may also emerge in populations encoding self-location combined with representations of landmarks or features. For instance, suppose we have a population encoding $\texttt{OBJ}(t) \circledast \phi(\boldsymbol{x}(t))$, where $\texttt{OBJ}(t)$ is an SP or bundle of SPs representing object(s) or feature(s) in view at time $t$ (this may be an SP representing a generic 'object' concept, an SP representation of a category of objects, or an SP representing a very particular object). If this population has encoders $\texttt{OBJ}_j \circledast \phi(\boldsymbol{x}_j)$, it will be a population of place cells tuned to particular landmarks, object types, or environmental features (see Figure 3.5 for examples).

Consequently, both grid and place cells in our models may thus represent the same information, but exhibit different tuning. In the NEF, the space of neural activity is distinct from the encoded state space, allowing different tuning for the same underlying information. Grid cells, which offer low decoding error over all space, are well-suited for general spatial representation. Place cells, with sparse tuning, are effective for learning functions in limited areas, as their low overlap facilitates learning with minimal interference, essentially reducing function approximation to a lookup table – though this scales poorly as the function domain

**Table 3.1:** The activity patterns and coding properties of a neural population depend not only on the information it encodes but also on the manner of encoding, which is determined by the encoders specified in the NEF. Populations representing the vector 'State' via the NEF and encoders as described, will have firing patterns similar to 'Neuron type.'

| State | Encoders | Neuron type |
|---|---|---|
| $\phi(\boldsymbol{x}(t))$ | $\phi(\boldsymbol{x}_j)$ | Place cells |
| $\phi_{\text{hex}}(\boldsymbol{x}(t))$ | $\phi_{\text{hex}}(\boldsymbol{x}_j)$ | Place cells |
| $\phi(\boldsymbol{x}(t))$ | Eq. (3.7) | Grid cells |
| $\texttt{OBJ}(t) \circledast \phi(\boldsymbol{x}(t))$ | $\texttt{OBJ}_j \circledast \phi(\boldsymbol{x}_j)$ | Place cells (object/landmark dependent) |
| $\phi(\Delta\boldsymbol{x}(t))$ | $\phi(\boldsymbol{x}_j)$ | Object vector cells |
| $\texttt{OBJ}(t) \circledast \phi(\Delta\boldsymbol{x}(t))$ | $\texttt{OBJ}_j \circledast \phi(\boldsymbol{x}_j)$ | Landmark vector cells |
| $B \circledast \phi(\Delta\boldsymbol{x}(t))$ | $B \circledast \phi(\boldsymbol{x}_j)$ | Boundary vector cells |

increases.

As well, object vector cells and landmark vector cells can be obtained by representing the displacement to objects in view as an SSP. For example, a population representing $\phi(\Delta\boldsymbol{x}(t))$,[6] where $\Delta\boldsymbol{x}(t)$ is the displacement vector to visible objects at time $t$, consists of object vector cells – neurons that fire when an animal has a particular displacement relative to objects – if it has encoders $\phi(\boldsymbol{x}_j)$ (see Figure 3.5C). A population representing $\texttt{OBJ}(t) \circledast \phi(\Delta\boldsymbol{x}(t))$, with encoders $\texttt{OBJ}_j \circledast \phi(\boldsymbol{x}_j)$, will be a population of landmark vector cells – neurons that fire when an animal has a particular displacement relative to particular landmarks.

Similarly, boundary cells, or boundary vector cells, make up a population encoding $B \circledast \phi(\Delta\boldsymbol{x})$, where $\Delta\boldsymbol{x}$ is displacement to an environment boundary and $B$ is an SP representing a boundary. An example is shown in Figure 3.5D. Note that we are not proposing a model here for how these kinds of spatial firing patterns (summarized in Table 3.1) are learned – we are simply demonstrating that they can be modelled using our methods.

### *3.3.5 Discussion*

The usefulness of SSPs, and HexSSPs in particular, can be attributed to several factors. One useful property of a spatial code is the approximate preservation of the geometry of the underlying space. Recent studies have highlighted the importance of local conformal isometry as a criterion for vector embeddings that generates grid cells (Gao et al., 2018; Xu et al., 2022; Schaeffer et al., 2024). This criterion ensures that angles and distances are maintained locally, so that displacements within a certain neighbourhood in the underlying space translate proportionally to displacements in the embedding space. For SSPs, this can be stated as

$$\langle \phi(\boldsymbol{x} + \Delta\boldsymbol{x}), \phi(\boldsymbol{x}) \rangle = s^2 \langle \boldsymbol{x} + \Delta\boldsymbol{x}, \boldsymbol{x} \rangle, \tag{3.9}$$

[6] If there are multiple objects or an object has an extended area, $\phi(\Delta\boldsymbol{x})$ may be an SSP bundle.

for small $\Delta \boldsymbol{x}$, or, equivalently,

$$\|\phi(\boldsymbol{x}+\Delta \boldsymbol{x})-\phi(\boldsymbol{x})\|=s\|\Delta x\|, \tag{3.10}$$

for some scalar $s$. For SSPs, this is achieved when the phase matrix columns are orthogonal,

$$\begin{aligned}\|\phi(\boldsymbol{x}+\Delta \boldsymbol{x})-\phi(\boldsymbol{x})\|_2 &= \left\|\mathcal{F}^{-1}\left\{e^{i\Theta(x+\Delta x)}-e^{i\Theta x}\right\}\right\|_2 && (3.11)\\ &\approx \left\|\operatorname{diag}\left(e^{i\Theta x}\right)\Theta\Delta \boldsymbol{x}\right\|_2 && (3.12)\\ &= \|\Theta\Delta \boldsymbol{x}\|_2, && (3.13)\end{aligned}$$

where a Taylor expansion about $\Delta \boldsymbol{x}$ is used at Eq. (3.12), and diag$(\cdot)$ is the matrix with the vector argument as its diagonal. The criteria $\|\phi(\boldsymbol{x}+\Delta \boldsymbol{x})-\phi(\boldsymbol{x})\|_2 = s\|\Delta \boldsymbol{x}\|_2$ is obtained when $\Theta^T\Theta = s^2 I$, *i.e.*, the columns of the phase matrix are orthogonal. This condition is satisfied by HexSSPs, but is certainly not unique to them. The property of local conformal isometry is crucial for accurate and robust spatial representations, suggesting that HexSSPs (and other SSPs whose phase matrix has orthogonal columns) maintain the integrity of spatial relationships.

Another requirement of a useful spatial code is a high representational capacity, which in the context of SSPs, is related to their periodicity and construction. For an SSP with $d = 3$, its similarity map will be given by $k(\boldsymbol{x}) = e^{i\boldsymbol{\theta}^T\boldsymbol{x}} + e^{-i\boldsymbol{\theta}^T\boldsymbol{x}} = 2\cos\boldsymbol{\theta}^T\boldsymbol{x}$ and thus will repeat over a period $2\pi/\|\theta\|$. The similarity map of an SSP with $d = 5$ will be given by $k(\boldsymbol{x}) = 2(\cos\boldsymbol{\theta}_1^T\boldsymbol{x} + \cos\boldsymbol{\theta}_2^T\boldsymbol{x})$, which will be aperiodic if $\|\theta_1\|/\|\theta_2\|$ is irrational; otherwise, it repeats with a period proportional to the least common multiple of the individual periods. Generally, a $d$-dimensional SSP will be aperiodic if at least one ratio of row magnitudes in $\Theta$ is irrational, meaning the probability of a repeating similarity map is essentially zero with randomly sampled phases. Alternatively, scaling the rows of $\Theta$ by a geometric progression with an irrational ratio can ensure representation uniqueness. In either case, the similarity function's maximum always occurs at the encoded variable, ensuring every $\boldsymbol{x} \in \mathbb{R}^n$ is uniquely represented by $\phi(\boldsymbol{x})$.[7] This property illustrates that SSPs can encode continuous variables with precision, providing a strong foundation for high-dimensional neural representations of spatial information.

Minimizing decoding error is another consideration for a good spatial code. Consider the standard NEF approach of non-linear encoding of a feature vector $\phi(\boldsymbol{x})$ in neural activities and linear decoding to approximate a given function $\boldsymbol{p}(\boldsymbol{x}) \in \mathbb{R}^{n_p}$:

$$a_i(t) = G_i\left[\alpha_i\langle \boldsymbol{e}_i, \phi(\boldsymbol{x})\rangle + \beta_i\right], \quad \hat{\boldsymbol{p}}(\boldsymbol{x}) = \sum_{i=1}^{N} a_i(t) * h_{\text{syn}}(t)\boldsymbol{d}_i. \tag{3.14}$$

As explained in the previous chapter, decoders $D \in \mathbb{R}^{n_p \times N}$ are computed to minimize the error of this approximation. We take $n_x$ samples of this function to form a matrix, $P \in \mathbb{R}^{n_p \times n_x}$ with

[7] Note that while the similarity function's global maximum is unique, recovering $\boldsymbol{x}$ can be challenging in low-dimensional SSPs due to multiple local maxima.

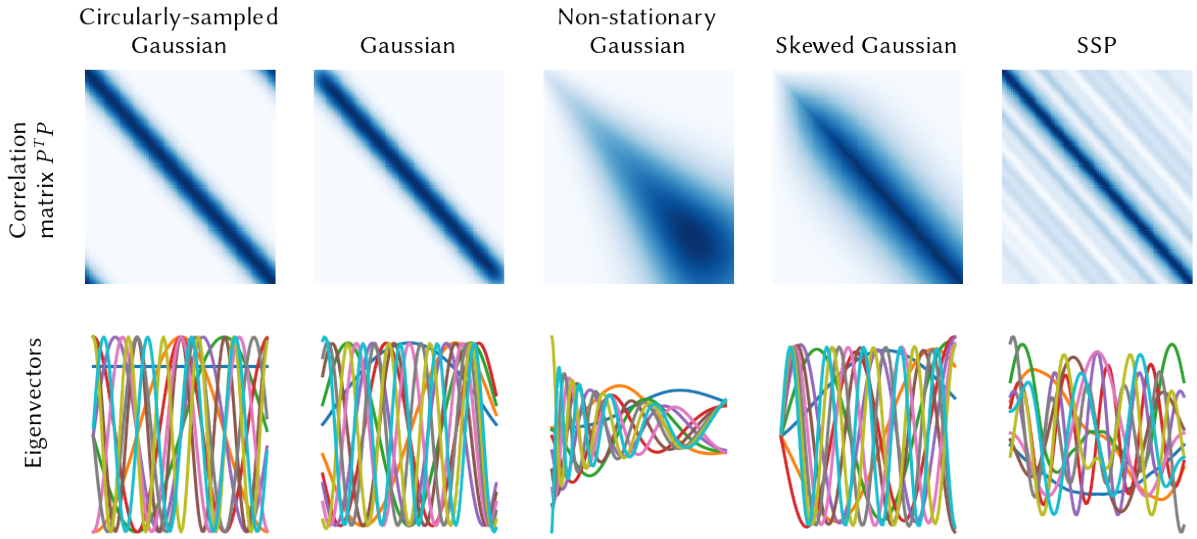



**Figure 3.6: Correlation matrices for different functions uniformly covering 1D space and their top eigenvectors.** The functions are Gaussian, sampled both circularly and non-circularly, a non-stationary Gaussian with width varying over space, a skewed Gaussian, and SSPs. In all examples, a matrix $P$ is constructed by sampling the function, with $P_{i,j} = p_i(\boldsymbol{x}_j)$ (for the Gaussians, $p_i$ refers to the $i$ Gaussian, while for the SSPs $p_i$ refers to the $i$ element of the SSP vector). The top row shows visualizations of the correlation matrices $P^T P$, while the bottom row depicts the real components of the top 10 eigenvectors of $P^T P$ (each row of the visualized matrix corresponds to an eigenvector). The circularly sampled Gaussians produce a circulant correlation matrix and Fourier mode eigenvectors. The non-circularly sampled Gaussians and SSPs produce a Toepliz correlation matrix. In these and even in the non-Toepliz examples, the eigenvectors resemble summed harmonics.

$P_{i,j} = p_i(\boldsymbol{x}_j)$, and solve the least-squares optimization problem:

$$\min \sum_{j=1}^{n_s} \left\| \boldsymbol{p}(\boldsymbol{x}_j) - D\boldsymbol{a}(\phi(\boldsymbol{x}_j)) \right\|^2 = \min \|P - DA\|_F^2 . \tag{3.15}$$

$$\arg\min_D \|P - DA\|_F^2 = PA^T(AA^T)^{-1} \tag{3.16}$$

where $A \in \mathbb{R}^{N \times n_x}$ is the activity matrix. For simplicity, we have removed the regularization term. In addition to optimizing the decoders to approximate $P$, we may also optimize the neural activities matrix. The optimal $A$ is the orthogonal matrix that, if $n_p \leq N$, diagonalizes $P^T P$. If instead we want to decode a function with a higher dimensionality than there are neurons (or, equivalently, we want to decode many functions, the total number of which is higher than the number of neurons), then this is a low-rank approximation problem, *i.e.*, a problem of fitting a data matrix using an approximating matrix that has a reduced rank. By the matrix approximation lemma, the columns of the optimal $A$ will span the top $N$ eigenvectors of the correlation matrix $P^T P$.

If we simply want to decode SSPs, then $p_i(\boldsymbol{x}_j) = \phi(\boldsymbol{x}_j)_i$, and the $i, j$ element of $P^T P$ will be $\phi(\boldsymbol{x}_i) \cdot \phi(\boldsymbol{x}_j)$. This is a matrix of similarities between the sample SSPs, which depends only on the distance between the samples they represent. Assuming that the sample points $\boldsymbol{x}_i$ uniformly cover space, the correlation matrix will be Toeplitz. This matrix and its top 10 eigenvectors are shown in Figure 3.6. If the space is circularly sampled, then the matrix will be circulant and its eigenvectors will be Fourier modes.

In Sorscher et al. (2023), this low-rank approximation problem was discussed in the context of decoding place cell output. That is to say, $P$ is a given matrix of place cell responses sampled across space, with $n_p \gg N$ being the number of place cells. These may be modelled via Gaussians,

$$P_{i,j} = p_i(\boldsymbol{x}_j) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2\sigma^2}(\boldsymbol{x}_j - \boldsymbol{\mu}_i)^T(\boldsymbol{x}_j - \boldsymbol{\mu}_i)}, \tag{3.17}$$

or a difference-of-Gaussians or difference-of-softmaxes (as used by Sorscher et al., 2023). They argue that when a non-negativity constraint is added to this optimization problem, grid cell-like activity patterns emerge.

More generally, suppose our goal is to decode any stationary kernel or a shift-invariant function (*e.g.*, Matern kernels, stationary probability distributions, successor features under a shift-invariant transition function) uniformly distributed over space. In all such cases, assuming periodic boundary conditions, the correlation matrix will be circulant. Thus, to decode these functions from a neural population, the optimal activity patterns are linear combinations of Fourier modes. Such neurons will have periodic and quasi-periodic spatial responses, similar to grid cells. This aligns with our finding that HexSSPs encoded via grid cells yield lower decoding error compared to linear encoding or place cell encoding (which also combines Fourier modes but uses many more components) of space.

In conclusion, HexSSP embeddings exhibit several promising features of a good spatial code, including preservation of local geometry, high representational capacity, and efficient decoding, which contribute to their low decoding error. Furthermore, in previous work, the utility of HexSSPs has been empirically validated in diverse contexts (Komer, 2020; Bartlett et al., 2023b).

## 3.4 Path integration

We have discussed how SSP representations of continuous variables can be encoded by the activity of place and grid cells. In this section, we address the question of how these representations are formed and continuously updated. Despite the fact that SSPs are defined in terms of a high-dimensional vector space, we can show that they naturally implement velocity controlled oscillators (VCOs).

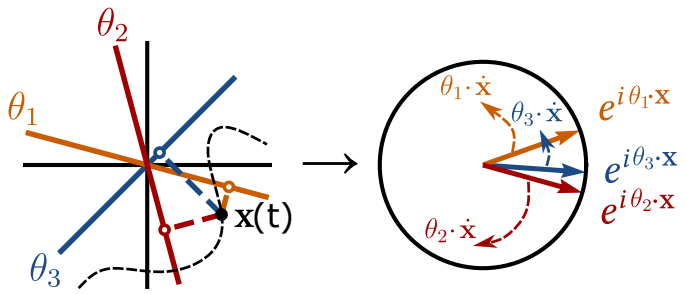



**Figure 3.7:** A variable encoded by an SSP changing at rate $\dot{x}(t)$ is equivalent to the SSP Fourier components oscillating at a frequencies $\omega(t) = \Theta\dot{x}(t)$.

### 3.4.1 SSP dynamics

Consider how $\phi(\boldsymbol{x}) \in \mathbb{R}^d$ changes if $\boldsymbol{x} \in \mathbb{R}^n$ is a function of time.

**Theorem 3.1.** *The dynamics of an SSP $\phi(\boldsymbol{x})$ are given by*

$$\dot{\phi}(\boldsymbol{x}(t)) = \mathcal{F}^{-1}\left\{ i\Theta\dot{\boldsymbol{x}}(t) \odot e^{i\Theta\boldsymbol{x}(t)} \right\}, \tag{3.18}$$

$$= \ln \phi(\dot{\boldsymbol{x}}(t)) \circledast \phi(\boldsymbol{x}(t)), \tag{3.19}$$

*where $\odot$ is element-wise multiplication, and the logarithm of an SSP is defined as the element-wise logarithm in the Fourier domain, $\ln \phi \equiv \mathcal{F}^{-1}\{\ln \mathcal{F}\{\phi\}\}$.*

*Proof.*

$$\dot{\phi}(\boldsymbol{x}(t)) = \frac{d}{dt}\mathcal{F}^{-1}\left\{ e^{i\Theta\frac{\boldsymbol{x}(t)}{\ell}} \right\} \tag{3.20}$$

$$= \mathcal{F}^{-1}\left\{ i\Theta\frac{\dot{\boldsymbol{x}}(t)}{\ell} \odot e^{i\Theta\frac{\boldsymbol{x}(t)}{\ell}} \right\} \tag{3.21}$$

$$= \mathcal{F}^{-1}\left\{ i\Theta\frac{\dot{\boldsymbol{x}}(t)}{\ell} \right\} \circledast \phi(\boldsymbol{x}(t)) \tag{3.22}$$

$$= \ln \phi(\dot{\boldsymbol{x}}(t)) \circledast \phi(\boldsymbol{x}(t)) \tag{3.23}$$

where exponentiation of a vector signifies element-wise exponentiation. □

Note the dynamics are unchanged if the SSP is bound with a non-time-varying vector (such as a SP representing an object or a fixed context):

$$\frac{d}{dt}\Big(A \circledast \phi(\boldsymbol{x}(t))\Big) = \ln \phi(\dot{\boldsymbol{x}}(t)) \circledast \Big(A \circledast \phi(\boldsymbol{x}(t))\Big). \tag{3.24}$$

Thus, a model that implements the dynamics of an SSP can be used, without modification, for continuous updating of a combined object-location representation – this suggest corresponds to a different initialization.

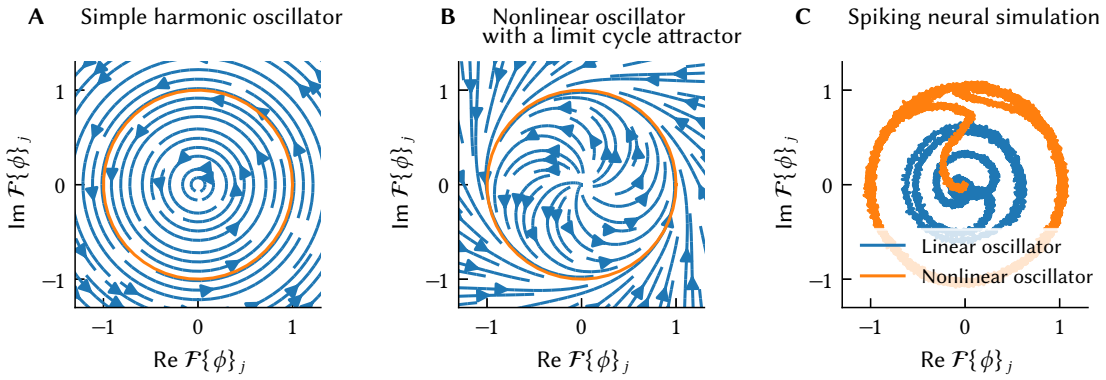



**Figure 3.8: VCO dynamics with and without a limit cycle attractor. (A)** The phase space of the dynamical system given in Eq. (3.26), the unmodified VCO. **(B)** The phase space of the dynamical system given in Eq. (3.28), the non-linear VCO with the unit circle as a limit cycle attractor. **(C)** Spiking neural network simulations of the linear oscillator described by Eq. (3.26) and the nonlinear oscillator described by Eq. (3.28). Note that the linear oscillator collapses, while the nonlinear oscillator remains stable.

### *3.4.2 SSP dynamics in the Fourier domain*

Now let us consider the dynamics of an SSP in the Fourier domain. Taking the Fourier transform of Eq. (3.18), we get,

$$\mathcal{F}\left\{\dot{\phi}(\boldsymbol{x}(t))\right\} = (i\Theta\dot{\boldsymbol{x}}(t)) \odot \mathcal{F}\left\{\phi(\boldsymbol{x}(t))\right\} . \tag{3.25}$$

Note that the dynamics of the Fourier components of an SSP are independent of each other. The dynamics of the $j^{\text{th}}$ Fourier component of the SSP can be written as

$$\frac{d}{dt}\begin{bmatrix}\text{Re}\mathcal{F}\{\phi(\boldsymbol{x}(t))\}_j \\ \text{Im}\mathcal{F}\{\phi(\boldsymbol{x}(t))\}_j\end{bmatrix} = \begin{bmatrix}0 & -\omega_j(t) \\ \omega_j(t) & 0\end{bmatrix}\begin{bmatrix}\text{Re}\mathcal{F}\{\phi(\boldsymbol{x}(t))\}_j \\ \text{Im}\mathcal{F}\{\phi(\boldsymbol{x}(t))\}_j\end{bmatrix}, \tag{3.26}$$

$$\text{where } \omega_j(t) \equiv \boldsymbol{\theta}_j \cdot \dot{\boldsymbol{x}}(t) = -i \ln \mathcal{F}\{\phi(\dot{\boldsymbol{x}}(t))\}_j. \tag{3.27}$$

where $\boldsymbol{\theta}_j$ is the $j^{th}$ row of the phase matrix $\Theta$. Each Fourier component of the SSP is thus the state of a time-varying harmonic oscillator whose frequency is given by $\omega_j$ (Figure 3.7). The oscillator frequencies are modulated by the velocity $\dot{\boldsymbol{x}}$; in other words, they are VCOs,the main component of OI models of path integration. This result shows that continuous fractional binding in the time domain is equivalent to controlled oscillations in the Fourier domain. Note that the dynamics of only $\lfloor \frac{d}{2} \rfloor$ VCOs need to be maintained to compute the complete SSP, due to the conjugate symmetry of $\mathcal{F}\{\phi\}$.

### *3.4.3 Attractor dynamics for PI*

Given this derivation of SSP dynamics, it becomes unsurprising that a neural path integrator can be constructed with a network that realizes the dynamics of an SSP representing self-location. These dynamics can be written different ways, either as continuous fractional binding

Eq. (3.19) or oscillation in the Fourier domain Eq. (3.26). An advantage of using VCOs is that the dynamics of each Fourier component are independent. This means that many sets of small neural populations, each recurrently connected only to itself, can realize these dynamics. In addition, simulations of the dynamics of each oscillator can be run in parallel.

However, a well-known issue with OI models is that noise causes drift in the oscillators and their accuracy quickly deteriorates (Zilli et al., 2009). This noise drift becomes even more acute when using spiking neurons (Figure 3.8C). However, the OI approach can be modified to improve the stability of the oscillators. VCOs can either be coupled together (Zilli et al., 2010; Burgess et al., 2014; Orchard et al., 2013; Ji, 2014) or attractor dynamics can be used to improve the stability of the dynamics (Bush et al., 2014). We use the latter approach with a novel choice of attractor dynamics. Cyclic attractors have been previously modelled with the NEF to describe the head-direction system (Eliasmith, 2005a). Here, instead of using the dynamics of a harmonic oscillator, we use a nonlinear oscillator with a stable limit cycle,

$$\frac{d}{dt}\begin{bmatrix}\mathrm{Re}\,\mathcal{F}\left\{\phi(\boldsymbol{x}(t))\right\}_j \\ \mathrm{Im}\,\mathcal{F}\left\{\phi(\boldsymbol{x}(t))\right\}_j\end{bmatrix} = \begin{bmatrix}-\omega_j(t)\mathrm{Im}\,\mathcal{F}\left\{\phi(\boldsymbol{x}(t))\right\}_j + \frac{1-r^2}{r}\mathrm{Re}\,\mathcal{F}\left\{\phi(\boldsymbol{x}(t))\right\}_j \\ \omega_j(t)\mathrm{Re}\,\mathcal{F}\left\{\phi(\boldsymbol{x}(t))\right\}_j + \frac{1-r^2}{r}\mathrm{Im}\,\mathcal{F}\left\{\phi(\boldsymbol{x}(t))\right\}_j\end{bmatrix}, \quad \text{where } r \equiv |\mathcal{F}\{\phi(\boldsymbol{x}(t))\}_j|. \tag{3.28}$$

A damping force is introduced when the amplitude of the VCO exceeds one and a driving force when the amplitude is less than one. This is similar to a Landau–Stuart oscillator. This system has the unit circle, $S^1$, as an attractor, ensuring that the SSP remains unitary as it evolves (Figure 3.8B). The attractor of the combined system of $\lfloor d/2 \rfloor$ VCOs that represent the complete SSP is the Cartesian product of the attractors of the individual VCOs, $S^1 \times S^1 \times \cdots \times S^1$, which is topologically equivalent to a $\lfloor d/2 \rfloor$-dimensional torus, $T^{\lfloor d/2 \rfloor}$. Thus, this system is a novel hybrid of OI and CANN models of PI.

### *3.4.4 SSP-PI network*

To implement these dynamics in spiking neurons using the NEF, we simulate $\lfloor d/2 \rfloor$ recurrent neural populations, each of which models a VCO (Figure 3.9). To approximate the nonlinear dynamics of Eq. (3.28), we need to represent both real and imaginary parts of $\mathcal{F}\{\phi\}_j$, as well as the frequency $\omega_j$, all in the same population (as is standard in the NEF). The $j^{\text{th}}$ neural population represents the real and imaginary parts of the Fourier components of the SSP. Thus, the vector being represented by the collective activity of the $j^{\text{th}}$ VCO population is

$$\begin{bmatrix}\omega_j(t) & \mathrm{Re}\mathcal{F}\left\{\hat{\phi}(t)\right\}_j & \mathrm{Im}\mathcal{F}\left\{\hat{\phi}(t)\right\}_j\end{bmatrix}^T. \tag{3.29}$$

We write $\hat{\phi}(t)$ here to emphasize that this is an *estimate* of $\phi(\boldsymbol{x})$. Due to noise inherent in neural encoding and the dynamics of Eq. (3.28) being approximated by recurrent connections, this estimate will drift from the SSP encoding of the ground truth location over time. In fact, the vector encoded by the path integrator will even drift from the toridal sub-space of SSP vectors

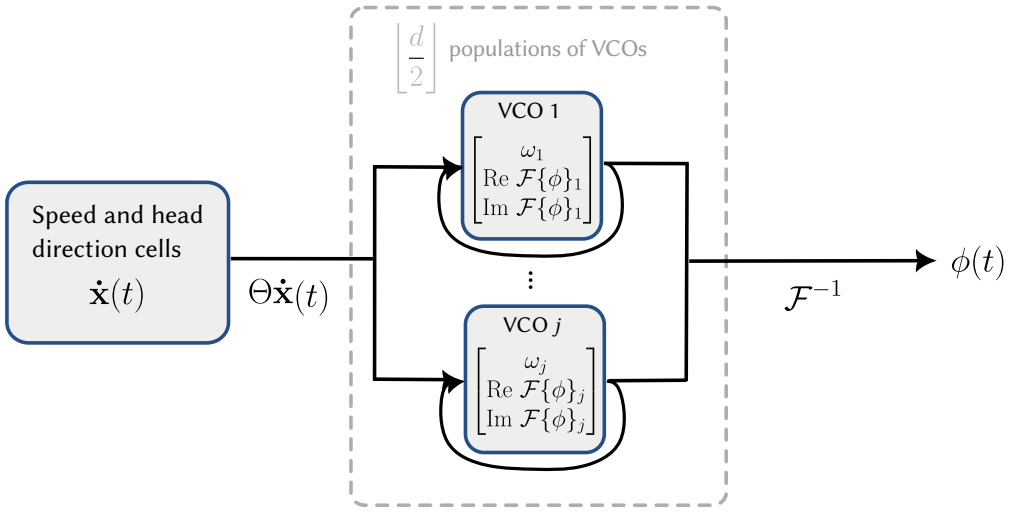



**Figure 3.9:** A diagram of the SSP-PI model. A population of speed and head direction cells represents velocity. This population provides input to a set of $\lfloor d/2 \rfloor$ VCOs, each of which is recurrently connected to implement the attractive oscillator dynamics. Blue boxes indicate neuron populations and arrows show the inter-population connections.

in $\mathbb{R}^d$ without some form of correction. In our model, correction comes from the dynamics described by Eq. (3.28). Other methods of error correction are presented in Chapter 4.

A population of speed- and heading-direction cells that encode the self-velocity projects onto the VCO populations. The connection weights compute the linear transform needed to obtain the input frequencies, $\Theta\dot{\boldsymbol{x}}(t) = \boldsymbol{\omega}$. Within each VCO, the neuron population is recurrently connected using weights optimized by least squares to implement the following dynamics, as per principle 3 of the NEF combined with Eq. (3.28):

$$\begin{bmatrix} 0 \\ -\tau\omega_j \mathrm{Im}\mathcal{F}\{\phi\}_j + \tau\frac{1-r^2}{r}\mathrm{Re}\mathcal{F}\{\phi\}_j + \mathrm{Re}\mathcal{F}\{\phi\}_j \\ \tau\omega_j \mathrm{Re}\mathcal{F}\{\phi\}_j + \tau\frac{1-r^2}{r}\mathrm{Im}\mathcal{F}\{\phi\}_j + \mathrm{Im}\mathcal{F}\{\phi\}_j \end{bmatrix}, \tag{3.30}$$

where $\tau$ is the synaptic time constant of the recurrent connections.

The complete model, SSP-PI, is depicted in Figure 3.9. Note that this model performs PI of continuous variables of any dimensionality, as long as they are represented by SSPs of the same size. The recurrent connection weights remain unchanged whether we are integrating $\boldsymbol{x} \in \mathbb{R}^n$ with $n = 2$ or 3, or higher. In such cases, the only required change to the network is the input connections that provide the frequency signals. We show results of PI using this model in Section 3.4.6.

### 3.4.5 Grid cells and PI

We have not yet addressed how this model can be used to generate grid cells. Each oscillator is a population representing a frequency (derived from speed and heading direction) and a single Fourier component of the SSP. This results in neurons with sensitivity to heading direction, speed, and spatial position (in a periodic fashion, resembling a plane wave). As a result, their firing patterns are velocity dependent bands or stripes and not hexagonal lattices.

Such banded firing patterns are not unique to the SSP-PI model. In most OI models, stripe/band cells perform PI and grid cells are the result of combining the output of such cells (Burgess et al., 2007). Strictly speaking, grid cells are not needed for PI, though some models use grid cells with CANN connectivity to provide corrections to VCOs (Bush et al., 2014). While there is some evidence for the existence of band cells (for example, the neural firing pattern shown in Figure 3.1G), they have not been as widely observed as grid cells (Krupic et al., 2012; Abad Perez et al., 2024).

It is possible to construct a model that uses conjunctive grid cells to perform PI in 2-dimensions. This requires the use of HexSSPs and modifications to the network structure of the SSP-PI model in Section 3.4.4. We shall refer to this modified model as GC-SSP-PI. Instead of having $\lfloor d/2 \rfloor$ independent populations of VCOs, we have $\lfloor d/6 \rfloor$ populations, each of which represents three VCOs. The vector encoded by the collective activity of the $j^{\text{th}}$ population is

$$\begin{bmatrix} \omega_{j_1} & \mathrm{Re}\mathcal{F}\{\phi\}_{j_1} & \mathrm{Im}\mathcal{F}\{\phi\}_{j_1} & \omega_{j_2}(t) & \mathrm{Re}\mathcal{F}\{\phi\}_{j_2} & \mathrm{Im}\mathcal{F}\{\phi\}_{j_2} & \omega_{j_3}(t) & \mathrm{Re}\mathcal{F}\{\phi\}_{j_3} & \mathrm{Im}\mathcal{F}\{\phi\}_{j_3} \end{bmatrix}^T. \tag{3.31}$$

where $\{j_1, j_2, j_3\}$ are indices such that $\boldsymbol{\theta}_{j_1} + \boldsymbol{\theta}_{j_2} + \boldsymbol{\theta}_{j_3} = 0$, *i.e.*, the associated rows of the phase matrix form a simplex. Using this representation in GC-SSP-PI means that the VCO triplet population encodes three SSP Fourier components (and their frequencies) whose phases sum to zero. This population will consist of conjunctive grid cells.

However, GC-SSP-PI is suboptimal in several respects. First, the dynamics of the SSP Fourier components are independent and do not need to be represented together. Representing vector components that are independent in separate populations is more dynamically stable, so there is no advantage for this unless some form of nonlinear coupling between triplets is required. Secondly, while the number of recurrent populations decreases from $\lfloor \frac{d}{2} \rfloor$ to $\lfloor \frac{d}{6} \rfloor$, the number of neurons per population must be increased to accurately represent the 9-dimensional vector in Eq. (3.31). In fact, we will see in the results presented in the next section that the SSP-PI model performs better than the GC-SSP-PI model given equal computational resources. Finally, this model does not generalize to PI in higher dimensions. To perform PI in 3D space, the recurrent populations must be changed to encode 12-dimensional vectors to obtain 'face-centred-cubic' cells. The network structure and recurrent connections depend on the underlying space, unlike the previous model. In short, there is no good theoretical reason to propose the GC-SSP-PI model, although it is more consistent with current experimental observations regarding the prevalence of grid cells compared to band cells.

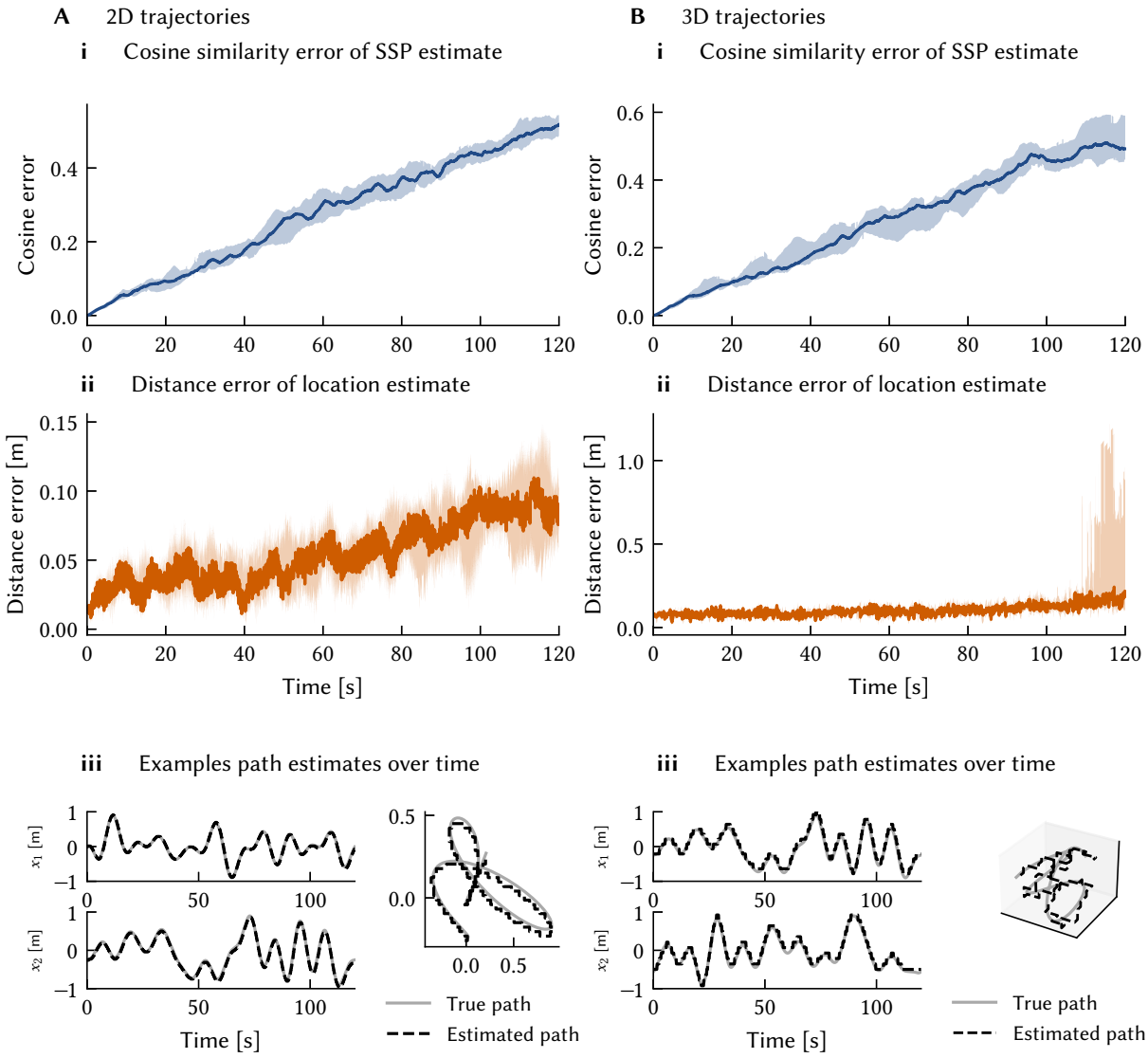



**Figure 3.10: Averaged results from the SSP-PI model** over ten two-minute long paths (each randomly generated via band-limited white noise, paths have equal power at all frequencies below 0.1 Hz). **(A)** PI results from 2D paths. **(B)** PI results from 3D paths. **(i)** The cosine distance between the SSP encoding of the ground truth path, $\phi(\boldsymbol{x}(t))$, and the model output $\hat{\phi}(t)$ over the simulation run time. The solid line is the median over the ten trials and the shaded area is the 95% bootstrapped confidence intervals. **(ii)** The Euclidean distance between the ground truth path, $\boldsymbol{x}(t)$, and the path decoded from the model output, $\hat{\boldsymbol{x}}(t)$. **(iii)** Examples of the randomly generated paths and the trajectories decoded from the model output. The grey line is the ground truth. Note that the path estimate plot on the right only shows 20 seconds on the trajectory for clearer visualization.

### 3.4.6 Results

**Path integration accuracy**. We test the SSP-PI on ten different two-minute-long paths randomly generated from band-limited white noise signals. The model is initialized with the SSP representation of the starting point of the path. It receives a velocity input signal (precomputed using finite differences on the path) over the simulation run time. We use 55-dimensional SSPs[8] and a total of 27000 LIF neurons. Figure 3.10(i) plots the median cosine distance between the model's output and the SSP representation of the ground truth,

$$1 - \frac{\hat{\phi}(t) \cdot \phi(\boldsymbol{x}(t))}{\|\hat{\phi}(t)\|_2 \|\phi(\boldsymbol{x}(t))\|_2}. \tag{3.32}$$

We see that this cosine error increases linearly over the simulation time. Figure 3.10(ii) plots the distance error of the model's location estimate (the MLE $\hat{\boldsymbol{x}}$ given $\hat{\phi}$ computed using the candidate set methods described in Section 2.4.2), $\|\boldsymbol{x}(t) - \hat{\boldsymbol{x}}(t)\|_2$.

Figure 3.10(iii) shows examples of the model's path estimate compared to the true path. Generally, the model follows the true path for the entire trajectory accurately, with a small amount of drift. Performance is not significantly different between the 2D and 3D models, except near the end of the simulation, where there is a jump in distance error for 3D trajectories (Figure 3.10B(ii)). We compare this model's performance to a recent artificial neural network (ANN) model of PI in the next section, showing similar levels of accuracy.

Sometimes near the end of the trajectories there are large 'jumps' in the model's estimate. This behaviour is shown in Figure 3.11, the results from running SSP-PI on 60-second path but with many fewer neurons per VCO population (300 versus 1000). With fewer neurons, the dynamics are more poorly approximated and there is significant phase drift. This does not result in a smooth drift of the model location estimate, but rather results in large sudden jumps (Figure 3.11A). The reason for this is apparent in Figure 3.11B. Early in the simulation, the model's output lies on the SSP manifold and $\hat{\phi}(t) \cdot \phi(\boldsymbol{x})$ looks as expected ($t = 10$s in Figure 3.11B). As error accumulates, the output leaves the SSP manifold and the similarity function develops multiple peaks due to constructive interference – although the distance error is still low, as the dominant peak follows the correct path ($t = 30$s in Figure 3.11B). Other peaks may begin to dominate, resulting in erratic jumps in the location estimate ($t = 32$s in Figure 3.11B). Destructive interference can destroy these while the correct peak remains – this can produce the odd behaviour seen at $t = 45$s in Figure 3.11, where the model's MLE is again accurate in spite of previous large errors.

The ability of SSP-PI to maintain and update multiple location peaks is actually advantageous in situations with positional uncertainty. For example, if the agent's initial position is uncertain, or if sensory cues or a cognitive map provide multiple hypotheses for self-location, it would be beneficial for a path integration model to represent and update multiple self-location estimates with self-motion cues. We demonstrate this capability in Figure 3.12. In this example,

[8] This corresponds to three grid modules of different scales and three possible grid orientations.

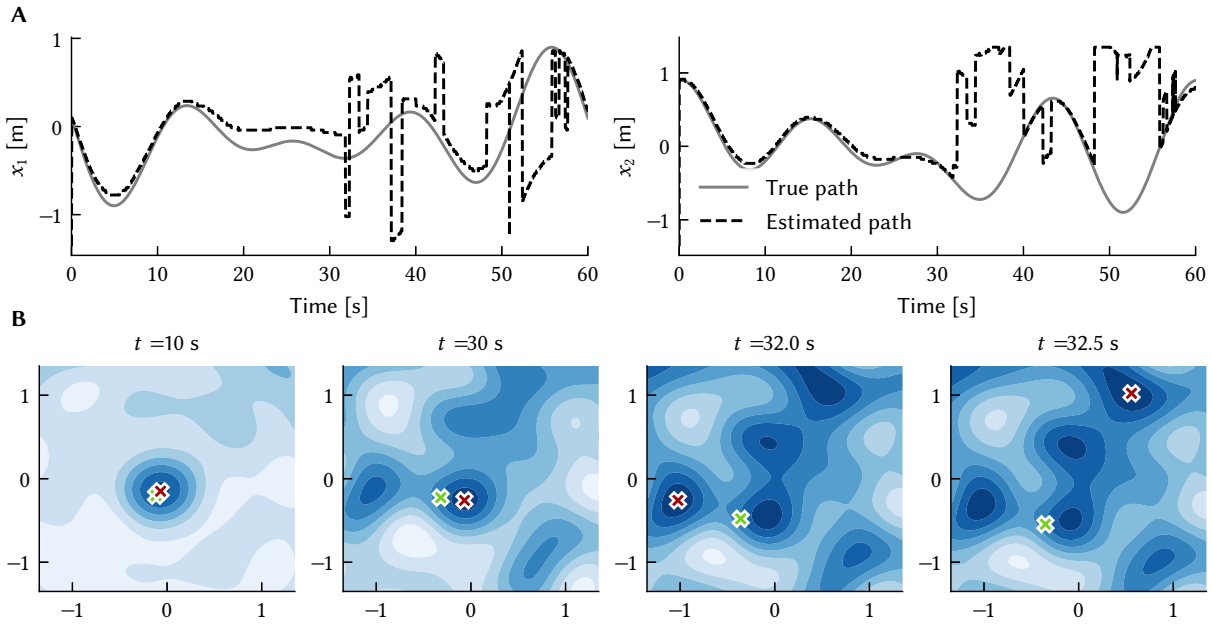



**Figure 3.11: Errors and the kernel evolution of SSP-PI when the number of neurons is reduced.** **(A)** Path estimate from the SSP-PI model. **(B)** The similarity map of the model's output, $\hat{\phi}(t) \cdot \phi(\boldsymbol{x})$, at four different times over the simulation. The green cross marks the true location at that time while the red cross marks the model's estimate.

SSP-PI is initialized with two starting locations, $\phi(0) = \text{unitary}(\phi_1 + \phi_2)$, where 'unitary' refers to normalizing the resulting vector such that all components in the Fourier domain have unit modulus. Figure 3.12A shows the resulting paths given both initial location hypotheses, along with the model's MLE that alternates between the two paths. Figure 3.12B illustrates SSP-PI's self-location probability estimates over time, derived from the similarity of the model's output to SSPs over space. The model maintains two distinct peaks, preserving both location hypotheses even as the relative prominence of each peak fluctuates slightly over time.

**Sensitivity to hyper-parameters.** The performance of the SSP-PI model given different model and task parameters is shown in Figure 3.13. Most of the trends are expected. As the number of neurons per VCO population is increased, the error decreases – though distance error plateaus beyond 800 neurons for an SSP dimensionality of 55 and maximum path frequency of 0.1 The distance error is high when SSP dimensionality is too low ($d = 7$ corresponds to a single grid, making the representation of a unique location impossible), but a size as low as 55 (corresponding to only 9 grids) is sufficient to minimize error with 800 neurons per VCO and a maximum path frequency of 0.1. Figure 3.13 also shows that the PI accuracy is most sensitive to the maximum frequency of band-limited white noise trajectories, with higher frequency paths accruing higher errors.

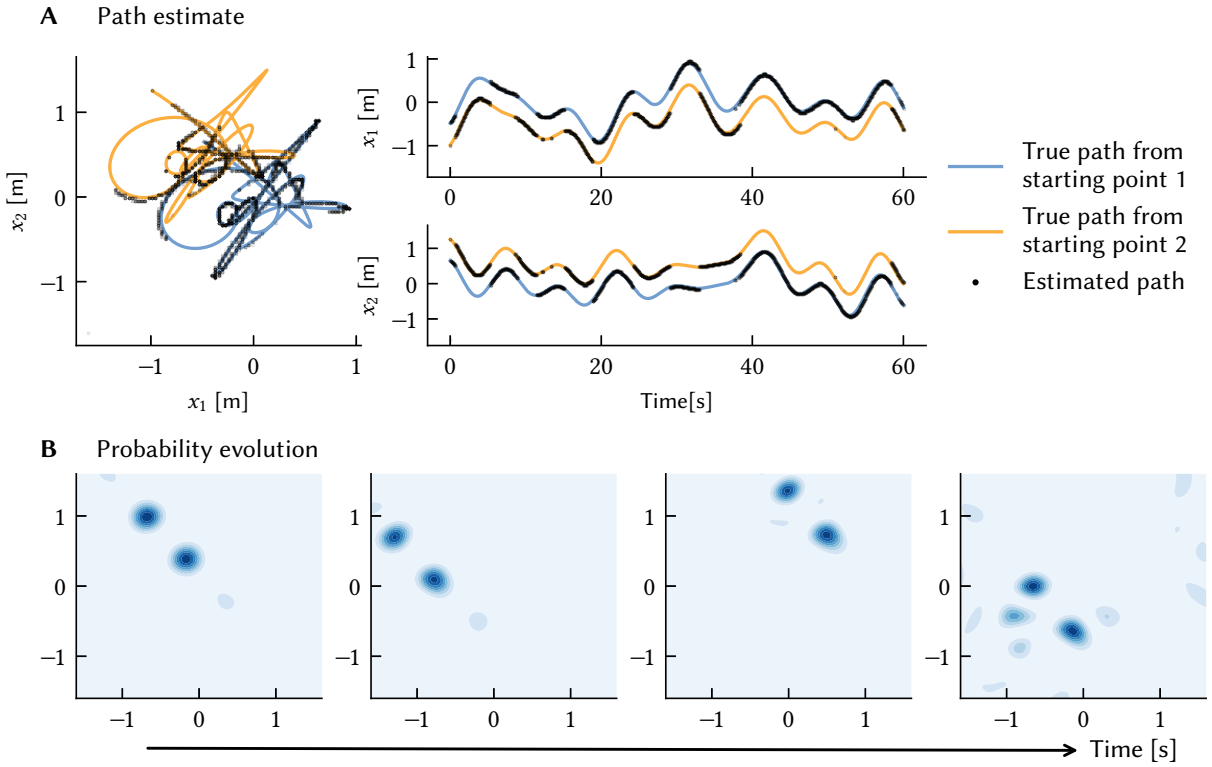



**Figure 3.12: Multiple hypothesis PI with SSP-PI.** The network is initialized with two starting positions, $\phi(0) = \text{unitary}(\phi(\boldsymbol{x}^{(1)}(0)) + \phi(\boldsymbol{x}^{(2)}(0)))$, and then run for 60 seconds. **(A)** The two solid coloured lines depict the ground truth trajectory given starting location 1 ($\boldsymbol{x}^{(1)}(0) = [-0.48, 0.65]$) and starting location 2 ($\boldsymbol{x}^{(2)}(0) = [-0.98, 1.25]$). The self-location MLE from SSP-PI is plotted in black. **(B)** A probability density function over self-location. This is obtained from squaring and normalizing the similarity map of the model's output, $\hat{\phi}(t) \cdot \phi(\boldsymbol{x})$, at four different times over the simulation.

**Model comparisons.** We also test the SSP-PI model on a different set of randomly generated trajectories (Figure 3.14). These 50 second trajectories are directed random walks and thus have much higher frequency content compared to the previous set of paths. The same trajectories were used in Xu et al. (2022) to test a recurrent (non-spiking) ANN model of path integration on 500-step trajectories. They use a 1800-dimensional learned vector representation compared to our 55 dimensional SSP and 1000 LIF neurons per VCO. They obtained an average (over 1000 random trajectories) distance error of $8.0 \times 10^{-2}$ at the end of the simulation and $3.0 \times 10^{-2}$ averaged over the whole trajectory.[9] In comparison, the mean distance error of our model's output is $9.4 \times 10^{-2}$ at the final step and $8.1 \times 10^{-2}$ averaged over the whole trajectory. This demonstrates that our spiking neural network (SNN) path integrator can achieve comparable accuracy to an optimized non-spiking ANN path integrator.

[9]This was reduced to $2.8 \times 10^{-2}$ and $1.7 \times 10^{-2}$, respectively, when the vector representation was regularly decoded to obtain the estimate in physical space and then re-encoded to correct the path integrator. Such re-encoding, or 'clean-up', was not done in the SSP-PI model.

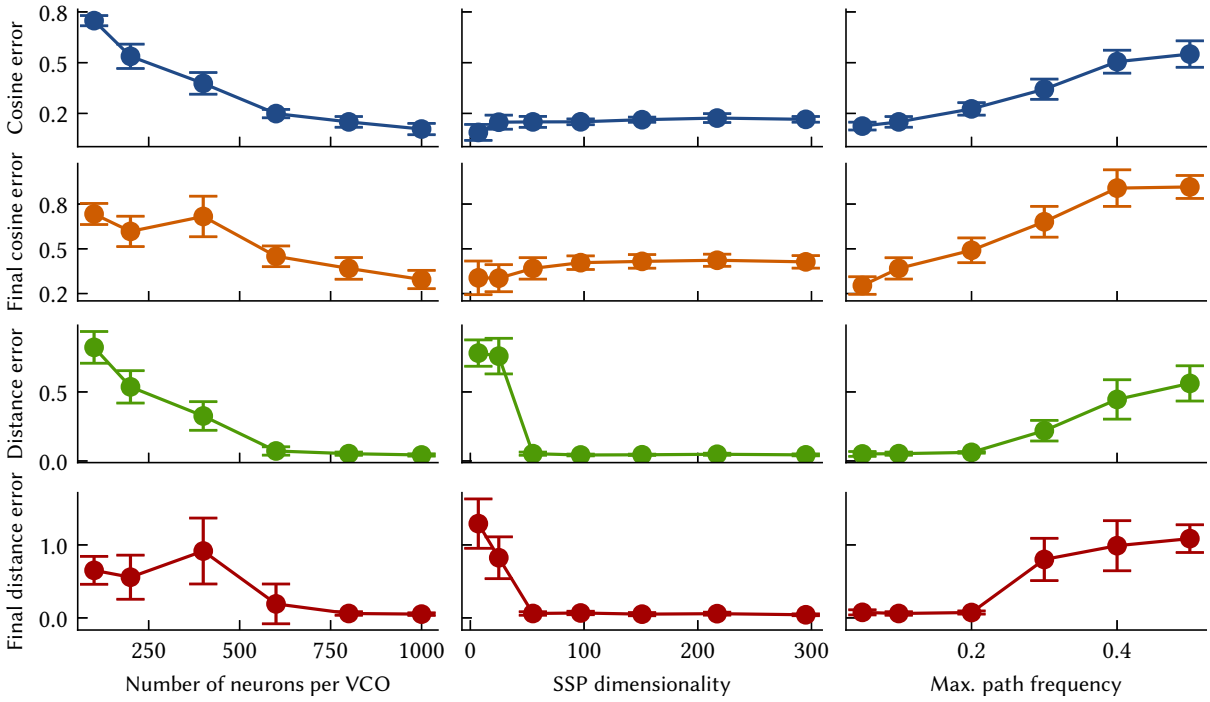



**Figure 3.13: SSP-PI performance given different parameters** on 60 second 2D paths. Each row corresponds to a different error measure – namely, cosine and distance error averaged over ten trials at the final simulation step and averaged over entire trajectories. Each column corresponds to a different parameter of the model that is varied. The number of neurons per VCO population is varied from 100 to 1000, while the SSP size and maximum path frequency are held constant at 55 and 0.1, respectively. The SSP sizes is varied from 7 to 295, while the number of VCO neurons and maximum path frequency are held constant at 800 and 0.1, respectively. The maximum path frequency is varied from 0.05 to 0.5, while the number of VCO neurons and SSP size are held constant at 800 and 55, respectively.

We also compared SSP-PI to GC-SSP-PI introduced in Section 3.4.5 (Figure 3.15). Both models were given equal resources – while the grid cell model has 9 independent recurrent populations (compared to the 27 VCOs in the standard SSP-PI) each is modelled with 3000 LIF neurons (compared to the 1000 used in SSP-PI). The grid cell version performs significantly worse than the original version of the model.

**Firing patterns**. To better understand how the firing patterns in SSP-PI compare to experimental observations, we recorded the activities of neurons in the VCO populations in several experiments. As shown in Figure 3.16, we ran SSP-PI on a 60 second wiggly spiral path. We used this path because it covers space well, allowing firing patterns to be easily discerned. Additionally, Figure 3.16 and Figure 3.17 show activity from SSP-PI over a 400 second trajectory recorded from a rodent (Sargolini et al., 2006).

The VCO neurons are expected to have striped or band-like patterns over space, but, as seen in Figure 3.16B, these patterns only appear in certain areas. This is because these neurons

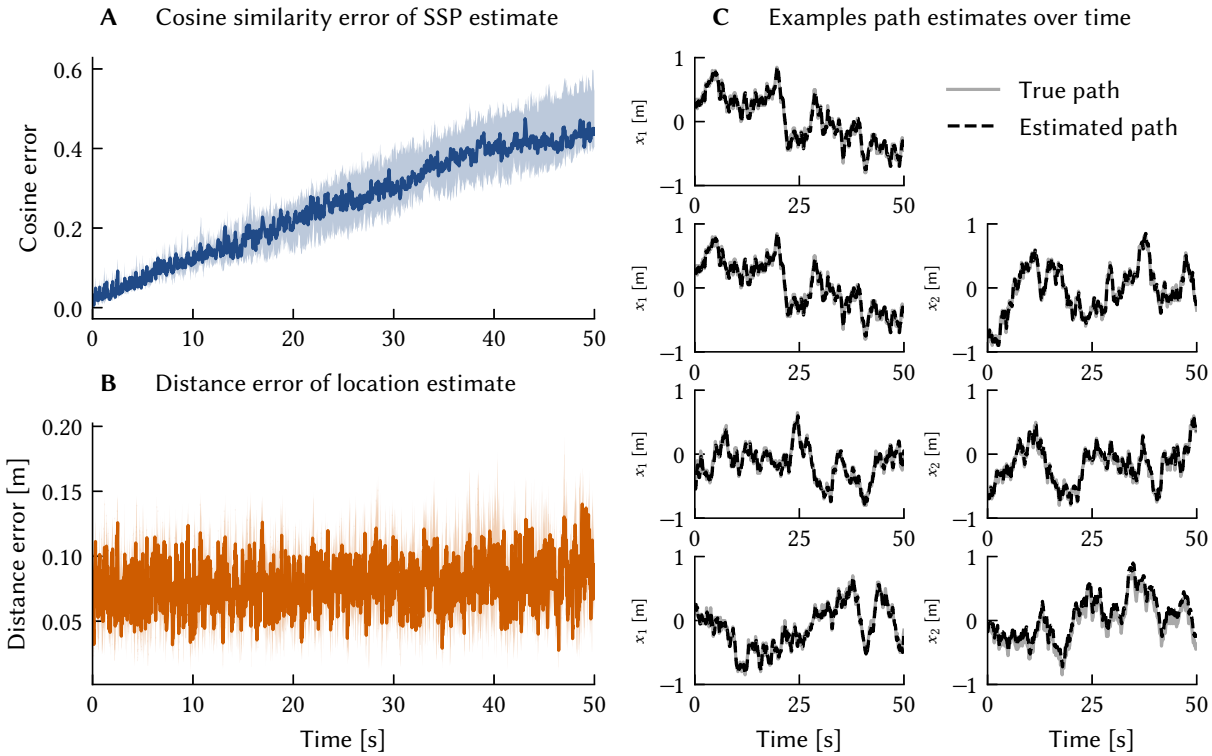



**Figure 3.14: Averaged results from the SSP-PI model** over ten 50 second long 2D paths, each of which is a directed random walk generated via the function used by Xu et al. (2022) to generate 500 steps trajectories. **(A)** The median and 95% bootstrapped confidence intervals of the cosine distance between the SSP encoding of the ground truth and the model's output. **(B)** The median and 95% bootstrapped confidence intervals of the Euclidean distance between the true path and the model's estimate. **(C)** Examples of the trajectories and the path integrator estimates. The first and second dimension of the paths are plotted in the first and second columns, respectively. Each row is a different trial.

are also sensitive to heading direction and speed, unlike the simulated 'perfect' band cell in Figure 3.17B. On the spiral path, spatial tuning patterns are clear due to the correlation between heading direction and spatial location. However, in natural foraging trajectories (Figure 3.16D-E and Figure 3.17C), the banded patterns are less discernible in rate maps or spatial autocorrelograms, although the sensitivity of the heading direction remains evident for some cells. This may explain why theoretical expectations for band cells differ from empirical observations: band cells that perform path integration, according to our model, must be conjunctive, making stripe patterns less prominent. Instead, such neurons might be classified as speed and heading direction cells (such as the neuron in Figure 3.17A from Sargolini et al. (2006)) due to their strong directional sensitivity.

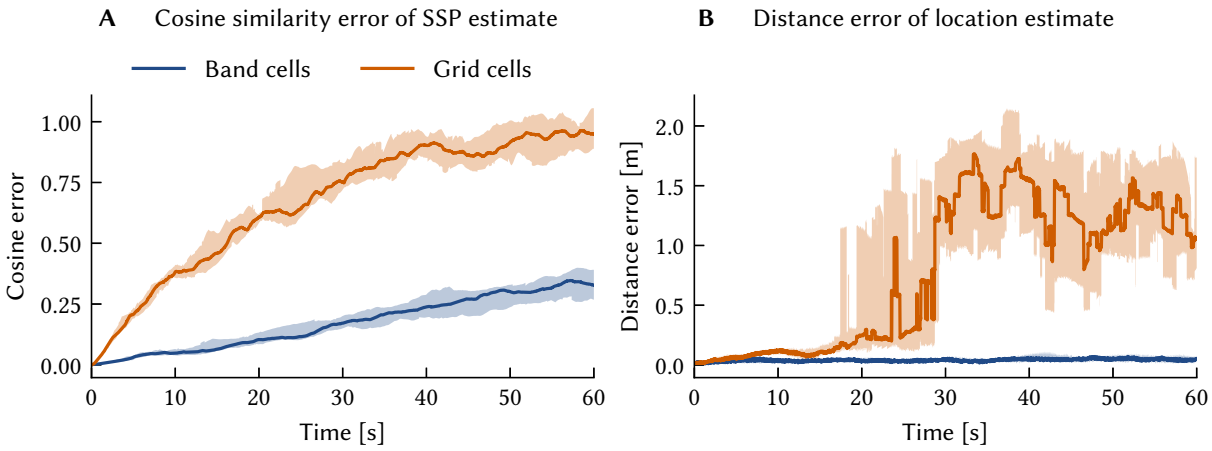



**Figure 3.15: Path integration results for SSP-PI versus the grid cell version.** **(A)** The median and 95% bootstrapped confidence intervals of the cosine distance between the SSP encoding of the ground truth and the model's output. **(B)** The median and 95% bootstrapped confidence intervals of the Euclidean distance between the true path and the model's estimate.

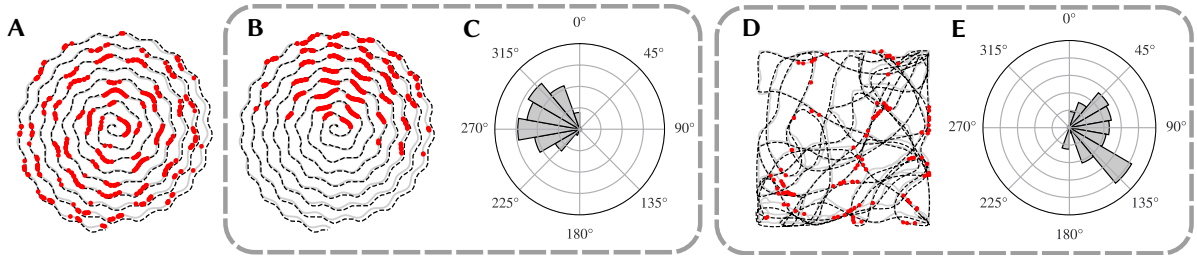



**Figure 3.16: Examples of firing patterns of neurons in SSP-PI.** The SSP-PI model is run using velocity input derived from the trajectories plotted as solid grey lines. The dotted black line shows the model's self-location MLE. Red dots are plotted along this line at locations where a particular neuron fired during the simulation. **(A)** An example grid cell from a population representing the output of SSP-PI. **(B)** An example neuron from a VCO population, which has a banded firing pattern. **(C)** Number of spikes binned by heading direction for the VCO neuron in (B). **(D)** and **(E)** Firing patterns of another VCO neuron when the model is run using velocity input from a rodent trajectories recorded by Sargolini et al. (2006).

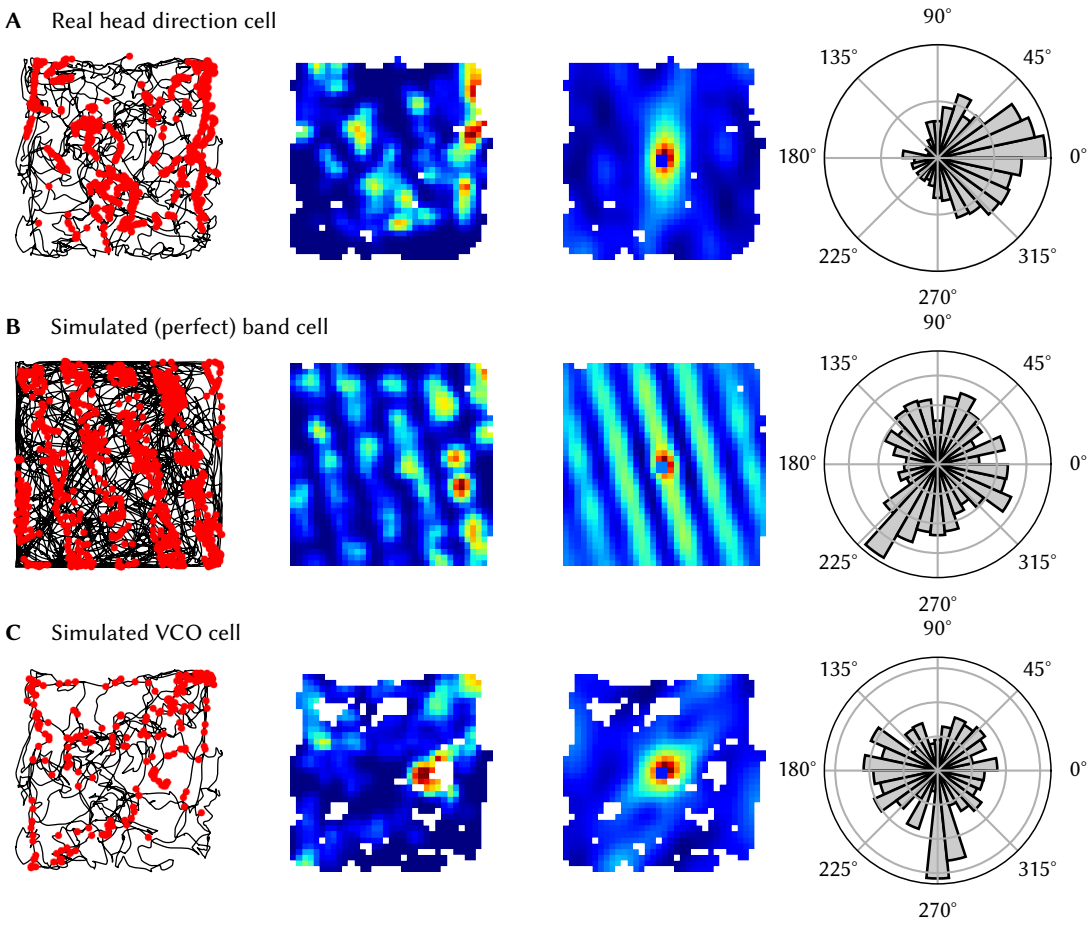



**Figure 3.17: Comparison between firing patterns of simulated neurons and real neurons.** Each row corresponds to a different neuron and shows its spiking activity (first column), a rate map obtained via a kernel density estimation (KDE) on the spiking data normalizing by visitation frequency (second column), a spatial autocorrelogram (third column), and directional tuning curves (last column). **(A)** A real head direction cell, data from Sargolini et al. (2006). **(B)** A simulated band cell from a population (with encoders that pick out individual Fourier components) encoding an SSP without path integration. **(C)** An example of a neuron from a VCO population in the SSP-PI model.

## 3.5 Temporal integration

We have focused primarily on the SSP encodings of the physical space, but these representations can be used to embed any continuous variable in a high-dimensional VSA space. For example, we can represent time in addition to self-location:

$$\phi(\boldsymbol{x}(t), t) = \phi_{\text{space}}(\boldsymbol{x}(t)) \circledast \phi_{\text{time}}(t) = \mathcal{F}^{-1}\left\{e^{i(A\boldsymbol{x}(t)+\boldsymbol{b}t)}\right\}, \tag{3.33}$$

where $\boldsymbol{b} \in \mathbb{R}^d$ is the phase vector used to encode time, $\phi_{\text{space}}$ and $\phi_{\text{time}}$ are the SSP maps for space and time, respectively. Since the rate of change of time is constant, integration of such an SSP can be accomplished by simply adding baseline frequencies (proportional to $\boldsymbol{b}$) to each oscillator. Baseline oscillations are frequently used in OI models and could correspond to hippocampal theta oscillations – but these are not necessary for integrating $\phi(\boldsymbol{x}(t))$ alone. [10]

Furthermore, since bundling can be used to represent multiple SSPs and SPs in a compressed fashion, we can encode trajectories:

$$\Phi(t) = \int_{\tau=0}^{t} \phi(\boldsymbol{x}(\tau), t-\tau) d\tau. \tag{3.34}$$

The result, $\Phi(t)$, represents the past history of the variable $\boldsymbol{x}$ in a high-dimensional vector space. The location along the trajectory $\tau$ seconds in the past can be approximately recalled by querying this vector: $\phi(\boldsymbol{x}(t-\tau)) \approx \Phi(t) \circledast \mathcal{F}^{-1}\left\{e^{i\boldsymbol{b}\tau}\right\}$. This compressed representation will gradually become less accurate as more information is added.

The dynamics of this trajectory representation are given by

$$\dot{\Phi}(t) = \mathcal{F}^{-1}\{i\boldsymbol{b}\} \circledast \Phi(t) + \phi(\boldsymbol{x}(t)). \tag{3.35}$$

The time of all locations stored in the vector are continuously shifted as new locations are added. Circular convolution of $\mathcal{F}^{-1}\{i\boldsymbol{b}\}$ and $\Phi(t)$ is equivalent to multiplying $\Phi(t)$ by a circulant matrix constructed from the vector $\mathcal{F}^{-1}\{i\boldsymbol{b}\}$, and so this is a linear time invariant (LTI) system. In fact, this system has the same dynamics matrix as the Modified Fourier memory system introduced in Stöckel (2022), which is a near-optimal compression of streaming temporal data.

The dynamics in Eq. (3.35) can be realized with fixed frequency oscillators that take $\mathcal{F}\{\phi(\boldsymbol{x}(t))\}$ as input:

$$\frac{d}{dt}\begin{bmatrix} \operatorname{Re}\mathcal{F}\{\Phi\}_1 \\ \operatorname{Im}\mathcal{F}\{\Phi\}_1 \\ \operatorname{Re}\mathcal{F}\{\Phi\}_2 \\ \operatorname{Im}\mathcal{F}\{\Phi\}_2 \\ \vdots \\ \operatorname{Re}\mathcal{F}\{\Phi\}_d \\ \operatorname{Im}\mathcal{F}\{\Phi\}_d \end{bmatrix} = \begin{bmatrix} 0 & -b_1 & & & & \\ b_1 & 0 & & & & \\ & & 0 & -b_2 & & \\ & & b_2 & 0 & & \\ & & & & \ddots & \end{bmatrix} \begin{bmatrix} \operatorname{Re}\mathcal{F}\{\Phi\}_1 \\ \operatorname{Im}\mathcal{F}\{\Phi\}_1 \\ \operatorname{Re}\mathcal{F}\{\Phi\}_2 \\ \operatorname{Im}\mathcal{F}\{\Phi\}_2 \\ \vdots \\ \operatorname{Re}\mathcal{F}\{\Phi\}_d \\ \operatorname{Im}\mathcal{F}\mathcal{F}\{\Phi\}_d \end{bmatrix} + \begin{bmatrix} \operatorname{Re}\mathcal{F}\{\phi\}_1 \\ \operatorname{Im}\mathcal{F}\{\phi\}_1 \\ \operatorname{Re}\mathcal{F}\{\phi\}_2 \\ \operatorname{Im}\mathcal{F}\{\phi\}_2 \\ \vdots \\ \operatorname{Re}\mathcal{F}\{\phi\}_d \\ \operatorname{Im}\mathcal{F}\{\phi\}_d \end{bmatrix}. \tag{3.36}$$

[10] In Orchard (2015), it is shown that theta oscillations are not strictly necessary for path integration in OI models.

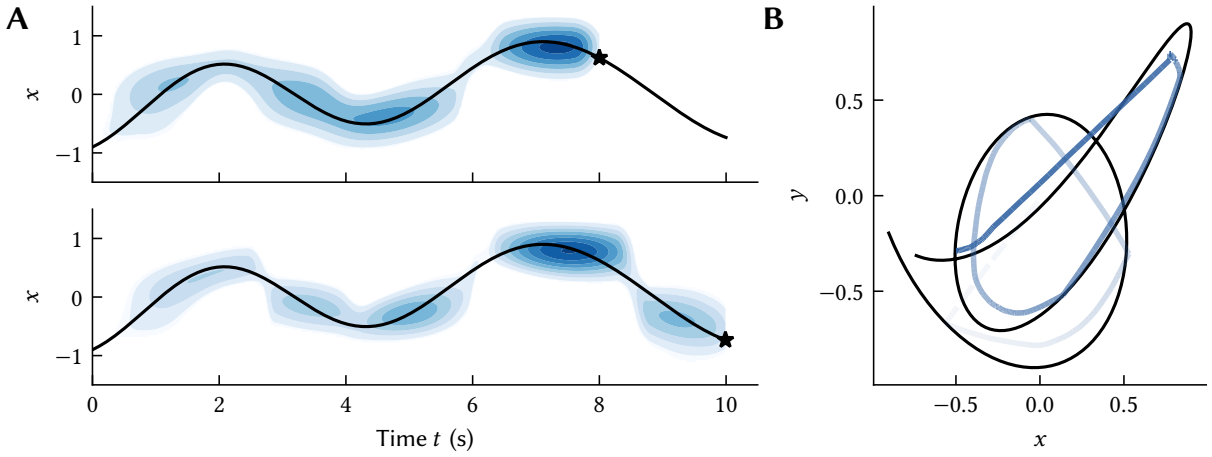



**Figure 3.18: Trajectory representation via temporal integration of SSPs.** Path integration is performed on a 2D path (the black line in **B**). The output of the path integrator is fed into a temporal integrator, with dynamics given by Eq. (3.34). **(A)** Two panels show the $x$-dimension of the trajectory output at different points (indicated by black stars) over the simulation time. The output $\Phi(t)$ is visualized by a contour plot of its similarity with SSP representations across $x$-space. This is analogous to a probability distribution of the $x$ position at different points in the past (see Section 2.4.4). **(B)** The 2D trajectory estimate, decoded from $\Phi(t)$ at the end of the simulation, as a blue line that fades with how far the estimate is into the past.

Figure 3.18 shows the output of an SNN that realizes these dynamics to maintain a trajectory memory. The encoded SSP memory $\Phi(t)$ is constantly updated by the recurrent neural network that approximates Eq. (3.34). To interpret $\Phi(t)$ at two points in time, we use shading to indicate the similarity between $\Phi(t)$ and the ideal SSP for that location in the graph. Darker shading can be interpreted as a higher probability that the trajectory was at that location at that point in time according to the integrator memory.

## 3.6 Discussion

In this chapter, we explored the application of SSPs in modelling biological representations and computations, specifically showing that SSPs can model grid cells for more accurate location decoding compared to random SSP projections and direct space encodings. Furthermore, we have derived the dynamics of SSP representations and used these to build an SNN model of path integration.

### *3.6.1 Performance of SSP-PI*

The SSP-PI model accurately integrates velocity cues in both 2D and 3D and generalizes to higher-dimensional spaces without requiring changes in the structure of the network or connection weights. Consider two SSP representations of different variables, $\boldsymbol{x}(t) \in \mathbb{R}^m$ and

$\boldsymbol{y}(t) \in \mathbb{R}^p$ where $p \neq m$. SSPs can be constructed to map to a space of size $d$ in both cases, but different phase matrices are used. In either case, the dynamics of the SSPs are given by Eq. (3.26). Thus, the same set of VCOs can model the dynamics of $\phi_x(\boldsymbol{x}(t))$ and $\phi_y(\boldsymbol{y}(t))$; the only difference between their dynamics is the calculation of the control frequencies, $\omega_j$. These frequencies are input to the path integrator, with incoming synaptic weights performing the linear transformation from self-motion to frequencies. The same path integrator network can receive input from multiple sources, and synaptic gating may be used to switch between localization in different dimensional spaces and coordinate frames.

For instance, an animal could flexibly switch from 2D to 3D PI when moving between land and water or air. Similarly, a PI network could be adapted for non-spatial tasks. For example, time perception might leverage a PI model where an internal state, representing an estimate of elapsed time or duration, is updated based on contextual "velocity" signals (*e.g.*, task difficulty or attention level). Alternatively, a PI model could be applied to music perception, updating a "position" along a scale or rhythm in response to tempo and tonal progression. A general-purpose network for maintaining internal estimates of dynamic variables, whether spatial or non-spatial and regardless of dimensionality, is broadly useful.

While the SSP-PI model can integrate accurately, the error incurred depends substantially on the frequency content of the velocity data and, to a lesser extent, the number of neurons used in recurrent populations. Furthermore, the characteristics of the error behaviour of SSP-PI are interesting. Errors in the model do not typically manifest as slow drift; instead, path estimates begin to jump wildly due to the constructive interference of plane waves creating new peaks.

Additional error correction mechanisms are required for long-term and stable self-location encoding. Such mechanisms could incorporate external cues (explored in the next chapter) or additional 'clean-up' via grid cells (Bush et al., 2014). The need for error corrections is expected since all noisy path integrators accumulate error over time. This is true for both models and animals. OI models are typically accurate on a time scale of seconds to a few minutes (Zilli et al., 2009), while spiking CANN models can remain stable for 1-10 minutes and non-spiking CANN models can remain stable for 10-30 minutes (Burak et al., 2009). Rodents can perform path integration for minutes to tens of minutes even when external cues are limited (Hafting et al., 2005; Chen et al., 2016; Dannenberg et al., 2020).

### *3.6.2 Biological realism of SSP-PI*

The VCO populations of SSP-PI consist of spatially sensitive neurons, but these neurons do not resemble place or grid cells. Each population represents a single Fourier coefficient of the SSP and its frequency (derived from velocity). This results in neurons with conjunctive sensitivity to heading direction, speed, and spatial position (in a periodic fashion, resembling a plane wave). Their firing patterns are velocity-dependent bands or stripes, as predicted by most OI models (Burgess et al., 2007). The SSP-PI model can be modified to use grid cells instead of

band cells, as done in OI models that couple VCOs (Burgess, 2008), but this results in poorer performance. However, note that grid cells have been effectively utilized in CANN models for highly accurate path integration (Burak et al., 2009), and we have not implemented coupling mechanisms to fully exploit the grid structure. Still, our results suggest that independent VCOs are better suited for generalizing path integration across spaces of varying dimensionality and different SSP parameterizations.

Although grid and place cells have been well documented and studied, band cells have not. Band-like cells, or 'stripe cells', exhibit periodic firing patterns in bands or stripes. There is evidence that such neurons may exist. For example, in Krupic et al. (2012), cells with elongated band-like firing fields were observed when the animals were in specific environments, although the grid cells were more stable. In Welday et al. (2011), neurons with periodic firing patterns (theta cells) with speed and directional tuning were observed in a high proportion of recording sessions in the anterior thalamus and at least half of their recording sessions in the medial septum and hippocampus.[11] More recently, Abad Perez et al. (2024) reported band-like cells in the mouse subiculum. This growing body of evidence suggests that the band cells in the SSP-PI and other OI models are not necessarily biologically unrealistic.

Furthermore, the conjunctive tuning of band cells in SSP-PI to speed and direction (required for implementing a controlled oscillator in the NEF) affects their appearance. We find that on natural paths the banded spatial pattern is not readily discernible while the heading direction sensitivity is (see Figure 3.16 and Figure 3.17). The band-like nature of such cells is most apparent in environments with strong correlations between heading direction and location, such as the spiral maze in Figure 3.16.

We hypothesize that neurons in deeper layers of the MEC categorized as conjunctive head direction cells may in fact have spatially periodic tuning curves and that such neurons are responsible for path integration. The deeper layers of the MEC receive input from the hippocampus and many cortical areas and project to the superficial layers of the MEC, where 'pure' grid cells are primarily found. These layers project back to the hippocampus via the trisynaptic and monosynaptic pathways. Grid cells may act as a clean-up step, in which additional attractor dynamics are used to produce a single valid SSP from VCO output (*e.g.*, 'picking out' a single peak from Figure 3.11), as well as providing an efficient basis for downstream decoding of place cells and the association of spatial context and memory.

### *3.6.3 Relation to prior PI models*

The SSP-PI model is quite similar to many existing PI models, combining features of both CANN and OI models. In OI models, position is encoded by the phase differences of VCOs. The dynamics of SSP components in the Fourier domain are exactly those of VCOs. CANN models assume a particular form of recurrent connectivity and periodic boundary conditions,

[11]The anterior thalamus projects onto the hippocampus and Neurons in the medial septum release neurotransmitters that modulate the neuronal activity of the entorhinal cortex (EC).

resulting in a circulant recurrent weight matrix. One can model a velocity signal as either an input that directly shifts the activity bump or as a signal that modulates the recurrent connections. Similarly, SSP dynamics can be described as continuous binding or circular convolution – in other words, continual multiplication with a circulant matrix derived from velocity. Furthermore, we have added attractor dynamics to SSP-PI to constrain its output to the toridal space of unitary vectors. This makes the SSP-PI a variety of CANN, with a unique nonlinear attractor.

A more recent model explored PI based on fractional binding of Fourier Holographic Reduced Representations (FHRRs) with hexagonal-structure, using an attractor network (Kymn et al., 2024). Although this model shares similarities with the approach in this chapter, SSP-PI precedes Kymn's model and demonstrates PI over long durations in both 2D and 3D spaces includes notable extensions.

Path integrators have been extensively researched, and their mechanisms are well established and have been repeatedly re-derived and rediscovered. The main contributions of the SSP-PI model, compared to previous work, include the use of nonlinear oscillators to enforce additional structure on the attractor, and the demonstration that mechanisms underlying OI and CANN models are equivalent to fractional binding, reinterpreting the grid and place cell activity as encoding SSPs.

This reinterpretation connects the brain's encoding of continuous variables to a specific VSA. Furthermore, situating the brain spatial code within the SPA framework has important theoretical and practical benefits. With this viewpoint, it is easy to see how the output of a neural path integrator may be used in various downstream tasks. The model can be extended to include time and construct trajectory memories, as shown in Section 3.5. Location encodings can be bound with symbol-like representations of objects, landmarks, features, or context. Moreover, it enables the integration of the model with pre-existing neural SPA models of cognitive functions, including associative memory (Stewart et al., 2011), serial memory (Choo et al., 2010), working memory, action selection, and decision-making (Stewart et al., 2010a).

## 3.7 Conclusion

We have proposed a model of path integration derived from SSP representations of space. This extends VSAs to support path integration and extends the representation of path integrator output to support binding with symbol-like representations. We believe that coupling low-level neural models with cognitive architectures in this manner is critical for building sophisticated models of biological cognition. The activity of hippocampal neurons has provided valuable insight into how the brain represents space, but does not reveal its underlying algorithms, such as PI. By constructing a path integrator out of spiking neurons, we have linked the activity of spatially sensitive neurons to a symbolic description of PI using SSPs.

Although the model we propose here is simple, it provides a new, scalable means of

integrating discrete symbol-like and continuous representations in a spiking neural substrate. However, integration with sensory input is critical for error correction, and we have yet to demonstrate the binding of PI output with other representations to, for example, construct cognitive maps.

# Chapter 4

# Simultaneous Localization & Mapping

> We shall argue that the hippocampus is the core of a neural memory system providing an objective spatial framework within which the items and events of an organism's experience are located and interrelated.
>
> — John O'Keefe and Lynn Nadel
> *The Hippocampus as a Cognitive Map*

**Summary**

This chapter extends the SSP-PI model to develop SSP-SLAM, a semantic spiking SLAM system that integrates a path integration network with an associative memory network for environment mapping. By leveraging SSPs and Semantic Pointers, SSP-SLAM represents spatial and semantic information to create queryable cognitive maps, which are used to correct drift in path integration. Tested in both 2D and 3D environments, SSP-SLAM demonstrates improved localization accuracy over SSP-PI through its error correction mechanisms. Additionally, the model is evaluated on a neuromorphic hardware emulator, showcasing its potential for deployment on energy-efficient neuromorphic systems, paving the way for biologically plausible and resource-efficient SLAM systems.

**Material in this chapter from prior publications**

The text and material in this chapter were adapted from Dumont et al. (2023a), originally published in *Frontiers in Neuroscience*.

## 4.1 Introduction

Simultaneous localization and mapping (SLAM) is the computational process of keeping track of one's location while navigating an unknown environment and, simultaneously, creating a map of the environment. Accurate localization is required for building a metric map from egocentric observations, but, as we saw in the previous chapter, errors in localization accumulate when relying solely on internally generated signals or self-motion cues. An allocentric environment map can be used to correct these errors via loop closure, making localization and mapping interdependent processes (Figure 4.1). SLAM is a core problem in mobile robotics, particularly in applications where high-precision GPS data is not available, such as in autonomous underwater vehicles or planetary exploration (Kim et al., 2013; Palomeras et al., 2019; Geromichalos et al., 2020).

Biological systems have evolved to solve exactly these problems. Animals are capable of navigating in novel environments, deducing their current location, and retracing their steps. While path integration (PI) plays a role in these functions, landmark navigation depends on other processes that use external stimuli (Griffin et al., 1998; Etienne et al., 2004; Fischler-Ruiz et al., 2021). The hippocampus has been identified as a key region where information from PI and external sensory landmarks are potentially integrated to build cognitive maps and provide corrections to self-location estimates, much like SLAM systems in robots (Save et al., 2000; Li et al., 2017). In fact, Safron et al. (2022a) have characterized the hippocampal-entorhinal system as "the most sophisticated of all biological SLAM mechanisms."

While SLAM is a well-studied problem, and modern mobile robots are capable of performing SLAM, animal navigational abilities are still superior; they are more robust, efficient and adaptive. Animals can use information from multiple sensory modalities (*e.g.*, visual, olfactory, auditory, magnetoreception, and idiothetic cues) for navigation and correction. Additionally, animals are able to navigate and map their environment in real-time using power-constrained computational resources, which is something that robots are still not able to achieve. Brains are far more energy efficient than the GPUs or CPUs used to execute typical SLAM algorithms – the brain consumes around 20 Watts of energy while a single modern graphics card requires around 350 Watts. Some researchers, taking inspiration from biology, have developed SLAM algorithms that are optimized for online processing and that can run on resource-constrained platforms. For instance, neuromorphic hardware – designed to mimic the functionality of biological spiking neural networks (SNNs) – is particularly well-suited for resource-constrained computing because it is designed to be energy-efficient and can perform brain-like computations using minimal resources (Bersuker et al., 2018; Thakur et al., 2018; Rathi et al., 2021).

Hippocampus research has influenced the development of a category of biologically-inspired SLAM models, including RatSLAM (Milford et al., 2004), DolphinSLAM (Silveira et al., 2015), and NeuroSLAM (Yu et al., 2019), among others. Some of these models have demonstrated performance comparable to contemporary state-of-the-art models. However, this is still an active area of inquiry, with questions remaining regarding scalability, biological

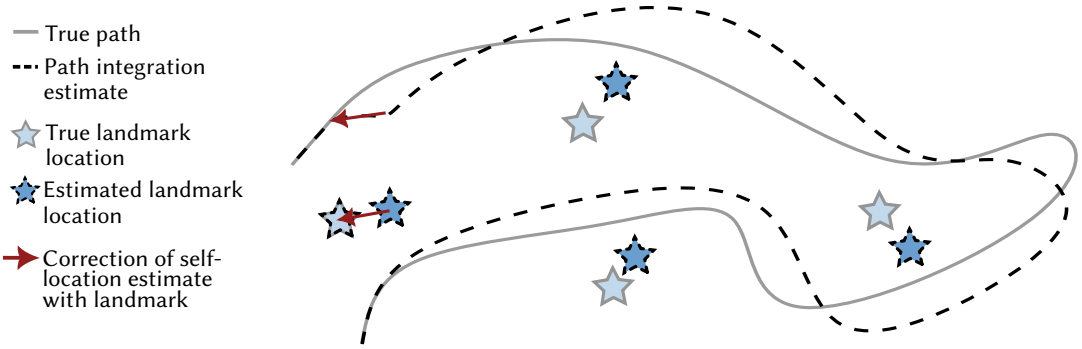



**Figure 4.1: A diagram illustrating simultaneous localization and mapping (SLAM).** An estimate of self-location is maintained and used to record landmark locations. When landmarks are re-encountered, the discrepancy between the observed and recalled landmark location is used to correct the self-location estimate ("loop closure"). Details regarding self-location updates, the form of observations, and the loop closure computation vary among particular SLAM algorithms.

plausibility, and deployability on neuromorphic hardware.

In this chapter, we take inspiration from cognitive science as well as neuroscience to develop a novel spiking neural network SLAM system, called Spatial Semantic Pointer (SSP)-SLAM, built using the Semantic Pointer Architecture (SPA) and SSPs. The SPA allows us to represent and compose neural representations of symbols (such as landmark identity) and continuous variables (such as locations) and thus construct environment maps. These maps are core to the functioning of SSP-SLAM, as they integrate semantic information, while being combined with the SSP-based path integrator. The resulting model provides the following contributions:

- We propose and implement a novel spiking neural network SLAM model. Additionally, we demonstrate that the model can be run on an emulator of Intel's Loihi neuromorphic chip.
- Our model is constrained to only use quantities that are known to be represented in hippocampal-entorhinal system, including the spatial information encoded by head direction cells, object vector cells, place cells, and grid cells. Furthermore, biologically plausible, Hebbian-like learning rules are used to learn an environment map in the form of an associative memory.
- We explore compositional semantic map representations using the SPA and the principles of Vector Symbolic Algebras (VSAs) more broadly. We demonstrate how such maps can be queried post-training to recall what landmarks were in particular areas, recall where landmarks belonging to certain categories or with certain features are located, and compute the vector between self-location and landmarks in memory.

The structure of this chapter is as follows. In Section 4.2, we review past research on biologically-inspired SLAM systems. The complete SSP-SLAM model is presented in Section 4.3.

It combines the path integrator model introduced in Section 3.4 of Chapter 3 and an associative memory approach to cognitive mapping. Following this are results and discussion in sections 4.4 and 4.5.

## 4.2 Background

### *4.2.1 Neuroscience*

Animals have access to a plethora of external sensory information, such as visual landmarks and odour trails. Such multi-sensory information is used in conjunction with path integration to create cognitive spatial maps of environments – the binding of spatial and non-spatial features. Animals use such cognitive maps to keep track of food locations, compute novel routes to goals (Tolman, 1948), and encode distance and paths to goals (Howard et al., 2014). Additionally, cognitive maps and external cues are required for accurate localization and navigation. For example, Fischler-Ruiz et al. (2021) found that rats could not accurately navigate distances over 2m on a virtual linear track using path integration alone. However, they could navigate and maintain stable place cell fields when sparse odour landmarks were added.

Several areas of the medial temporal lobe likely contribute to environment mapping and place recognition for loop closure (Figure 4.2). The parahippocampal cortex, which surrounds the hippocampus, processes information from the visual cortex and other sensory areas and is involved in landmark and scene encoding and recognition (Epstein et al., 1998; Aminoff et al., 2013). Spatial view cells, which respond when an animal looks at a specific region of an environment regardless of its location, are found in both the parahippocampal gyrus and hippocampus (Feigenbaum et al., 1991; Rolls et al., 1997). Additionally, the hippocampus associates landmarks and sensory cues with location, containing place cells that respond to features and landmarks - both distal and local (Knierim, 2002). It also contains boundary vector cells that fire when an animal is at a given distance and direction from any wall or environment boundary (Lever et al., 2009), and landmark vector cells that fire when an animal is at a given distance and direction from a particular landmark or type of landmark/object (Deshmukh et al., 2013). Similar neurons, called object vectors cells, are found in the medial entorhinal cortex (MEC), though these fire independently of object identity (Høydal et al., 2019).

Neurons across various regions of the medial temporal lobe thus encode allocentric spatial information, forming an environmental map that may be used in relocalization, *i.e.*, for SLAM. Environmental boundaries and familiar landmarks/objects help in realigning an animal's internal map with physical space. Jayakumar et al. (2019) demonstrated that rats could correct their internal maps when presented with consistent external landmarks, even after long periods of disorientation. Moreover, the integration of multi-sensory cues, such as combining visual, auditory, and olfactory information, further refines spatial awareness and reduces navigational errors (Wang, 2016). Some work has reported correction in the form of resetting self-location estimates (Zhao et al., 2015; Harootonian et al., 2022), while others have observed estimate

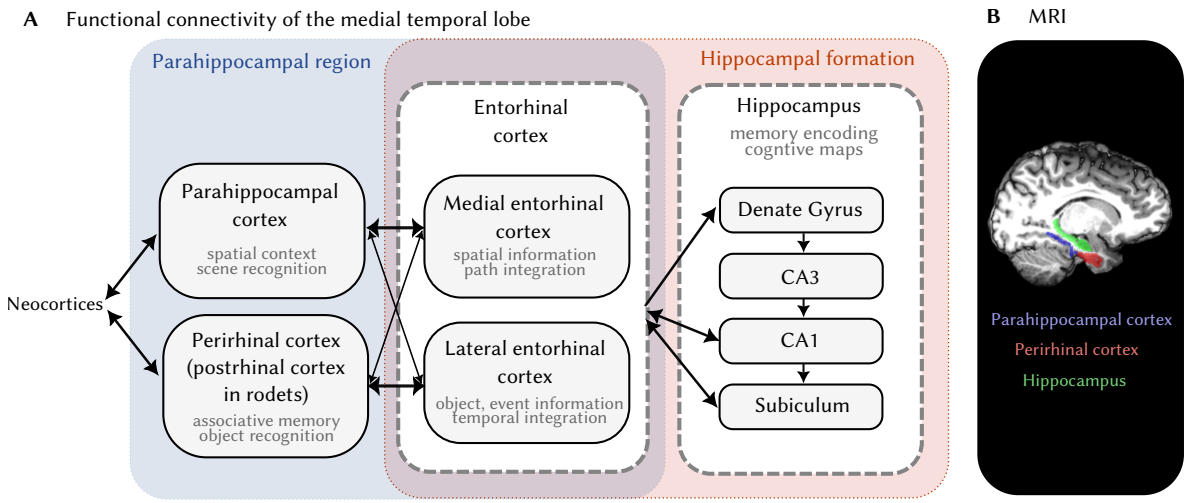



**Figure 4.2: Anatomy of the medial temporal lobe. (A)** Connectivity within the medial temporal lobe, which includes the parahippocampal cortex, perirhinal cortex, entorhinal cortex, and hippocampus. Note that the parahippocampal region and hippocampal formation overlap in this figure; the entorhinal cortex is part of the parahippocampal region and is sometimes included in a broader definition of the hippocampal formation. Both the parahippocampal and perirhinal cortices project to the medial and lateral entorhinal cortices. However, the parahippocampal cortex – involved in processing spatial context and landmark encoding (Eacott et al., 2005; Brown et al., 2010; Marchette et al., 2015) – projects most strongly to the MEC, and the perirhinal cortex – involved in processing object-related information (Clarke et al., 2014) – projects strongly to the lateral entorhinal cortex (LEC; Witter et al., 2017). Figure adapted from Weilbächer et al. (2016). **(B)** An anatomical MRI of a human brain, with certain regions of the medial temporal lobe highlighted. Figure adapted from Libby et al. (2012).

shifts through reliability-weighted averaging (Chen et al., 2017; Harootonian et al., 2022).

### *4.2.2 Models of neural mapping and SLAM*

There are many models of spatial mapping and memory formation in the hippocampal formation. Associative learning between latent representations of self-location and environmental cues – often using Hebbian principles – is frequently introduced in PI models to correct drift, enabling stable and accurate localization. In some cases, associative learning is used to store a cognitive map in the connections between place or grid cells and populations representing landmarks (Burgess et al., 1994), local views (Samsonovich et al., 1997), or environment boundaries (Barry et al., 2006; Hardcastle et al., 2015). Other models introduce a further level of flexibility by representing space through hidden state sequences derived from sensory inputs, allowing them to differentiate contexts even when observations are similar. For instance, the Clone-Structured Cognitive Graph (George et al., 2021) and Tolman-Eichenbaum Machine (Whittington et al., 2020) are two examples; these models represent environments as dynamic, context-sensitive graphs, with nodes representing spatial or relational states and edges captur-

ing transitions or sequential context. By structuring cognitive maps as latent graphs, these models support generalization across both spatial and abstract domains. Most pertinent to our work however, is past work on biologically inspired SLAM systems.

The development and implementation of SLAM algorithms for mobile robots has garnered significant attention. These approaches generally involve recursive Bayesian estimation, implemented using methods such as Kalman Filters (Smith et al., 1990; Brossard et al., 2018) and Particle Filters (Montemerlo et al., 2002; Sim et al., 2005). The uncertain self-location estimate is represented as a probabilistic distribution over possible robot poses (position and orientation). This representation can take various forms, including the parameters of a Gaussian distribution, a set of particles, a graph structure, or, in one instance, through Random Fourier Features (RFFs) (Kapushev et al., 2021). The form of the environment map in these algorithms also varies – some models use occupancy grids to capture the geometry of the environment (Stachniss et al., 2004; Bircher et al., 2016), while others use graphs to represent topological maps (Thrun et al., 2006; Sünderhauf et al., 2012).

In recent years, researchers have focused on incorporating semantic information into SLAM systems, using deep artificial neural networks, particularly convolutional or recurrent neural networks for object detection and semantic segmentation. The use of semantic information in SLAM has been found to improve the performance and robustness of robot localization (Frost et al., 2016; Stenborg et al., 2018; Bowman, 2022). Furthermore, robots equipped with semantic SLAM have the potential to perform higher-level tasks, such as creating routes based on human instructions that refer to objects within their environment.

Concurrently, an alternative approach to SLAM, drawing inspiration from the brain, has continued to develop novel algorithms with the goal of improving the efficiency and robustness. In this line of research, neural models of PI are used for localization (Milford et al., 2004; Milford et al., 2016; Silveira et al., 2015; Yu et al., 2019). The seminal application of neural-inspired methods to SLAM is RatSLAM, in which visual odometry is used to drive a continuous attractor neural network (CANN; Milford et al., 2004; Milford et al., 2016). The CANN consists of "pose cells" (similar to the place and head direction cells found in the hippocampal formation) and maintains an estimate of self-location and orientation. Sensor data is processed outside the neural network to create a template array.[1] When a new template is observed, a new "local view cell" (similar to spatial view cells found in the hippocampus) is added to the network. The population of these cells is sparsely connected to the CANN, with associations learned with Hebbian rules. Additionally, a graph is constructed and updated with a graph relaxation algorithm online to create a topological environment map. Its nodes store experiences, which contain pose and local view cell activity vectors along with estimated pose.

While RatSLAM and many other biologically-inspired SLAM algorithms draw on findings in neuroscience, they are not intended as biologically-plausible models of cognitive mapping. For instance, the graph relaxation algorithm is not neurally implemented – though some SLAM

[1]Typically visual input is converted to an intensity profile vector, however other forms of template arrays are admissible.

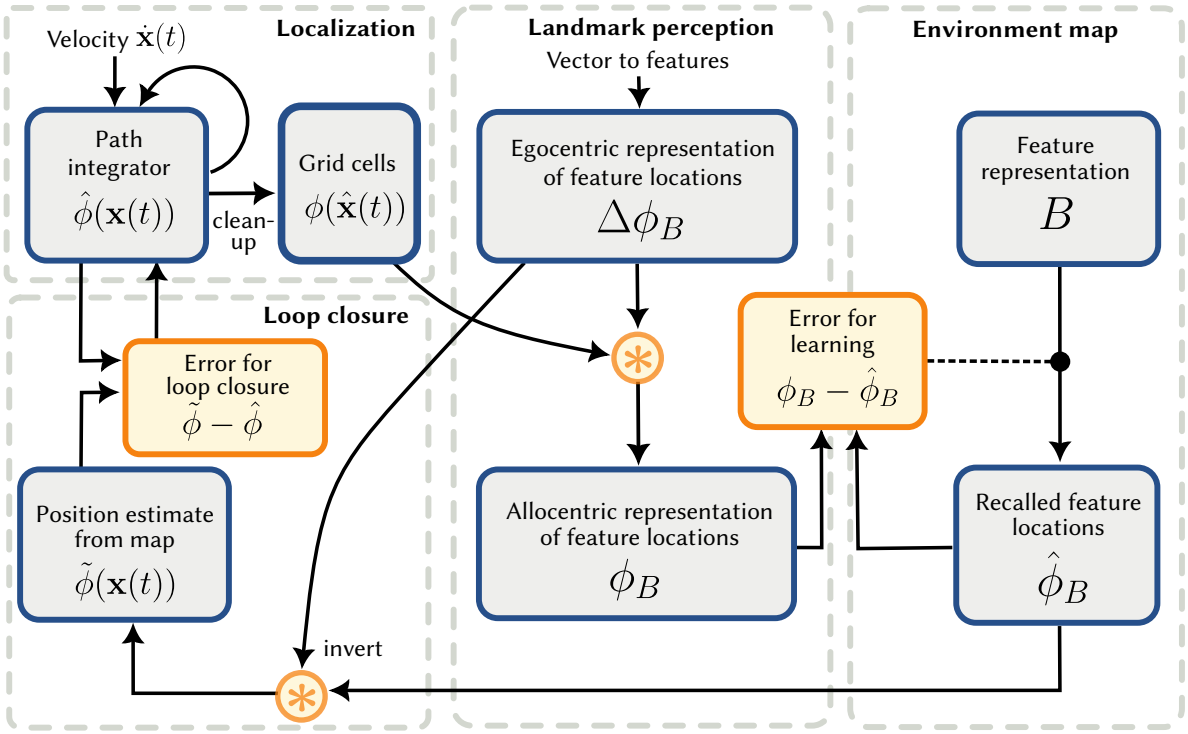



**Figure 4.3:** The SSP-SLAM model. Output of the localization module is used (along with the egocentric feature locations encoded by the object vector population) to train an associative memory network, which can be thought of as an environment map. The output of this map is, in turn, used for error correction of the PI model. Note that, while this shows only a single association network (the one required for loop closure), other association networks may be trained concurrently (*e.g.*, maps from location to features) as is done in Section 4.4.1. In this diagram, blue blocks denote neural populations, orange blocks are error populations, and orange ⊛ icons are circular convolution networks. Solid lines indicate neural projections and dashed lines indicate a modulatory effect on a set of synapses.

models use only the associative network as their map (Tang et al., 2019; Kreiser et al., 2020b). However, to our knowledge, no biologically plausible SLAM algorithm has yet used semantic or compositional representations for mapping. VSAs, which link symbolic representation with neural implementation, combined with SSPs, provide a suitable framework for modelling semantic SLAM in spiking neural networks, and we employ this approach in this chapter.

## 4.3 SSP-SLAM model

In this section, we develop a spiking neural network SLAM model using semantic pointers, SSPs and the Neural Engineering Framework (NEF). The model, SSP-SLAM, consists of six main neural populations, grouped into four modules, that provide all the necessary functionality (Figure 4.3).

- **Localization module**
  - *Path integrator (SSP-PI):* A network maintaining an allocentric self-location estimate, represented as an SSP, $\hat{\phi}(\boldsymbol{x}(t))$, that is dynamically updated using a velocity signal. Specifically, this is a recurrent neural network, consisting of many sub-populations representing controlled oscillators.
  - *Grid cell population:* A population representing a 'cleaned-up' version of the SSP self-location estimate, $\phi(\hat{\boldsymbol{x}}(t))$.
- **Landmark perception module**
  - *Object vector population:* A population that encodes the SSP representation of distances and directions to landmarks and environmental features in view – *i.e.*, an egocentric representation of feature locations.
  - *Object location population:* A population that performs circular convolution to obtain an allocentric SSP representation of feature locations.
- **Environment map module**
  - *Associative memory network:* A network that learns a mapping between landmarks and locations using the biologically plausible prescribed error sensitivity (PES) and Voja learning rules.
- **Loop closure module**
  - *Map estimate population:* A population that performs circular convolution to obtain an alternative estimation of self-location using the environment map. This provides corrections to the path integrator.

Each element of the SSP-SLAM model is described in more detail below.

### *4.3.1 Localization Module*

The localization module consists of the SSP-PI model introduced in Chapter 3, together with a population of grid cells. The output of the PI network is an SSP estimate of self-location, $\hat{\phi}(\boldsymbol{x}(t))$. However, this estimate will become noisy over time, eventually leaving the manifold of SSPs and hence requiring some sort of projection back to the SSP space. Consequently, a 'clean up' operation is required to improve its performance.

We use the projection to grid cells to perform this clean up. Recall as described in Section 2.4.4, SSPs can be used to construct probability distributions. Here, the PI network is initialized to encode the SSP $\phi(\boldsymbol{x}(0))$, from which a prior probability distribution can be computed. At every simulation timestep, this belief state is updated according to the dynamics given in Eq. (3.28). Then, the probability density of the agent being at a location $\hat{\boldsymbol{x}}$ is

$\hat{f}(\hat{\boldsymbol{x}}) \approx (\phi(\hat{\boldsymbol{x}}) \cdot \hat{\phi}(\boldsymbol{x}(t)) - \xi)^+$, where $\xi$ is a normalization constant. The position estimate of the PI model is taken to be the $\hat{\boldsymbol{x}}$ that maximizes this probability, *i.e.*, the maximum likelihood estimate (MLE). The SSP representation of the MLE, $\phi(\hat{\boldsymbol{x}})$, is computed as a part of the 'clean-up' process and supplied to the grid cell population.

### 4.3.2 Landmark Perception Module

In the SSP-SLAM model, the agent not only receives a self-velocity signal as input, but additionally receives observations of its local environment. The construction of SNNs that use raw sensory data to recognize objects and estimate their displacement from the observer remains an active area of research. For example, Osswald et al. (2017) presented an SNN model and neuromorphic demonstration of stereo-correspondence in 3D space. Spiking neural algorithms for object detection (Kim et al., 2020a) and place recognition (Hussaini et al., 2022) have also been developed. Moreover, deep learning has proven to be highly successful in computer vision tasks such as semantic segmentation (Lateef et al., 2019), and these pre-trained artificial neural networks can be converted to spiking neural networks (Cao et al., 2015; Hunsberger et al., 2015). However, visual processing of raw sensory data is outside the scope of this thesis. Instead, we assume that information regarding distance to landmarks and landmark identity is provided directly as input to SSP-SLAM.

Specifically, we let $\{B_1, B_2, \dots\}$ be a set of Semantic Pointers (SPs) that represent features or landmarks in an environment, at locations $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \dots\}$. The SSP representation of the vector from the agent to each landmark within the agent's field of view, $\phi(\boldsymbol{x}_i - \boldsymbol{x}(t))$, is given as input to SSP-SLAM. The neurons in this population have activity patterns like those of object vector cells (OVCs) in the MEC, so we call the population the OVC population. The output of the path integrator and OVC populations are bound together to compute allocentric features locations, $\hat{\phi}(\boldsymbol{x}(t)) \circledast \phi(\boldsymbol{x}_i - \boldsymbol{x}(t)) = \hat{\phi}(\boldsymbol{x}_i) \approx \phi(\boldsymbol{x}_i)$. This is stored in the object location population.

Like the SSP estimate of self-location, the allocentric SSP estimate of an landmark location, $\hat{\phi}(\boldsymbol{x}_i)$, can be converted into probabilities. The probability density of landmark $B_i$ being at a location $\boldsymbol{x}$ is $f(\phi(\boldsymbol{x}) \cdot \hat{\phi}(\boldsymbol{x}_i))$ (where $f$ is a function that converts similarities to probabilities as described in Section 2.4.4; see Figure 4.6 for examples).

### 4.3.3 Environment Map Module

In SSP-SLAM, an environment map is stored in the weights of a heteroassociative memory network. We use an existing SNN model of heteroassociative memory, presented in Stewart et al. (2011) and adapted for online learning by Voelker et al. (2014).

The memory network architecture consists of two neural populations (Figure 4.4). The first receives some input and projects to the second population – the transformation is trained online to map input-output pairs. The PES learning rule, given in Eq. (2.20), is used to train the decoders (*i.e.*, the outgoing synaptic weights) of the first population. Concurrently, the Voja

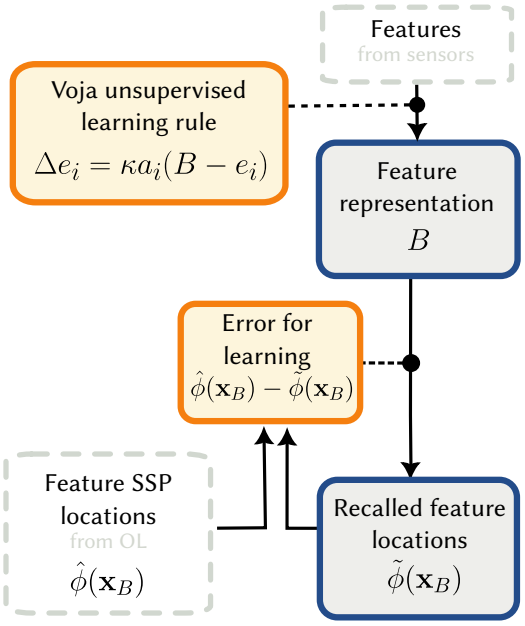



**Figure 4.4:** Diagram of the heteroassociative memory model, used as the environment mapping module of SSP-SLAM.

learning rule, given in Eq. (2.21), is used to modify the encoders of the first population. This shifts neuron encoders to be more similar to the input they receive. The Voja learning rule sparsifies and separates activity in the population, which helps prevent interference.

Voelker et al. (2014) analyzed this heteroassociative memory model and found good storage capacity and scaling – with 2500 leaky integrate-and-fire (LIF) neurons in the first layer and input-output vectors of size $d = 128$, the model was able to correctly recall[2] over 90% of the values stored in memory after training on 150 associations. It correctly recalled 80% of the values stored in memory after training on 250 associations (Figure 4 of Voelker et al., 2014).

In SSP-SLAM, this heteroassociative memory network is used to map representations of features (*e.g.*, objects, landmarks, barriers, colours, etc.) in the agent's field of view to the current estimate of those feature's locations as SSPs, $\hat{\phi}(\boldsymbol{x}_i)$, obtained from the landmark perception module. Notably, these environmental features can be structured representations. For example, vector representations of a colour, smell, and shape can be bound or bundled together to create a multi-sensory landmark. Using such representations, complex semantic environment maps can be learned.

Other mappings can also be learned. For example, a network can be trained to map feature locations $\hat{\phi}(\boldsymbol{x}_i)$, to feature symbols. Or, alternatively, a mapping from feature locations to feature symbols bound with their location, $\hat{\phi}(\boldsymbol{x}_i) \circledast B$, can be learned. Given an SSP input

[2]Correct recall was defined in Voelker et al. (2014) to be recall with a root mean square error (RMSE) $< 0.02$.

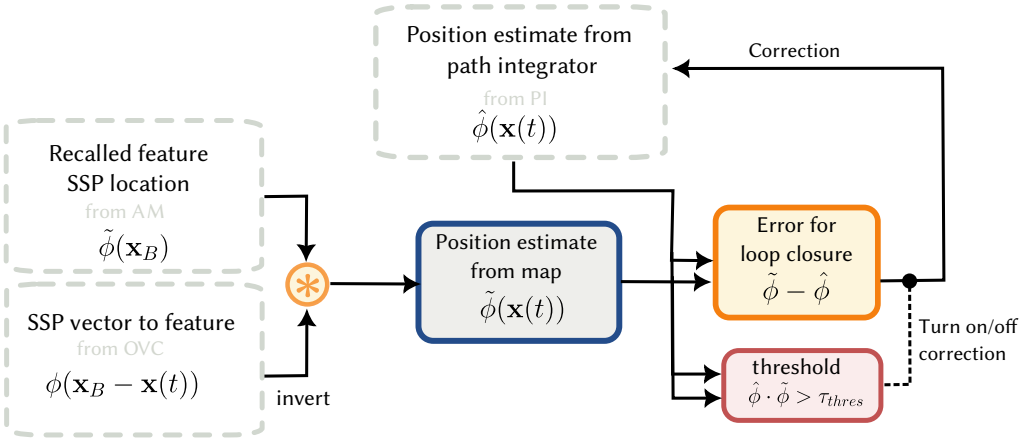



**Figure 4.5:** Diagram of the loop closure module of SSP-SLAM.

that represents the whole area of an environment, the network will approximately recall $\sum_i \phi(\boldsymbol{x}_i) \circledast B_i$, and so a single vector representation of a complete map can be recovered. We demonstrate a variety of these mappings and explore their accuracy in Section 4.4.1.

### 4.3.4 *Loop Closure Module*

The combination of the SSP-PI model and the associative memory network (for environment mapping) provides the core components of a SLAM system. As landmarks are discovered, their perception drives the training of a memory network. When landmarks are re-encountered, the past estimate of their location is recalled by the memory network. This might be different than the current estimate of their locations computed in the object location population, due to errors accumulating in the PI computation. The difference in estimations is used to correct the PI model, but only when the similarity between estimates exceeds a threshold $\tau_{\text{thres}} = 0.2$ (Figure 4.5). This ensures that corrections are not made before the environment map is learned or with estimates obtained from incorrect landmarks with similar features but different locations.

## 4.4 Results

### 4.4.1 *Querying maps*

In this section, we focus on a single example environment to demonstrate map querying and accuracy in SSP-SLAM. As shown in Figure 4.6A, we use a simple 2D environment that contains four point landmarks (a blue square, blue triangle, orange square, and orange triangle) as well as a wall region. To provide a path, we generate a random, frequency-limited trajectory through the environment and use finite differences to obtain velocities along the path. The velocity input signal is represented by a spiking neural population, introducing noise to the

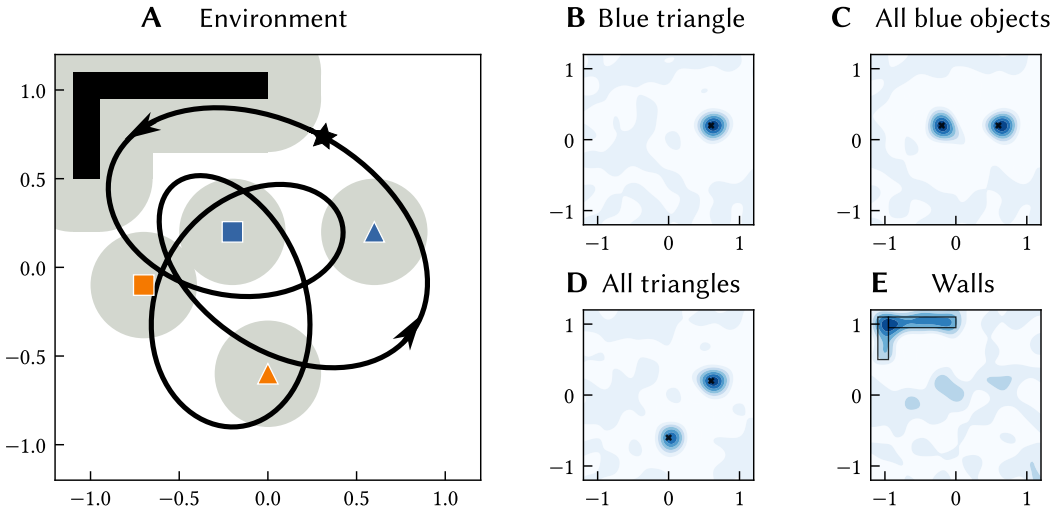



**Figure 4.6: (A)** An example 2D environment consisting of four point landmarks (blue and orange squares and triangles) and a wall region. An agent transverses a 60 second long trajectory through the environment (the black line, starting at the star). The agent has a limited field of view. The grey circles around landmarks indicates the distance at which the landmark is visible to the agent. SSP-SLAM is used to estimate the agent's location and learn a map of this environment. **(B-E)** The environment map network, trained to map features to locations, is probed at the end of the simulation. Plotted are the similarity maps obtained from these probes along with ground truth locations plotted in black. In **(B)**, the location of the blue triangle is recalled by passing in $\texttt{BLUE} \circledast \texttt{TRIANGLE}$ to the environment map network. The heat map shows the similarity of the output of the network to SSPs representing points over the 2D, capturing the probability over locations. **(C)** The recall of the location of all blue landmarks, plotted as a similarity map. **(D)** The recall of the location of all triangle landmarks, plotted as a similarity map. **(E)** The recall of the wall area.

signal. Model parameters used in this and subsequent experiments in this chapter are listed in Appendix C.

The environment map network is trained to map SPs representing environment features to the feature locations as SSPs. Given the map in Figure 4.6, it ideally learns the following associations:

$$\texttt{BLUE} \circledast \texttt{SQUARE} \rightarrow \phi([-0.2, 0.2]), \qquad \texttt{BLUE} \circledast \texttt{TRIANGLE} \rightarrow \phi([0.6, 0.2]) \tag{4.1}$$

$$\texttt{ORANGE} \circledast \texttt{TRIANGLE} \rightarrow \phi([0, -0.6]), \qquad \texttt{ORANGE} \circledast \texttt{SQUARE} \rightarrow \phi([-0.6, -0.1]) \tag{4.2}$$

$$\texttt{WALL} \rightarrow \int_{0.2}^{1.1} \int_{-1.1}^{-0.95} \phi(x, y) dx dy + \int_{0.95}^{1.1} \int_{-0.95}^{0} \phi(x, y) dx dy \tag{4.3}$$

where BLUE is a semantic pointer representing the colour 'blue', SQUARE is the semantic pointer representing the shape 'square', etc.

At the end of the simulation, the actual mapping learned by the environment map network

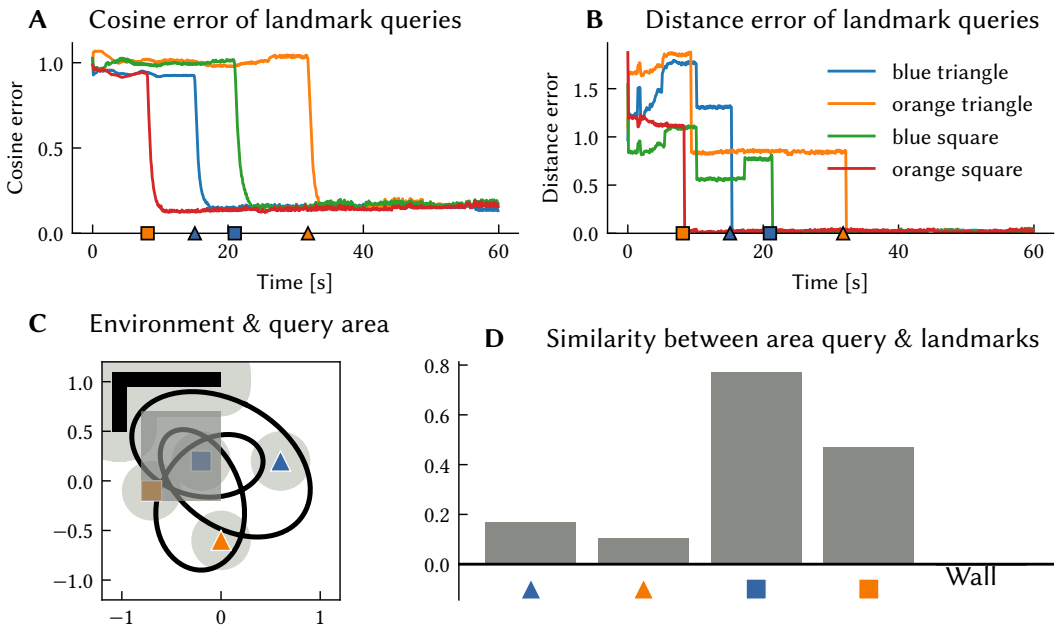



**Figure 4.7: Results from querying different associative memory networks,** trained on the environment in Figure 4.6A. **(A-B)** The association network mapping features to locations is queried over the course of training to compute estimates of the vectors between self-location and landmarks. Each feature $B_i$ is passed through the network to obtain a recollection of its location, $\hat{\phi}(\hat{\boldsymbol{x}}_i)$, which is bound with the model's self-location estimate to obtain $\hat{\phi}(\boldsymbol{x}_i - \boldsymbol{x}(t))$. **(A)** The cosine similarity error between the estimate of the SSP vector to landmarks in memory and the ground truth over the simulation time $t$. **(B)** The distance between the MLE obtained from $\hat{\phi}(\boldsymbol{x}_i - \boldsymbol{x}(t))$ and the ground truth vector between self-location at time $t$ and landmark locations, $\boldsymbol{x}_i - \boldsymbol{x}(t)$. **(C-D)** The association network that instead maps locations to features is queried at the end of the simulation. **(C)** The 2D environment and a query area (the dark grey shaded region). The SSP representing the query area is given as input to the location-to-features associative memory network. **(D)** The similarity of the output of the associative memory network to all object SPs in the environment.

is probed. The locations of particular point landmarks are recalled by feeding in SP input, *e.g.*, `BLUE` $\circledast$ `TRIANGLE` as shown in Figure 4.6B. Additionally, the map is queried for locations of all landmarks sharing certain characteristics. For example, in Figure 4.6C the locations of all blue landmarks is queried by giving the network input `BLUE` $\circledast$ (`SQUARE` + `TRIANGLE`).

The recalled landmark locations can be used in various downstream computations. For example, we compute the vector between the model's self-location estimate to all recalled landmark locations. The output from querying the environment map network for each landmark's SSP location, $\hat{\phi}(\boldsymbol{x}_i)$, is combined with the output of the localization module to compute these vectors over the simulation run time. This is done by taking the inverse of the SSP output of the localization module, $\phi(\hat{\boldsymbol{x}}(t))^{-1}$, and binding it with recalled locations from the associative memory, $\phi(\hat{\boldsymbol{x}}(t))^{-1} \circledast \hat{\phi}(\boldsymbol{x}_i) = \hat{\phi}(\boldsymbol{x}_i - \boldsymbol{x}(t)) \approx \phi(\boldsymbol{x}_i - \boldsymbol{x}(t))$. This produces an estimate of the

vector distance between the agent and landmark $i$ – a useful quantity for navigation. The error in this computation is plotted in Figure 4.7A-B. At the beginning of the simulation, the environment map has not yet been learned and so the output $\hat{\phi}(\boldsymbol{x}_i - \boldsymbol{x}(t))$ is inaccurate. After an item has been encountered, the error drops and remains low over the entire simulation.

An associative memory that maps landmark location SSPs to landmark SPs is also trained in this experiment. After learning, SSPs are passed into this network to recall the SPs representation of landmarks or features at particular locations or over particular areas. An example of querying an area is shown in Figure 4.7C-D.

### *4.4.2 Replicating neural activity patterns*

The activity patterns of spiking neurons in various components of the SSP-SLAM are presented and shown to capture the classes of hippocampal complex cells discussed in Section 4.2.1. We run SSP-SLAM on a 150 second path, recorded from a rodent by Sargolini et al. (2006), with ten landmarks at random locations added to the environment for our experiment. Spike trains are recorded from neurons in the PI network, grid cell population, object vector population, and the associative memory network during the simulation. Activity patterns from example neurons are shown in Figure 4.8.

In Figure 4.8A, we see that a neuron in the grid cell population indeed has hexagonal-patterned activity, as expected. However, this pattern deteriorates when using SSP-PI alone. The corrections computed using the trained environment map module ensure the pattern's stability. This environment map is learned by modifying the outgoing connection weights in the associative memory population using the PES rule, while the Voja learning rule is used to modify the encoders of the associative memory population. Learning thus results in neurons developing selective sensitivity to particular encountered landmarks, similar to some hippocampal place cells (Geiller et al., 2017; Kim et al., 2020b), as shown in Figure 4.8C. In Figure 4.8B, the activity of a neuron in the object vector population is shown and, as expected, its activity is like that of the object-vector cells of the MEC.

### *4.4.3 Localization*

Figure 4.9 shows the results of comparing the localization accuracy of SSP-SLAM and SSP-PI over ten different environments. For each environment, 100 locations are selected for point landmarks,[3] and trajectories are randomly generated from band-limited white noise signals. Note that in these experiments, we assume known correspondences – in other words, all landmarks are unique and identifiable. For 2D environments, trajectories are 500 seconds long and have an average path length of 75.91 m ($\pm$2.48 m). For the 3D environments, 250 second trajectories are used with an average path length of 55.29 m ($\pm$3.91 m). The model is initialized with the 97-dimensional SSP representation of the starting point of the path, and receives the

[3]Locations are generated via the Rd quasi-random sequence over $[-1, 1]^n$.

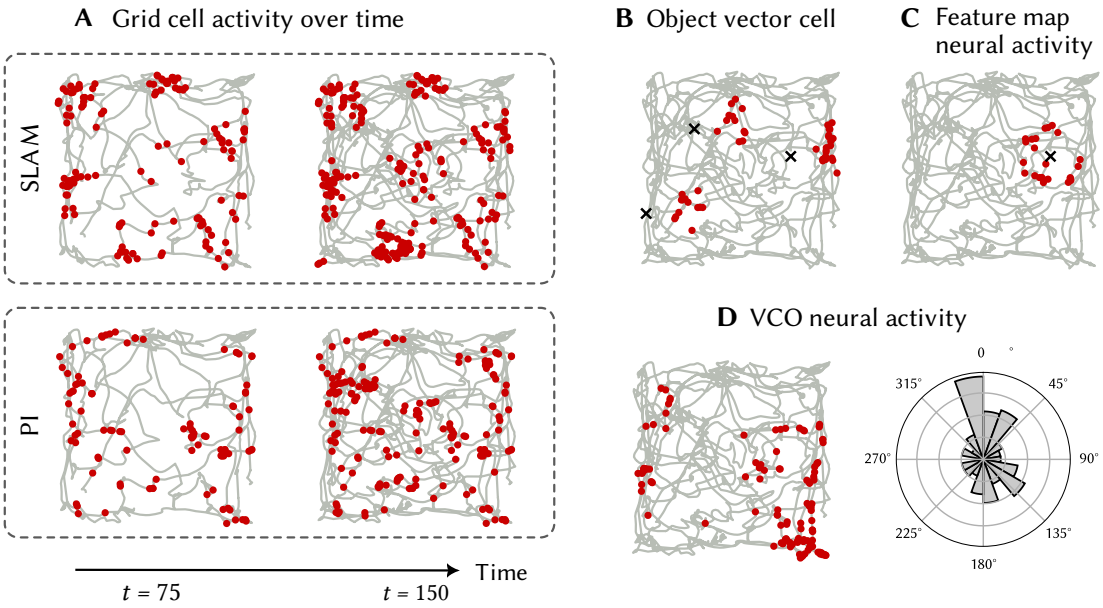



**Figure 4.8: Firing patterns of neurons in SSP-SLAM.** The agent's trajectory is shown in grey (obtained from Sargolini et al. (2006)). Red dots indicate the positions at which the given neuron fired. **(A)** A neuron from the grid cell population encoding $\phi(\hat{\boldsymbol{x}}(t))$. Spikes are recorded with the normal functioning of SSP-SLAM in the top row. For the second row, error correction from the learned environment map is turned off. The first column of both rows presents result half way through the simulation, while the second column displays the complete recording. The corrections from the environment map help maintain the grid-cell-like activity pattern over time. **(B)** A neuron from the OVC population encoding the SSP representation of the vector between $\boldsymbol{x}(t)$ and any landmarks in view. Object locations are marked with an 'x'. For this simulation, only three landmarks are included for clarity of the visualization. This neuron fires when a landmark is west of the agent. **(C)** A neuron from the associative memory population. The Voja learning rule shifts the neuron's encoder towards its input, resulting in the neuron firing when a particular landmark (marked with an 'x') is in view. **(D)** A neuron from a velocity controlled oscillator (VCO) population in the path integrator. Neurons in this population have conjunctive sensitivity to velocity and position. On the right panel, spike counts are binned by the heading direction of the path, demonstrating the neuron's preference for a particular heading direction (337.5° to 360°).

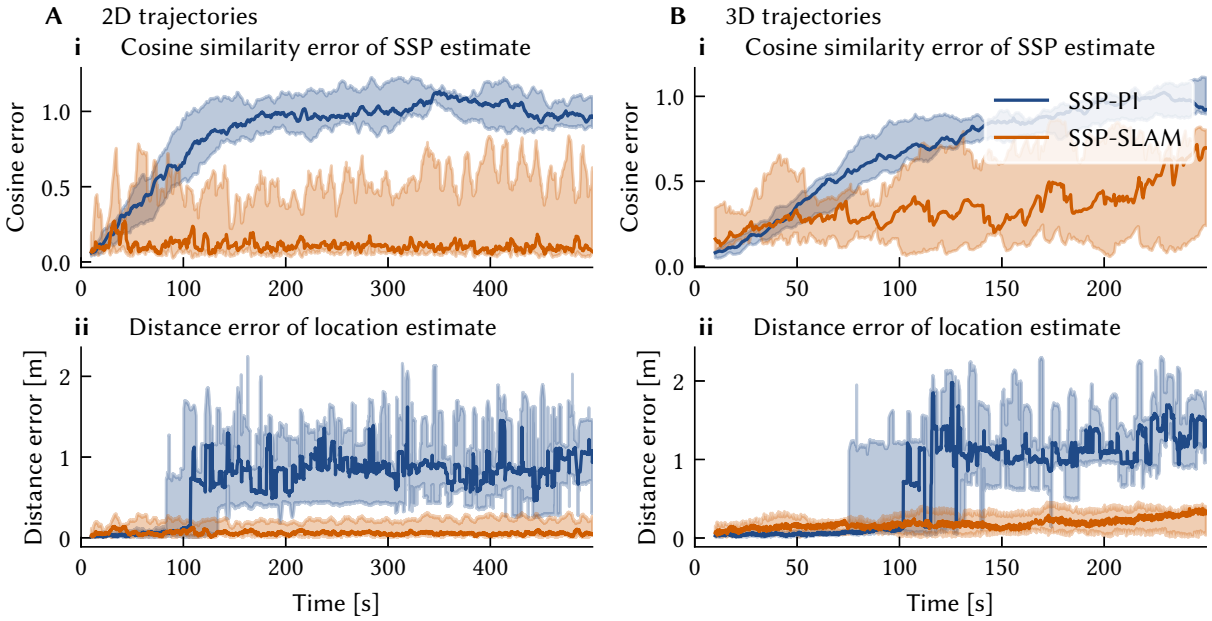



**Figure 4.9: Localization error in SSP-SLAM versus SSP-PI for (A) 2D and (B) 3D trajectories.** The solid lines are the median errors over ten trials and the shaded areas are the 95% bootstrapped confidence intervals. **(i)** The cosine similarity error over the simulation – *i.e.*, dissimilarity between the vector output of the path integrator and the SSP encoding of the ground truth. **(ii)** The distance between the model MLE of self-location and the ground truth over the simulation.

velocity along the path (computed using finite differences) as input during simulation run time. There are 800 neurons per VCO population in the localization module and a total of 82280 neurons in the full SLAM network. Recall that the same SNN model is used for both 2D and 3D inputs.

The average accuracy (in particular the cosine similarity Eq. (3.32) and distance errors) of SSP-SLAM localization output is shown in Figure 4.9. Unlike the SSP-PI model, the errors of the SSP-SLAM remain low over the entire simulation for both 2D and 3D trajectories. For the 2D trajectories, the range of errors remains consistent – suggesting that accuracy would continue to be maintained over even longer run times – while for the 3D trajectories there is a clear trend of increasing cosine error. More landmarks may be needed in higher-dimensional environments for accurate long-term localization.

For the 2D trajectories, the average deviation from ground truth trajectory over the simulation is 0.09 m for SSP-SLAM, compared to 0.72 m for SSP-PI. Integrating the RMSE between SSP-SLAM's MLE and the ground truth over the entire simulation time yields 0.43 m for SSP-SLAM and 3.51 m for PI. These results and the results for the 3D environments are given in Table 4.1. The error corrections provided by the environment map in SSP-SLAM result in roughly an eight-fold improvement in localization error.

Figure 4.10 shows examples of self-location estimates from SSP-SLAM compared to ground

**Table 4.1: The 2D and 3D localization errors for SSP-SLAM and SSP-PI averaged over ten trials.** The RMSE of all rows is the distance between the models' MLE of self-location and the ground truth. The first row is the distance error averaged over the trajectories. The second row is the error at the final timestep (500s for 2D paths, 25-s for 3D paths). The third row is the error integrated over the run time. Shown are the means and standard deviation of these quantities.

| | 2D | | 3D | |
|---|---|---|---|---|
| | SSP-SLAM | SSP-PI | SSP-SLAM | SSP-PI |
| Avg. RMSE | $0.086 \pm 0.014$ | $0.716 \pm 0.337$ | $0.184 \pm 0.042$ | $0.763 \pm 0.529$ |
| Final RMSE | $0.087 \pm 0.088$ | $0.914 \pm 0.375$ | $0.279 \pm 0.122$ | $1.451 \pm 0.244$ |
| Integrated RMSE | $0.423 \pm 0.330$ | $3.507 \pm 0.621$ | $0.442 \pm 0.178$ | $1.830 \pm 0.351$ |

truth trajectories and the output of SSP-PI (*i.e.*, dead reckoning). The full SSP-SLAM model accurately follows the true path for the entire trajectory in the examples shown. In contrast, the path from the SSP-PI is very inaccurate. About 100 seconds into the simulation, the vector represented by the path integrator drifts sufficiently far from the SSP manifold that an accurate location can no longer be decoded, resulting in the self-location estimate jumping wildly in the space. In contrast, the corrections computed using the environment map in SSP-SLAM keep the path integrator output near the ideal result, though drift can be seen in some estimates.

### *4.4.4 Neuromorphic simulation*

To investigate the feasibility of deploying SSP-SLAM on neuromorphic hardware, we simulated the network on the NengoLoihi emulator. This Python package allows SNNs built in Nengo to be run on Intel's Loihi 1 architecture. It includes both support for running models on Loihi hardware and a Loihi emulator. We use the emulator for these experiments. Notably, the NengoLoihi emulator implements the same limited precision mathematics as the actual hardware, using 8-bit weights and a quantized neuron response function. This quantized response is less smooth than the standard LIF tuning curves, resulting in higher decoding error in Loihi networks.

A much smaller version of SSP-SLAM is used for these experiments due to the constraints of the Loihi architecture. The Loihi 1 chip consists of 128 neuromorphic cores, each containing 1024 spiking neurons, 4096 incoming and outgoing axons, and 128,000 synapses. This places limitations on the size of the recurrent VCO populations – for NengoLoihi experiments we use only 500 neurons per VCO as opposed to the 800 in previous experiments. This means a total of 13500 neurons were used in the path integrator alone to maintain the dynamics of the 55-dimensional SSPs. In addition, parts of the SSP-SLAM network are modified for these simulations. In particular, the clean-up process is removed, as this would have to occur off-chip, and the Voja learning is disabled as this learning rule is not supported in NengoLoihi.

The accuracy of this pared-down SSP-SLAM network, compared to the full SSP-SLAM

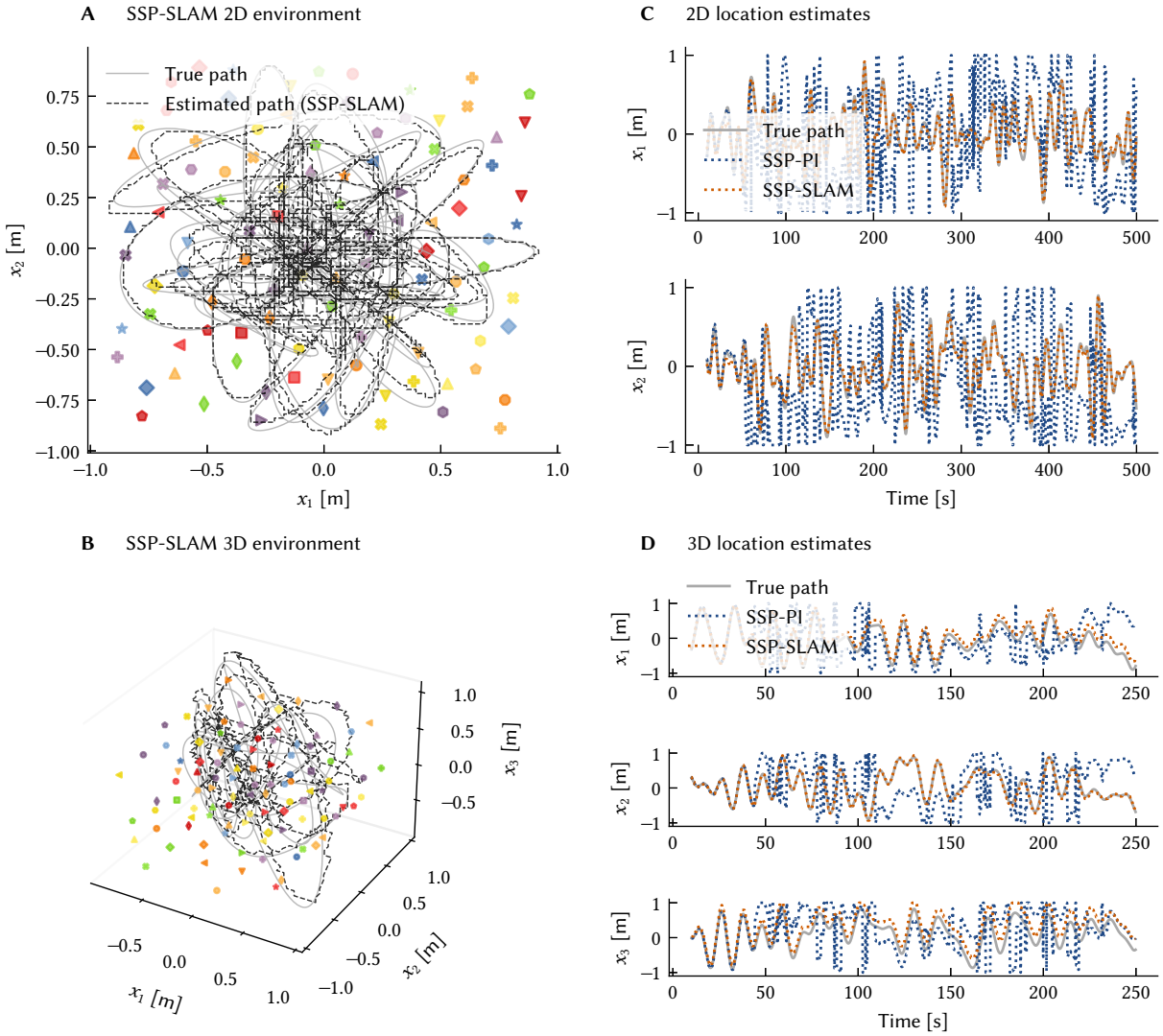



**Figure 4.10: An example of an environment and the models self-location MLEs.** The ground truth trajectory is plotted as a solid grey line. **(A-B)** Plotted are the 2D (3D) trajectory, the SSP-SLAM path estimate, and the locations of 100 point landmarks. **(C-D)** Each dimension of the path estimates obtained from SSP-SLAM and SSP-PI without any corrections from the environment map network (*i.e.*, dead reckoning) over the simulation time.

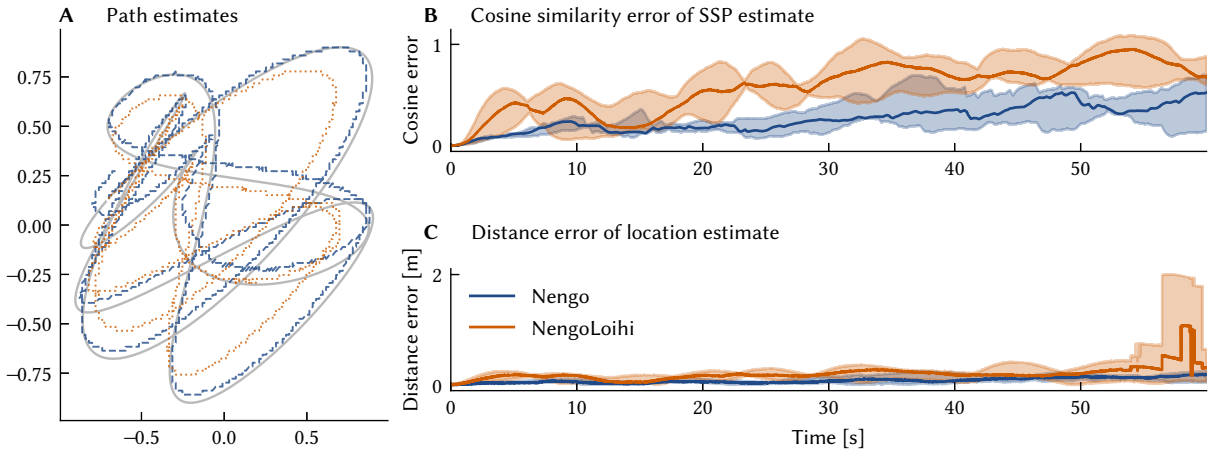



**Figure 4.11: Results from SSP-SLAM simulated on the NengoLoihi emulator aggregated over ten random 2D trajectories. (A)** The self-location estimates from SSP-SLAM simulated on a GPU with Nengo and in a Loihi emulator with NengoLoihi. **(B-C)** The solid lines are the median errors over ten trials and the shaded areas are the 95% bootstrapped confidence intervals.

network simulated with full-precision on Nengo, is shown in Figure 4.11. Both models use 500 neurons per VCO. While the NengoLoihi implementation performs worse than the Nengo version, it is able to maintain an accurate location estimate for 50 seconds.

The NengoLoihi model uses 1763 cores, which would require 14 chips if run on hardware and incur significant lag due to chip-to-chip communication. However, this is due to non-optimal partitioning of resources. Although some cores are fully utilized (namely those storing the VCO neurons and their incoming and outgoing connections), most are not. In fact, on average, the cores use only 2.43% of the available neurons, 1.01% of the axons, and 0.70% of the synapses. This is because NengoLoihi does not natively support putting multiple separate neural populations on a single core. Networks with many small populations will have poor core utilization, and this is the case for SSP-SLAM – the circular convolution networks are modular and break down the computation by splitting vectors amongst many populations, each with only 100 neurons in our experiments. Consequently, running the model on the hardware would require updating the NengoLoihi backend, or manually restructuring the network to avoid current assumptions made in the backend.

## 4.5 Discussion

### *4.5.1 Prior Research*

Standard approaches to SLAM in robotics have tackled the problem of SLAM as one of statistical inference, while biologically-inspired SLAM methods do not use this method. Instead, most models build upon the RatSLAM template: a CANN consisting of pose cells (neurons sensitive to place and heading direction) or grid cells is used to perform PI, while a Hebbian learning

rule is used to drive associative learning between this population and a population of local view cells (similar to the spatial view cells in the hippocampus; Milford et al., 2004; Milford et al., 2016; Zeng et al., 2017; Yu et al., 2019). The learned weights between these populations represent a map of sorts. Some neural SLAM methods only use these associations to error correct PI, while others, including the original RatSLAM model, additionally use a graph as a topological environment map. This graph is constructed and updated with a graph relaxation algorithm and used to perform loop closure and correct the CANN.

In contrast, a hybrid OI-CANN model is used for path integration in SSP-SLAM and a graphical environment map is not learned – instead, the outgoing connection weights from the memory network implicitly store a map which can be retrieved by querying the network. The Voja rule, which is used to shift the associative memory population encoders towards observed input in SSP-SLAM, plays a similar role to the template novelty detection and addition of local view cells that occurs in RatSLAM. Furthermore, we have not implemented an external module for pre-processing of sensory data, and we use landmark semantic pointers and displacement SSPs in lieu of templates. Object detection and depth estimation algorithms would be required to obtain this input from visual data.

Many models have since extended the original RatSLAM. CANN SLAM models with place cell-like activity were also used by BatSLAM (Steckel et al., 2013), an extension to RatSLAM for handling environment information from sonar sensors, and DolphinSLAM (Silveira et al., 2015), developed for 3D SLAM in underwater environments. A CANN consisting of conjunctive grid cells is used in the SLAM model presented in (Zeng et al., 2017). Three-dimensional SLAM in realistic environments with grid cells was also explored in NeuroSLAM (Yu et al., 2019). Unlike our work, none of these models use spiking neural networks.

More recent research has focused on developing spiking networks for SLAM and testing them on neuromorphic hardware. Spiking 2D SLAM models were presented in Tang et al. (2018) and Tang et al. (2019) and Kreiser et al. (2020a) and Kreiser et al. (2020b). In Kreiser et al. (2020a), a SLAM system on the Loihi chip was used to estimate the head position of an iCub robot as it visually explored a wall with a dot pattern acting as the environment. Tang et al. (2019) made use of a depth camera and Bayesian updates on a posterior distribution represented by a neural population. They found that their SLAM system, when run on Loihi, was more energy efficient by two orders of magnitude compared to a baseline method on a CPU. While the models discussed here use raw sensory input, it should be noted that non-spiking visual modules are used to process this information and obtain input for SLAM. For instance, intensity profile vectors or feature colours and distances from the observer are used. In contrast to SSP-SLAM, none of the models mentioned incorporate any elements of oscillator-interference (OI) to perform path integration, or perform 3D SLAM. Furthermore, some of these models employ "localist"/discrete representations, such as using one neuron to represent each integer value for heading direction or discretized distance to features. This approach does not support generalization and does not scale well to higher dimensional representations, unlike SSPs.

Taken together, and summarized in Table 4.2, past work provides examples of spiking and

non-spiking networks, using CANNs for path integration. However, unlike SSP-SLAM, none of these approaches provides a methodology for incorporating semantic information or for online learning of semantic environmental maps. In addition, none of these employ SSPs, or the same combination of a OI-CANN in a fully spiking model capable of functioning equally well in both 2D and 3D spatial environments, as demonstrated above.

### *4.5.2 Performance*

We have presented the results of several experiments on SSP-SLAM to assess its performance. The model demonstrates accurate localization capabilities on different paths, both two-dimensional and three-dimensional. Notably, this is the only SLAM model (to our knowledge) that uses OI techniques for localization. This approach has the advantage of easy generalization to higher-dimensional spaces. Typically, CANN models describe a neural population as a 2D sheet or 3D array (often with periodic boundary conditions), where the geometry specifies the recurrent connectivity pattern required for localization. However, this only supports unimodal position estimates, and the connectivity pattern must be modified and made more complicated to move to higher dimensional path integration.[4] In contrast, in our approach the recurrent connectivity of the path integrator network remains the same regardless of spatial dimensionality. This allows the same model to switch seamlessly between SLAM in different spaces and domains.

Furthermore, SSP-SLAM encodes environment maps in the outgoing connections of an associative memory network, which are learned online using biologically plausible learning rules. The map generated is a semi-metric, semantic map that uses symbol-like vector representations that have been leveraged in a variety of large-scale cognitive models (Eliasmith, 2013; Arora et al., 2018; Kajić et al., 2019b; Kelly et al., 2020; Gosmann et al., 2021).

By working in the SSP and VSA paradigm, we are able to formulate the problem in such a way that combines metric and semantic SLAM. These features distinguish SSP-SLAM from other bio-inspired SLAM models, and make it the first spiking semantic SLAM model to our knowledge. The environment map – associations between SPs and SSPs – is used for loop closure which significantly improves localization accuracy over PI alone (4.9). After training, the map can be queried to obtain object locations given their symbol-like representation as a semantic pointer. Alternatively, item representations can be obtained by querying specific areas, or vectors between the agent and landmarks can be computed. Such direct queries of semantic map knowledge cannot be easily made with past spiking network map representations.

[4]Recent research has explored variants to traditional CANNs that overcomes these limitations. A multi-modal CANN model was presented in Wang et al. (2022) and research exploring CANNs with arbitrary dimensional attractor manifolds and more biologically realistic asymmetries in synaptic connectivity was presented in Darshan et al. (2022). Such CANN variants have not been used in SLAM systems.

**Table 4.2:** Comparison of bio-inspired SLAM models

| Model | Sensors | Input representation | Dim. | Localization | Env. map | Cells | Spiking | Neuromorphic hardware |
|---|---|---|---|---|---|---|---|---|
| SSP-SLAM | None | Displacement to features as an SSP and feature identities as SPs | Any, tested on 2D & 3D | OI-CANN hybrid | Weights between landmark population to landmark locations | HDC, GC, landmark cells, OVC | Yes | In sim. |
| RatSLAM (Milford et al., 2016) | Monocular camera | Greyscale image intensity profile | 2D | CANN | Topological map associating local views with position, in the form of a graph | Pose cells, local view cells | No | No |
| BatSLAM (Steckel et al., 2013) | Biomimetic sonar | Intensity difference between left and right Echolocation Related Transfer Functions | 2D | CANN | Topological map associating local views, in the form of a graph | Pose cells, local view cells | No | No |
| DolphinSLAM (Silveira et al., 2015) | Sonar & visual | One-hot representation obtained from FabMAP algorithm on top of a Bag of Words model | 3D | CANN | Graph with nodes storing local view, place cell and position while edges store displacements | 3D PC, local view cells | No | No |
| NeuroSLAM (Yu et al., 2019) | Panoramic camera | Greyscale image intensity profile | 3D | CANN | Topological map storing activities of local view cells, GCs, HDCs, and estimated pose | 3D PC, conjuctive 3D GC and HDC, local view cells | No | No |
| Kreiser et al. (2020a) | Event-based camera | Detection of blinking LEDs at different frequencies | 2D | CANN | Weights from landmark population to a HDC population | HDC, landmark cells | Yes | Yes |
| Tang et al. (2019) | RGB-Depth camera | Discretized distances to landmarks | 2D | CANN | Weights from PC to a displacement-from-border population | 2D PC, HDC, border cells, Bayesian cells | Yes | Yes |

### 4.5.3 Limitations

We have presented a novel model that employs biologically-inspired mechanisms to solve SLAM. However, SSP-SLAM has several limitations. First, the model was tested on artificial environments, which restricts what conclusions we can draw as to its generalizability to more complex, real-world environments. Second, the SSP-SLAM model has not been deployed on an actual neuromorphic hardware platform.

To improve the model's utility, it is essential to test it on real-world input and integrate it with a network that can process raw sensory data. Such an approach would demonstrate the model's ability to handle more complex and diverse environmental conditions. Moreover, the current model's accuracy is inferior to that of non-biologically inspired SLAM methods, which limits its usefulness to mobile robotics. Observing an accuracy drop and the use of small scale test environments is true of current spiking SLAM models more generally. Despite this, the use of neuromorphic computing and hardware has the potential to improve energy efficiency of SLAM systems, which is particularly useful in mobile robotics applications. Reduced power demands permits the deployment of SLAM systems in progressively more power-constrained environments, such as in edge computing or operations in GPS-denied settings, like space or sub-sea exploration. An increasing number of algorithms have harnessed the advantages of spike-based computing to make gains in efficiency and speed (Yakopcic et al., 2020; Yan et al., 2021; Davies et al., 2021).

Therefore, while the current model shows promise in enabling biologically-inspired SLAM, its limitations in terms of testing and accuracy must be addressed to justify its wider application in real-world scenarios. Further research should focus on testing the model on more complex environments and realistic input data. Nevertheless, it may be uniquely well-positioned to realize the promise of energy-efficient SLAM using neuromorphic hardware.

### 4.5.4 Future work

One obvious area of future work is to address each of the limitations discussed in the previous section. Beyond this, there are several other directions that warrant further exploration – for example, explicit modelling of sensor uncertainties using SSPs, demonstrating higher-dimensional SLAM, and integrating SSP-SLAM with other cognitive models.

Higher-dimensional SLAM could be a promising area of investigation. The proposed model can be extended to localization and mapping in any dimension of space by modifying the input without changing the model or hyperparameters. Although SLAM is mainly applied to navigation and mapping in physical spaces, operating in dimensions equal to or less than three, it is possible that the same neural mechanisms underlying spatial navigation and mapping could be applicable to non-spatial domains, such as mapping in high-dimensional conceptual space. The idea that similar computations to those behind SLAM may be understood as core cognitive processes has been proposed in Safron et al. (2022b).

Another promising direction for future work is to integrate the proposed model with more complex memory and reasoning systems. Since the proposed model is developed using the SPA, it fits naturally within the context of NEF and other SPA models, including Spaun (Stewart et al., 2012; Choo, 2018). Integration of the proposed SLAM model with other models constructed with these tools could lead to more sophisticated memory and reasoning capabilities. Exploiting such capabilities in large spatial environments remains a central challenge for models of biological cognition.

## 4.6 Conclusion

We have proposed a novel spiking semantic SLAM model, SSP-SLAM, inspired by the hippocampal formation in the mammalian brain. The model is unique in its integration of a hybrid OI-CANN path integrator, online biologically-plausible learning of an environment map, and use of symbol-like object representations in a spiking network. This combination enables the model to perform SLAM accurately in simulated environments and learn representations that can be queried in powerful ways. For example, it can provide information about what is located in a particular area of the map, report vectors between landmarks, and identify the location of objects based on their properties, such as their colour. Thus, the model extends the capabilities of biologically plausible SLAM systems by using VSA representations, alongside neural mechanisms inspired by the hippocampus and entorhinal cortex, while demonstrating functionality in both 2D and 3D environments. These advancements not only deepen our understanding of how the brain may implement spatial mapping and memory but also offer promising directions for developing neuromorphic systems capable of robust, efficient, and interpretable SLAM in real-world applications.

# Chapter 5

# Biologically-Constrained Reinforcement Learning

> However information is stored in nervous systems, it appears to be radically unlike information storage in a digital computer, where storage and processing are separated and items are stored in memory according to addressable locations. In nervous systems, information seems to be stored in the connections between the same neurons that process the information. There does not appear to be a distinct storage location for each piece of stored information, and information is content addressable rather than location addressable.
>
> — Patricia Smith Churchland, Terrence J. Sejnowski
> *Neural Representation and Neural Computation*

**Summary**

This chapter explores biologically-constrained reinforcement learning (RL) in small environments by developing spiking neural networks that integrate SSPs and the Legendre Delay Network (LDN). The LDN, a neural working and episodic memory system, is used to compute temporal traces for continuous-time temporal difference (TD) learning under biologically plausible learning rules. SSPs are used in models of successor features, reward distribution learning, and policy embeddings – demonstrating the versatility of neural oscillator-based representations for encoding probabilistic functions.

**Material in this chapter from prior publications**

Some material in this chapter was adapted from two prior publications in conference proceedings. The LDN-TD($\theta$) rule was presented at the *20th International Conference on Cognitive modelling* (Bartlett et al., 2022). This rule was conceived of by the author of this thesis and the experiments using it in this chapter were not presented in Bartlett et al. (2022). The SSP sampling model was presented at the *33rd International Conference on Artificial Neural Networks* (Furlong et al., 2024). The SSP sampling algorithm was developed by co-authors Dr. P. M. Furlong and Dr. Kathryn Simone. The spiking implementation of this algorithm was built by the author of this thesis.

## 5.1 Introduction

Reinforcement learning (RL) is a powerful framework for training agents to make decisions by learning through trial and error. In addition to its success in various AI and ML applications, it has also provided a framework for interpreting neural mechanisms within the mammalian brain. Most notably, dopaminergic neurons in the midbrain appear to signal temporal difference (TD) errors, which are used to drive learning in many RL algorithms. This observation has led to the dopamine reward prediction error hypothesis, which posits that the activity of dopaminergic neurons signals an error between predicted and experienced reward (Hollerman et al., 1998; Montague et al., 1996). More recently, researchers have proposed theories that dopamine may signal predictive errors more generally (Gershman et al., 2024; Lerner et al., 2021). Additionally, it has been hypothesized that the dorsal and ventral subdivisions of the striatum in the basal ganglia may function as an actor and critic, the components of a common architecture in RL models (Joel et al., 2002; O'Doherty et al., 2004).[1]

However, gaps exist between the mechanisms employed by many computational models of RL and the brain's biological mechanisms for reinforcement learning. When attempting to map standard RL methods to neural circuits, several questions arise.

- How are the functions in RL algorithms learned using local learning rules in continuous time?
- What neural representations are a good basis for state encoding?
- How are temporal traces stored in a neural circuit, and how are they sampled and used to compute TD errors?
- How are policy distributions stored or encoded in neural populations, and how are decisions and actions sampled from them?

While these questions have been extensively explored by the research community, they remain incompletely answered. The majority of RL models rely on biologically implausible learning rules and operate in discrete time. Although spike-timing-dependent plasticity and other local learning rules have been used to in some RL models, the biological implementation of distributed or multi-scale TD learning remains an open research area. In addition, there is ongoing research that aims to understand how neural circuits in regions such as the prefrontal cortex and basal ganglia might encode more abstract states or contextual information for RL tasks.

[1] While the cortico-basal ganglia-thalamo-cortical loop is often modelled as an RL system of action section, RL-related computations are not confined to this circuit. Other regions, such as the cerebellum, also contribute to RL, and RL is observed in animals and invertebrates that lack homologous structures to the basal ganglia or cerebellum. The underlying mechanisms of RL are quite general purpose and have likely evolved independently across diverse lineages (Murray et al., 2017).

Synaptic eligibility traces have been proposed as a mechanism for storing the temporal information needed for TD learning, but they are not well suited for retaining long, complex sequences. How the brain maintains and utilizes these traces is still under investigation. Additionally, the scalability of such mechanisms to more complex tasks is not fully understood. There is evidence that neural populations in the basal ganglia and cortex encode probabilistic information about actions, but it is still unclear how biological circuits represent complex, high-dimensional policy distributions, and whether the brain employs a form of stochastic sampling akin to what is commonly used in RL.

This chapter explores these questions and presents novel biologically plausible models of RL with spiking neural networks (SNNs). In particular, we focus on RL in continuous space and time and leverage particular methods: Spatial Semantic Pointers (SSPs) for continuous representations (including spatial and distributional) and the Legendre Delay Network (LDN) for memory. As described in more detail in Section 5.2.3 below, the LDN is a state-space model that has been used to model time cells and can store accurate, adaptive memory traces over long time scales. As well, it has not previously been used for RL. We present TD learning rules using the LDN, and demonstrate their functionality in value learning, trace conditioning, and successor feature learning. We also present a model of policy learning using SSPs and action sampling using Langevin dynamics in the SSP space. Lastly, we discuss future directions this work might take, and the many challenges in scaling SNN models with local learning to more complex navigation tasks.

## 5.2 Background

In this section, we first review RL and its mathematical formulation as a Markov decision process in both discrete and continuous time settings. Next, we review existing RL models, with an emphasis on biologically plausible systems. Last, we discuss the neural correlates of RL and the role of memory in RL and neural systems.

### *5.2.1 Reinforcement Learning Methods*

RL is a category of machine learning in which agents learn in response to partial environment feedback rather than from correct stimulus-response examples. Spatial navigation problems can be formulated as RL problems (Anggraini et al., 2018; Stoianov et al., 2018). RL is arguably a better description of learning processes in animals compared to supervised learning, since animals are typically not provided explicit instructions on how to act. Instead, behavioural strategies are often acquired by active engagement with the environment. Consequently, the study of animal learning in psychology has significantly shaped the field of RL, influencing its fundamental algorithms. RL, in turn, has influenced our understanding of the neuroscience of learning processes and decision making in the brain.

RL is typically modelled as a Markov decision process (MDP) (Figure 5.1). There is a set of

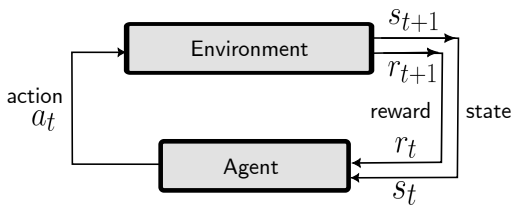



**Figure 5.1:** The basic MDP setting in RL.

environment *states*, $\mathcal{S}$, and a set of agent *actions*, $\mathcal{A}$. At timestep $t$, the environment will be in some state, $s_t \in \mathcal{S}$. The agent outputs an action at each time step, based on a stochastic *policy*, $a_t \sim \pi(s_t)$.[2] The environment will return a reward, $r_t$, and the environment will transition to a new state based on the state transition function, $Pr(s_{t+1} \mid s_t, a_t)$. The task in discrete-time RL is to learn a policy to maximize the expected discounted sum of future rewards (called the *return*):

$$\max_{\pi} \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right], \tag{5.1}$$

where $\gamma \in [0, 1)$ is the discount factor. The above expectation, when beginning at some particular state $s$ at timestep $t$, is the value function.

$$V(s) = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \;\middle|\; s_t = s\right] \tag{5.2}$$

One can also define the *Q-function*, $Q(s, a)$, in which the above expectation is also conditioned on the action taken at timestep $t$, $Q(s, a) = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid s(t) = s,\, a(t) = a\right]$.

RL algorithms can generally be divided into two categories, *model-free* and *model-based*, each of which has advantages and disadvantages. In model-free RL, the value function, Q-function, or policy are learned. An advantage of this approach is that action selection is computationally cheap – either the action that maximizes the Q-function is chosen or a trained policy is directly sampled. In model-based RL, the state transition function and reward function are learned by interacting with the environment. An advantage of this approach over model-free RL is that it quickly adapts to distal changes to the reward or state transition structure. Consider a maze with a goal at a certain location. If the goal location is changed, the model of the state transitions is still correct, and so a model-based agent only needs to locally update the reward function at affected states. This is not the case in model-free RL. A local change in the goal location may radically alter the value and Q-functions globally, and so additional experience and training is needed to adapt.

[2]We denote the probability distribution over actions given that the agent is in state $s$ as $\pi(s)$. We also use $\pi(a \mid s)$ to refer to the probability mass or density of the distribution at action $a$ given state $s$.

Furthermore, model-free RL is not capable of latent learning – learning when there is no explicit reward signal – unlike animals. Rats placed in a maze and allowed to explore without receiving rewards will learn to navigate to a goal in the maze faster than rats who did not go through the exploration phase (Tolman et al., 1930). Model-based agents will be able to learn a state transition function when there are no rewards, and this information can be leveraged in future tasks. Unfortunately, optimal action selection in model-based RL is computationally expensive. Value estimation must be performed iteratively over the entire state space, or a tree search method must be used to plan ahead.

This trade-off presents a difficulty, particularly when one is concerned with animal-like learning and biologically-plausible models. Animals display latent learning, transfer learning, and can learn after few environment interactions, unlike model-free agents that require huge amounts of data. But the computational complexity of action selection in model-based RL poses a challenge when working within the limitations of SNNs. In reality, a wealth of evidence suggests animal decision making relies on both model-free and model-based-like learning systems (Gläscher et al., 2010; Dezfouli et al., 2013; Lee et al., 2014). Nevertheless in this work, we focus primarily on biologically plausible model-free algorithms for learning value functions and policies.

**Value learning**. The update of a value function in discrete-time has the form,

$$V(s_t) \leftarrow V(s_t) + \eta \left[G_t - V(s_t)\right] = V(s_t) + \eta\delta_t. \tag{5.3}$$

where $\eta$ is the learning rate, $G_t$ is some estimate of the return, and $\delta_t$ is an error – the difference between the current value estimate and the return. This update, as written, is for tabular RL, in which the values of all states are stored in a table. To generalize to a continuous state space, one can model $V(s)$ with a neural network trained using the error signal $\delta_t$.

To compute returns, one can use the full sequence of future rewards (*i.e.*, a Monte Carlo estimate) or TD algorithms that take advantage of the recursive relationship between successive values (Sutton, 1988). The simplest such update is called TD(0),

$$V(s_t) \leftarrow V(s_t) + \eta \underbrace{\left[r_t + \gamma V(s_{t+1}) - V(s_t)\right]}_{=\delta_t^{(0)}}. \tag{5.4}$$

A single reward observation and the value estimate one timestep ahead is used to compute error. Alternatively, return may be estimated with a longer sequence of $n$ reward observations:

$$G_t^{(n)} = \sum_{k=0}^{n} \gamma^k r_{t+k} + \gamma^n V(s_{t+n}), \qquad \delta_t^{(n)} = G_t^{(n)} - V(s_t). \tag{5.5}$$

This is called the TD($n$) error, a compromise between the full Monte Carlo return estimate and the 1-step TD(0) method. The optimal $n$ is task dependent.

The $\lambda$-return algorithm provides a different compromise: it uses the weighted average of the $n$-step return estimates over all $n$ (from 1 to the full episode length, or some maximum number of steps). Empirically, this estimator tends to produce better results than using any one of the $n$-step estimates.

$$\delta_t^{(\lambda)} = (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}G_t^{(n)} - V(s_t) = \sum_{k=0}^{\infty}(\gamma\lambda)^k\delta_{t+k}^{(0)}. \tag{5.6}$$

An online version of $\lambda$-return is the TD($\lambda$) algorithm, which updates the value function according to

$$V(s) \leftarrow V(s) + \eta\delta_t^{(0)}e_t(s), \tag{5.7}$$

$$\text{where} \quad e_t(s) = \gamma\lambda e_{t-1}(s) + \nabla V(s_t). \tag{5.8}$$

The TD($\lambda$) algorithm effectively combines the advantages of both TD(0) and the full Monte Carlo method by maintaining an eligibility trace, $e_t$, which decays over time according to the parameter $\lambda$.[3]

**Policy learning**. In Q-learning, actions can be obtained by maximizing the Q-function at each timestep, $\max_{a\in\mathcal{A}}Q(s,a)$, with occasional random action selection to encourage exploration. If a value function is learned instead, then a policy must also be learned for action sampling: $a_t \sim \pi_W(s_t)$, where $W$ are the parameters of the policy. This is an Actor-Critic (AC) model: both a value function (the *critic*) and a policy (the *actor*) are learned simultaneously (Barto et al., 1983). The actor is used to select actions, while the critic is used to train the actor by modifying the parameters $W$ via the policy gradient:

$$\nabla J(W_t) = \mathbb{E}_{\pi_{W_t}}\left[\nabla_{W_t}\log\pi_{W_t}(s_t,a_t)\Psi(s_t,a_t)\right], \tag{5.9}$$

$$W_{t+1} = W_t + \eta\nabla_W J(W_t), \tag{5.10}$$

where $J(W_t) = \mathbb{E}_{\pi_{W_t}}\left[\sum_{k=0}^{\infty}r_{t+k}\right]$ is total expected reward, and $\Psi(s_t,a_t)$ can be any one of a number of estimates: the Q-function, the return, a TD error, or the advantage function, $A(s,a) = Q(s,a) - V(s)$ (Schulman et al., 2015).

The other factor in the policy gradient is $\nabla_W\log\pi_W(s,a)$, the score function, and it depends on the policy type and parameterization. For example, if the action space is discrete with size $n_a$ then $\pi(\boldsymbol{s}_t)$ is given by a categorical distribution. The parameters $W$ may be the weights of a neural network that outputs a vector of size $n_a$, which is given as input to a softmax function to obtain the probabilities of the distribution. If the action space is continuous, then the policy is some continuous distribution, usually a Gaussian, whose parameters (*e.g.*, a mean vector and covariance matrix) are the output of a neural network that takes state as input. The policy can still be updated using the policy gradient, the score function just takes a different form.

[3]While TD($\lambda$) is more frequently referenced in recent literature, $\lambda$-return remains widely used, particularly in the form of Generalized Advantage Estimation (GAE), which applies $\lambda$-return to advantage function estimation.

**Successor Representation**. There are RL algorithms are neither model-based nor model-free. Successor representations (SRs) form the basis for such an algorithm that provides a compromise between these two categories (Dayan, 1993). In this paradigm, the model learns a reward function, $r(s)$ – often assumed to depend only on state – and a representation of long-term transitions in the environment called the SR, $M : \mathcal{S} \times \mathcal{S} \to \mathbb{R}$. Assuming a discrete state space, $M(s, s')$ is the expected number of discounted visits to state $s'$ given an initial state $s$ and policy $\pi$:

$$M(s, s') = \mathbb{E}_\pi \left[ \sum_{t=0}^{\infty} \gamma^t \, \mathbb{1}[s_t = s'] \,\middle|\, s_0 = s \right], \tag{5.11}$$

where $\mathbb{1}[\cdot]$ is the indicator function. The SR represents temporal correlations between states. Just as the value function accumulates rewards discounted over an episode, the SR accumulates state occupancies. Also like the value function, it follows a Bellman-like equation,

$$M(s_t, s') = \mathbb{1}[s_t = s'] + \gamma M(s_{t+1}, s'). \tag{5.12}$$

Thus, model-free algorithms used for value function learning (*e.g.*, TD learning, eligibility traces, etc.) can be applied to SRs. Furthermore, given the SR and a reward function one can easily compute state values,

$$V(s) = \sum_{s' \in \mathcal{S}} M(s, s') \, r(s'), \quad \boldsymbol{V} = M\boldsymbol{r} \tag{5.13}$$

where $M$ is the matrix such that $M(s_i, s_j)$ is its $(i, j)$ element, and $\boldsymbol{r}$ is the vector of the expected rewards for each state. An alternative version of the SR for actions, $M(s, s', a)$ (the expected discounted future state occupancy starting from state $s$ and taking action $a$), can be used to compute the Q-function instead. Thus, in the SR framework, decisions can be made cheaply by selecting actions that maximize the Q-function. Or, alternatively, a value function computed via Eq. (5.13) could drive learning of a policy as done in AC models. A SR model is capable latent learning and can more quickly adapt to distal changes in the reward structure, facilitating transfer learning (Gershman, 2018). However, changes in the state transition structure of the environment may be slow to propagate to the SR of all states.

To generalize this to continuous state spaces, the indicator function is replaced by a feature function (Barreto et al., 2017). Let $\phi : \mathcal{S} \to \mathbb{R}^d$ be a function that maps states to a vector representation called the state's features. The features can be learned from raw state observations, or engineered in some way (*e.g.*, representations from a Vector Symbolic Algebra (VSA) or SSPs). The successor feature (SF) function $\boldsymbol{M} : \mathcal{S} \to \mathbb{R}^d$ is defined as

$$\boldsymbol{M}(s_i) = \mathbb{E}_\pi \left[ \sum_{t=0}^{\infty} \gamma^t \phi(s_t) \,\middle|\, s_0 = s_i \right], \tag{5.14}$$

that is, it is expected discounted sum of future features. If states are in fact discrete and finite and $\phi$ is defined as a one-hot encoding, then the successor representation is recovered by $M(s, s') = \boldsymbol{M}(s) \cdot \phi(s')$, and the successor features $\boldsymbol{M}(s_i)$ are simply the rows of the SR matrix.

The successor features follow a Bellman-like equation and so can be learned with TD learning rules.

$$\delta_t^{\mathrm{SF}} = \phi(s_t) + \gamma \boldsymbol{M}^{\pi}(s_{t+1}) - \boldsymbol{M}^{\pi}(s_t). \tag{5.15}$$

Usually a linear function is used to represent the reward function when used alongside SFs:

$$r(s) = \phi(s) \cdot \boldsymbol{w}_r. \tag{5.16}$$

This is so that the value function can be recovered linearly: $V(s) = \boldsymbol{M}(s) \cdot \boldsymbol{w}$. Note that this requires the underlying reward function to be expressible as a linear combination of the feature components or be sufficiently close to a function that is (Barreto et al., 2017).

**Continuous time RL.** Markov decision processes and, thus, RL can be generalized to continuous-time settings. The state of the environment and action signal are continuous functions, $s(t)$ and $a(t) \sim \pi(s(t))$, the discrete rewards are replaced by a reward rate function, $r(t) = r(s(t), a(t))$, and the state evolves according to a (potentially stochastic) state dynamics function, $\dot{s}(t) = f(s(t), a(t))$. Rather than maximizing the expected discounted sum of future rewards, the goal in continuous-time RL is maximizing the expected discounted *integral* over the reward rate function,

$$\max_{\pi} \mathbb{E}_{\pi}\left[\int_{t=0}^{\infty} \gamma^t r(t) dt\right], \tag{5.17}$$

where $0 < \gamma < 1$ is the discount per time unit. The definition of the value function in continuous-time RL likewise replaces sums with integrals:

$$V(s) = \mathbb{E}_{\pi}\left[\int_{k=0}^{\infty} \gamma^k r(t+k) dk \;\middle|\; s(t) = s\right]. \tag{5.18}$$

Note that the optimal value function satisfies

$$V(s(t)) = \max_a \mathbb{E}\left[\int_{k=t}^{\infty} \gamma^{k-t} r(k) dk\right], \tag{5.19}$$

$$= \max_a \mathbb{E}\left[\int_{k=t}^{t+\Delta t} \gamma^{k-t} r(k) dk + \gamma^{\Delta t} V(s(t+\Delta t))\right]. \tag{5.20}$$

Taking the Taylor expansion of the term $V(s(t+\Delta t))$ and the limit as $\Delta t \to 0$, we arrive at the discounted version of the Hamilton–Jacobi–Bellman equation,[4]

$$-\log(\gamma) V(s(t)) = \max_a \mathbb{E}\left[r(s(t), a(t)) + \frac{\partial V(s(t))}{\partial s} \frac{ds(t)}{dt}\right]. \tag{5.21}$$

[4]This is the Hamilton–Jacobi–Bellman equation for deterministic optimal control; there are other versions of this equation that incorporate stochastic dynamics.

Thus, the following learning rule can be derived for this integral formulation of the value function:

$$V(s(t)) \leftarrow V(s(t)) + \eta \underbrace{\left[ r(t) + \log(\gamma) V(s(t)) + \frac{dV(s(t))}{dt} \right]}_{=\delta^{(0)}(t)}. \tag{5.22}$$

We refer to this as the continuous-time version of the TD(0) rule, as it uses only instantaneous information (Doya, 2000). The derivative of the value function can be approximated by a difference quotient. Using this, continuous-time value functions can be learned – either in conjunction with a policy, or the gradient of the value function can be used for greedy control (Doya, 2000).

While TD updates can drive value function learning in continuous-time, they cannot always be applied to learn a useful Q-function. In continuous-time, the Q-function collapses to the value function, and, thus, is problematic as a tool for decision making (Baird, 1994). One can define a Q-function with fine time discretization, but the result is highly sensitive to the size of the timestep, making this problematic (Tallec et al., 2019).

### *5.2.2 Prior Work*

Our interest is in the biological plausibility of RL methods, which means that algorithms operating in continuous time and space are of particular interest, as are those that have been explicitly mapped to biological mechanisms. In this section, we review past work in these directions.

*Continuous time RL.* While RL in continuous-time is not a new subject, it is often neglected. Many tasks that are naturally described in continuous-time (*e.g.*, navigation in continuous spaces, motor control, robotics simulations) are solved via fine time discretization (Fu et al., 2020). But this is not always the case. In Doya (2000), the integral value definition given in Eq. (5.17) was used to derive the continuous-time TD(0) rule in Eq. (5.22), along with a TD($\lambda$) rule via eligibility traces, to learn a value function approximated by a radial basis function network. An SNN model using this TD(0) rule for value learning and a winner-take-all (WTA) network for deterministic action selection was presented in Frémaux et al. (2013).

In other work, continuous-time value and advantage functions are learned by discretizing the value integral (Baird, 1994; Tallec et al., 2019). Model-based approaches that learn continuous state dynamics using neural ordinary differential equations have been presented as well (Du et al., 2020; Yildiz et al., 2021). Stochastic policy gradient and AC models in continuous-time settings have been explored in a control theoretic framework, for example in Wang et al. (2020), which assumes a linear–quadratic form, and more generally in Jia et al. (2022).

*Local learning.* To make biologically constrained RL models, there are several important considerations, including the plausibility of learning rules, the mechanisms of action selection, methods for action representation, and the role of memory. Some models draw inspiration

from the role of dopamine in the reward prediction error hypothesis – using local Hebbian learning rules extended to include external error signals for TD learning (Izhikevich, 2007), as well as spike-timing-dependent plasticity (STDP) learning rules (Yan et al., 2019). In contrast, other approaches use backpropagation, adapted for deep spiking neural networks, despite the acknowledged biological implausibility (Tang et al., 2021; Liu et al., 2022).

*Action selection.* Regarding action selection in SNNs, various strategies have been explored – most of which are limited to discrete action settings. One approach is first-spike coding, where each neuron represents a discrete action with the selected action determined by the first neuron to fire after entering a new discrete state (Potjans et al., 2009). Another strategy involves a WTA mechanism over discrete action spaces, where individual neurons or groups of neurons represent actions and integrate input and, upon reaching a threshold, inhibit other groups (Frémaux et al., 2013; Zambrano et al., 2015; Yan et al., 2019). Alternatively, Q-learning over discrete action spaces has been implemented (Liu et al., 2022). Moving away from RL settings, biologically plausible spiking models of the basal ganglia have been proposed as discrete action selectors (Joel et al., 2002; Stewart et al., 2010b).

More recently, the basal ganglia model of Stewart et al. (2010b) was extended to use distributional coding of actions, making it more suitable for continuous action scenarios (Bartlett et al., 2023a). For continuous control in robotics tasks, some models use the weighted output of filtered neural activity as a deterministic continuous action signal or as the mean of a Gaussian distribution, which may then be sampled outside the network for action selection (Tang et al., 2021). Others have used membrane voltages at output layers as continuous action signals (Chen et al., 2024).

*Memory.* There are other considerations in biologically-plausible RL. For truly online learning, the unavailability of future values for TD updates necessitates the use of current estimates to compute errors of past estimates. This challenge requires a memory system, which may involve replaying past experiences, storing past neural activity, or integrating activity over time. Decaying traces of neural activity, functionally equivalent to a low-pass filter over spikes, are often employed for this purpose (Potjans et al., 2009; Zambrano et al., 2021; Zambrano et al., 2015). While the time scales of synapses is much shorter than that normally used for eligibility traces, long time constant low pass filters can be realized by recurrent neural networks rather than in synapse dynamics.

In contrast to work using traces, the hierarchical reinforcement learning (HRL) model proposed by Rasmussen et al. (2017) addresses the memory challenge by employing two identical neural populations to represent current and delayed Q-functions, with mechanisms in place to copy learned weights from one population to the other, akin to a discrete-time TD(0) algorithm with a target network. Alternatively, Duggins (2023) presented an SNN model that uses regular replay over a one-step state memory to obtain the neural activities necessary for TD(0) learning in a time-blocked task.

*Successor Representation.* Compared to model-free and model-based learning, SR has not been as widely used, but it has seen a resurgence in recent years, in part, due to work in deep successor RL for transfer learning and research into the connections between SR and hippocampal place cells. For example, in Zhang et al. (2017), deep SFs were used for transfer learning in navigation tasks. In Madarasz et al. (2019), Bayesian Successor Representations were introduced, which maintain a belief distribution over multiple successor maps. An SR is sampled from this distribution, used for greedy action selection, and then all maps are updated, allowing transfer of policies between environments with similar reward functions.

Connections have also been drawn between SRs and neuroscientific data. When an agent explores an environment with a random walk, neurons in a population representing learned SRs will have place cell-like activity (Stachenfeld et al., 2017). During other tasks, there will be some differences between the learned SRs and the classical idea of place fields. For example, if an agent learns a SR in an environment where it moves down a corridor in only one direction, the SR place fields will be asymmetric – distorted in the direction of motion. When there is a reward in the environment, SR place fields cluster more around the goal. These features have also been observed in hippocampal place cell populations (Mehta et al., 2000).

In Cothi et al. (2020), place cells were used to represent SRs in a discrete time environment, with boundary vector cells encoding features and grid cells encoding the eigenvectors of the SFs. This work was able to explain grid cell data in structured mazes. Additionally, STDP has been used to train spiking neural populations that encode SR in discrete state environments (George et al., 2023; Bono et al., 2023).

The proposal that place cells, or some subset of them, encode an SR suggests that statistical learning occurs in the hippocampus. A more classical view is the complementary learning systems theory, which posits that slow, statistical learning occurs in the neocortex, rapid learning of episodic memories occurs in the hippocampal formation, and the hippocampus replays episodes offline during sleep to the neocortex for consolidation (McClelland et al., 1995). This achieves an interweaving of learning experiences, which can eliminate catastrophic forgetting (Rolnick et al., 2019). However, there is also evidence for rapid statistical learning in the hippocampus (Covington et al., 2018; Schapiro et al., 2014; Schapiro et al., 2016). Indeed, Schapiro et al. (2017) suggest a more complicated picture, theorizing that a complementary learning system exists within the hippocampus, with different pathways involved in statistical learning and episodic memory (one-shot learning).

Contrary to the reward prediction error hypothesis, dopaminergic neurons also show activity changes in response to value-neutral sensory prediction errors (Takahashi et al., 2017) and errors in stimulus–stimulus associations (Sharpe et al., 2017). Furthermore, there is heterogeneity in dopaminergic neurons responses, suggesting a vector-valued signal Lee et al. (2022). Gardner et al. (2018) theorize that dopmaine-producing neurons encode the more general, and vector-valued TD error for SRs.

*Distributional RL.* Another explanation for these findings, suggesting a non-scalar, distributed dopamine signal, is distributional RL (Lowet et al., 2020). In distributional RL, rather than simply learning the expected future return, one learns the distribution of future return. This allows for risk-aware decision making. In Dabney et al. (2020), recordings from rodents performing tasks with probabilistic rewards indicated that midbrain dopaminergic neurons were encoding distinct parts of the reward distribution rather than just the mean expected reward.

### *5.2.3 Memory*

In Section 5.2, we presented TD rules as updates on current estimates using future observations. As mentioned in the last section, these errors are not available until future observations have in fact been observed. In practice, we update past estimates using current and past observations when learning online – which requires memory. Memory replay buffers, now ubiquitous in RL, were introduced by Lin (1992). Experiences – tuples consisting of state transitions, action, and reward, $(s_t, a_t, r_t, s_{t+1})$ – are stored in a list that is accessed so that agents might learn from past experiences (and so that long sequence TD errors can be computed during online learning). Buffer sampling may be done with forward replay (stepping through the buffer forward in time, starting from most distant observations), backward replay, or in a shuffled fashion, randomly sampling batches of experiences. Random sampling has the advantage of breaking temporal correlations – preventing over-fitting to recent experiences and reducing instability (Mnih et al., 2015).

Replay can also be done in a more principled fashion. In prioritized experience replay, experiences associated with larger TD errors are sampled more frequently. In this way, agents can learn from their biggest mistakes, improving sample efficiency and performance (Schaul et al., 2015). Model-based algorithms provide another way to improve sample efficiency, as trained state and reward models can be used to simulate experiences for training value functions and policies.

A memory replay buffer is quite different from memory in the brain. The brain's neural circuitry does not store memory in a simple list, and instead information over time is encoded in either neural activities or via changes in synaptic weights. It is crucial to note that, unlike replay buffers, neural memory is far from perfect. For instance, human working memory can retain information on a limited scale, typically ranging from 3 to 5 items[5] (Cowan, 2001) and, without rehearsal, the duration of information retention in working memory is very short, on the order of tens of seconds (Reitman, 1974; Ricker et al., 2010).

In contrast to replay buffers, neural memory exhibits distinctive temporal effects during recall. Memories of the early and most recent events in an episode are generally more accurately recollected (Murdock Jr, 1962). Additionally, the act of recalling a specific event can trigger the

[5]Different numbers for the limit of working memory have been suggested – from 4-9 by Miller (1956) to only 3-4 in Gilchrist et al. (2008). Furthermore, there are different findings given different memory tasks – whether items are numbers, words, images, or phrases.

recall of similar events or those experienced in close temporal proximity (Tulving et al., 1973). These nuances are absent from conventional replay buffers used in RL.

Furthermore, the brain harbours diverse types of memory, each serving distinct roles, with some potentially better suited for RL applications. For instance, the prefrontal cortex is often linked to working memory, which encapsulates the temporary storage and manipulation of information required for cognitive tasks and decision-making processes (Funahashi, 2017). But working memory has a limited capacity and is susceptible to degradation (Constantinidis et al., 2016). Long-term memory persists for much greater lengths of time – for whole lifetimes even – and has a high capacity (Brady et al., 2008). It includes general knowledge, associations between concepts, and episodic memory. Episodic memory involves the storage of events including their spatio-temporal context and is generally associated with the hippocampal formation (Vargha-Khadem et al., 1997; Burgess et al., 2002).

Given our specific interests in neural coding and RL, the role of "time cells" in memory and learning is noteworthy. These cells, found in the hippocampus and other brain regions, contribute to temporal coding, aiding in the organization of memories over time (MacDonald et al., 2011; Eichenbaum, 2014; Umbach et al., 2020). These cells fire at particular moments during a behavioural sequence, essentially creating a temporal map of events. This mapping is essential for episodic memory and encoding of sequences and timing of events. In the context of RL, time cells may provide a neural basis for understanding how agents compute errors and learn from sequences of actions and their outcomes over time. Furthermore, they may be a useful basis for state memory – in Lin et al. (2023) time cells and ramping cells emerged in populations of recurrent units encoding state features in a deep RL agent.

An interesting model of time cells is the delay network presented in Voelker et al. (2018). This recurrent SNN was derived to optimally retain information over a fixed time window and contains neurons with responses qualitatively comparable to those of time cells. It maintains a memory of a continuous input signal by projecting the signal onto an orthogonal temporal basis of Legendre polynomials. This model was later extended to the Legendre Memory Unit (LDN), an artificial neural network (ANN) recurrent unit that addresses the scaling problems of traditional recurrent neural networks (RNNs), such as the Long Short-Term Memory (LSTM) or GRU (Voelker et al., 2019). It has influenced the development of many modern deep learning models that incorporate state space representations.

**Legendre Delay Network**. In Section 5.3, we use a state space model (SSM), in particular the LDN, to encode and recall information over time to compute TD errors. SSMs are an approach to computing on sequential or temporal data that leverage many of the benefits of continuous dynamical system neural networks, recurrent neural networks, and convolutional neural networks. One of the early SSMs specifically tailored for deep learning was the LDN (Voelker et al., 2019), which has since been shown to achieve state-of-the-art performance on several benchmark NLP datasets (Chilkuri et al., 2021a). A more recent SSM, proposed by Gu et al. (2022a), achieved state-of-the-art performance on every task in the Long Range Arena

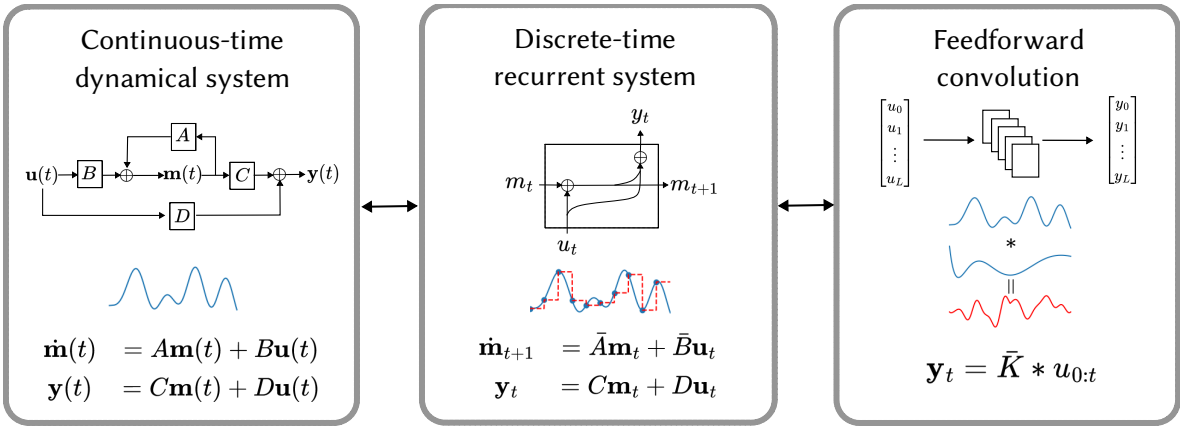



**Figure 5.2:** SSMs can be realized and analyzed in several ways. An SSM can be viewed as a continuous time dynamical system. Alternatively, it can be discretized and viewed as a recurrent unit in a network. Both the continuous and discrete time realizations can be re-formulated into a feed-forward convolution by the SSM kernel. This diagram is inspired by Figure 1 of Gu et al. (2022a).

benchmark (Tay et al., 2020), including the previously unsolved Pathfinder-X task, leading to increased attention to SSMs in the deep learning community. More recently, SSMs have appeared in deep networks applied to partially-observable tasks in RL that require a memory over observations, and were found to outperform other recurrent units (Lu et al., 2024).

Let us review SSMs. Given a continuous time input signal $\boldsymbol{u}(t) \in \mathbb{R}^p$, a continuous state space model is defined as

$$\dot{\boldsymbol{m}}(t) = A\boldsymbol{m}(t) + B\boldsymbol{u}(t) \tag{5.23}$$

$$\boldsymbol{y}(t) = C\boldsymbol{m}(t) + D\boldsymbol{u}(t) \tag{5.24}$$

where the memory vector $\boldsymbol{m}(t)$ is a $q$-dimensional latent representation of the signal $\boldsymbol{u}(t')$ from $t' = 0$ to $t$, $q$ is called the order of SSM, and $\boldsymbol{y}(t) \in \mathbb{R}^q$ is an output vector. The matrices that characterize the SSM are the the state matrix $A \in \mathbb{R}^{q \times q}$, the input matrix $B \in \mathbb{R}^{q \times p}$, the output matrix $C \in \mathbb{R}^{m \times q}$, and the feedthrough matrix $D \in \mathbb{R}^{m \times p}$ – though we shall generally assume $D = 0$.[6] These matrices may be fixed or learned, and can potentially be functions over the input $\boldsymbol{u}(t)$.

While the matrices $A, B, C$ can be defined as time-varying, we shall assume they are constant and so this is a linear time invariant (LTI) system. Thus, the system's output can be computed using continuous convolution:

$$K(t) = Ce^{tA}B, \quad y(t) = \{K * u\}(t) \tag{5.25}$$

where $K(t)$ is called the system's impulse response. The discretized version of Eq. (5.25) can be computed quickly using the fast Fourier transform (FFT). This insight was used in Chilkuri

[6]The inclusion of $D$ is a type of skip connection, but it is typically omitted for simplicity.

et al. (2021b) and Chilkuri et al. (2021a) and Gu et al. (2022a) to speed up deep SSM training with parallelization, similar to transformers and CNNs, while at inference time or sequence generation, the SSM could still be run as a standard recurrent unit (Figure 5.2).

The memory vector $\boldsymbol{m}(t)$ serves a similar role as the hidden state in an RNN unit. One might question if a linear recurrent system could compete with more sophisticated recurrent units, such as LSTM. However, SSMs models, even with fixed state and input matrices, have been found to outperform LSTMs and other recurrent units, particularly in tasks that require learning long-range dependencies (Voelker et al., 2019; Chilkuri et al., 2021a). Nonlinear RNNs can suffer from vanishing and exploding gradients, which complicates learning long-term dependencies. In contrast, linear SSMs do not suffer from these issues to the same extent and their gradients remain more stable during training. The eigenvalues of the system matrix $A$ in linear SSMs directly influence the stability and dynamics of the model. If these eigenvalues are well-conditioned (*i.e.*, they do not lead to divergent or decaying states rapidly), the system can maintain stability and performance.

A naive approach to training an SSM would be to randomly initialize the matrices $A, B, C$ and learn them via backpropgation. However, in practice, this performs poorly (Gu et al., 2022b), and in fact, this was not the approach taken by the LDN. The LDN (Voelker et al., 2019), originally inspired by time cells in the mammalian brain, uses fixed state and input matrices designed to be optimal for retrieving delayed information from the state vector over a fixed time window. These optimal matrices are given by

$$A_{i,j} = \frac{1}{\theta}(2i+1)\begin{cases} -1, & i < j \\ (-1)^{i-j+1}, & i \geq j \end{cases} \tag{5.26}$$

$$B_i = \frac{1}{\theta}(2i+1)(-1)^i, \quad \text{for } i, j \in [0, n-1], \tag{5.27}$$

where $\theta$ is a parameter that determines the 'window size' of the LDN – how far into the past information can be decoded. When using the LDN as a layer in a network applied to some task, the $A$ and $B$ matrices are generally fixed and the $C$ matrix (along with any subsequent layers in the network) is usually learned to produce the desired output vector (*e.g.*, a prediction of the input at some future time, a classification of the input sequence, or a function over the history window). If we wish to compute a delay, there is a closed-form solution for the output matrix $C$:

$$C_j = \mathscr{P}_j\left(\frac{\tau}{\theta}\right), \quad 0 \leq \tau \leq \theta, j \in [0, n-1], \tag{5.28}$$

$$\text{where } \mathscr{P}_j(t) = \frac{1}{2^j j!}\frac{d^j}{dt^j}((2t-1)^2 - 1)^j. \tag{5.29}$$

We then have that $\boldsymbol{y}(t) = C\boldsymbol{m}(t) = \boldsymbol{P}\left(\tau/\theta\right)\boldsymbol{m}(t) \approx u(t-\tau)$, *i.e.*, the output of the LDN is approximately the input signal $\tau$ seconds in the past. That is, the memory vector at time $t$ is a compressed representation of the input signal over a $\theta$ second time window. Note that

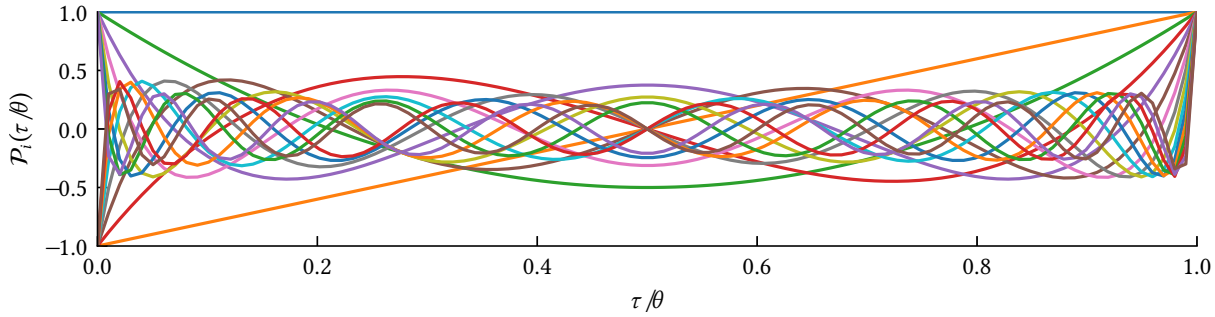



**Figure 5.3:** The shifted Legendre polynomials of orders $i = 1, \ldots, 16$.

$\mathcal{P}_j$ is the $j^{th}$ order shifted Legendre polynomial (see Figure 5.3 for a visualization of these polynomials) and $\boldsymbol{P}$ is a vector of shifted Legendre polynomials. The memory vector $\boldsymbol{m}(t)$ is approximately the projection of the input signal $u(t)$ onto the orthonormal Legendre basis, and the LDN is solving the problem of online function approximation.

A larger LDN state size $q$ allows for the representation of higher frequency components of the signal or longer delay periods without increasing the error. In particular, the accuracy of the delay approximation by the network is determined by the product $\theta \cdot f$, where $f$ is the maximum frequency in the input (Voelker, 2019). This scaling property implies that for a fixed error tolerance, we can adjust $\theta$ and $q$ proportionally: doubling $\theta$ (doubling the memory window) requires that we double $q$ to maintain the same frequency range for accurate delay representation. If $q = 1$ then the LDN simply acts as a first order low-pass filter on the input scaled by $1/\theta$ and with time constant $\tau = \theta$ – see Eq. (2.8):

$$A = -\frac{1}{\theta},\ B = \frac{1}{\theta} \ \rightarrow\ \dot{m}(t) = -\frac{1}{\theta}m(t) + \frac{1}{\theta}u(t) \tag{5.30}$$

Although the LDN state and input matrices are theoretically optimal for retrieving delays under some assumptions, this does not mean the Legendre basis and LDN is always the ideal choice of SSM. Some tasks may have signals with characteristics that make a different choice more appropriate. Additionally, the optimality of the LDN matrices is in terms of the computation of delays up to a fixed predetermined length $\theta$,[7] as accurately as possible. In some cases, it may be acceptable, or preferred, for delay recall to be unbounded but with accuracy drop-off.

More recent work has expanded on this family of models, introducing new state and input matrices. Stöckel (2022) explained how many function bases can be converted to LTI systems to define the SSM matrices. Examples included the cosine basis, a modified Fourier basis, and the Haar basis. Gu et al. (2022b) proposed several more matrices for SSMs. Like the LDN matrices, these were derived for the task of signal memorization. Unlike the LDN matrix,

[7]Note that the length of the LDN time-window can learned for a given task, or controlled over the course of a simulation.

which is optimal for recall over a fixed time window, these were derived for memorization over an unbounded window with exponential memory decay. Other approaches have focused on efficient training of SSMs and more expressive dynamics – such as Gu et al. (2023), which featured input-dependent state matrices.

## 5.3 Continuous Time Temporal Difference Learning

In this section, we will develop methods for online, on-policy TD learning in spiking neural networks that operate in continuous-time and use biologically plausible memory mechanisms. LDNs allow us to encode windowed histories of continuous signals. Given an LDN memory state, $\boldsymbol{m}(t)$, computed from input signal, $u(t)$, we can linearly decode not only delays, $u(t-k) \approx \boldsymbol{P}\left({}^{k}/_{\theta}\right)\boldsymbol{m}(t)$, but any linear function of the signal history.[8] For example, we can approximate its rate of change in the past, $\frac{du(t-k)}{dk} \approx \frac{d\boldsymbol{P}({}^{k}/_{\theta})}{dk}\boldsymbol{m}(t)$, or integrate the signal over the memory window, $\int_{k=0}^{\theta} u(t-k)dk \approx \int_{k=0}^{\theta} \boldsymbol{P}({}^{k}/_{\theta})dk\ \boldsymbol{m}(t)$. In these examples, the only term that changes is the matrix applied to $\boldsymbol{m}(t)$. So, given a recurrent neural population that encodes the memory state of an LDN, we can project linear functions over a signal's history to different populations, with the weights set to approximate the desired transformation. Thus, from an LDN we can compute continuous-time TD errors.

We will first derive TD errors for continuous-time value functions and, in later sections, generalize those to successor features and value distributions. Note that standard TD rules are written as updates on current estimates using future observations. This is how TD updates were presented in Section 5.2.1. To perform *online* RL, we now reverse this perspective and write learning rules as TD updates on *past* estimates using *current* memories and estimates.

### *5.3.1 TD(0)*

To begin, assume that we have a neural population encoding state, $s(t) \in \mathbb{R}^d$, received as input from the environment. The state population projects onto a value population. The synaptic weights between these populations (and perhaps between other intermediate hidden layers) will be trained with an error signal (Figure 5.4).

Recall that the continuous-time version of the TD(0) error is given by $\delta^{(0)}(t) = r(t) + \log(\gamma)V(s(t)) + \frac{dV(s(t))}{dt}$. Let $\boldsymbol{m}_r(t) \in \mathbb{R}^{q_r}$ be an LDN memory of the reward signal and $\boldsymbol{m}_v \in \mathbb{R}^{q_v}$ be an LDN memory of the output of the value population. Assume that all LDNs have a time window of $\theta$. We can decode the TD(0) error for our estimate $t-k$ seconds in the past from these memory traces:

$$\delta^{(0)}(t-k) = r(t-k) + \log(\gamma)V(s(t-k)) - \frac{dV(s(t-k))}{dk} \tag{5.31}$$

[8] Nonlinear functions over a signal history can be approximated as well, but not accurately with linear decoding.

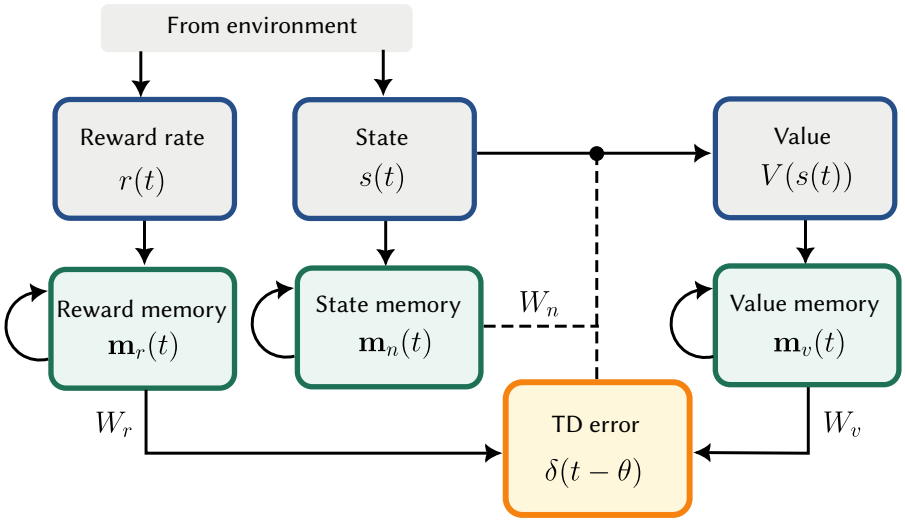



**Figure 5.4:** Diagram of a value network that uses LDN memories for TD learning. Blue blocks denote neural populations, green blocks denote LDNs, and the orange block is an error signal. Solid lines indicate neural projections and dashed lines indicate a modulatory effect on a set of synapses.

$$= \boldsymbol{P}^{q_r}\left(\frac{k}{\theta}\right)\boldsymbol{m}_r(t) + \left[\log(\gamma)\boldsymbol{P}^{q_v}\left(\frac{k}{\theta}\right) - \frac{d\boldsymbol{P}(k/\theta)}{dk}\right]\boldsymbol{m}_v(t) \tag{5.32}$$

To train a critic network with this error, we can replay through a $\theta$ second long history of the state via an LDN, $\boldsymbol{m}_s(t) \in \mathbb{R}^{q_s \times d}$. This would require interweaving periods of experience gathering (during which the LDN memory traces are built through recurrent dynamics) with replay and learning (during which the LDN states are frozen and sampled). Replay through state memories can be done with forward replay, backward replay, or via random sampling. This is similar to the approach taken by Duggins (2023) – training a spiking RL network by alternating between interacting with the environment and updating in phases – though unlike our approach, their learning phase only updates the last estimate, rather than any set of estimates within a longer memory window.

This TD(0) error could be used to drive any error-based learning algorithm. In Figure 5.6, we use it to train a spiking neural network with the biologically plausible prescribed error sensitivity (PES) learning rule. Let $T$ be the end time of the last experience-gathering phase. The decoder on the $i^{\text{th}}$ neuron of the state population is updated via

$$\Delta \boldsymbol{d}_i(t) = \eta\delta^{(0)}(T - k(t)\theta)a_i(t) \tag{5.33}$$

$$= \eta\left[\boldsymbol{P}^{q_r}(k(t))\,\boldsymbol{m}_r(t) + \left(\log(\gamma)\boldsymbol{P}^{q_v}(\tau(t)) - \frac{1}{\theta}\frac{d\boldsymbol{P}(k(t))}{dk}\right)\boldsymbol{m}_v(t)\right]a_i(t), \tag{5.34}$$

$$\text{while } s(t) = \boldsymbol{P}^{q_s}(k(t))\boldsymbol{m}_s(t),$$

where $\eta$ is a learning rate and $k(t) \in [0, 1]$ determines the replay mechanism: $k(t) = 0$ corresponds to sampling the most recent memories from the LDNs (the state, reward, and

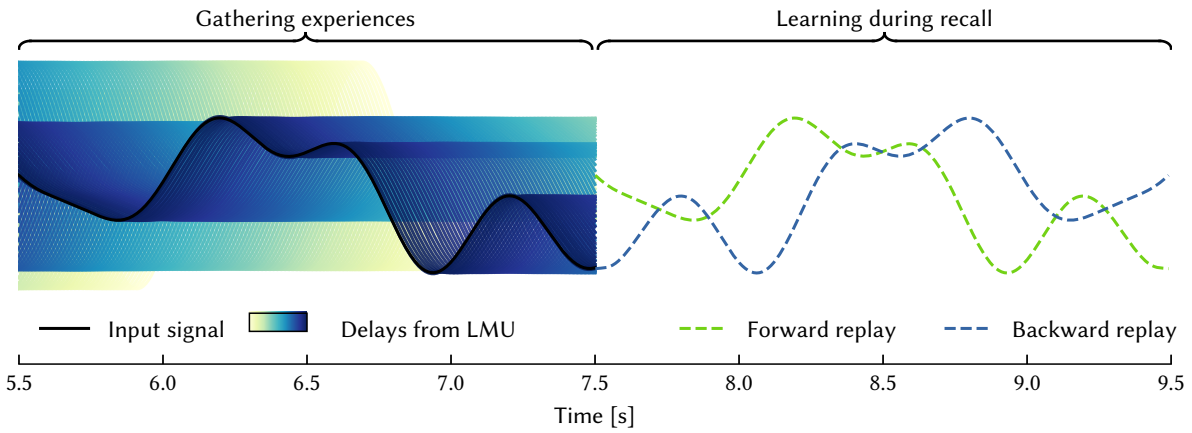



**Figure 5.5: LDNs store a compressed representation of a continuous signal, which can be later recalled and replayed for learning.** Here $u(t)$, an input signal (solid black line), is used to continuously update $m(t)$, an LDN memory (with $q = 10$ and $\theta = 2$s), in the 'gathering experiences' phase. Visualizing $m(t)$ in the 10-dimensional space in which it resides is not possible, so instead we depict the evolving memory by plotting the delayed information that can be obtained from it. The coloured lines in 'gathering experiences' phase show $\boldsymbol{P}\left({}^{k}/_{\theta}\right)\boldsymbol{m}(t)$, an approximation of $u(t-k)$, for $k \in [0, \theta]$. Darker colours correspond to smaller $k$ (*i.e.*, a small delay, the memory of recent experiences) and lighter colours correspond to larger $k$ (*i.e.*, a large delay, the memory of more distant experiences). After the 'gathering experiences' phase, the input signal and LDN memory are frozen and the last 2 seconds of the signal are approximately replayed (either forward or backwards in time) and used for TD learning in the 'learning during recall' phase, as described in Eq. (5.34).

value estimate at time $T$) while $k(t) = 1$ corresponds to sampling the most distant memories from the LDNs (the state, reward, and value estimate at time $T - \theta$). Forward replay over a learning time window of length $w$, is obtained by $k(t) = {}^{w-t}/_{w-T}$, while backward replay occurs when $k(t) = {}^{t-T}/_{w-T}$. See Figure 5.5 for a high-level picture and Figure 5.6 for an example of this type of learning.

Note that to compute the update in Eq. (5.34), we must compute the time-varying transformations, $\boldsymbol{P}(\tau(t))$, and compute the products, $\boldsymbol{P}(\tau(t))\boldsymbol{m}(t)$, with a separate neural population. The update cannot be computed directly from the LDN populations so additional neural resources are required.

By separating experience and learning phases, we are, in effect, learning offline. However, explicit replay is not strictly required. For fully online learning without replay, we do though require additional memory. Assume that we have a population of $N$ neurons representing the state. We need an LDN, $\boldsymbol{m}_n(t) \in \mathbb{R}^{q_n \times N}$, that encodes the history of neural activities of the state population. With this we can compute the updates to the decoders without stepping through the memory of the state signal,

$$\Delta \boldsymbol{d}_i(t) = \eta \delta^{(0)}(t-\theta)\boldsymbol{P}^{q_n}(1)\boldsymbol{m}_n(t) \tag{5.35}$$

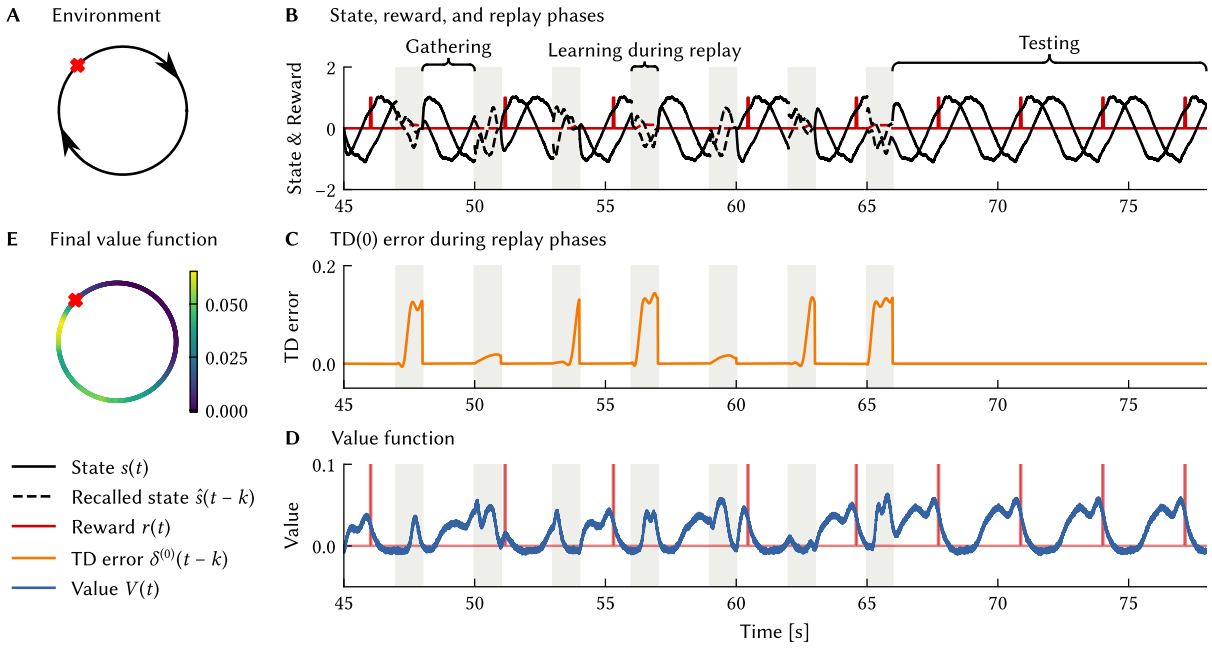



**Figure 5.6: Learning a value function with replay phases. (A)** In this example, an agent travels around a circle and receives reward at a particular location. We expect that a value function will anticipate this reward. **(B)** A spiking neural population encoding the value function was trained via Eq. (5.34) and backwards replay. During the 'gathering' phase, the agent travels clockwise at a constant frequency.LDNs store a record of the observed state (the agent $(x, y)$ location), reward, and value. During the 'learning' phase, states are replayed backwards from the LDN and learning occurs. During the simulation time labelled as 'testing', learning was turned off. **(C)** The TD(0) error, given by Eq. (5.34), is only computed during the learning phases. **(D)** The value function over time. By the test period, the value function increases leading up to the reward location as expected. **(E)** The final value function.

$$= \eta \Bigg[ \underbrace{\boldsymbol{P}^{q_r}(1)}_{=W_r} \boldsymbol{m}_r(t) + \underbrace{\left( \log(\gamma) \boldsymbol{P}^{q_v}(1) - \frac{1}{\theta} \frac{d\boldsymbol{P}(k)}{dk}\bigg|_{k=1} \right)}_{=W_v} \boldsymbol{m}_v(t) \Bigg] \underbrace{\boldsymbol{P}^{q_n}(1)}_{=W_n} \boldsymbol{m}_{n_i}(t), \tag{5.36}$$

$$= \eta \big( W_r \boldsymbol{m}_r(t) + W_v \boldsymbol{m}_v(t) \big) W_n \boldsymbol{m}_{n_j}(t) \tag{5.37}$$

or, writing the update in terms of the weights,

$$\Delta w_{i,j}(t) = \eta \underbrace{\delta^{(0)}(t-\theta)}_{\text{error}} \underbrace{\boldsymbol{P}^{q_n}(1) \boldsymbol{m}_{n_i}(t)}_{\text{pre-neuron activity}} \underbrace{e_j}_{\text{post-neuron encoder}}. \tag{5.38}$$

We will call this the LDN-TD(0) update. This is slightly different from standard PES or other three-factor Hebbian learning rules – rather than updating weights using an error signal and current local activities, we update using an error signal and an external activity memory signal. In this version of TD(0), value estimates from $\theta$ seconds in the past are continuously updated, while the agent concurrently interacts with the environment and obtains new experiences

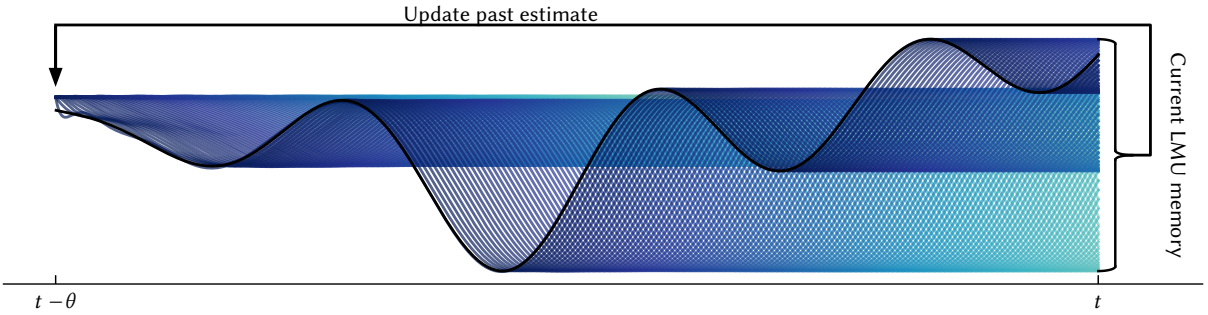



**Figure 5.7: Errors for learning can be directly computed from an LDN state without explicit replay.** An input signal $u(t)$ (solid black line) is used to continuously update $m(t)$, an LDN memory (with $q = 18$). The coloured lines show $\boldsymbol{P}\left(^{k}/_{\theta}\right)\boldsymbol{m}(t)$, an approximation of $u(t-k)$, for $k \in [0, \theta]$. Darker colours correspond to smaller $k$ (*i.e.*, a small delay, the memory of recent experiences) and lighter colours correspond to larger $k$ (*i.e.*, a large delay, the memory of more distant experiences). The current LDN memory at time $t$ (the far right of the image) can be used to compute TD errors for estimates in the past (at time $t-\theta$, the far left of the image), as described in Eq. (5.36).

(Figure 5.7). In this formulation, unlike in Eq. (5.34), the LDNs are never frozen. Furthermore, the matrices applied to the LDNs ($W_r$, $W_v$, $W_n$ in Eq. (5.36) and Figure 5.4) are fixed and therefore can be pre-computed and used to set weights between the recurrent LDN populations and the population encoding the TD error signal.

The use of an activity memory trace makes this quite similar to TD($\lambda$) with eligibility traces (ET-TD($\lambda$)). In continuous time ET-TD($\lambda$), the decoder and eligibility trace updates are given by $\Delta \boldsymbol{d}_i(t) = \eta \delta^{(0)}(t) \boldsymbol{e}_i(t)$ and $\boldsymbol{e}_i(t) = -^{1}/_{\tau} e_i(t) + a_i(t)$, where $\tau = {}^{1}/_{1-\gamma\lambda}$ (Doya, 2000). The eligibility trace is simply a low-pass filter applied to the vector of neural activities. Recall from Eq. (5.30), that when the order of the LDN activities memory $q_{n_i} = 1$, the LDN is also a low-pass filter with time constant $\theta$. Thus, with a particular choice of parameters ($q_{n_i} =$ and activities memory window length $\theta = {}^{1}/_{1-\gamma\lambda}$) ET-TD($\lambda$) can be recovered from LDN-TD(0).

### *5.3.2 TD($\theta$)*

Rather than using the instantaneous reward rate to update the value estimate (derived by clipping the integral at $\Delta t \to 0$), we can integrate the reward signal over a $\theta$ second time window to estimate returns,

$$V(s(t-\theta)) \leftarrow V(s(t-\theta)) + \eta \underbrace{\left[ \int_{k=0}^{\theta} \gamma^k r(t+k-\theta) dk + \gamma^{\theta} V(s(t)) - V(s(t-\theta)) \right]}_{=\delta^{(\theta)}(t-\theta)}. \tag{5.39}$$

We will call this the TD($\theta$) rule – a continuous-time version of the TD($n$) rule[9] – or LDN-TD($\theta$) when computed using LDNs. The discounted integral over the reward rate history can be directly computed from its LDN representation and used to drive PES learning:

$$G^{(\theta)}(t-\theta) = \left(\frac{1}{\theta}\int_0^1 \gamma^{\theta(1-\tau)}\boldsymbol{P}^{q_r}(\tau)d\tau\right)\boldsymbol{m}_r(t) + \gamma^{\theta}V(s(t)) \tag{5.40}$$

$$\delta^{(\theta)}(t-\theta) = \underbrace{\left(\frac{1}{\theta}\int_0^1 \gamma^{\theta(1-\tau)}\boldsymbol{P}^{q_r}(\tau)d\tau\right)}_{=W_r}\boldsymbol{m}_r(t) + \underbrace{\left(\gamma^{\theta}\boldsymbol{P}^{q_v}(0) - \boldsymbol{P}^{q_v}(1)\right)}_{=W_v}\boldsymbol{m}_v(t) \tag{5.41}$$

$$\Delta \boldsymbol{d}_i(t) = \eta\delta^{(\theta)}(t-\theta)\boldsymbol{P}^{q_a}(1)\boldsymbol{m}_{n_j}(t) = \eta(W_r\boldsymbol{m}_r(t) + W_v\boldsymbol{m}_v(t))W_n\boldsymbol{m}_{n_j}(t) \tag{5.42}$$

This continuously updates the value estimate in the past, like LDN-TD(0), but uses all reward information obtained since. This update has the same form as Eq. (5.36), $\Delta \boldsymbol{d}_i(t) = \eta\big(W_r\boldsymbol{m}_r(t) + W_v\boldsymbol{m}_v(t)\big)W_n\boldsymbol{m}_{n_j}(t)$, just with different $W$ matrices.

An advantage of this error signal over the TD(0) signal from Eq. (5.36) is that a longer sequence of reward observations is used to update the approximation of the value function. This improves data efficiency, as TD(0) requires many passes through the same state sequence for distal rewards to affect value estimates. However, there is a key limitation to the online implementation of LDN-TD($\theta$). The value estimate associated with a state seen at time $t$ can only be updated $\theta$ seconds later. This introduces a lag to learning that becomes more pronounced as $\theta$ is increased.

### 5.3.3 TD($\lambda$)

Here we discuss how the $\lambda$-return estimate can be defined in continuous time. Furthermore, we present a truncated approximation of the continuous time $\lambda$-return using LDN memories that allows immediate updating of value estimates without the lag of LDN-TD($\theta$).

First, let us define $\lambda$-return in continuous-time with a forward view. An error at time $t$ can be derived by integrating over all $\theta$ returns weighted exponentially, or, equivalently, by integrating over all exponentially weighted future TD(0) errors.

**Definition 5.1.** *The $\lambda$-return error is defined as*

$$\delta^{(\lambda)}(t) = \int_{k=0}^{\infty} w_\lambda(k)G^{(k)}(t)dk - V(t), \tag{5.43}$$

$$= \int_{k=0}^{\infty} W_\lambda(k)\gamma^k\delta^{(0)}(t+k)dk, \tag{5.44}$$

[9]The rule we call TD($\theta$) is similar to the error called $\text{TD}_2^u$ in Zhou et al. (2021). We instead call this TD($\theta$) to indicate that the window size $\theta$ is a parameter and because we do not use the notation of $u$ for control.

*where* $0 < \lambda < 1$ *is a weighting parameter, and* $w_\lambda(k)$ *and* $W_\lambda(k)$ *are weighting functions such that*

$$W_\lambda(k) = \int_{\tau=k}^{\infty} w_\lambda(\tau)d\tau, \quad \frac{dW_\lambda(k)}{dk} = -w_\lambda(k), \\ \text{and } W_\lambda(0) = \int_{\tau=0}^{\infty} w_\lambda(\tau)d\tau = 1. \tag{5.45}$$

Many weighting functions satisfy the requirements of Eq. (5.45). In our experiments, we use the following exponential weighting functions,

$$w_\lambda(k) = -\log(\lambda)\lambda^k, \quad W_\lambda(k) = \lambda^k. \tag{5.46}$$

Other valid choices include power-law weighting, $w_\lambda(k) = {}^{n}/_{(\lambda+k)^{n+1}}$ and $W_\lambda(k) = {}^{1}/_{(\lambda+k)^{n}}$ with $n$ set to normalize $W_\lambda(0)$, or a Gamma distribution weightings.

By using the weighting functions given by Eq. (5.46), we retain some of the key characteristics of the discrete-time $\lambda$-return error. The weighting function has a constant area: $\int_{k=0}^{\infty} -\log(\lambda)\lambda^k dk = 1$. We also have equivalence between averaging over all $\theta$ or $n$-returns and averaging over TD(0) errors. As $\lambda \to 1$, the $\lambda$-return approaches the full Monte-Carlo return estimate. But we do not obtain the property that the $\lambda$-return error collapses to the TD(0) error given in Eq. (5.22) as $\lambda \to 0$. In fact, no matter our choice of weighting function it is not possible to have both $\lim_{\lambda\to 0} \delta^{(\lambda)}(t) = \delta^{(0)}(t)$ and $\lim_{\lambda\to 1} \delta^{(\lambda)}(t) = \delta^{(T-t)}(t)$. Now we shall prove these facts.

**Lemma 5.1**. *The* $\lambda$*-return error can be expressed as an average over* $\theta$ *returns, or as an average over TD(0) errors,*

$$\int_{k=0}^{\infty} w_\lambda(k)G^{(k)}(t)dk - V(t) = \int_{k=0}^{\infty} W_\lambda(k)\gamma^k\delta^{(0)}(t+k)dk, \tag{5.47}$$

$$\text{if } W_\lambda(k) = \int_{\tau=k}^{\infty} w_\lambda(\tau)d\tau \text{ and } W_\lambda(0) = 1. \tag{5.48}$$

*Proof.* We have

$$\delta^{(\lambda)}(t) = \int_{k=0}^{\infty} w_\lambda(k)G^{(k)}(t)dk - V(t), \tag{5.49}$$

$$= \int_{k=0}^{\infty} w_\lambda(k)\left[\int_{\tau=0}^{k} \gamma^\tau r(t+\tau)d\tau + \gamma^k V(t+k)\right]dk - V(t), \tag{5.50}$$

$$= \int_{\tau=0}^{\infty}\left[\int_{k=\tau}^{\infty} w_\lambda(k)dk\right]\gamma^\tau r(t+\tau)d\tau + \int_{k=0}^{\infty} w_\lambda(k)\gamma^k V(t+k)dk - V(t). \tag{5.51}$$

This is obtained by noting that $\int_{k=0}^{\infty}\int_{\tau=0}^{k} \cdot\, d\tau dk = \int_{\tau=0}^{\infty}\int_{k=\tau}^{\infty} \cdot\, dk d\tau$. Now, we recall that $W_\lambda(\tau) = \int_{k=\tau}^{\infty} w_\lambda(k)dk$ and $W_\lambda(0) = 1$:

$$\delta^{(\lambda)}(t) = \int_{\tau=0}^{\infty} W_\lambda(\tau)\gamma^\tau r(t+\tau)d\tau - \left[W_\lambda(k)\gamma^k V(t+k)\right]_{k=0}^{\infty} \tag{5.52}$$

$$+ \int_{k=0}^{\infty} \left( W_\lambda(k) \log(\gamma) \gamma^k V(t+k) + W_\lambda(k) \gamma^k \frac{dV(t+k)}{dk} \right) dk - V(t), \tag{5.53}$$

$$= \int_{k=0}^{\infty} W_\lambda(k) \gamma^k r(t+k) dk + V(t) + \int_{k=0}^{\infty} W_\lambda(k) \log(\gamma) \gamma^k V(t+k) dk \tag{5.54}$$

$$+ \int_{k=0}^{\infty} W_\lambda(k) \gamma^k \frac{dV(t+k)}{dk} dk - V(t) \quad \text{(relabelling } \tau \text{ as } k\text{)}, \tag{5.55}$$

$$= \int_{k=0}^{\infty} W_\lambda(k) \gamma^k \left[ r(t+k) + \log(\gamma) V(t+k) + \frac{dV(t+k)}{dt} \right] dk, \tag{5.56}$$

$$= \int_{k=0}^{\infty} W_\lambda(k) \gamma^k \delta^{(0)}(t+k) dk. \tag{5.57}$$

Note that if we set $w_\lambda(k) = -\log(\lambda)\lambda^k$, then $W_\lambda(k) = \int_{\tau=k}^{\infty} w_\lambda(\tau) d\tau = \lambda^k$ and $W_\lambda(0) = 1$, as required. □

**Lemma 5.2.** *The $\lambda$-return given in Eq. (5.43) approaches the Monte Carlo return estimate as $\lambda \to 1$ if $W_\lambda(0) = 1$, $\lim_{\lambda \to 1} W_\lambda(k) = 1$ and $\lim_{\lambda \to 1} w_\lambda(k) = 0$.*

*Proof.* To obtain the Monte-Carlo error when $\lambda \to 1$, we require that

$$\lim_{\lambda \to 1} \delta^{(\lambda)}(t) = \int_{k=0}^{\infty} \gamma^k r(t+k) dk - V(t) = \delta^{(\infty)}(t). \tag{5.58}$$

Note that we have

$$\delta^{(\lambda)}(t) = \int_{k=0}^{\infty} W_\lambda(k) \gamma^k \delta^{(0)}(t+k) dk, \tag{5.59}$$

$$= \int_{k=0}^{\infty} W_\lambda(k) \gamma^k \left[ r(t+k) + \log(\gamma) V(t+k) + \frac{dV(t+k)}{dt} \right] dk, \tag{5.60}$$

$$= \int_{k=0}^{\infty} W_\lambda(k) \gamma^k \left[ r(t+k) + \log(\gamma) V(t+k) \right] dk + \left[ W_\lambda(k) \gamma^k V(t+k) \right]_{k=0}^{\infty} \tag{5.61}$$

$$- \int_{k=0}^{\infty} (-w_\lambda(k) \gamma^k + W_\lambda(k) \log(\gamma) \gamma^k) V(t+k) dk, \tag{5.62}$$

$$= \int_{k=0}^{\infty} W_\lambda(k) \gamma^k r(t+k) dk - W_\lambda(0) V(t) + \int_{k=0}^{\infty} w_\lambda(k) \gamma^k V(t+k) dk. \tag{5.63}$$

When $W_\lambda(0) = 1$, $\lim_{\lambda \to 1} W_\lambda(k) = 1$ and $\lim_{\lambda \to 1} w_\lambda(k) = 0$:

$$\lim_{\lambda \to 1} \delta^{(\lambda)}(t) = \int_{k=0}^{\infty} W_\lambda(k) \gamma^k r(t+k) dk - V(t) + \int_{k=0}^{\infty} w_\lambda(k) \gamma^k V(t+k) dk \tag{5.64}$$

Note that these conditions hold for the weighting functions specified in Eq. (5.46). □

**Lemma 5.3.** *The $\lambda$-return given in Eq. (5.43) cannot both approach the Monte Carlo return estimate as $\lambda \to 1$ and collapse to the TD(0) error as $\lambda \to 0$.*

*Proof.* To obtain the equivalence of Eq. (5.43) and Eq. (5.44) and, additionally, the property that the weighting over returns integrates to one, we required that $W_\lambda(0) = 1$. To further obtain the Monte-Carlo estimate when $\lambda \to 1$ we require $\lim_{\lambda\to 1} W_\lambda(k) = 1$.

While for the $\lambda$-return error to collapse to the TD(0) given in Eq. (5.22) as $\lambda \to 0$, the weighting function $W_\lambda(k)$ must be a 'nascent' delta function, *i.e.*,

$$\lim_{\lambda\to 0} W_\lambda(k) = \delta(k), \text{ the Dirac delta} \tag{5.65}$$

Given this we find that

$$\lim_{\lambda\to 0} \int_{k=0}^{\infty} W_\lambda(k)\gamma^k \delta^{(0)}(t+k)dk = \delta^{(0)}(t) \tag{5.66}$$

as desired.

However, this contradicts the earlier requirement that $\lim_{\lambda\to 1} W_\lambda(k) = 1$, as we need the weight function $W_0(k)$ to be both one and infinite at the origin. □

Turning back now to a backward view, a truncated version of the error in Eq. (5.43) can be estimated from LDN memory vectors to update the estimate at a past time, $t-\theta$. This however has the same problem as TD($\theta$) – there is a lag in learning that grows with $\theta$. Alternatively, we may update all estimates over the $\theta$ second memory window using subsequent reward information to obtain a rule LDN-TD($\lambda$), akin to online $\lambda$-return. This is accomplished by integrating over the error signal and activity memory,

$$\Delta \boldsymbol{d}_i(t) = \eta \left[ \int_{k=0}^{\theta} w_\lambda(k)\delta^{(k)}(t-k)\boldsymbol{P}^{q_n}(k/\theta)dk \right] \boldsymbol{m}_{n_i}(t), \tag{5.67}$$

$$= \eta \int_{k=0}^{\theta} w_\lambda(k) \Bigg[ \left( \int_{\tau=0}^{k} \gamma^{k-\tau} \boldsymbol{P}^{q_r}(\tau/\theta)d\tau \right) \boldsymbol{m}_r(t) + \gamma^k V(s(t)) - \boldsymbol{P}^{q_v}(k/\theta)\boldsymbol{m}_v(t) \Bigg] \boldsymbol{P}^{q_n}(k/\theta)dk \boldsymbol{m}_{n_i}(t), \tag{5.68}$$

$$= \eta \Bigg[ \left( \boldsymbol{m}_r(t)^T \int_{k=0}^{\theta} \int_{\tau=0}^{k} w_\lambda(k)\gamma^{k-\tau} \boldsymbol{P}^{q_r}(\tau/\theta)^T \boldsymbol{P}^{q_n}(k/\theta)d\tau dk \right) + \boldsymbol{m}_v(t)^T \left( \int_{k=0}^{\theta} w_\lambda(k) \left( \gamma^k \boldsymbol{P}^{q_v}(0)^T \boldsymbol{P}^{q_n}(k/\theta) - \boldsymbol{P}^{q_v}(k/\theta)^T \boldsymbol{P}^{q_n}(k/\theta) \right) dk \right) \Bigg] \boldsymbol{m}_{n_i}(t) \tag{5.69}$$

At every time step, we update the estimation of $V(s(t-\theta))$ to $V(s(t))$ with the maximal window lengths of return estimates, *i.e.*, update all past estimates in memory. Over time, this means that all past estimates will get updates computed using window sizes from 0 to $\theta$. The rule, LDN-TD($\lambda$), is an approximation of online truncated $\lambda$-return. This approach may be preferable

to online TD($\lambda$) in cases where you must adaptively tune the update horizon,[10] discount rate, or discount function,[11] *e.g.*, to ensure that distant rewards are properly accounted for.

### *5.3.4 Examples*

We now demonstrate the derived learning rules on two tasks: T-maze value learning; and trace conditioning.

**T-maze value learning**. Figure 5.8 shows the differences between the value function output learned using different LDN learning rules during a T-maze navigation task. In this task, an agent navigates through a corridor and then takes either the left or right branch. Upon reaching the end of a branch, the agent receives reward if ending at the right branch (the reward function is a Gaussian bump), the agent is then removed (the state is set to coordinates outside the maze), and there is a one-second wait time that marks the end of the episode. At this point, the agent's memories are reset by briefly inhibiting the activity of the LDN populations.

Our focus is solely on value learning within spiking neural networks that use LDN memory, without considering action selection or policy learning. Consequently, the decision regarding which branch the agent takes is random and is not determined by a trained policy.

The model is configured with $q_v = q_r = q_a = 8$ dimensional memory vectors and 400 neurons in both the state and the value populations. The agent learns over six episodes, each lasting six seconds, corresponding to 6000 simulation timesteps. Additionally, the state is the agent's location in the maze, a 2D position that is represented as an SSP, and encoded by a population of place cells. The LDN memory over neural activities is non-spiking in this example to speed up computations.

The results, as illustrated in Figure 5.8B-C, demonstrate that all learning rules produce a reasonable value function, which is consistent with the results from a tabular comparison using TD($\lambda$) with eligibility traces on a discretized 2D state space (called 'Tabular ET-TD($\lambda$)' in the legend). The value function remains near zero for most of the first corridor and the left branch, with a marked increase near the end of the right branch where the reward is located. However, the degree to which the reward information is propagated back from the goal state varies among the rules. LDN-TD(0) shows a peak in the value function just before the goal location, while both LDN-TD($\theta$) and LDN-TD($\lambda$) exhibit an increasing value function earlier on the right branch.

[10]The matrices we apply to the LDN memories to compute the error are memory horizon invariant. Thus, to change $\theta$, we need only apply neuromodulation to the recurrent and incoming connections of the memories networks, and do not need to change the way errors are computed from those populations. Changing $\theta$ would, however, change the effective discount rate.

[11]Different discount functions would require different transforms on the LDN output. While this could be learned, it would not be as straightforward as modifying $\theta$ on-the-fly.

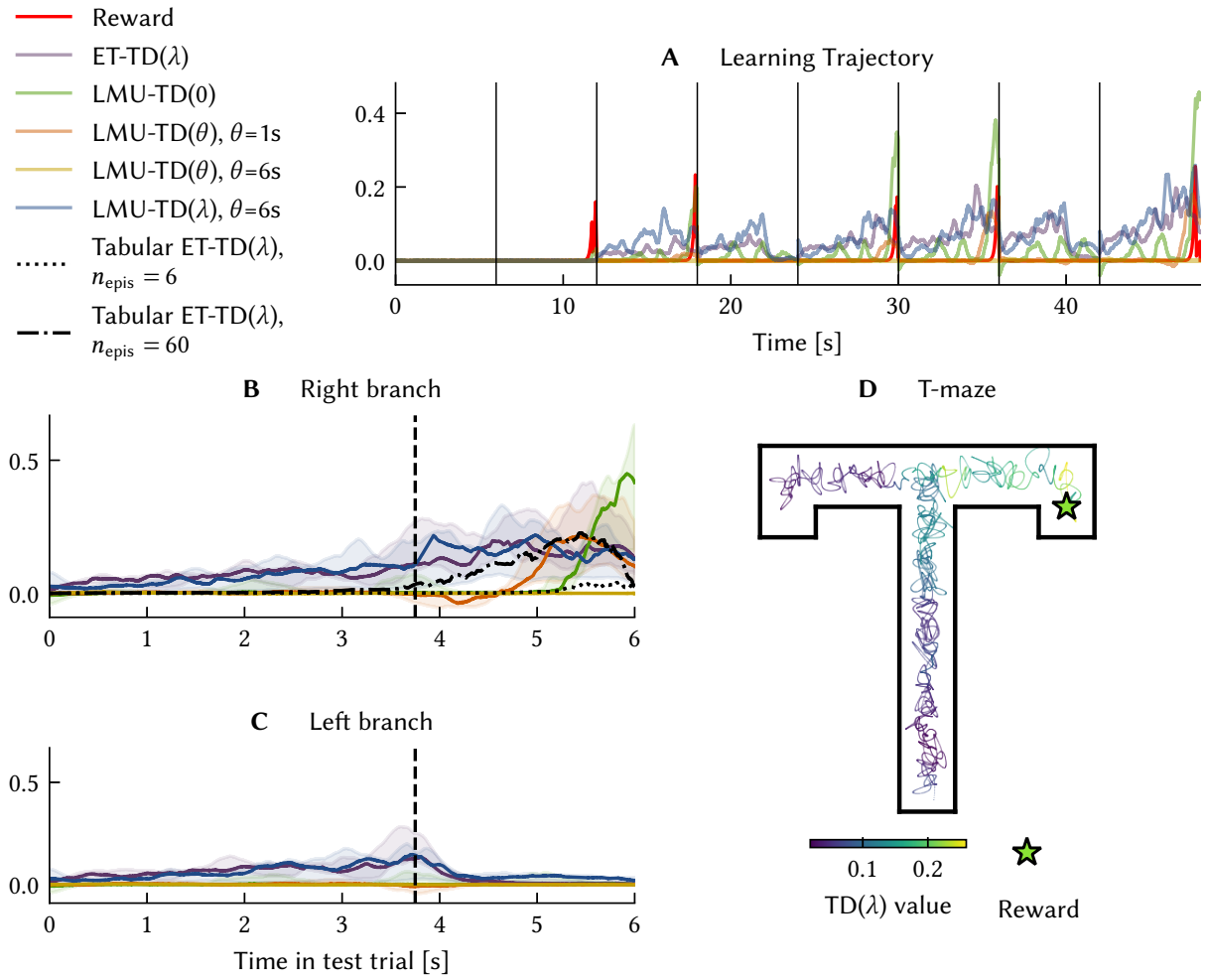



**Figure 5.8: Value functions learned with LDN-TD rules in an SNN for a T-maze navigation task.** **(A)** The reward signal observed over a whole training sequence, with episodes delineated via the vertical lines. The output of the value network trained online with different LDN-TD rules is also plotted. **(B-C)** Value networks were trained five times over 6 episodes – each with a different seed determining the sequence of branches the agent follows and the tuning curve parameters in the network. The median value output during two final test episodes is plotted along 95% confidence intervals. Also included in these figures is the output from a tabular ET-TD($\lambda$) algorithm trained over 6 and 60 episodes. **(D)** The value output when trained with LDN-TD($\lambda$) over the T-maze environment.

LDN-TD($\theta$) performs well with a memory window, $\theta$, of 1 second, but fails with a window of 6 seconds. This is because the value estimate at the goal location is only updated *theta* seconds after the reward is received, and the waiting period at the end of each episode is only 1 second long. When $\theta = 1$s, the value estimate at the goal is updated by the end of the waiting period. While with $\theta = 6$s, the wait period is insufficient and so value estimates at the end of the branches are never updated. In contrast, LDN-TD($\lambda$) updates all estimates in memory, allowing it to quickly learn the correct value function shape, even with a theta of 6 seconds. In

fact, TD($\lambda$) can learn a function of the correct shape after observing each branch only once.

Also included in Figure 5.8 are results using (non-tabular) ET-TD($\lambda$). This is an online TD($\lambda$) algorithm used to train the SNN value function. It is similar to the LDN-TD(0) rule but, unlike in the TD(0) example, an order $q = 1$ state activity LDN is used with a long time window, as described at the end of Section 5.3.1. This algorithm also quickly learns the correct value function. Both ET-TD($\lambda$) and LDN-TD($\lambda$) with a long enough $\theta$ (an approximation of online truncated $\lambda$-return) will produce the same results if learning is offline. In an online learning context, such as this example, there will be differences in the learning dynamics of these two approaches. In this case, these differences do not produce different final results.

In summary, the T-maze experiment demonstrates that learning rules based on LDNs can be used to learn value functions. LDN rules leverage explicit, windowed memory representations, offering more precise and flexible updates compared to the decaying trace mechanism in ET-TD($\lambda$). Among the LDN-based rules, LDN-TD($\lambda$) stands out as the most robust, effectively propagating reward information and adapting to long memory windows. In contrast, LDN-TD($\theta$) is sensitive to the memory window size due to misaligned update timing. Overall, LDN-TD($\lambda$) emerges as the most reliable and efficient approach, combining the strengths of explicit memory with the ability to learn effectively from sparse and online observations.

**Trace conditioning**. In this experiment, a passive agent observes a state signal and a temporally delayed reward, learning a value function based on a state population with memory. Each 5-second trial involves a tone sounding at a random time, followed by a reward 1.5 seconds later. This setup models a trace conditioning task, where both the tone and reward are represented as temporally extended, differentiable events (narrow Gaussian bumps) rather than instantaneous spikes. The state used to learn the value function is the memory state of an LDN over the auditory signal. Importantly, the time window of the LDN memory representing state, $\theta_{\text{state}}$, is independent of the time window used to compute TD errors.

The results, shown in Figure 5.9, demonstrate that all rules partially learn to anticipate the reward before it begins. However, LDN-TD($\lambda$) performs best, with the value estimate increasing immediately after the tone and peaking as the reward begins. In contrast, LDN-TD(0) shows delayed propagation, with the value increasing closer to the reward event. These trends mirror those observed in the T-maze experiment. However, this experiment demonstrates that LDNs can also be used to represent state in tasks that require one to learn temporal dependencies and delays, as well as for computing TD errors.

### *5.3.5 Discussion*

We have introduced a set of new learning rules – LDN-TD(0), LDN-TD($\theta$), and LDN-TD($\lambda$) – for training continuous time value functions in SNNs using LDNs and biologically plausible, local learning rules. A type of three-factor Hebbian rule, PES, is used to train our network, with the memory-dependent errors and activity required for TD updates linearly decoded from LDNs.

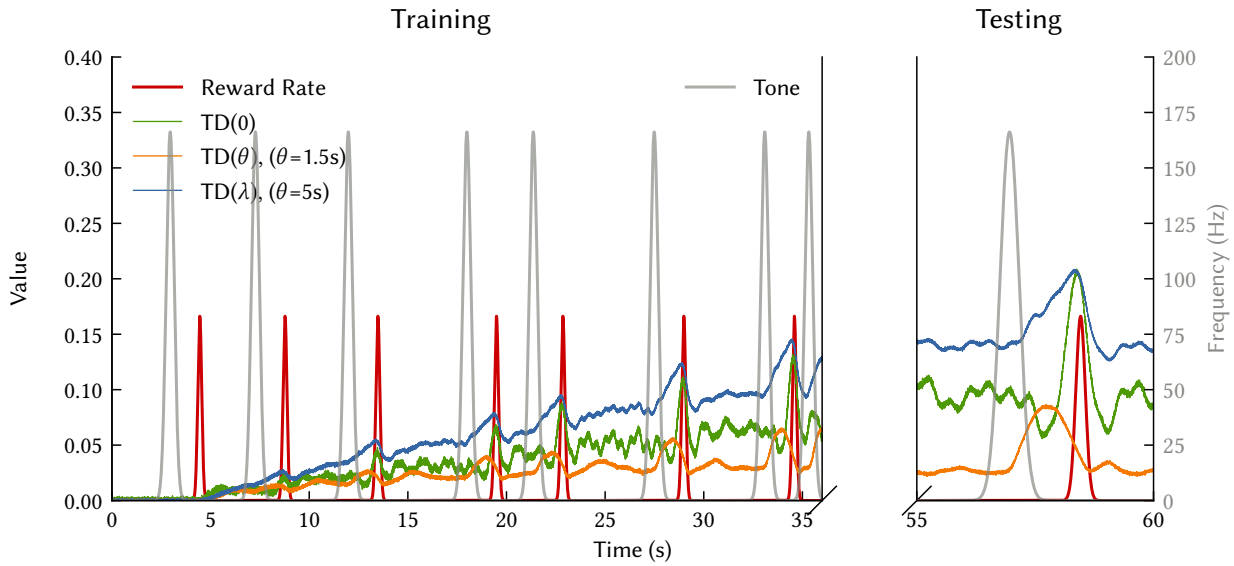



**Figure 5.9: Value population output during testing and training for the trace conditioning task.** The tone is plotted in grey and the reward, always received 1.5 seconds after the tone, is shown in red. The other lines are the value function estimates obtained with different LDN-TD learning rules.

In this framework, the LDN models short-term episodic memory, storing information over time for error signal computation. However, this memory is noisy and constrained by the biological limitations of the recurrent spiking network, which mirrors the imperfections of biological memory. Indeed, LDN memory effects have been compared to human memory performance (Jong et al., 2024).

Our results show that these LDN-TD critic models can learn qualitatively accurate value functions in simple environments. Notably, using LDN-TD($\lambda$), the learning process can occur rapidly – sometimes after just one or two examples. Although our experiments were conducted in simple environments, the primary focus of this work is not on optimizing performance but on exploring biologically realistic RL mechanisms that make use of new tools and techniques. Biological systems operate under constraints, like noisy representations and imperfect memory storage, and designing systems that respect these constraints may reveal insights into how RL operates in the brain.

It is important to note that the errors in LDN-TD(0) and LDN-TD($\theta$) are not novel. TD(0) has been used in continuous-time RL (Doya, 2000), and TD($\theta$) is a natural extension of TD($n$) to continuous time. However, these methods have not previously been implemented using a neural memory model or by projecting value and reward signals onto a temporal basis. Prior work in continuous-time RL has used either instantaneous reward information and Euler differentiation to approximate value derivatives (Doya, 2000; Frémaux et al., 2013), eligibility traces, or numerical integration by discretizing time with fixed (Baird, 1994; Tallec et al., 2019) or adaptive timesteps (Zhou et al., 2021).

In contrast, we do not discretize the value integral over time. Instead, we project the value function and reward signals onto a Legendre polynomial basis using the LDN, effectively shifting from the time domain to the frequency domain. The integral approximation comes from truncating the temporal basis, rather than from time discretization.

In contrast to the other LDN-TD rules, LDN-TD($\lambda$) is novel both in the form of the error computation and its implementation within an SNN using LDNs. This method is conceptually similar to TD($\lambda$) computed via eligibility traces (ET-TD($\lambda$)), which has been previously implemented in SNNs (Seung, 2003; Yuan et al., 2019; Vasilaki et al., 2009). Eligibility traces have historically been used to capture temporal correlations without memory replay buffers. For methods like TD(0), simple delays or twin value networks (Rasmussen et al., 2017) have been used, but learning longer-term correlations requires a more flexible memory system.

Similar to ET-TD($\lambda$) approach, we use a type of activity trace to compute updates. Eligibility traces provide a minimal resource solution, storing a trace of the activity of $N$ neurons in a vector of size $N$ (equivalent to a order $q = 1$ LDN), whereas an LDN memory of such activity is vector of size $qN$. This higher dimensional trace maintains higher frequency information about a signal's history, allowing for more accurate decoding of delays and replay. Additionally, eligibility traces store information at a fixed temporal resolution and discount rate, while LDN-TD($\lambda$) allows for the computation of different TD errors with varying discount rates and $\lambda$-weighting functions from the same memory populations.

Recurrent memory units are commonly used in RL models, both spiking (Karamanis et al., 2018) and non-spiking (Bakker, 2001; Zambrano et al., 2015; Rivest et al., 2014), but they are typically used to handle working memory tasks (*e.g.*, the trace conditioning task) and partially observable environments, not for error computation. Episodic memory and replay mechanisms have been explored in RL (Pritzel et al., 2017; Zhang et al., 2021), but these models often use external buffers, not biologically plausible memory systems.

The use of the LDN to compute value integrals and derivatives is reminiscent of spectral methods, which numerically solve differential equations by transforming them into the frequency domain using Fourier, Chebyshev, or Legendre bases. The time derivatives of the variables are approximated using differentiation matrices based on the polynomial basis – similar to how we approximate the derivative of the value function with a linear transform applied to its Legendre representation (the LDN memory vector). Spectral and pseudospectral methods have been used for learning state transition dynamics (Azizzadenesheli et al., 2016; Ren et al., 2022) and solving optimal control problems (Mehrali-Varjani et al., 2018; Dolgov et al., 2021). Though these approaches to control are more akin to learning the coefficients of the decomposition of the value function rather than only using the basis projection to compute updates to the value function.

Our approach has limitations, but offers many avenues for future research. While we propose a biologically plausible mechanism for learning predictive value associations, we have not yet addressed action selection – a topic for Section 5.5. Moreover, real biological systems

often combine both model-free and model-based strategies for learning, and this conjunctive use of systems is not covered here. We will explore a hybrid approach – successor features – in the following section.

Memory processes in humans and animals are far more complex than the models presented here. Replay and sleep consolidation are thought to be crucial for learning, with random sampling of past experiences helping break temporal correlations and prevent catastrophic forgetting. modelling these aspects would require longer-term memory systems than the LDN currently provides. While LDNs and other state-space models have proven effective in long time series tasks, they are still limited in the amount of information they can store, especially when noise in representation and dynamics, as seen in spiking implementations, is considered. To address these challenges, it may be necessary to store memory not just in instantaneous neural activity but in synaptic weights. Future work might explore developing a memory network that learns to associate states with LDN memory vectors. Relevant episodic memories could later be recalled and replayed. Such extensions will be essential for applying our methods to more complex problems and bridging the gap between biological plausibility and computational efficiency.

Alternative temporal bases could also be explored. While we use the Legendre polynomial basis, a Fourier basis could make the system more like the SSP trajectory memory model discussed in Chapter 3. Alternatively, choosing a basis orthogonal to the weighting function in Eq. (5.69) could simplify the TD update, though this would not generalize to other discount functions or rates.

Lastly, using past activity in weight updates opens new research possibilities. We use learning rules like PES as it uses only local information, but by storing neural activity history in an external memory system, neurons could potentially receive information on approximate past activity and activity gradients of neighbouring neurons, allowing for more complex learning schemes. Although this would likely be resource-intensive and scale poorly, this direction may be worth investigating, especially given recent work on SNN training methods using eligibility traces (Bellec et al., 2020).

## 5.4 Temporal Difference Learning of Distributions

The methods discussed so far in this chapter are not limited to learning value functions; they are broadly applicable to any function that can be learned using a TD algorithm. This includes learning representations of future state occupancies – *i.e.*, successor representations (SRs) or, in continuous state spaces, successor features (SFs); see Section 5.2.1 – and even distributions over future reward. Importantly, when learning such functions, the decoder update equations we have introduced remain unchanged, except the scalar reward signal is replaced by a vector signal.

Working with SFs presents challenges, particularly due to their sensitivity to the choice of

state features and the difficulties in learning state features concurrently with the successor function. Ideally, the features should be generalizable and transferable, rather than being specialized to a particular problem. Here we use SSPs as features in navigation tasks, as they provide a general and effective representation of continuous variables. As well, the probabilistic interpretation of SSP bundles allows us to view the successor features as representations of distributions over future locations in the state space. We discuss extending this approach to reward distribution learning, by approximating the future reward distribution with SSP bundles that may be learned with LDN-TD rules.

### *5.4.1 Successor features*

Recall from Section 5.2.1 that the successor feature function computes the expected discounted accumulation of features under some policy $\pi$. In continuous-time, it is defined as

$$\boldsymbol{M}(s) = \mathbb{E}_{\pi}\left[\int_{k=0}^{\infty} \gamma^k \phi(s(t+k))dk \;\middle|\; s(t) = s\right], \tag{5.70}$$

where $\phi(s(t+k))$ are the *feature* representations of space – we use SSP encodings as our features, assuming $\mathcal{S} \subseteq \mathbb{R}^n$. Using SSPs as features has several advantages. First, this is a neurally grounded feature representation of space that does not have to be learned concurrently with SFs. Additionally, these features could, in principle, be combined with other representations using VSA operations to represent more complex state spaces. Furthermore, the resulting SFs has a useful probabilistic interpretation (Furlong et al., 2022; Furlong et al., 2023).

The SF function can be learned in a similar manner as the value function or Q-function. The optimal SF function satisfies a Bellman-like equation,

$$\boldsymbol{M}(s(t-\theta)) = \int_{k=0}^{\theta} \gamma^k \phi(s(t+k-\theta))dk + \gamma^{\theta}\boldsymbol{M}(s(t)). \tag{5.71}$$

Thus, we can define TD(0), TD($\theta$), and TD($\lambda$) updates for the successor function. For example, the TD($\theta$) error is given by

$$\delta_{\text{SF}}^{(\theta)}(t) = \int_{k=0}^{\theta} \gamma^k \phi(s(t+k-\theta))dk + \gamma^{\theta}\boldsymbol{M}(s(t)) - \boldsymbol{M}(s(t-\theta)). \tag{5.72}$$

This error can be computed in the same fashion as the LDN-TD($\theta$) error we defined for value functions in Eq. (5.42). The same is true for the LDN-TD(0) and LDN-TD($\lambda$) rules. The only difference when training the SF function is that the LDN memory of reward is replaced by a memory of features, and the TD error signal is vector-valued instead of a scalar (see Figure 5.10).

To obtain a value function with the SFs, we also need to learn a linear model of reward rate, $\hat{r}(s) = \mathbf{w}_r(g) \cdot \phi(s)$, so that we can compute $V(s(t)) \approx \mathbf{w}_r(g) \cdot \boldsymbol{M}(s(t))$. This network learns a mapping from a context or goal signal $g$ to a population encoding the reward vector $\mathbf{w}_r(g)$ – this means multiple reward functions corresponding to different contexts or goals can be

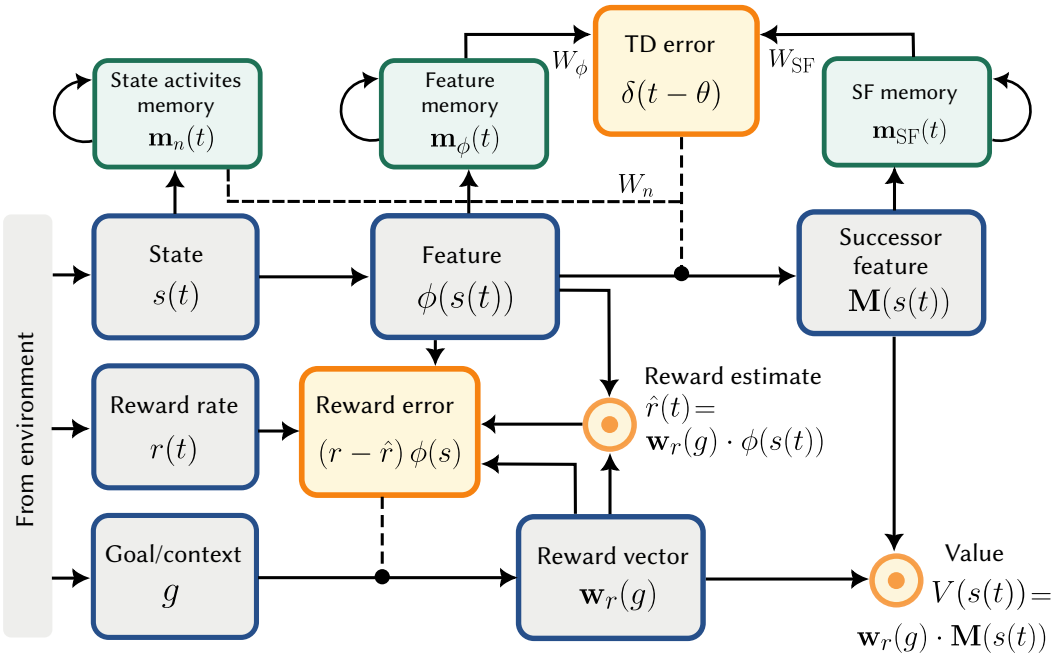



**Figure 5.10:** Diagram of a network that learns the successor feature function using LDNs and a linear reward function, from which value can be computed. Blue blocks denote neural populations, green blocks denote LDNs, and the orange block is an error signal. The concentric circles icon indicates a product network. Solid lines indicate neural projections and dashed lines indicate a modulatory effect on a set of synapses.

learned simultaneously, as done in Ma et al. (2020). For environments without multiple goals or context conditions, the signal $g$ simply remains constant at an arbitrary value. The decoders on the connections from the context to reward vector populations are updated using the standard PES learning rule with the following error signal:

$$\text{Reward error at } t = [r(t) - \mathbf{w}_r(g)\cdot\phi(s(t))]\,\phi(s(t)), \tag{5.73}$$

$$= [r(t) - \hat{r}(s(t))]\,\phi(s(t)). \tag{5.74}$$

Next, we test this network's ability to learn in two example scenarios: a hairpin maze, to show the general ability of the network to learn; and a two room maze, to demonstrate its ability to learn context sensitivity.

**Hairpin maze.** We test the LDN-TD($\lambda$) update given in Eq. (5.69) for SF learning using SSP features. In this hairpin maze task, the agent repeatedly travels though the maze, starting at the bottom left and ending at the top left, where reward is received. The agent's 20 second path is constructed by adding noise to a pre-computed, fixed path though the maze and smoothing with spline interpolation. The added noise creates small variations between episodes (Figure 5.11A). The context/goal signal is constant over the whole experiment. The features are the SSP

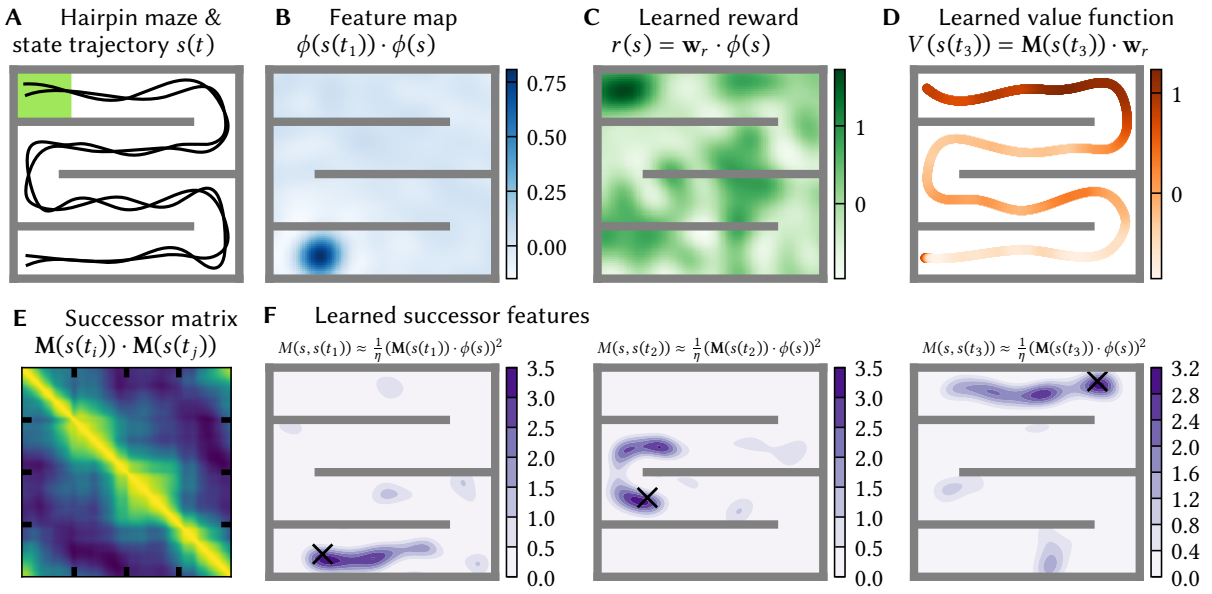



**Figure 5.11: Successor features in a hairpin maze trained with LDN-TD($\lambda$). (A)** The hairpin maze environment and the agent's trajectory $s(t)$ in different episodes. A goal is in the upper left corner of the maze. **(B)** A similarly map of the SSP feature encoding of the state at time $t_1$. **(C)** The learned linear reward function. **(D)** The value function obtained via the final SF and reward functions. **(E)** The matrix $\frac{\boldsymbol{M}(s(t_i)) \cdot \boldsymbol{M}(s(t_j))}{\|\boldsymbol{M}(s(t_i))\| \|\boldsymbol{M}(s(t_j))\|}$ – the cosine similarity of successor features across the test trajectory. The axis ticks mark the turning points in the maze. **(F)** Examples of the SF at different states. These plots are obtained by computing the similarity between the given SF $\boldsymbol{M}(s(t_i))$ and the features of points gridded over space $\phi(s)$ and then converting the similarities into probabilities via squaring.

encodings of the agent's location in the maze (Figure 5.11B). The population encoding the features has SSPs representing points scattered over the maze,[12] *i.e.*, place cells.

Only two training episodes are used and the results shown in Figure 5.11C-F are from a final test episode. In Figure 5.11C, the final reward function is plotted. In this case the local reward function is simply an indicator function at the end of the maze, so it is easy to approximate it with a linear function on the SSP features – essentially the learned $\boldsymbol{w}_r$ is proportional to the SSP representation of the goal location.

Examples of the final successor features at different time points along the trajectory, $\boldsymbol{M}(s(t_j))$, are shown in Figure 5.11F – visualized by their similarity to the features, $k(s) = \boldsymbol{M}(s(t_j)) \cdot \phi(s)$, mapped to a probability distribution over the maze. We see that, as expected, the similarity plots of the successor features show a trial of high similarity ahead of the current position.

The value function computed from the SF and reward function during the final test trajec-

[12] In particular, the points are given by a Sobol sequence, a type of quasi-random sequences used to cover the space fairly evenly.

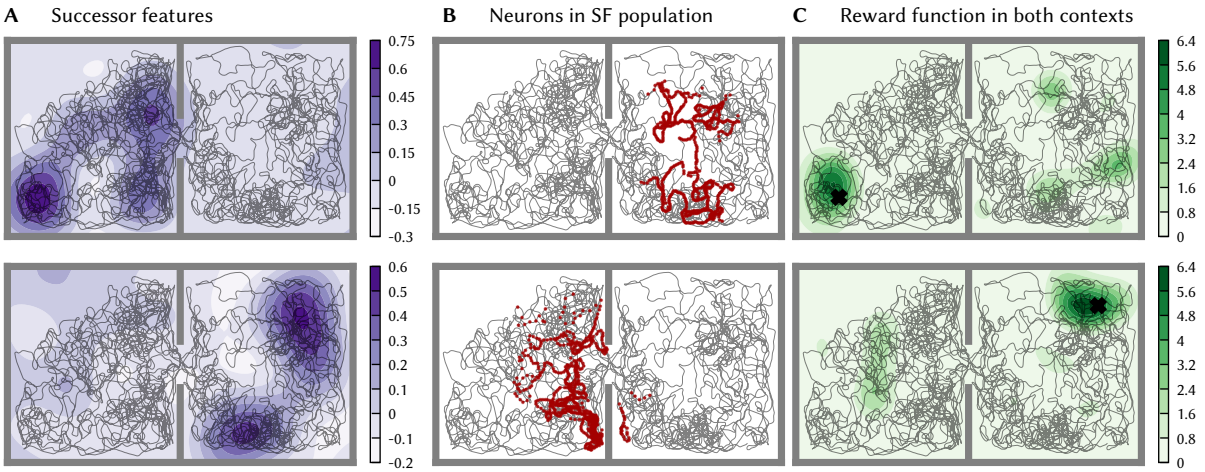



**Figure 5.12: Successor features in two connected rooms trained with LDN-TD($\lambda$).** The solid black line shows the path taken through the environment. **(A)** The similarity between example SFs $\boldsymbol{M}(s(t_i))$ and the features of points gridded over space $\phi(s)$. **(B)** Firing locations of two example neurons in the population encoding SF. **(C)** The learned linear reward function in the two contexts (left and right rooms). The black 'x' marks the goal locations.

tory is plotted in Figure 5.11D and is qualitatively correct, with the highest values just before the goal location and the value decaying further back in the maze. Note that we also plot the matrix of similarity between SFs along the final test trajectory (Figure 5.11E). We can see that it has a diagonal structure since, as expected, SFs closer in time and closer along the path will have overlap as a result are more similar. The blocking structure corresponds to the turning points in the maze (marked with ticks in the figure). This experiment clearly shows the ability of the SF network to solve an RL task using online updates with continuous-time and space representations.

**Two room maze**. We test the same model on an environment consisting of two joined rooms – an environment used in George et al. (2023). The path through the maze is a directed random walk, generated in the same fashion as the trajectory in George et al. (2023)[13]. The context signal was not constant in this experiment – instead it was a scalar signal indicating which of the two rooms the agent was in. Each room had a particular goal location where the agent received reward (Figure 5.12C). We can see that reasonable SFs were learned in this environment (Figure 5.12A) and that the reward function can be learned for both contexts (Figure 5.12C).

[13]It is a random walk with momentum and bias towards room crossings and edges – this is meant to mimic natural foraging paths of rodents.

### 5.4.2 Reward distributions

In RL, understanding not only the expected return but also the distribution of possible outcomes can improve decision-making, particularly for risk-sensitive tasks. Here, we present an approach to represent reward distributions using SSPs. Unlike traditional distributional RL, our method focuses on the expected distribution of future rewards rather than the full distribution of returns.

Recall from Section 2.4.4 that we may represent distributions via a bundle of SSPs encoding data samples. Define $\boldsymbol{R}(s)$ as

$$\boldsymbol{R}(s) = \mathbb{E}\left[\int_{k=0}^{\infty} \gamma^k \phi(r(t+k))dk \;\middle|\; s(t) = s\right], \tag{5.75}$$

where $\phi(r(t+k))$ is an SSP representing the reward rate at time $t+k$, and $\boldsymbol{R}(s)$ is a finite-dimensional feature mean embedding. This is closely related to successor features, with $\boldsymbol{R}(s)$ representing the successor features of reward rather than state. We can approximate the density of the distribution of future reward, given that we start in state $s$, with

$$\mathbb{P}(R = r \mid s) \approx \frac{1}{\xi}(\boldsymbol{R}(s) \cdot \phi(r))^2, \tag{5.76}$$

where $\xi$ is a normalization constant. Note that other mappings from similarity to density can be used (Section 2.4.4), but here we use squaring and normalization to obtain a density from the similarity measure.

Like the successor feature function, $\boldsymbol{R}(s)$ follows a Bellman-like equation and so we can learn it via a TD learning rule. For example, the TD($\theta$) error is given by

$$\delta_{\mathrm{R}}^{(\theta)}(t) = \int_{k=0}^{\theta} \gamma^k \phi(r(t+k-\theta))dk + \gamma^{\theta}\boldsymbol{R}(r(t)) - \boldsymbol{R}(r(t-\theta)). \tag{5.77}$$

While the sprit of this – learning a distribution of future outcomes rather than the expected return – is similar to that of distributional RL, this approach is quite different. Distributional RL extends traditional RL by modelling the full distribution of returns, law($Z(s)$), where the return $Z(s)$, a random variable, is defined as:

$$Z(s) = \int_{k=0}^{\infty} \gamma^k r(t+k)dk, \quad \text{where } s(t) = s. \tag{5.78}$$

The goal in distributional RL is to approximate law($Z(s)$) and, by extension, capture the variability and uncertainty in outcomes. In practice, the infinite-dimensional return distribution is approximated with a finite dimensional representation – a parameterization of the return distribution or a set of finite statistics or statistical functionals. For instance, categorical distributional RL parameterizes the return distribution as a mixture of Dirac delta functions at fixed points, effectively binning the returns (Bellemare et al., 2017). Another approach, quantile

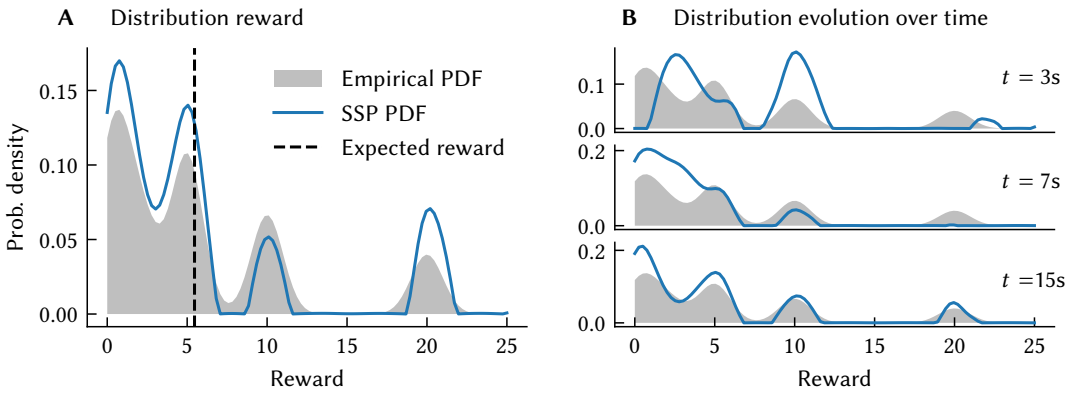



**Figure 5.13: Learning a distributional reward embedding with LDN-TD(0).** The grey area is a KDE computed from the reward data, while the solid line is densities obtained from similarity to SSPs. **(A)** Density computed with the kernel of the reward distribution embedding at the end of the simulation, $1/\xi(\boldsymbol{Z} \cdot \phi(r))^2$. The dashed vertical line shows the expected value of the rewards – we see that this does not capture nearly as much information as the full distribution. **(B)** Reward distribution density at different points in time over the simulation.

regression distributional RL, represents the distribution as a mixture of Diracs at learnable points (Dabney et al., 2018).[14] Other work has approximated the return density with mixtures of Gaussians (Choi et al., 2019) or splines (Luo et al., 2021).

Our proposed approach diverges from distributional RL by learning an approximate distribution over future rewards rather than returns. This difference allows us to re-use the architecture and methods from Section 5.4.1. We can learn $\boldsymbol{R}(s)$ via TD updates directly in the feature space, making it suitable for biologically plausible implementations.

*Variable-magnitude reward.* To demonstrate the application of Eq. (5.75), we learn the distribution over reward in the variable-magnitude task from Dabney et al. (2020). Each trial lasts 0.5 seconds and during each trial a random reward is received – either (0.1, 0.3, 1.2, 2.5, 5, 10, 20). In this experiment, state is held constant. The distributional embedding learned is encoded in the activity of a population that receives input from the constant state population and the weights of those projections are learned with LDN-TD(0). As shown in Figure 5.13, the SSPs are capable of learning the distribution of rewards, whereas an expected reward calculation loses much of the information in the distribution (Figure 5.13B).

### *5.4.3 Discussion*

In this section, we have described how the LDN-TD learning rules, previously presented in the context of value function learning, can be applied to learning successor representation and reward distributions. These paradigms for RL have several advantages over traditional

[14] Quantile regression distributional RL has been implemented in an SNN in Sun et al. (2023), which is, to the best of our knowledge, one of the only spiking distributional RL model.

approaches and have connections to how the brain might represent and process information, making them valuable to study and model.

*Successor features with SSPs.* In the hairpin maze experiment, we showed that an SNN could successfully learn SFs using recurrent memory units. RL agents equipped with SFs are capable of latent learning, linear value evaluation (assuming a linear reward function over features), and the ability to estimate value under a previously unobserved reward function. In goal-reaching problems, for example, when the agent's goal state, $s_{\text{goal}}$, has a unique SSP feature representation $\phi_{\text{goal}}$, the reward function can be approximated as $r(s) = \phi(s) \cdot \phi_{\text{goal}}$. Given an SF function trained under policy $\pi$ (*e.g.*, an exploratory policy) and a known goal location, one can immediately compute the expected value under $\pi$ without additional training. Previous research has leveraged a single SF across multiple reward functions for transfer and multi-task learning (Ma et al., 2020; Mozifian et al., 2022).

SRs bear some resemblance to the coding schemes used by place cells in the hippocampus. Past models have made these connections with the discrete SR (Gershman, 2018; Fang et al., 2023) and have even modelled SRs with spiking networks and biologically plausible plasticity rules (Bono et al., 2023). George et al. (2023) presented a spiking models of SFs with place cells providing the feature basis, much like our model, and demonstrated skewing effects in place cells, resembling results from rodent experiments. Our model is unique in its use of neurosymbolic features and suggests interesting areas of future development. For example, the resulting SFs have a probabilistic interpretation, which we discuss next and could be explored more in future work. In addition, temporal SSPs may also be incorporated in features and SFs in such a fashion that allows disambiguation of visitation probability and temporal distance.

In 5.4.1 we demonstrated how to learn the SF function $\boldsymbol{M}(s)$, which represents the expected sum of future abstract features. However, this abstraction comes at the cost of losing the explicit occupancy information of successor representations, as the focus shifts from the state distribution itself to the accumulation of feature representations. The successor measure $\mathscr{M} : \mathcal{S} \to \mathcal{P}(\mathcal{S})$ is an alternative formulation, which maintains the notion of state occupancy and distribution. Rather than accumulating abstract features, it directly measures the discounted probability of the agent visiting states in a set $S \in \mathcal{S}$ in the future. This keeps track of the actual distribution of states.

$$\mathscr{M}^{\pi}(S|s) = -\frac{1}{\log(\gamma)} \int_{k=0}^{\infty} \gamma^k \mathbb{P}\left(s(t+k) \in S | s(t) = s\right) dk, \quad S \subset \mathcal{S}, \tag{5.79}$$

where $S \subseteq \mathcal{S}$ is a measurable set. Note that the successor feature function computed with SSPs is a weighted bundle of samples and thus approximates the distribution of those samples. Thus, we have $\mathscr{M}(s, S) \approx {}^{1}/_{\xi}(\boldsymbol{M}(s) \cdot \phi(S))^2$. This is another benefit of the SSP features – it supports a probabilistic interpretation.

Furthermore, while we have used pure SSP features, we could incorporate other information via binding. For example, we could represent mixed discrete-continuous states or incorporate temporal information. Recall the SSP trajectory model presented in Section 3.5 in Eq. (3.33)

and Eq. (3.34) earlier in the thesis. We can define

$$\boldsymbol{M}^{\pi}(s(t)) = \mathbb{E}_{\pi}\left[\int_{k=0}^{\infty} \gamma^{k}\phi(s(t+k),k)dk \;\middle|\; s(t)=s\right]. \tag{5.80}$$

This variant of the successor feature function uses features that contain both spatial and temporal information, thus disentangling visitation probability and temporal distance. This can be learned in the following way: let the feature at time $t$ be $\phi(s(t),t) = \phi_{\text{state}}(s(t)) \circledast \phi_{\text{time}}(t)$, where $\phi_{\text{state}}$ and $\phi_{\text{time}}$ are the SSP maps for state and time and will have different parameterizations. Then the SF can be updated by

$$\boldsymbol{M}(s(t-\theta)) = \phi_{\text{time}}(-t) \circledast \int_{k=0}^{\theta} \gamma^{k}\phi(s(t+k-\theta),t+k-\theta)dk + \gamma^{\theta}\phi_{\text{time}}(\theta) \circledast \boldsymbol{M}(s(t)). \tag{5.81}$$

Such an SF will be the prospective version of the retrospective trajectory memory presented in Section 3.5. Exploring such extensions to the model is an avenue for future research.

*Reward distributions with SSPs.* We have sketched out how the SF model might be used for learning representations of reward distributions in Section 5.4.2. In this view, both SFs and reward distributions are predictive functions that can be learned in the same fashion – they just operate over different features. Our proposed approach approximates the distribution of future rewards (with an immediacy bias) with a high-dimensional feature vector. The Bellman updates reduces the L2 distance between distribution embeddings. This has some similarities to distributional RL methods that use moment matching (Nguyen-Tang et al., 2021) or mean Random Fourier Feature (RFF) embeddings (Wenliang et al., 2024). In these methods, finite statistics are used to represent the return distribution and the Bellman update results in a reduction in the maximum mean discrepancy (MMD). MMD is distance measure between two probability distributions obtained by evaluating the distance between their mean embeddings in a reproducing kernel Hilbert space (RKHS). For two distributions $p$ and $q$, their kernel mean embeddings are defined as:

$$\mu_p = \mathbb{E}_{x\sim p}[k(x,\cdot)],\ \mu_q = \mathbb{E}_{x\sim Q}[k(x,\cdot)] \tag{5.82}$$

where $k(\cdot,\cdot)$ is a positive definite kernel function, and $\mathbb{E}_{x\sim p}[k(x,\cdot)]$ is the expectation of the kernel function with respect to the distribution $P$. The MMD between the distributions $p$ and $q$ is then defined as the distance in the RKHS between these mean embeddings:

$$\text{MMD}(p,q) = \|\mu_p - \mu_q\|_{\mathcal{H}} \tag{5.83}$$

The MMD satisfies the properties of a distance function if the kernel is characteristic (*i.e.*, the mean embedding of any distribution is unique). The MMD can also be empirically estimated – this estimate was used in Nguyen-Tang et al. (2021).

Our proposed method represents the distribution of future reward – not returns – in a finite dimensional RKHS with a non-characteristic kernel (like the kernel mean embedding (KME)

method of Wenliang et al. (2024)) and so the L2 distance between SSP distribution embeddings is not a proper distribution distance function – but it does estimate MMD. The form of features we use, SSPs, are essentially RFFs equipped with an algebra. Thus, like RFF, the error scaling of the MMD estimate decays as $\mathcal{O}(1/\sqrt{d})$, where $d$ is the SSP dimensionality (Sutherland et al., 2015).

SSPs provide a compact and biologically plausible encoding mechanism, enabling RL algorithms to move beyond learning only expected returns (value functions) to capturing richer reward distributions that include uncertainty. Representing $\boldsymbol{R}(s)$, the distribution of potential instantaneous rewards, offers advantages over a value function by providing an agent with information about possible outcomes. This can aid in avoiding actions leading to negative future rewards or assessing the achievability of specific rewards from a given state, which is particularly useful in risk-sensitive or uncertainty-driven decision-making. Additionally, evidence suggests that animals encode such information (Lowet et al., 2020; Muller et al., 2024), aligning this approach with biological plausibility. Future work should consider how this information might be used in risk-aware planning and could extend this approach to learn return distributions via SSP bundles, rather than just reward distributions.

## 5.5 Action selection

So far we have presented methods for obtaining value functions. But, to fully model RL in continuous-time environments with SNNs, we must consider how a biological system might represent, learn, and sample a policy. Here we present a model that uses SSP bundles to represent policy distributions, which can be learned with policy gradient methods. Additionally, we describe how Langevin dynamics, realized by a recurrent network, can sample such distribution embeddings.

### *5.5.1 Learning policies*

VSAs, in particular Holographic Reduced Representations (HRRs), provide methods for representing various actions as high-dimensional vectors. Discrete actions can be represented by random high-dimensional vectors $\phi_j$, continuous vector actions can be encoded by SSPs, $\phi(\boldsymbol{a}) \in \mathbb{R}^d$, and other action spaces (mixed discrete and continuous, trajectory, trajectories of multiple agents) can be represented using the algebra of HRRs (Plate, 1995). In this section, we focus primarily on continuous action spaces.

Using SSPs to encode continuous actions, we require methods for representing distributions over actions and the ability to sample distributions. However, we have already discussed how distributions can be encoded by SSP bundles (see Section 2.4.4). A probability distribution can be represented as a single vector, $\mu \in \mathbb{R}^d$. This vector can be constructed by taking the mean embedding – the normalized bundle over samples from the distribution, $\boldsymbol{x}_i$, encoded with an

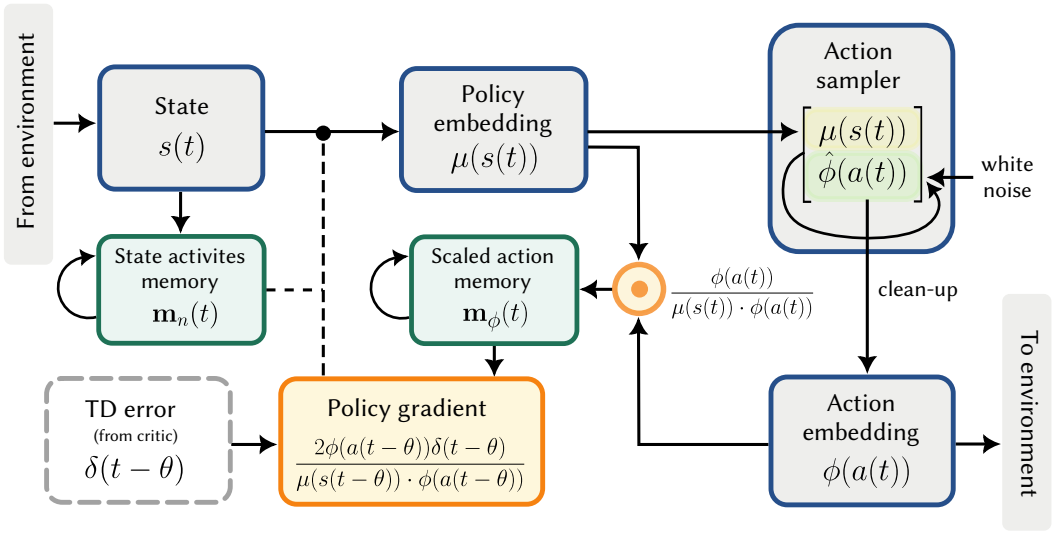



**Figure 5.14:** Diagram of actor network that uses LDN memories for computing the policy gradient given in Eq. (5.90) and a noisy neural for action sampling as described in Section 5.5.2.

SSP:

$$\mu = \frac{1}{n}\sum_{i=1}^{n}\phi(\boldsymbol{x}_i), \quad \Pr(\boldsymbol{x}) = \frac{1}{\xi}\left(\mu \cdot \phi(\boldsymbol{x})\right)^2 \tag{5.84}$$

where $\xi$ is a normalizing constant, set so that $\int_{\boldsymbol{x}\in\mathfrak{X}} \Pr(\boldsymbol{x})d\boldsymbol{x} = 1$. The probability of data $\boldsymbol{x}$ is proportional to the square of the similarity of the data's representation in an SSP to the mean vector $\mu$. Thus, the vector $\mu$ is a representation of a probability distribution. It can approximate some true distribution if we are given samples, or it can be learned. Additionally, conditioning and marginalization can be performed via algebraic manipulations on $\mu$ using VSA operations (Furlong et al., 2023).

In the case of RL, we would like to learn a representation of a distribution over actions that depends on an environment's state – which we call the *policy embedding*. We use $\mu(s) \in \mathbb{R}^d$ to refer to the vector embedding of the policy and $\pi(a \mid s)$ to refer to the probability of selecting action $a$ at state $s$ under the policy:

$$\pi(a \mid s) = \frac{1}{\xi(s)}\left(\mu(s) \cdot \phi(a)\right)^2. \tag{5.85}$$

A neural population that receives input from a state population can encode the policy embedding $\mu(s)$. Thus, $\mu(s(t)) = D\boldsymbol{a}(t)$ where $\boldsymbol{a}(t) \in \mathbb{R}^N$ is the vector of filtered neural activities of the state population and $D(t)$ is the matrix of decoders on the synaptic weights between the state and policy embedding populations at time $t$.

Consider the policy gradient Eq. (5.9), which in this case is given by:

$$\nabla J(D(t)) = \mathbb{E}_\pi\left[\nabla_{D(t)} \log \pi(a(t) \mid s(t))\delta(t)\right], \tag{5.86}$$

$$= \mathbb{E}_\pi \left[ \frac{2\phi(a(t))\boldsymbol{a}(t)^T}{\mu(s(t))^T \phi(a(t))} \delta \right], \tag{5.87}$$

$$\Delta D(t) = \eta \nabla J(D(t)). \tag{5.88}$$

where $\phi(a(t))$ is the SSP encoding of action $a(t)$, $\mu(s(t))$ is the policy embedding, $\boldsymbol{a}(t)$ is the vector of filtered activities of the state population, and $\delta(t)$ is the TD error signal for value.

This is a problem with this approach: the policy gradient of Eq. (5.9) is derived for discrete-time settings. In continuous-time, the variance of the likelihood ratio estimate of the return gradient grows without bound (Munos, 2006), making this policy learning method problematic. There are alternative approaches, such as the pathwise derivative approach in which the gradient of the return is computed by differentiating the trajectory of the system dynamics with respect to the policy parameters (Munos, 2006; Ainsworth et al., 2021). These methods can be computationally expensive and it is unclear how they might be translated to an SNN implementation. For our purposes, we employ the likelihood ratio estimate, but note that this component of the resulting AC model may have poor performance.

The policy gradient update in Eq. (5.88) can be performed using the PES learning rule:

$$\Delta d_i(t) = \eta \underbrace{\left[ \frac{2\phi(a(t))}{\mu(s(t))^T \phi(a(t))} \delta(t) \right]}_{\text{error signal}} \underbrace{\boldsymbol{a}_i(t)}_{\text{pre- activity}}. \tag{5.89}$$

In Eq. (5.89), we are modifying decoders in the direction of the SSP representation of a taken action weighted by the advantage of the action and the inverse of its similarity to the policy embedding. For example, if an action has a low similarity to the policy embedding (low density in the policy) but is then followed by high return, the policy embedding will be updated in the direction of that action. If it already had high similarity to the embedding, then the update will be smaller. If instead an action is followed by negative return, the policy embedding will be updated to move away from the action.

To learn a policy online, we need to take a backwards view and update past policy embeddings using current information via LDNs. To update the policy embedding $\theta$ seconds in the past using the TD($\theta$) error $\delta^{(\theta)}(t-\theta)$ in place of the term $\delta(t)$, we modify decoders by

$$\Delta \boldsymbol{d}_i(t) = \eta \boldsymbol{P}^{q_\phi}(1) \boldsymbol{m}_\phi(t) \delta^{(\theta)}(t-\theta) \boldsymbol{P}^{q_n}(1) m_n(t) \tag{5.90}$$

where $m_\phi(t)$ is the LDN memory of SSP action encodings $\phi(a)$ scaled by the inverse of similarity to the policy embedding, and $m_{n_i}(t)$ is the LDN memory of the state population's $i^{\text{th}}$ neuron activity.

A diagram of a model that learns a map from state to an SSP policy embedding that is updated via Eq. (5.89) and LDN memories is given in Figure 5.14. We show examples of using this network in Section 5.5.3.

### 5.5.2 Sampling policies

Once a policy embedding is learned (or while it is being learned), we need to actually use it to select actions in a neural network, balancing the need for both exploration and exploitation. Given a policy embedding, we can sample the encoded distribution with Langevin dynamics (Furlong et al., 2024). Langevin dynamics simulates the time evolution of a system that follows a stochastic differential equation consisting of deterministic forces and stochastic noise and can be used as a tool for sampling. Langevin dynamics can be viewed as a continuous-time analogue of Monte Carlo sampling, as, over time, the trajectories generated by Langevin dynamics will sample the same distribution as certain Monte Carlo algorithms (*e.g.*, the Metropolis-Hastings algorithm).

Let us define a 'sampler' population. This layer takes the policy embedding $\mu(s(t))$ (represented by a population of spiking neurons) as input and outputs a vector $\phi(t)$, which should be an (approximate) SSP encoding of an action sampled from the policy. The sampler population is recurrently connected to realize Langevin dynamics: the deterministic force is the gradient of the log probability of the policy (driving the output trajectory $\phi(t)$ towards a local maximum) and the stochastic noise is given by white noise (for exploration of the action space). That is,

$$\dot{\phi}(t) = \alpha \nabla_{\phi} \log \pi(s(t), a(t)) + \nu, \tag{5.91}$$

$$= \frac{2\eta\mu(s(t))}{\phi(t) \cdot \mu(s(t))} + \nu, \tag{5.92}$$

where $\alpha$ is a rate parameter that determines the speed at which the dynamics follow the policy gradient, and $\nu \sim \mathcal{N}(0, {}^{\sigma^2}/_{d}I)$ is noise that introduces stochasticity to the sampling dynamics. A variance scale factor of $\frac{1}{d}$ reflects that the variance of the dot product between randomly generated SSPs has a variance of ${}^{1}/_{d}$ (Gosmann et al., 2016; Voelker et al., 2017). By increasing $\sigma$, we can introduce more random exploration in the policy. Using the dynamics principle of the NEF, the recurrent weights on the sampler population can be set to realize the dynamics in Eq. (5.92). The action output is a continuous signal decoded from the sampler output., *i.e.*, $a(t) = \text{argmax}_a \phi(t) \cdot \phi(a)$.

*Gaussian mixture example.* An example of the samples obtained from a fixed distribution using a recurrent spiking population is given in Figure 5.15. This is a demonstration of the sampling abilities of this recurrent network, and not an example application of these dynamics in an RL context. A Gaussian mixture model with two modes is sampled to create a KME approximated by a normalized sum of SSPs, $\mu$. The recurrent population weights are optimized with least-squares to approximate Eq. (5.92) with a fixed $\mu$. The population is simulated for four seconds and the output of the spiking sampler population is decoded and plotted in Figure 5.15A. Also shown are the empirical CDF obtained from the samples and distance measures between the empirical distribution given by the original samples and the empirical distribution given by the SNN SSP sampler output. Hyper-parameters, including the SSP length scale, the rate $\alpha$, and noise variance $\sigma$ in the Langevin dynamics, are optimized to minimize

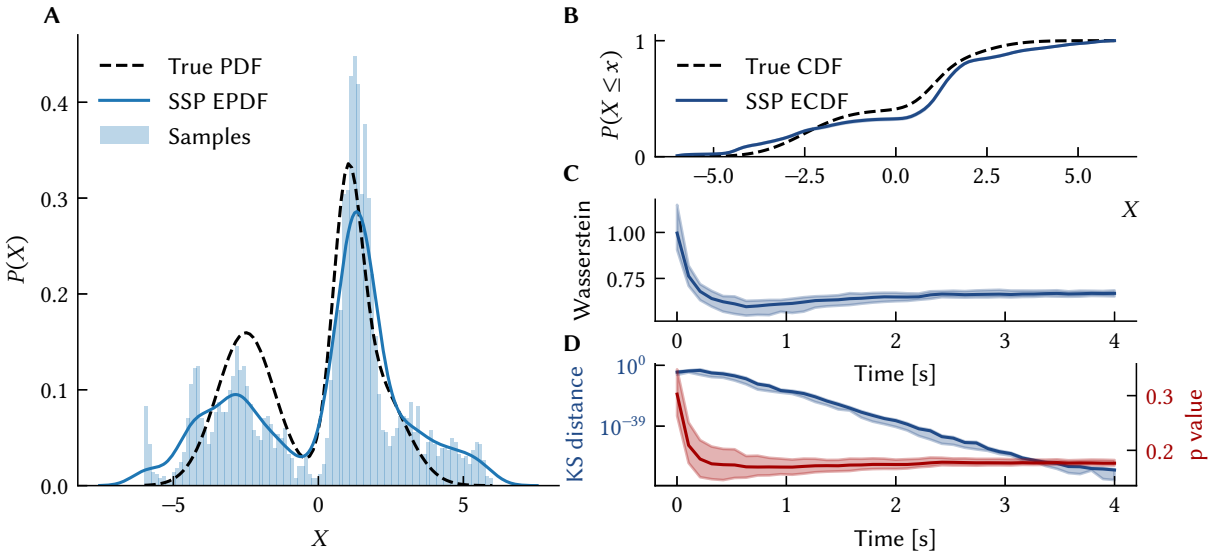



**Figure 5.15:** Output of the Langevin dynamics model of SSP bundle sampling. **(A)** A histogram of the output of the SNN SSP sampler, along with the PDF of the true distribution and a kernel density estimation (KDE) obtained from the model's samples. Note that the model dynamics are determined by the distribution embedding $\mu$, which was constructed with 2000 samples from the true distribution – we call these the original samples. **(B)** The empirical CDF, obtained via the sampler output, versus the true CDF. **(C)** The Wasserstein-1 distance between the empirical distributions of original samples and model samples obtained over the course of the simulation. It's final value is fairly small, approximately 5% of the total range of the data. **(D)** The statistic (KS distance) and p-value the two-sample KS test. The null hypothesis is that the two empirical distributions are identical. The statistic is the maximum absolute difference between the empirical distributions. At the end of the simulation, the KS statistic is small and p-value is greater than 0.05 – which means we cannot reject the null hypothesis.

the final Kolmogorov–Smirnov (KS) distance. The results in Figure 5.15C-D show that the model samples capture the true distribution well.

When using Eq. (5.92) to sample from a policy, the dynamics will change as they depend non-linearly on $\mu(s(t))$, which will change with state and over the course of learning. This means that the sampler population must represent the vector $[\phi(t), \mu(s(t))]^T$. It receives the current policy embedding $\mu(s(t))$ and a filtered white noise as input. Recurrent connection weights are optimized to approximate the transform Eq. (5.92) as a function over both $\phi(t)$ and $\mu(s(t))$. The state of the sampler is decoded from the neural activities at every timestep. The SSP action samples are, in turn, decoded from this output.

Note that the network shown in Figure 5.14 includes this sampler. The sampler is a key part of the complete SSP actor model, which we now test on two example applications: a bandit problem, showing the basic usage of Eq. (5.89) and Eq. (5.92) to learn a spiking policy in a discrete action space without the use of memory units; and a maze task, showing the complete spiking AC model that trains the SSP actor with LDNs and a value network, trained with one

of the LDN-TD rules from Section 5.3.

### 5.5.3 Examples

*Bandit problem.* We first test the spiking actor model on a 10-arm bandit task, where each action is associated with a stochastic reward. For action $j$, the reward is $r \sim \mathcal{N}(m_j, \sigma_j^2)$, with randomly selected means and variances (Figure 5.16A). This reward signal is used for policy learning (as the $\delta(t)$ term in Eq. (5.89)), so neither memory nor a value network is needed.

Since the action space is discrete, random high-dimensional semantic pointers represent actions rather than SSPs. The state population encodes a constant scalar, and the policy gradient maps this to the policy embedding population. Langevin dynamics govern the sampler population, whose output is decoded back into the action space. Figure 5.16F shows the similarity of the sampler population's output to each action SSP over time, while Figure 5.16E plots the policy embedding population's similarity to each action, converted into probabilities.

Initially, both populations' outputs are dissimilar to any action embeddings, causing the sampler to select random actions. The neural dynamics and synaptic smoothing slow down rapid switching between actions. Over time, the probability assigned to actions with higher rewards increases. When action 8 (the action with the highest expected reward) is chosen, its probability mass rises significantly. While the noisy sampler continues to select other actions for a short while, the gradient log probability term in Eq. (5.92) eventually drives the sampler output toward action 8, which becomes the dominant choice by the end of the simulation, as expected (Figure 5.16C).

*Water maze.* In this experiment, we demonstrate the full AC SNN applied to a simple water maze task. The environment is a square arena with obstacles and a goal located in the bottom-right corner (Figure 5.17A). The value function is learned using the LDN-TD(0) rule, while the policy is represented as a policy embedding – a single vector (interpreted as an SSP bundle) – that is learned through the policy gradient rule in Eq. (5.88). The policy embedding guides the recurrent sampler network with stochastic dynamics, which samples the agent's actions.

Results show that the SNN agent successfully learns to navigate to the goal, progressively earning more reward over time (Figure 5.17C). Initially, the stochastic dynamics of the sampler lead to random exploration, allowing the agent to discover the goal through a random walk. As the policy embedding is learned, it influences the sampler's dynamics, improving action selection and enabling more efficient navigation. The final learned policy, shown in Figure 5.17B, generally points toward the goal, demonstrating that the actor-critic SNN approach can effectively learn both value functions and goal-directed policies in a dynamic environment.

### 5.5.4 Discussion

In this section, we introduced a spiking RL model, adding an actor network to the critic network discussed in Section 5.3. We demonstrated that policy distributions can be represented using

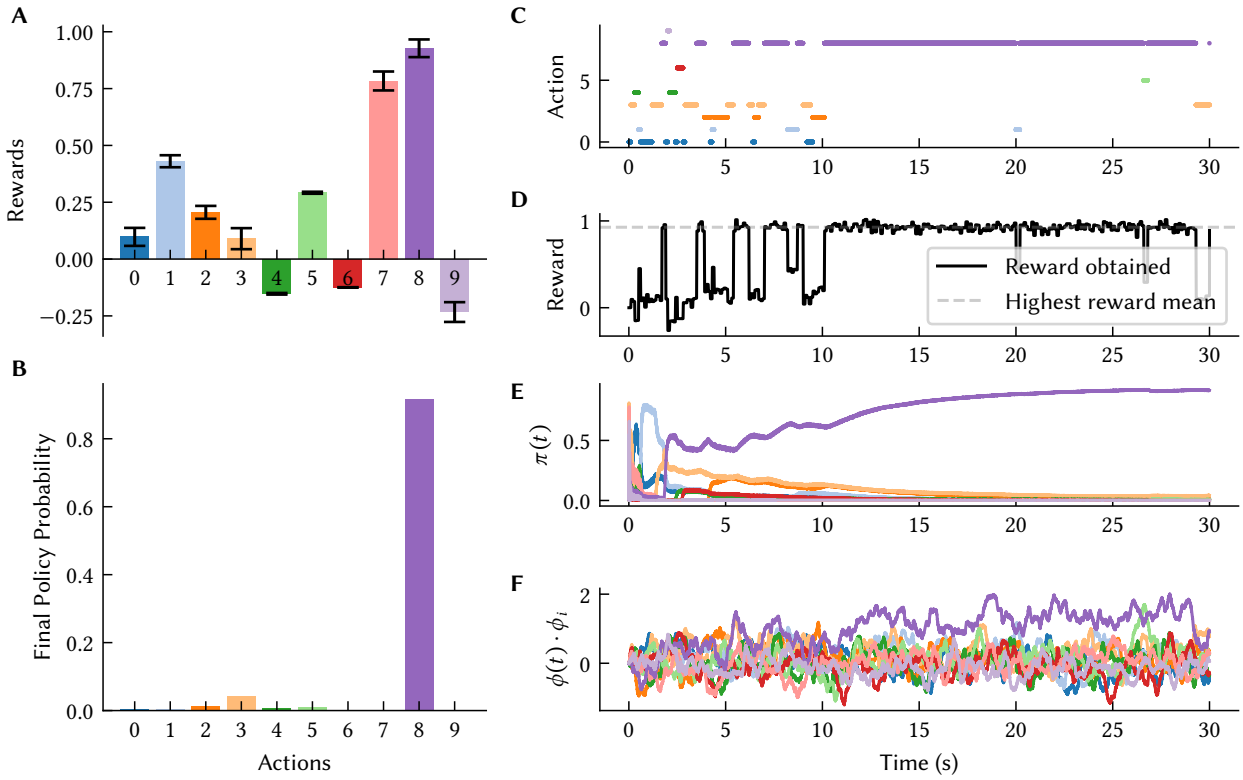



**Figure 5.16: A 10-arm bandit problem. (A)** The means rewards and standard derivations associated with each action. **(B)** The probability assigned to each action by the policy at the end of the simulation. **(C)** The actions selected by the sampler over time. **(D)** The reward received over time. The dashed line marks the highest expected reward. **(E)** The values of $\pi(a)$ for each action. **(F)** The similarity of the sampler output to all action SSP encodings.

SSPs, and policy gradients can be used to learn such policy embeddings. We also showed that a recurrent spiking network can sample from these embeddings. Our key contributions are: 1) representing policies and learning policy gradients in SSP spaces; and 2) using a spiking model of Langevin dynamics for action selection over policy embeddings in continuous and discrete action spaces.

Previous methods for representing probability distributions in neural systems have used probabilistic population codes, where neuron activity reflects the probability of that a random variable (some input to the network) takes on certain values (Ma et al., 2014; Barber et al., 2003). Some models include biologically plausible sampling through linear decoding (Grabska-Barwinska et al., 2013; Hennequin et al., 2014; Savin et al., 2014). Unlike those local population codes, SSPs use high-dimensional distributed representations. SSPs and operations on SSPs have a probabilistic interpretation and thus offer a way for neural circuits to compute under uncertainty (Furlong et al., 2022).

We also demonstrated that SSPs support sampling in a biologically plausible SNN, which

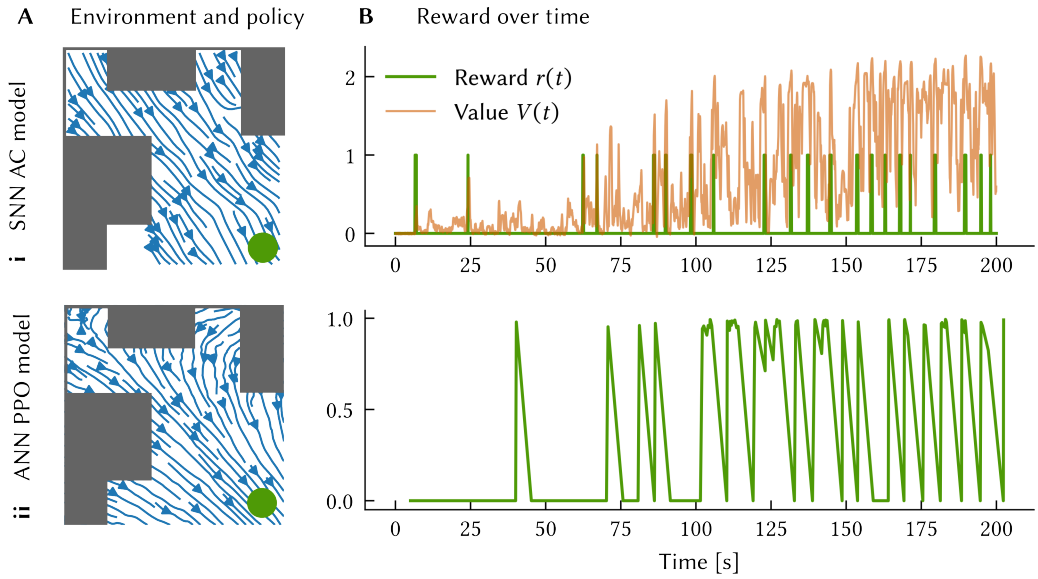



**Figure 5.17: A water maze task. (i)** The SNN AC model (Figure 5.14), with the value function learned via Eq. (5.36) and the SSP policy embedding learned via Eq. (5.88). **(ii)** A baseline ANN model that uses Proximal Policy Optimization (PPO) to learn a policy. **(A)** The environment, goal location (in green), and the final policy (visualized by computing the most likely action given $\boldsymbol{\mu}(s)$ over space, plotted as a stream plot). **(B)** The reward received (and the value estimate of the SNN model) over simulation run time.

can be used to support decision-making and control. Currently, many neural models of decision making, including those that model the basal ganglia, operate over finite, discrete choices with winner-take-all networks, where options compete through evidence accumulation (Frémaux et al., 2013; Zambrano et al., 2015; Yan et al., 2019; Stewart et al., 2010a). Existing spiking models of decision making in continuous action spaces use neural activity as a deterministic control signal or as the parameters of a distribution sampled externally (Tang et al., 2021).

In contrast, our model uses a non-parametric policy gradient approach, with Langevin dynamics implemented in a recurrent spiking network for stochastic sampling. Langevin dynamics has been used in deep RL, but in a different use case – typically in a stochastic gradient descent algorithm applied policy gradient and Q-learning (Kamalaruban et al., 2020; Hsu et al., 2024).Furthermore, Langevin dynamics has been implemented in recurrent neural networks (Grabska-Barwinska et al., 2013; Hennequin et al., 2014; Savin et al., 2014). Our work extends previous work on Langevin dynamics in the SSP space, (Furlong et al., 2024), by using time-varying policy embeddings rather than fixed distributions and realizing the dynamics with an SNN.

One limitation of our model is that the exploration-exploitation trade-off is controlled by the noise versus gradient terms in the Langevin dynamics. While Langevin dynamics has been proposed as a method for improving exploration in RL, relying solely on random

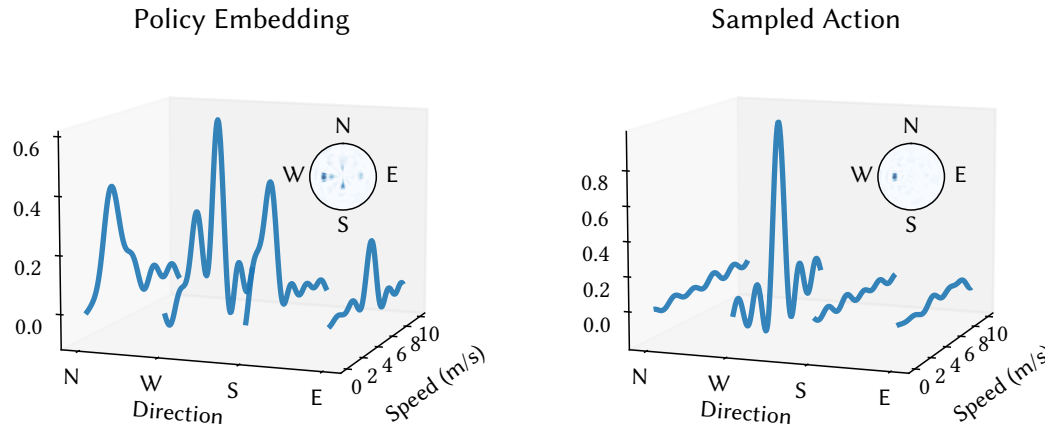



**Figure 5.18: HRR supports representations of mixed discrete and continuous action spaces.** For example, one can encode a continuous speed in a discrete direction by binding an SSP encoding speed and Semantic Pointer (SP) – a random vector embedding – encoding a cardinal direction. Pictured here is the similarity of such an action embedding (right) and the similarity of a policy embedding in this space (left). Similarity is visualized by both a set of line plots, and as a heat map in the inset plots.

noise might not be ideal. Future work could incorporate principles such as information gain or curiosity-driven exploration. For example, Furlong et al. (2021) used SSPs in a Bayesian optimization algorithm for efficient black-box optimization. Mutual information was used to encourage exploration in the sample space with high uncertainty. A similar approach could be incorporated into our RL model to guide exploration more intelligently. Integrating BO-SSP with our RL agent could lead to a more principled exploration-exploitation strategy within a biologically plausible actor-critic SNN framework.

The most significant limitation of the model proposed here is the performance of the policy learning method in continuous-time. We use the likelihood ratio method to estimate the policy gradient; however, as noted in Section 5.5, when using this method in continuous time, the number of decisions grows infinitely, leading to a "variance explosion" Munos (2006). This method is ill-suited to continuous-time settings, which we are primarily interested in. Perhaps this method might be better suited to selecting temporally extended actions at a coarse time-scale, rather than fine-grain control. Indeed, decision-making over more complex action spaces, such as trajectories, mixed continuous-discrete spaces, and multi-agent trajectories, should be explored in future work. Such actions can be represented using the HRR algebra (Figure 5.18) and so learning and sampling policies over these spaces is theoretically possible in our model.

In addition, other methods for computing the policy embedding error signal that reduce variance and improve stability should be explored in future work. This could involve learning a model of state transitions or estimating the policy gradient by computing the state gradient along observed trajectories, without explicitly learning a model of state transitions.

## 5.6 Conclusion

In this chapter, we developed models for biologically plausible RL using SNNs and local learning rules. A key contribution is the LDN-TD rules, which use the LDN as an episodic memory over observations, estimates, and neural activity. The LDN projects inputs onto temporal basis functions, enabling a vector-valued memory trace. We demonstrated that these rules support learning of value functions and SFs in SNNs using local, three-factor Hebbian-like updates. Additionally, we represented SFs and distributions over rewards and actions with SSPs and used Langevin dynamics to sample actions from noisy SSP policy embeddings.

Biologically plausible RL models are important because they bridge the gap between neuroscience and AI by grounding learning computations in neural principles. Our work advances this field by proposing TD error computations with vector memory traces, developing neurally grounded distribution encoding, and demonstrating local learning of complex representations in SNNs. These contributions take steps toward understanding how the brain may process RL tasks and integrating biological insights into AI systems. However, significant gaps remain, particularly in the integration of long-term memory and model-based learning, which are important components of learning in brains.

As well, there are several limitations to the models proposed here. Memory and error computations are noisy and imperfect, limiting the precision of learned value functions and SFs. Moreover, we only trained shallow, single-layer function approximators, which is sufficient for simpler problems, but inadequate for more complex tasks. Another practical challenge is the large number of neurons required to represent memories and policy populations, which, when implemented using leaky integrate-and-fire (LIF) neurons, leads to slow and expensive simulations. Running these models on neuromorphic hardware could significantly improve simulation speed and energy efficiency, but this remains an area for future exploration.

Although these biologically plausible RL models hold potential value for AI, they are still far from being directly applicable to modern AI systems due to these limitations. However, there are clear avenues for improvement that could bring these models closer to practical use and closer to comprehensive models of learning in the brain. Refining the models, by incorporating deeper networks, and exploring alternative learning mechanisms, such as predictive coding or feedback alignment, could allow for learning multilayered functions with biologically plausible feedback signals. Extending the model's memory capabilities with long-term memory stored in synaptic weights, for instance by storing LDN memory vectors in an associative network, and replaying memory traces could enable learning over longer time horizons. Additionally, integrating model-based learning with cognitive maps – highly relevant to modelling the hippocampal formation – could further enhance the framework.

Notably, while this chapter hints at the potential of VSA operations for constructing more complex and structured state and action representations, this idea was not a primary focus. Similarly, although we use SSP state embeddings, we did not compare them to alternative approaches or explore them in non-spatial contexts. These limitations motivate the next chapter,

where we shift focus toward deep RL models. By leveraging larger networks, we explore how neural and cognitively inspired state representations might improve the effectiveness of RL systems on a variety of tasks. This transition marks a shift from the biologically constrained, model-free methods discussed here to more scalable deep learning approaches with non-spiking networks.

# Chapter 6

# Neurosymbolic Representations for Reinforcement Learning

> In virtually all domains of cognition – from vision and speech to reasoning and planning – empirical investigation over millennia has consistently shown how cognitive functions can be well-approximated as computation over appropriate compositional structures
>
> — Paul Smolensky et al.
> *Neurocompositional computing: From the Central Paradox of Cognition to a new generation of AI systems*

**Summary**

This chapter examines the application of SSPs and VSA operations to construct state embeddings for RL networks. We find that SSPs embeddings of non-spatial continuous variables can be used in control tasks, but the benefits of this approach emerge in navigation settings. Leveraging the compositional operations of VSAs, we embed spatial and object-related knowledge into a single, composite vector, providing an effective state representation. This approach significantly improves sample efficiency and final returns compared to tested alternative representations in navigation tasks. Our findings highlight the utility of structured, compositional representations, inspired by neuroscience and cognitive science, in RL.

**Material in this chapter from prior publications**

All content – text and experiments – in this chapter are original work. However, related work which used HexSSPs as a state encoding in shallow RL networks was presented in Bartlett et al. (2023b), at the *21th International Conference on Cognitive modelling*. This work inspired the experiments presented here. The author of this thesis is a co-author of Bartlett et al. (2023b).

## 6.1 Introduction

Successful reinforcement learning (RL) agents learn feature representations from raw observations and simultaneously map these to value functions, policies, and predictive models. The quality of these feature representations is critical to agent performance, as they are the foundation upon which decisions are made. Useful features capture the essential structure of the environment and are robust to noise. They must balance expressiveness and simplicity, ensuring meaningful similarity and separability in state embeddings for accurate learning. However, the representations learned by deep RL systems are often highly task-specific and lack generalizability – in stark contrast to those seen in biological systems.

For example, when navigating a new environment or performing a new task in a familiar environment, a rat does not need to relearn a spatial representation from scratch. Neurons in the medial entorhinal cortex (MEC) quickly adapt, exhibiting grid cell firing patterns within minutes of exposure to a new environment (Hafting et al., 2005). These grid patterns remain remarkably consistent across a wide variety of environments (Hafting et al., 2005; Fyhn et al., 2007),[1] demonstrating a level of neural feature representation versatility that is not commonly observed in artificial systems. The grid cell representation of space may thus provide a general purpose feature basis useful for navigation and RL.

Vector Symbolic Algebras (VSAs), and, more specifically, Spatial Semantic Pointers (SSPs), provide a hypothesis about high-level neural representations in the brain. We suggest that SSPs may serve as effective features or a basis for features in RL for several reasons. First, SSPs capture the structure of Euclidean space and can be endowed with grid cell-like structure. Hence, they may be a good general purpose representation of space – and perhaps continuous variables more generally – in RL tasks. Indeed, using SSP input generally improves the performance of the networks on various machine learning benchmarks, even on non-spatial tasks (Komer, 2020). Second, within the framework of a VSA, SSPs facilitate integration of structured, symbolic and sub-symbolic data. This capability aligns with the requirements of navigation tasks that involve symbolic reasoning and interaction with multiple objects or agents. These properties motivate our research into the utility of SSPs specifically, and Holographic Reduced Representations (HRRs) more generally, as features in RL.

In this chapter, we explore neuro-cognitive feature representations in RL, specifically SSP and HRR state encodings. We pose several questions:

- Can grid-cell-inspired state representations enhance RL algorithm performance, including in non-spatial tasks?
- Can RL systems learn polices over the structured, semi-symbolic representations of a VSA? Do these representations offer benefits over alternatives?

[1] Though there are some small variations in grid patterns in different environments and tasks. In particulate, grid patterns can distort according to environment geometry (Krupic et al., 2014) and their scale can change in response to environment novelty (Barry et al., 2012).

To address these questions, we test SSP and HRR state encoding strategies across benchmark control problems and navigation tasks, aiming to determine if emulating biological and cognitive frameworks through these representations can substantively benefit RL applications. Our findings suggest that such integrations do provide advantages in navigation tasks, laying the groundwork for further research into the utility of VSAs in RL systems.

## 6.2 Related prior work

Previous research has explored biologically inspired features and neurosymbolic techniques within the field of RL. In this section, we review some of this work.

### *6.2.1 Grid cells in RL*

Research in RL has successfully integrated grid-cell-inspired representations in navigation agents. For example, Banino et al. (2018) embedded a grid cell network trained for path integration into an RL system. This system used the output of grid cells combined with an encoding of goal location as input to a policy network. They found that the grid cell RL agent outperformed a place cell and baseline Advantage Actor-Critic (A2C) model on a navigation task. Furthermore, the grid cell RL agent demonstrated robustness to choice of hyperparameters and showcased the ability to generalize to new environments and modified environments. These findings substantiated the authors' claim that grid cell-like representations endow RL agents with a Euclidean spatial metric, which aids agents in navigation tasks.

Dang et al., 2021 also used a grid-cell-code in an A2C RL algorithm for goal-directed navigation in a grid world. They used a Fourier basis to store projections of current and goal locations, which, bound together, provided a representation of the vector to the goal. This representation was then concatenated with a small field of view processed by a CNN and provided as input to the A2C model. The study found that the agent leveraging grid cell coding achieved faster convergence and demonstrated improved generalization compared to an agent using place cell representations.

Extending the application of grid cells from spatial to non-spatial domains, Yu et al. (2021) noted that grid cell patterns are superpositions of Fourier modes and, crucially, that combinations of Fourier modes are the eigenvectors of state transition matrices in environments with a translation-invariant transition structure – such as navigation in an open environment. Based on this insight, they proposed GridCell-DQN, an RL system that integrates Fourier modes into a Deep Q-Network (DQN). GridCell-DQN contained two sub-networks: one which used unmodified state and action observations as input, and another that operated on Fourier modes over a discretized state and action space. A final layer computed action values from the concatenated sub-networks' output. They tested GridCell-DQN on the Cart Pole task, and reported improved performance compared to a vanilla DQN.

Random Fourier Features (RFFs) are a well-established method for projecting data onto a high-dimensional Fourier basis. This approach shares many characteristics with the methods discussed above, except that it does not involve an explicit parameterization to enforce a grid-cell-like code. RFF and Fourier features have been used as a feature pre-processing step in RL – in value linear models (Konidaris et al., 2011), shallow policy networks (Rajeswaran et al., 2017), deep Q models (Morere et al., 2018; Brellmann et al., 2023), and deep successor models Yu et al. (2024). Li et al. (2021) proposed an architecture that used RFFs concatenated with state observations as the first layer of a Q or policy network. Unlike previous work, they learn the parameters of the RFF projection. They report improved sample efficiency in an array of control tasks, which is attributed to robustness to noise during training.

SSPs use a mapping similar to RFFs, with the primary differences being the grid-cell-inspired Fourier basis (*i.e.*, HexSSPs) and the use of the HRR algebra to compose representations. HexSSP representations of state have been used in a goal-conditioned RL system trained to navigate via curriculum learning (Komer, 2020). Additionally, HexSSPs have been used to train a shallow network on the Cart Pole task (Bartlett et al., 2023b). These prior studies used fixed SSP projections rather than learning the SSP parameters. Furthermore, SSPs have not been tested on any other non-spatial RL tasks, and compositional HRRs have not previously been used in RL.

Although the integration of grid cells into RL models shows promising results, the scope of previous studies has been relatively narrow. Spatial applications have been limited to mazes, while non-spatial applications have been almost exclusively limited to the Cart Pole task. This restricts our understanding of how well these biologically inspired features can generalize across a broader spectrum of RL problems. RFFs have been tested in larger suites of control tasks, but such past work did not explore navigation or the use of grid-cell structure. Even prior research using SSP state encodings did not explore the full utility of these representations. SSPs offer more than just a means of representing continuous variables. They also function within a VSA, equipping these spatial representations with operations for incorporating symbol-like representations in a structure-preserving manner.

### *6.2.2 Neurosymbolic RL*

In recent years, there has been a growing interest in leveraging neurosymbolic modelling techniques in RL to make more interpretable and safe systems. Various strategies have been explored, from using neural networks to extract features for symbolic planning systems (Anderson et al., 2020), and the incorporation of symbolic knowledge (*e.g.*, knowledge graphs) into neural RL agents (Silva et al., 2021; Xiong et al., 2017), to the use of symbolic policies (Landajuela et al., 2021; Trivedi et al., 2021).

Hierarchical reinforcement learning (HRL) shares a close relationship with neurosymbolic methods. Many HRLs algorithms construct 'meta-policies' over semantically meaningful actions (such as 'go to the door' instead of direct motor control signals), *e.g.*, 'options', which

can be either designed manually (Sutton et al., 1999) or learned (Bacon et al., 2017). Nachum et al. (2019) argues that performance improvement in HRL over standard RL is primarily due to improved exploration via semantic actions, as opposed to hierarchical network structure or improved policy learning. Both Jin et al. (2022) and Mitchener et al. (2022) have presented neurosymbolic HRL models that generate symbolic meta-policies over options in various scenarios, including games and animal cognition tests.

In this work, we employ HRRs, a type of VSA, for feature embeddings. VSAs use distributed representations to encode symbols as vectors, making them suitable for manipulation in a way that resembles neural computations. This differs from purely symbolic approaches, which manipulate discrete categories. Typically working with continuous, real-numbered values, VSAs are particularly suited for integration with gradient-based optimization methods used in neural networks

Other neurosymbolic methods, such as capsule networks or graph-based neurosymbolic models, also attempt to bridge the gap between symbolic modelling and neural networks, but do so by augmenting networks with additional hierarchical or graph-based structures. Tensor Product Representations (TPRs) are a different approach that, like VSAs, includes methods for binding symbols using tensor products. However, as more symbols are added, the dimensionality of these tensor products increases, leading to scalability challenges due to the exponentially growing size of the representations. In contrast, VSAs offer an efficient method to incorporate symbolic structures into deep RL models without necessitating structural changes to the underlying networks and without the scaling problems of tensor products.

## 6.3 SSPs for control

The SSP encoding is not limited to encoding spatial variables; it can be applied to any continuous variable, including velocities, angles, and more. This section aims to investigate the effectiveness of using SSPs, and HexSSPs in particular, as a general encoding mechanism for continuous states within the context of control.

### *6.3.1 Methods*

We test SSP representations on standard RL benchmarks from the DeepMind control suite (Tassa et al., 2018). This suite consists of a variety of simulated control tasks, with continuous state and action spaces, built using the MuJoCo physics engine. We primarily focus on tasks in this suite with relatively low-dimensional ($< 20$) state spaces, as we believe that low-dimensional variables are likely to benefit the most from projection into the high-dimensional SSP space. The state vectors, $\boldsymbol{s} \in \mathbb{R}^n$, in the environments tested range in size, from $n = 3$ to 78, and contain a mixture of variables, including positions, velocities, orientations, torques, and accelerations.

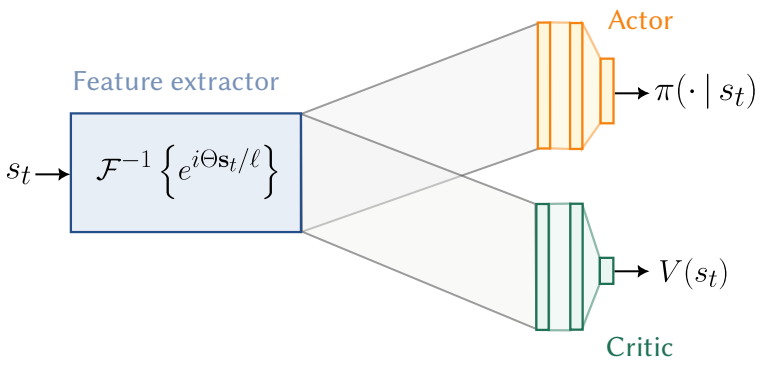



**Figure 6.1:** The actor-critic architecture with an added SSP feature embedding layer.

For all environments, state observations, $\boldsymbol{s} \in \mathbb{R}^n$, are directly encoded as $d$-dimensional SSPs, given by

$$\phi(\boldsymbol{s}) = \mathcal{F}^{-1}\left\{e^{i\Theta \boldsymbol{s}/\ell}\right\}, \tag{6.1}$$

where $\Theta \in \mathbb{R}^{d \times n}$, the phase matrix, and $\ell \in \mathbb{R}^n$, the length scales, are parameters. The SSP mapping is differentiable, allowing us to train these mapping parameters using standard back-propagation. Note that, as discussed in Definition 2.1, the phase matrix must have conjugate symmetry to ensure that $\phi(\boldsymbol{s})$ is real-valued. This means that there are only $n\lfloor d/2 \rfloor$ free parameters in the matrix, and the rest are computed from this subset. In total, there are $n(1 + \lfloor d/2 \rfloor)$ trainable parameters in the SSP encoding. We initialize the phase matrix with the HexSSP scheme given in Algorithm 1.

We compare the performance of models using these embeddings against a baseline model that uses the default state observations, $\boldsymbol{s}$. In addition, we compare our method to the learned RFFs approach of Li et al. (2021), because this is in an existing approach similar to SSPs that, unlike most others mentioned in Section 6.2.1, has been benchmarked on a large number of non-spatial tasks. In Li et al. (2021), states are embedded as

$$\gamma(\boldsymbol{s}) = \begin{bmatrix} \sin(2\pi B \boldsymbol{s}) \\ \cos(2\pi B \boldsymbol{s}) \\ \boldsymbol{s} \end{bmatrix}, \tag{6.2}$$

where $B \in \mathbb{R}^{d/2 \times n}$ is learned (and initialized as $B_{ij} \sim \mathcal{N}(0, \sigma^2)$ with $\sigma = 0.001$), and the sin and cos are applied element-wise. Both SSPs and RFFs project state variables into a Fourier basis; as done by Li et al. (2021), we learn this projection. There are a few differences however, between SSPs and these RFFs. The RFFs stay in the Fourier space, while we convert back to a pseudo-spatial domain via the IDFT, but this difference is just a single linear operation. They also concatenate the state vector to the Fourier embedding, while we consider SSPs with and without such concatenation. Another difference is that we use the structured HexSSP initialization of the projection rather than a random projection. The biggest difference between

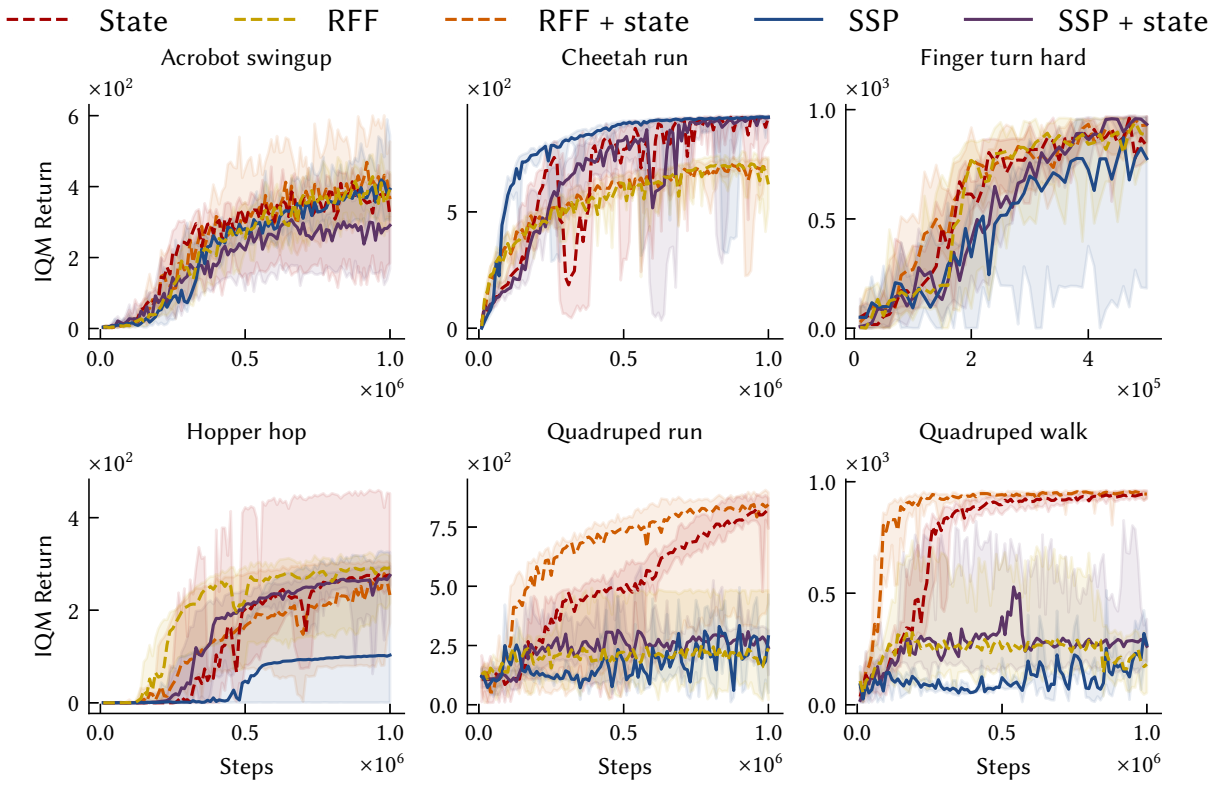



**Figure 6.2: Sample efficiency curves for control benchmarks**. The interquartile mean (IQM) of episodic reward over the course of training is plotted, along with stratified bootstrap 95% confidence intervals, for models with the baseline states, learned RFFs, and SSPs for all control benchmarks.

SSPs and RFFs is that SSPs reside in the HRR algebra and thus can be composed, via bundling and binding, with representations of discrete symbols. This property of SSPs is not used here but is shown to be important in Section 6.4.

For all state encoding methods, the embeddings are input to an Actor-Critic network (Figure 6.1) trained with the Soft Actor-Critic (SAC) algorithm. For the SSP and RFF methods, two layer critic and policy networks are used, while the baseline state input model uses three layer networks, so that the number of trainable parameters are similar between all approaches. Otherwise, the network architecture, algorithm implementation, and hyperparameters – all sourced from Li et al. (2021) – are the same amongst all methods.

### *6.3.2 Results*

The results are presented in Figure 6.2, which plots the interquartile mean (IQM) of episodic rewards over training across five trials, each conducted with different random seeds. The figure includes stratified 95% bootstrap confidence intervals for these IQM values. IQM is used as the reported point estimate for reporting performance, over mean or median metrics, due to

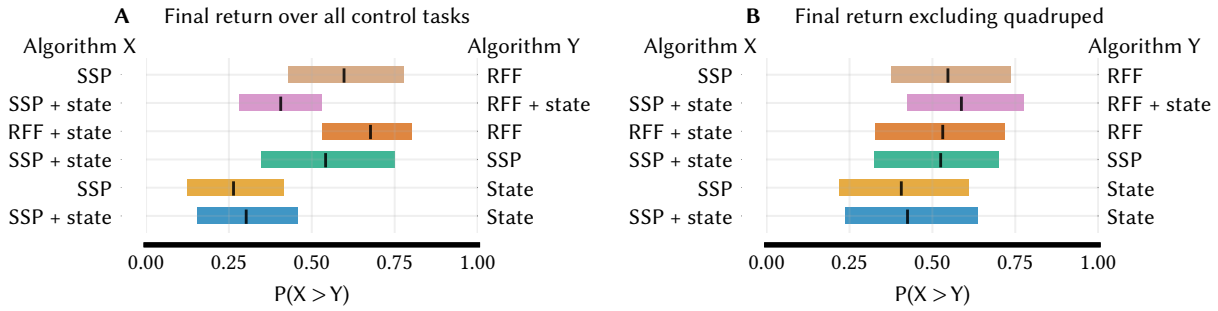



**Figure 6.3: Probability of improvement on control benchmarks**. The average probability of improvement of Algorithm X over Algorithm Y, $P(X > Y) = \frac{1}{M}\sum_{m=1}^{M} P(X_m > Y_m)$ where $P(X_m > Y_m)$ is the probability that the final return given Algorithm X is higher than that of Algorithm Y on task $m$.

its resilience against outliers and its ability to robustly represent central tendencies. This is in contrast to the mean, which may be highly skewed by outliers, and median, which can require a large number of samples for valid performance comparisons (Agarwal et al., 2021).

Additionally, to determine the statistical strength of observed differences in final performance, we report the average probability of improvement in Figure 6.3. These plots show how likely Algorithm X (left column) is to outperform Algorithm Y (right column) on a random task. This is computed by a Mann-Whitney U rank test (Agarwal et al., 2021). It is important to note that this does not take into account the magnitude of improvement or the effect size..

For all tasks except the quadruped benchmarks, the agent can learn reasonable polices with SSP input. However, in terms of final performance and sample efficiency, the SSP embeddings are generally the same or worse than the baseline state model and the learned RFFs. We see from Figure 6.3A, differences in final performance between the SSPs and baseline state methods are statistically robust and that concatenating state vectors to RFFs at least helps. Excluding the quadruped benchmarks, which had by far the largest state size ($n = 78$), we see that improvements in final returns are not statistically robust.

Overall, these results suggest that while SSPs can be used as features in RL control tasks, there are no clear benefits from using Fourier embeddings with grid-cell-inspired structures. In fact, there are significant drops in performance in tasks with high-dimensional states, such as quadruped walk and run. Note that for all environments, we use RFFs and SSPs of approximately the same size[2] which means that the high-dimensional states are not as accurately represented by the encoding. It is not entirely unexpected that SSPs do not shine in these control tasks. While it is not entirely unexpected that SSPs do not excel in these control tasks, we hypothesize that SSPs, as models of neural spatial representations, will provide an effective feature basis in spatial domains.

[2]The RFFs used $d = 1024$. The HexSSP initialization means that the SSP size must be $d = 2(n+1)n_{\text{scales}}n_{\text{rotates}}+1$, where $n$ is the size of the state space, and $n_{\text{scales}}$ and $n_{\text{rotates}}$ are the number of scales and rotations, respectively, used to construct the phase matrix. We selected $n_{\text{scales}}$ and $n_{\text{rotates}}$ for each task so that $d$ was close to 1024. The exact values of $d$ are listed in the appendix.

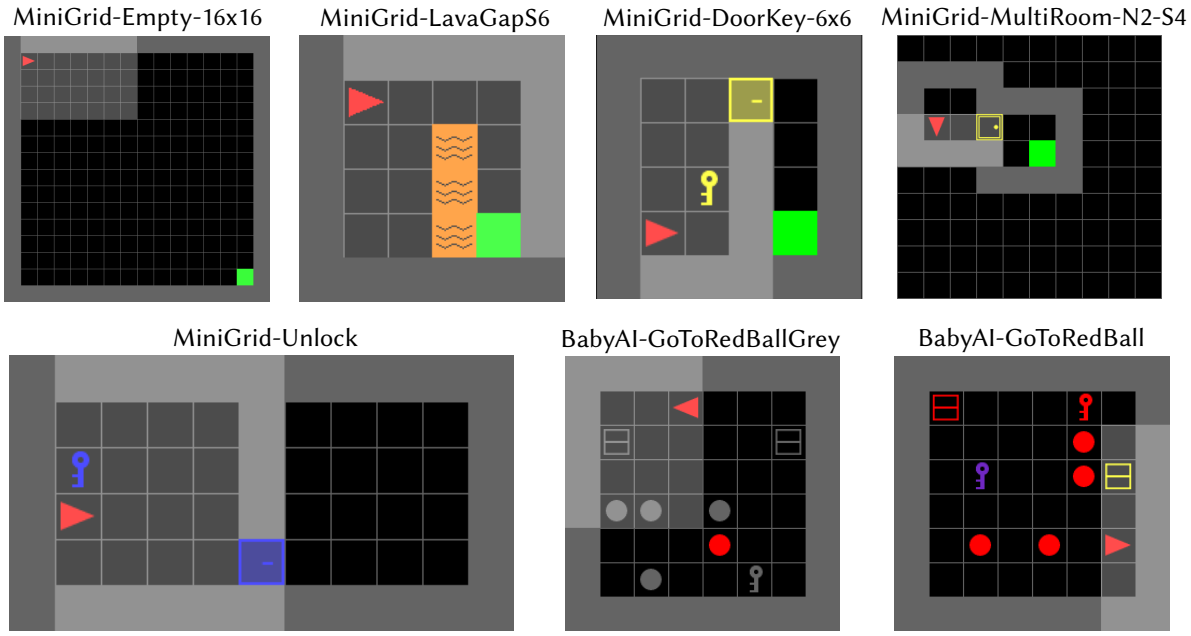



**Figure 6.4: Examples of the MiniGrid and BabyAI tasks.** The red triangle is the agent and the lighter coloured area depicts the agent's field of view. Objects included in these environments include goals (green squares), keys, boxes (squares with a horizontal line), doors, balls, and lava.

## 6.4 VSAs for structured scene representation

Now we investigate SSP state encodings of purely spatial variables and make use of the algebra of HRRs to create compositional state observation embeddings in navigation settings.

### *6.4.1 Methods*

We test our approach on a subset of the Minimalistic Gridworld Environments (MiniGrid; Chevalier-Boisvert et al., 2018b) and BabyAI environments (Chevalier-Boisvert et al., 2018a). MiniGrid is an open-source collection of 2D grid-world environments (see Figure 6.4). BabyAI also features grid-world settings but was originally designed for studying language acquisition. The BabyAI tasks used here do not require language.

In the grid world environments, each cell can contain, at most, one object, which is specified by its type, colour, and state. Possible object types include wall, door, key, ball, box, goal, and lava, with each object having attributes like colour (from a predefined set) and states (open, closed, locked) that are specific to certain object types.

The agent has a limited $7 \times 7$ field of view and cannot see through walls. The default observations are represented as a $7 \times 7 \times 3$ integer matrix, where each vector $(i, j, :)$ denotes the type, colour, and state of the object at position $(i, j)$ within the agent's field of view. The

agent can perform seven actions: turn left, turn right, move forward, pick up an object, drop an object, open a door or box, and complete a task (which is not applicable in the tasks considered here).

The agent receives a reward only upon meeting a goal condition, which may involve reaching a specific location, moving objects to designated spots, or carrying certain objects. The reward is dependent on the number of steps taken to obtain the goal condition,

$$r(s) = \begin{cases} 1 - 0.9\frac{\text{steps taken}}{\text{max. num steps}} & \text{if } s \text{ is the goal state,} \\ 0 & \text{if } s \text{ is not the goal state.} \end{cases} \tag{6.3}$$

MiniGrid's navigation tasks challenge the agent to navigate around obstacles and complete sequential actions such as obtaining keys or opening doors, with only a sparse reward signal. The more challenging scenarios often require curriculum learning (Patra et al., 2022), HRL (Dossa et al., 2022), or the use of intrinsic reward signals (Andres et al., 2022).

We evaluate the performance of an A2C model trained with Generalized Advantage Estimation (GAE) given various feature extraction methods (Figure 6.5) across several MiniGrid environments (Figure 6.4). The baseline model employs a convolutional neural network (CNN) to generate a 64-dimensional feature vector from the agent's $7 \times 7 \times 3$ observations (Figure 6.5A), which is then input into the actor and critic networks, illustrated in Figure 6.1. The network architecture and algorithm are adapted from Willems (2023).

Additionally, we test SSP and HRR state embeddings. Our first method involves representing the agent's pose within the environment as an SSP (Figure 6.5B),

$$\phi_{\text{pose}} = \phi\left([x, y, \theta]\right) \tag{6.4}$$

where $x, y$ is the agent's global position in the grid and $\theta \in \{0, 1, 2, 3\}$ is an integer indicating the direction the agent is facing. Although the state variables are discrete due to the finite number of possible agent positions and orientations, they are treated as continuous variables in this embedding.

This representation is consistent with the output of the SSP path integration model, SSP-PI (Chapter 3), and the localization output of SSP-SLAM (Chapter 4). If the agent can learn to navigate environments with this embedding, it demonstrates that the output from SSP-PI can be effectively integrated into RL systems.

The second method we test uses the algebra of HRRs to encode both the agent's field of view and its pose, as shown in Figure 6.5C. The complete state encoding is constructed via binding and bundling operations:

$$\begin{aligned} \Phi_{\text{view}} = \phi([x, y, \theta]) + \texttt{HAS} \circledast \sum_{\text{objects carried}} \texttt{ITEM}_i \circledast \texttt{COLOUR}_i \circledast \texttt{STATE}_i \\ + \sum_{\text{objects in view}} \Delta\phi_i \circledast \texttt{ITEM}_i \circledast \texttt{COLOUR}_i \circledast \texttt{STATE}_i. \end{aligned} \tag{6.5}$$

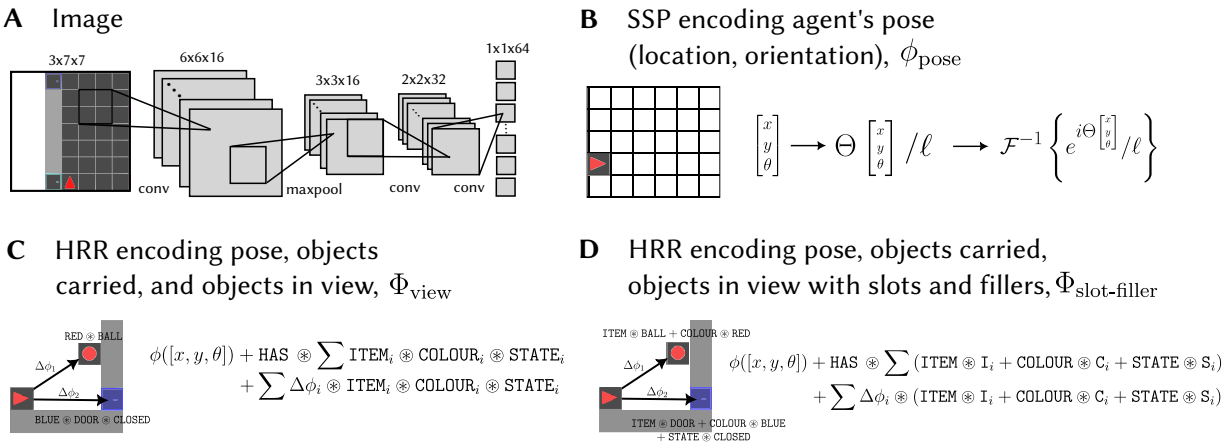



**Figure 6.5: MiniGrid feature extraction methods. (A)** The baseline model uses a CNN to obtain a 64-dimensional vector from the $7 \times 7 \times 3$ 'image' representation of the agent's field of view. **(B)** The SSP pose representation, $\phi_{\text{pose}}$, encodes the agents global location and orientation as a 201-dimensional vector. **(C)** The HRR view representation, $\Phi_{\text{view}}$, encodes the agent's pose and objects in its field of view and inventory. **(D)** The HRR slot-filler representation, $\Phi_{\text{slot-filler}}$, also encodes the agent's pose and objects in its field of view and inventory, but it does so using a slot-filler representation of objects.

This bundled representation combines SSPs with symbol-like vectors to encode comprehensive state information. The bundle includes:

- A representation of the agent's pose (global position and orientation), $\phi([x, y, \theta])$;
- A representation of the object the agent is carrying (bound with a semantic pointer, `HAS`), if the agent is carrying an object;[3] and
- A representation of objects in the agent's field of view and their location relative to the agent. The vector, `ITEM`, indicates the 'type' of an object in view, and can take on values `DOOR`, `KEY`, `BALL`, `BOX`, `GOAL`, or `LAVA`. The vector, `COLOUR`, indicates the colour of the associated object. The vector, `STATE`, indicates the 'state' of an object, and can take on values `OPEN`, `CLOSED`, or `LOCKED` (objects with fixed states, such as lava or balls, are encoded as being in the 'open' state). Finally, $\Delta\phi_i$, encodes an object-in-view's location relative to the agent.

Such an embedding can be obtained from the SSP-SLAM model. The localization module provides a representation of the agent's pose (or rather, an estimate of it), while the input we assume is provided to the landmark perception module is precisely what is needed to construct the bundle representing objects in view. After accumulating experience, the SSP-SLAM associative memory network can generate a comprehensive bundle of all objects in the

[3] In these environments the agent is limited to carrying a single object, so the sum over objects carried in Eq. (6.6) and Eq. (6.5) is over at most a single object.

environment. However, in this chapter, we do not use an environment map or global object information.

Note that there are other ways to encode an agent's surroundings with a VSA. The third encoding method we test uses a slot-filler approach (Figure 6.5C),

$$\begin{aligned}\Phi_{\text{slot-filler}} = \phi([x, y, \theta]) + \texttt{HAS} \circledast \sum_{\text{objects carried}} (\texttt{ITEM} \circledast \texttt{I}_i + \texttt{COLOUR} \circledast \texttt{C}_i + \texttt{STATE} \circledast \texttt{S}_i) \\ + \sum_{\text{objects in view}} \Delta\phi_i \circledast (\texttt{ITEM} \circledast \texttt{I}_i + \texttt{COLOUR} \circledast \texttt{C}_i + \texttt{STATE} \circledast \texttt{S}_i)\,,\end{aligned} \tag{6.6}$$

where ITEM, COLOUR, and STATE are random vectors that represent *slots* – they indicate the type of the vector they are bound with – while $\texttt{I}_i$, $\texttt{C}_i$, and $\texttt{S}_i$ denote the actual values of item type, colour, and state. The main difference between $\Phi_{\text{slot-filler}}$ and the prior HRR embedding, $\Phi_{\text{view}}$, is representational overlap. In $\Phi_{\text{view}}$, objects differing in any attribute (*e.g.*, an open blue door versus a closed blue door) are dissimilar, whereas in $\Phi_{\text{slot-filler}}$, objects sharing properties have greater similarity (*e.g.*, the representation of an open blue door is more similar to a closed blue door or a blue key compared to a red box).

The representational overlap may be a desirable representation property in some scenarios (*e.g.*, if the goal condition involves reaching an object of a particular type, regardless of its colour), but it may also be disadvantageous, due to confusion between objects and reduced separation of states. For instance, given a large length scale parameter (*i.e.*, a low spatial resolution), the terms $\Delta\phi_i$ and $\Delta\phi_j$ will be highly similar for $i \neq j$. This would make it difficult to determine which colours and states are associated with which object types given $\Phi_{\text{slot-filler}}$ when there are multiple objects in the agent's field of view. As another example, consider two states, one in which the agent is in front of orange lava and one in which the agent is in front of an orange door. Such states would be more similar when embedded with $\Phi_{\text{slot-filler}}$ compared to $\Phi_{\text{view}}$, and such similarity would make learning distinct policy output more challenging.

For all of these encoding methods, the SSP parameters, the phase matrix $\Theta$ and length scales $\ell$, are trainable, affecting both the $\phi([x, y, \theta])$ and $\Delta\phi_i$ terms. The SSP parameters are initialized to produce HesSSPs. However, the semantic pointers representing colours, items, states, and slots are fixed, random vectors. While the algebra of HRRs is differentiable and permits training these components, we believe that this would introduce too many additional parameters and is unnecessary.

### *6.4.2 Results*

Our experiments are conducted in six MiniGrid and two BabyAI environments, comparing six feature representations:

1. Image: The $7 \times 7 \times 3$ agent view matrix, which is processed by a CNN with 10544 trainable parameters.

2. Pose: The raw agent pose vector, $[x, y, \theta]$, which is processed by a two layer network with 17152 trainable parameters.
3. Image + pose: The agent view matrix and the pose vector – the view is processed by a CNN, the output of which is concatenated with the pose vector before it is passed to critic and actor networks.
4. SSP pose: The SSP encoding of pose, $\phi_{\text{pose}}$ – the only trainable parameters are the SSP parameters of which there are $n(1 + \lfloor {}^{d-1}/_{2} \rfloor) = 2031$.
5. SSP + HRR view: The HRR encoding of pose and view, $\phi_{\text{view}}$, which also has 2031 trainable parameters.
6. SSP + HRR slot-filler: The HRR encoding of pose and view that uses a slot-filler structure, $\phi_{\text{slot-filler}}$, which also has 2031 trainable parameters.

Additionally, we compare using RandSSPs versus HexSSPs.[4] Our research questions focus on whether the agent can learn using SSP and HRR state embeddings, which representations yield the best performance, and the sensitivity of performance to the phase matrix structure.

Figure 6.6 plots the IQM of episodic reward over five seeds against the number of environment observations. In all environments, except MultiRoom-N2-S4, the HRR encoding of pose and view, $\Phi_{\text{view}}$, performs the best or on par with other methods, consistently achieving the highest final return and the best sample efficiency. In some tasks, such as DoorKey-6x6 and LavaGap-S7, it is the only feature embedding that enables the agent to solve the task within the given observation limit. Although the image representation facilitates learning across many environments, and performs the best on MultiRoom-N2-S4, it consistently demonstrates lower data efficiency compared to $\Phi_{\text{view}}$.

Aggregate results across all seeds and environments are presented in Figure 6.7. The claim that $\Phi_{\text{view}}$ performs the best is supported by these results: the probability that it improves final return over alternative approaches is 65%-95% (Figure 6.7A) and, in the performance profiles (Figure 6.7B), the $\Phi_{\text{view}}$ curve is strictly above all others – meaning that across these tasks the $\Phi_{\text{view}}$ final return is statistically dominant over the returns from all other methods.

In the Empty-16x16 environment, where global position information is more useful than vision, the methods with the best sample efficiency are the direct representation of global position $[x, y, \theta]$, the SSP encoding of pose, and HRR view. The agent fails to consistently solve the non-empty tasks given pose input. Likewise, the SSP encoding of pose alone, $\phi_{\text{pose}}$, is only adequate in the empty environments.

Both $\Phi_{\text{view}}$ and $\Phi_{\text{slot-filler}}$ encode the same information with HRRs, but differ significantly in performance, suggesting that the structural representation of view/carry information critically

[4]There are additional experiments and results in Appendix B.2. These include results with a fixed, not learned, phase matrix/SSP parameters; using the HRR view embedding without pose information; results with much lower-dimensional embeddings; and results using different activation functions.

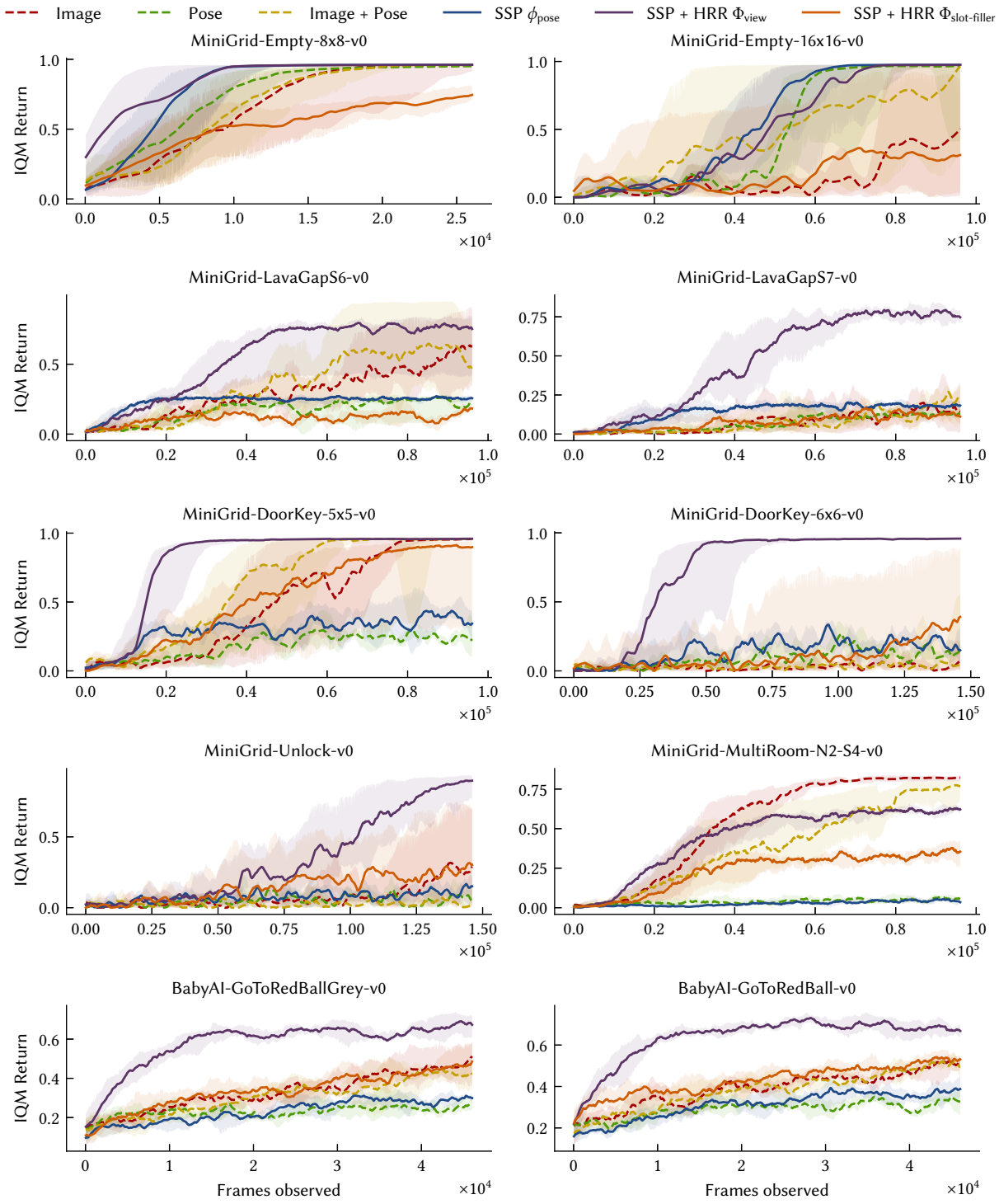



**Figure 6.6: Sample efficiency curves for navigation benchmarks**. The IQM of episodic reward over five seeds, along with stratified bootstrap 95% confidence intervals, versus the number of observations for different MiniGrid and BabyAI environments.

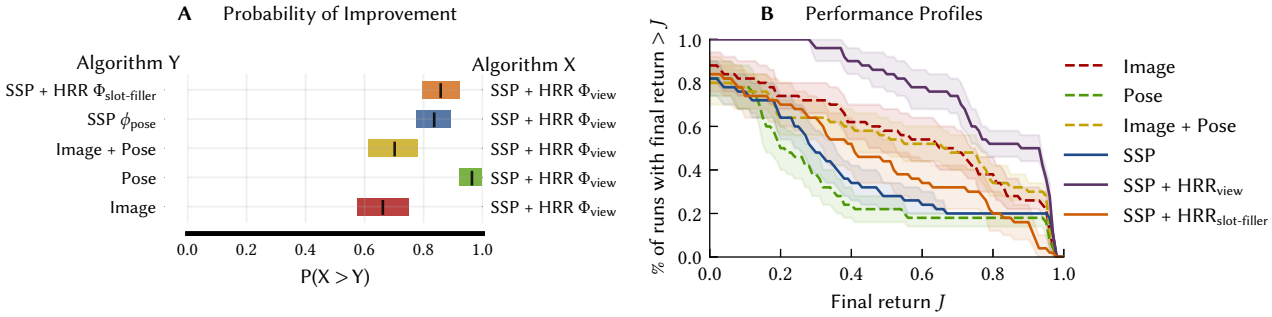



**Figure 6.7: Probability of improvement and performance profiles on navigation benchmarks. (A)** The average probability of improvement of Algorithm X over Algorithm Y, $P(X > Y) = \frac{1}{M}\sum_{m=1}^{M} P(X_m > Y_m)$ where $P(X_m > Y_m)$ is the probability that the final return given Algorithm X is higher than that of Algorithm Y on task $m$. **(B)** Return distributions after training.

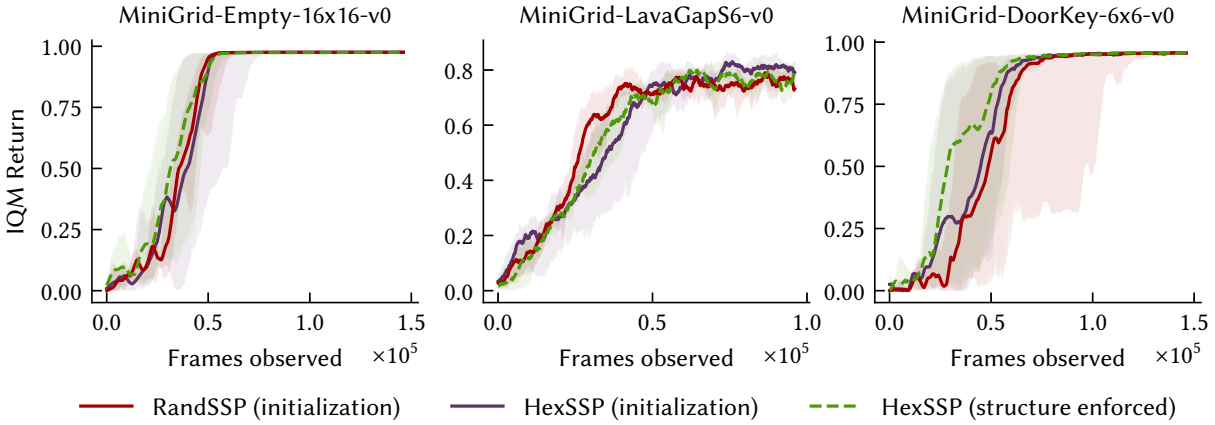



**Figure 6.8:** The IQM of episodic reward over five seeds for the HRR view embedding using a random phase matrix initialization (RandSSP), a HexSSP initialization, and a version that maintains the HexSSP structure over training.

impacts learning outcomes. $\Phi_{\text{slot-filler}}$ shows limited success, possibly due to the bundling capacity or representational overlap between objects.

We also compare RandSSPs and HexSSPs in Figure 6.8. The phase matrix of RandSSPs is sampled from a normal distribution, while the HexSSP phase matrix is structured by stacking scaled and rotated simplex vertices (see Section 3.3). In the 'initialization' runs, we use these schemes to initialize the phase matrix. For the 'HexSSP (structure enforced)' runs, the HexSSP structure is maintained during training – the scales applied to the base simplex (but not the rotations) are learnable. We find no major differences in performance, except higher variance given RandSSP initialization on one task. This suggests that the effectiveness of SSP representations in this context is largely robust to the specific details of initialization or structure, as the training process can compensate for and normalize initial differences.

Despite successes in these simple MiniGrid environments, solvable by a vanilla A2C

model, more challenging tasks likely require additional mechanisms, such as intrinsic rewards, curriculum learning, or HRL. HRRs and SSPs enhance navigation capabilities but do not negate the need for advanced strategies in complex scenarios. For example, multi-step tasks might require that agents be provided with, or compute, exploratory rewards signals or sub-goals. However, HRRs, which we have demonstrated are an effective feature basis for navigation, could be easily integrated with such additional mechanisms.

## 6.5 Discussion

### *6.5.1 Findings*

In this chapter, we investigated the use of SSPs and HRRs as a method for encoding state features – both continuous valued state features and structured states that include information regarding objects and their properties. The control experiments demonstrated that while SSPs can be used for encoding general continuous state variables, they do not outperform, and often underperform, the baselines in standard control tasks. In navigation experiments, HRRs showed a marked improvement in task performance, both in sample efficiency and final return. These results highlight the potential of HRRs to enhance spatial reasoning and decision-making.

### *6.5.2 Limitations*

Our work has several limitations that warrant further consideration. First, while we can project any continuous variables into the SSP space, the resulting space may not accurately reflect the topology of the original variables. For instance, certain variables in the control tasks were angles, which ideally should be encoded in such a way that preserves their periodicity. However, this issue is partially alleviated by training the SSP parameters, allowing adjustments to the kernel shape and width.

Another limitation involves the use of HRRs for structured representations, as the mapping from observations to embeddings is currently not learned and must be manually specified. As we saw in the MiniGrid experiments, there is flexibility in how one might represent a scene with a VSA, and this choice can greatly affect performance. Optimizing the structure of HRR encoding could potentially overcome this limitation.

### *6.5.3 Future work*

This section outlines several promising directions for further research based on our findings with SSPs and HRRs in deep RL networks.

**Action and policy representation**. One promising direction for future research involves expanding the use of SSP and HRR encodings beyond just states to include action and policy

representations. For example, we could use an actor network that outputs an SSP policy embedding rather than conventional action weights or Gaussian parameters, a change that would necessitate modifications to the underlying network structure. Such advancements could be informed by prior work in Bayesian optimization with SSPs (Furlong et al., 2021).

**Goal representation**. Prior work has used grid-cell codes for goal-conditioned navigation. Such work could be extended to incorporate HRRs and more complex goal representations. For instance, some MiniGrid and BabyAI tasks contain mission statements written in simple natural language (*e.g.*, what object type to pick-up, what colour door to open). These tasks could benefit from the ability of HRRs to encode mission statements that combine symbolic data, such as object types and colours with spatial information. This would enhance the agent's ability to interpret and execute language-based instructions within a dynamic environment.

**Intrinsic rewards**. Another intriguing area of research is the development of intrinsic reward systems that leverage our existing representations. By computing a type of novelty score based on the similarity of new states to a running mean state embedding vector, we can dynamically adjust the intrinsic reward to favour exploration of novel states.

$$r_t^{\text{intrinsic}} = \left(1 - \boldsymbol{\mu}_t \cdot \phi(\boldsymbol{s}_t)\right), \tag{6.7}$$

$$\text{where } \boldsymbol{\mu}_t = \frac{\sum_{t'=0}^{t-1} \phi(\boldsymbol{s}_{t'})}{\| \sum_{t'=0}^{t-1} \phi(\boldsymbol{s}_{t'}) \|}, \tag{6.8}$$

$$\rightarrow r_t^{\text{modified}} = r_t + \kappa r_t^{\text{intrinsic}} \tag{6.9}$$

where $\kappa$ is a scaling factor and $\boldsymbol{\mu}_t$ is a running mean of visited features vectors. If a state $\boldsymbol{s}_t$ (or similar states) has been visited many times before, $r_t^{\text{intrinsic}}$ will be small. If a state is dissimilar to past states (*i.e.*, novel), the intrinsic reward $r_t^{\text{intrinsic}}$ will be higher. This approach could significantly alter the reward dynamics, encouraging the agent to explore unvisited or rarely visited states.

**Successor features**. An important avenue for future investigation is the potential application of SSPs in successor feature (SF) models. In the last chapter, we demonstrated that fixed SSP features could be used to train a shallow SF function. Here we discuss the potential of SSP and HRR embeddings with trainable parameters to act as the features of a deep SF Actor-Critic (AC) network, and present preliminary results.

There are different ways to encode and learn features for SF. Recall that the SF function $\boldsymbol{M} : \mathcal{S} \rightarrow \mathbb{R}^d$ is defined as

$$\boldsymbol{M}(s_i) = \mathbb{E}_\pi \left[ \sum_{t=0}^{\infty} \gamma^t \phi(s_t) \,\middle|\, s_0 = s_i \right]. \tag{6.10}$$

This follows a Bellman-like equation, and so can be trained with model-free methods. We use a generalized advantage estimation (GAE) algorithm with off-line $\lambda$-return to train an SF

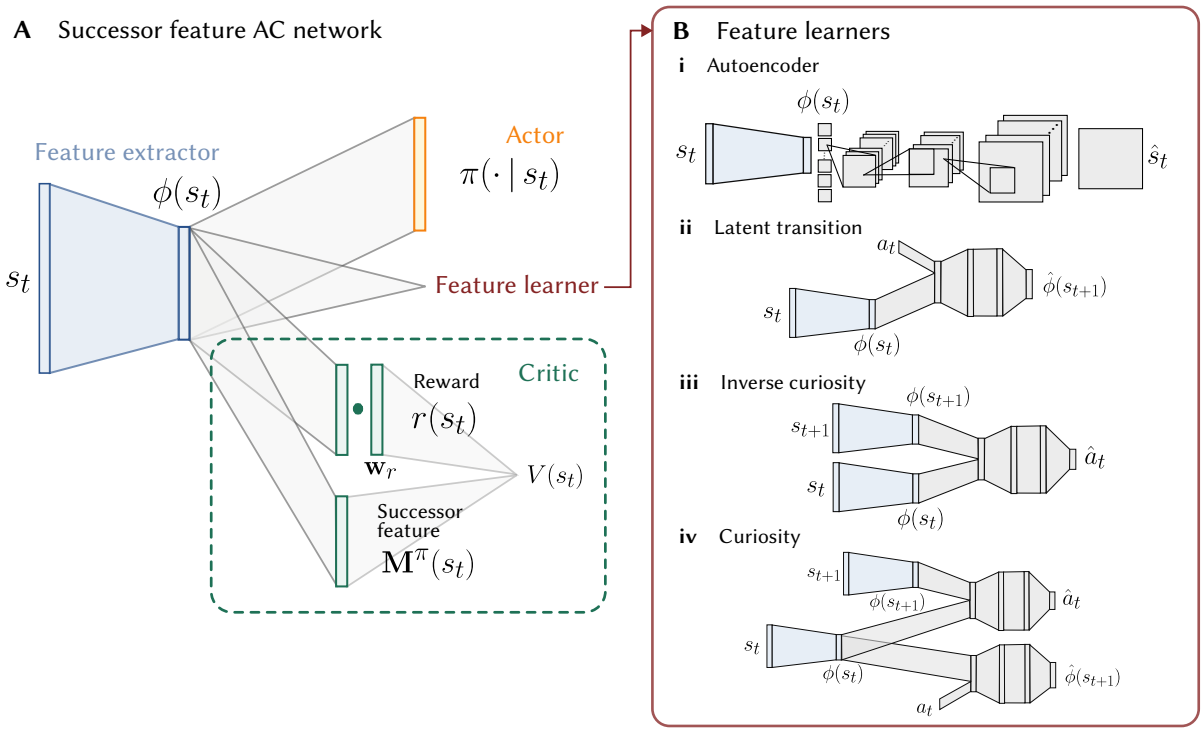



**Figure 6.9: The networks used for successor feature learning. (A)** Diagram of the AC network that uses successor features. A feature extractor (one of Figure 6.5A-C) takes the raw observation, $s_t$, and outputs its feature embedding, $\phi(s_t)$. This is supplied as input to a policy network, SF network, linear reward function, and a feature learner network. **(B)** The different feature learner networks are listed. Each has its own associated loss function used to train the feature extractors. The methods used are: **i)** an autoencoder, the feature learner network is a CNN decoder; **ii)** a latent transition model, the feature learner network is a model of forward dynamics; **iii)** an inverse curiosity module, the feature learner network is a model of inverse dynamics; and **iv)** a curiosity module, the feature learner network includes models of forward and inverse dynamics

network. Learning this alongside a linear reward function over features, $r(s) = \phi(s) \cdot \boldsymbol{w}_r$, one can obtain a value function. A full actor-critic model based on SFs is shown in Figure 6.9A.

The default feature extractor for the Minigrid tasks is a CNN (Figure 6.5A). The resulting feature is passed as input to the actor and critic (SF network and linear reward function) and additional networks for feature learning. The loss from the SF and actor are not propagated back to the feature extractor, as this results in poor performance and pathological feature representations (*e.g.*, all zeros).

Instead, the feature extractor is trained by the feature learner networks, which include:

- **Autoencoder**: Learn a decoder, $f : \mathbb{R}^d \to \mathcal{S}$, to recover states from their feature representations (Figure 6.9B(i)). The reconstruction error is the loss signal used to train

the feature extractor, $\mathcal{L}^{\text{feature}}_{\text{auto}} = \|f(\phi(s)) - s\|_2^2$. This was used in the first deep SF model (Kulkarni et al., 2016).

- **Latent transition model**: Learn a model of forward dynamics, $f : \mathbb{R}^d \times \mathcal{A} \to \mathbb{R}^d$, that predicts the next feature from the current feature and action (Figure 6.9B(ii)), $\mathcal{L}^{\text{feature}}_{\text{latent}} = \|f(\phi(s_t), a_t) - \phi(s_{t+1})\|_2^2$ (Grill et al., 2020; Touati et al., 2022).

- **Inverse curiosity**: Learn a model of inverse dynamics, $b : \mathbb{R}^d \times \mathbb{R}^d \to \mathcal{A}$, that predicts actions given consecutive features (Figure 6.9B(iii)). For continuous actions, an L2 loss can be used, but, for discrete actions, the probabilities of each action is the output of $b$, and cross entropy is used to train features. The idea here is to learn aspects of the state that are controllable (Pathak et al., 2017; Touati et al., 2022).

- **Curiosity**: Learns both a forward model (on latent representation), $f : \mathbb{R}^d \to \mathcal{S}$, and inverse dynamics model, $b : \mathbb{R}^d \times \mathbb{R}^d \to \mathcal{A}$ (Figure 6.9B(iv)). The loss used to train the features is the average of the losses from latent transitions and inverse curiosity, $\mathcal{L}^{\text{feature}}_{\text{cm}} = 0.5\mathcal{L}^{\text{feature}}_{\text{latent}} + 0.5\mathcal{L}^{\text{feature}}_{\text{icm}}$.

We test the network using SSP, $\phi_{\text{pose}}$, and structured HRR encodings of state, $\Phi_{\text{view}}$, following the approaches outlined in Section 6.4 (Figure 6.5B-C), on an empty 8×8 MiniGrid environment, but with the agent's starting position randomly varying trial-by-trial. The methods listed above were applied to train the SSP parameters. Here we use lower dimensional SSPs, with $d = 201$ and 303 trainable parameters in the feature extractors as opposed to Section 6.4, which uses $d = 1353$ and 2031 trainable parameters.

The preliminary results are presented in Figure 6.10. In this environment, curiosity (Figure 6.10A) and latent transition (Figure 6.10C) perform well, and for both of these methods the HRR representation of agent pose and local view performs best. The SSP encoding of just pose performs worse than the HRR except when trained with inverse curiosity (Figure 6.10B), in which case they are comparable. Interestingly, we see an increasing, though not converged, return for the image feature extractor when it is trained via latent transitions or inverse curiosity, but not the full curiosity module, which simply combines the losses of the prior two. The worst performing variations are the autoencoder with the image representation (Figure 6.10D) – this favours features from which the entire $7 \times 7 \times 3$ observations can be decoded from, rather than features that capture the environment dynamics – and the fixed SSPs and HRRs (Figure 6.10D). This shows that the SSP parameters require some form of tuning to the environment dynamics.

Future work should consider more tasks, particularly more complex tasks, and other methods for feature learning, such as the Laplacian eigenfunction method from Touati et al. (2022) or forward-backward method from Touati et al. (2021).

**Non-stationary representations**. Non-stationary SSP representations could be used to introduce adaptability to varying environmental demands. By allowing phase matrices and length

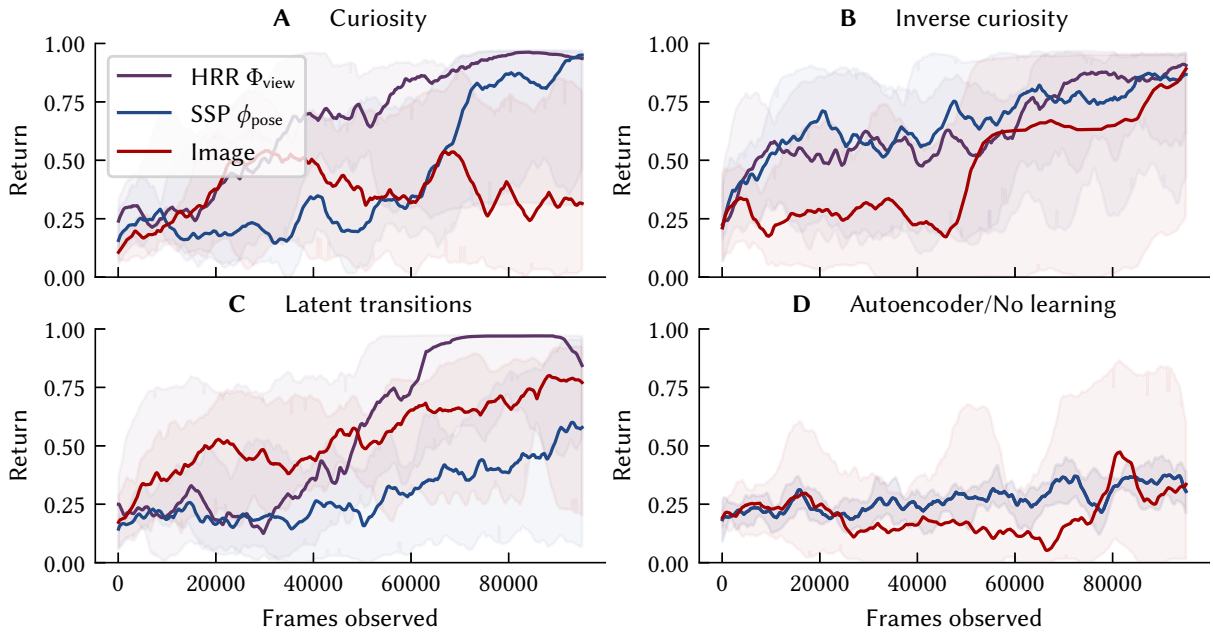



**Figure 6.10: Results using SSPs and HRRs as the features in a successor feature model** on MiniGrid Empty Random $8 \times 8$. Plotted is the IQM of episodic reward versus the number of observed environment transitions for 5 seeds. Each plot includes results using different feature extraction methods: HRR encoding of the agent's pose and view, $\Phi_{\text{view}}$, the SSP encoding of agent pose, $\phi_{\text{pose}}$, and the default image observations, processed by a CNN. Different methods were used to train the feature extractors (the CNN weights and theSSP mapping parameters): **(A)** curiosity, **(B)** inverse curiosity, and **(C)** latent transitions. **(D)** Plotted are results using fixed SSP and HRR representations – no feature learning used – and results using an autoencoder to train the image CNN. The autoencoder method was not used with SSP and HRR representations.

scales to be outputs of a neural network based on raw state inputs, the system could automatically adjust the resolution of spatial representations to match the requirements of different regions within an environment. This would be particularly useful in complex navigation tasks where different areas require varying levels of detail in state representation.

**Meta-learning over HRRs.** Integrating meta-learning over HRRs would be a major advancement. Current models employ fixed strategies for encoding states, which, as evidenced in the MiniGrid results, may limit their effectiveness. Both $\Phi_{\text{view}}$ and $\Phi_{\text{slot-filler}}$ were used to encode the same information with the algebra of HRRs, just in slightly different ways, and yet $\Phi_{\text{view}}$ consistently performed better than $\Phi_{\text{slot-filler}}$. A meta-learning model that could learn and adapt the composition and structure of symbol-like vectors within a VSA framework would offer a way to refine state or goal embeddings continuously based on environmental feedback. This could lead to more sophisticated and adaptable systems capable of handling a broader range of tasks and environments.

## 6.6 Conclusion

In this chapter, we explored the potential of neurocognitive feature representations, specifically HRRs and SSPs, in improving reinforcement learning (RL) algorithms. We found that SSPs can indeed be used to encode state in RL algorithms and improve performance on navigation tasks. There is thus potential to integrate outputs from SSP-SLAM – specifically landmark identities, properties, and locations derived from its associative memory networks – directly into RL systems.

This approach underscores the utility of SSPs as an easily integrated feature processing layer, enhancing standard RL models without necessitating substantial architectural changes. Moreover, the use of HRRs enables the incorporation of semantic knowledge into state embeddings, presenting a method to use symbolic knowledge seamlessly within the existing frameworks of deep neural networks.

# Chapter 7

# Conclusion

> Deciphering the code of the brain will have a lasting impact on our society. It is not simply an intellectual exercise for a handful of esoteric individuals anymore. It is also more than a "just" a brain-health–related issue, which affects millions in the United States and many more worldwide. As Robert Noyce, the co-inventor of the integrated circuit, once put it: In order to understand the brain, we have used the computer as a model for it. Perhaps it is time to reverse this reasoning. To understand where we should go with the computer, we should look to the brain for some clues.
>
> — György Buzsáki
> *Rhythms of the Brain*

This thesis posed the question: How does the mammalian brain represent and manipulate spatial variables to navigate complex environments? We propose that Vector Symbolic Algebras (VSAs), specifically Holographic Reduced Representations (HRRs), offer a compelling framework for understanding and modelling essential aspects of spatial cognition. We argue that SSPs and HRRs are more than abstract representational tools; by considering their dynamics, we reveal a foundation in oscillatory neural processes. This perspective grounds the mathematical operations of HRRs, such as binding, in plausible neural mechanisms, bridging the gap between their origins in cognitive modelling and their biological relevance.

Although decades of research on the hippocampal formation has revealed possible functions and uncovered classes of spatially sensitive neurons, the precise encoding and manipulation of spatial information in the brain remains incompletely understood. Neural tuning curves, though informative, do not fully reveal how the brain represents spatial information or how it builds cognitive maps and navigates using these representations. Thus, a high-level, general-purpose framework is needed – one that can articulate the principles underlying these mechanisms and that can be used to build testable models.

We put forth VSA as such a framework and Spatial Semantic Pointers (SSPs) as a biologically plausible model for encoding and manipulating spatial information within this framework. Using SSP encodings and the Neural Engineering Framework (NEF), we developed spiking neural models for key navigational processes – such as path integration, mapping, and learning. These models present possible neural mechanisms of these cognitive functions and have potential applications in AI. This work has a dual focus, intersecting theoretical neuroscience and AI, as understanding spatial cognition in the brain can not only advance our knowledge of neural mechanisms, but also inspire new approaches to navigation and learning in artificial

systems.

## 7.1 Contributions

One key contribution is further developing SSPs as a model for neural spatial representation. In VSAs, concepts and symbols are represented by high-dimensional vectors that can be composed or bound to create structured, distributed representations. HRRs propose a particular method for binding information through circular convolution of vectors, or equivalently, the Hadamard product of complex vectors in the Fourier domain. This approach has a straightforward implementation. We extend this view by noting that SSPs and other unitary vectors can be thought of as sets of oscillators, and focus on that as a novel implementational route. Information, such as an animal's location in space or even discrete concepts, is thus encoded by the configuration of many neural oscillators constrained by SSP relations. The alignment between two configurations of oscillator sets reflects the similarity between their associated concepts, providing a biologically plausible mechanism for representing both continuous and discrete information.

This thesis demonstrates that SSPs can effectively model spatial encoding in the brain by presenting several biologically plausible models of navigation-related cognitive functions using SSPs. First, in Chapter 3, we describe a non-normative model of grid cells using SSPs with a specific simplex structure. We introduce a general algorithm for constructing these structured SSPs, termed HexSSPs, and extend this approach to higher-dimensional spaces (three and beyond). Consistent with prior findings on grid cells, we show that this representation achieves low decoding error, which decays exponentially with the number of neurons—surpassing the power-law decay typical of place cell or linear encoding. Our contribution lies in the development of the HexSSP representation, which offers an efficient and scalable encoding of space.

In Chapter 3, we derive the dynamics of SSPs and present a spiking neural network (SNN) model of path integration – *i.e.*, the continual updating of SSP-encoded self-location using only self-motion signals. Circular convolution, the binding operation specific to HRRs, directly informs the structure of our models, including the path integration model. Continuous binding is realized through the dynamics of velocity-controlled oscillators, stabilized with attractor dynamics. Furthermore, we demonstrate that additional variables – such as time – can be incorporated within the SSP framework to model memory of spatial trajectories. Our contribution includes the development of an SSP-based path integration model (the SSP-PI model) and a trajectory memory model, each grounded in biologically plausible mechanisms.

In Chapter 4, we extend the SSP-PI model to incorporate mapping and relocalization mechanisms. We present SSP-SLAM – the first semantic spiking SLAM model. SSP-SLAM combines the path integration network with a hetero-associative memory network that learns and stores the locations of objects and features within an environment, using the biologically plausible Voja and PES learning rules. This memory network can later be queried to retrieve global

locations of specific objects or groups of objects sharing particular features. We demonstrate that this environment map can be used to correct errors in the path integration network. Additionally, we implement SSP-SLAM on a neuromorphic hardware emulator, demonstrating its practical viability for robotics applications. Our contribution is the development of the complete SSP-SLAM model.

In Chapter 5, we apply SSPs and a neural memory network, the Legendre Delay Network (LDN), to reinforcement learning (RL), creating spiking RL models for active navigation and decision-making. We use the LDN to encode state histories and compute temporally extended TD errors. Using SSPs as feature embeddings, we demonstrate their applicability in learning successor features and reward distributions. Furthermore, we demonstrate that SSP bundles can be used to encode stochastic policies. Although further work is needed to refine the learning method for these SSP policy embeddings, we show that the policy embeddings can be accurately sampled in a biologically plausible manner using Langevin dynamics. Our contributions include 1) using LDNs in RL, 2) developing the LDN-TD learning rules for training shallow spiking value and successor feature networks, 3) leveraging SSPs in a spiking successor feature model, 4) proposing the use of SSPs to represent reward distributions, 5) using SSP bundles as policy embeddings, and 6) presenting a spiking model of Langevin dynamics for sampling non-stationary policy embeddings. These advancements position SSPs as a potentially valuable tool in RL models, and lay the groundwork for further exploration of bio-inspired RL architectures

In Chapter 6, we shift to deep artificial neural networks, exploring applications of SSPs to standard benchmarks in RL. We investigate the utility of SSPs and HRRs as general-purpose feature embeddings for navigation and control tasks, demonstrating that compositional embeddings – constructed using the bundling and binding operations of HRRs to represent mixed discrete-continuous information – can be effectively used by deep networks. These embeddings enable deep RL agents to learn navigation tasks in sparse reward grid world environments more effectively than alternative representations, including default representations and those augmented with global location information. Additionally, we show that SSPs can represent high-dimensional, non-spatial variables in control tasks, though performance varies by task. These results underscore the value of HRR operations for constructing compositional representations and emphasize the critical role of how complex, multi-faceted information is represented for effective learning. Our contribution is a detailed investigation into the potential of SSPs and HRRs as state encoding mechanisms in deep RL.

## 7.2 Key findings

Having outlined the contributions of this thesis, we now turn to the insights gained from the methods and results presented in each Chapter. This research set out to investigate how biologically plausible spatial representations, specifically SSPs, could model essential cognitive functions like navigation, memory, and decision-making. Beyond developing these models,

each Chapter contributes to our perspective on the role of spatial encoding in neural and artificial systems. In the following section, we discuss these perspectives and highlight the key findings from each Chapter, situating these insights within the broader context of past research.

In Chapter 3 we introduced HexSSPs, a representation of continuous variables that can be encoded by grid cell activity. The advantages of HexSSPs over RandSSPs align with previous research on the information-theoretic properties of grid cell coding. Previous grid cell models have been either mechanistic – grid cells in a continuous attractor neural network (CANN) or derived from output of a oscillator-interference (OI) network – or normative – claiming grid cells emerge in path integration networks given particular constraints or as a result of certain unsupervised feature learning rules. We do not contest these normative accounts; in fact, such mechanisms may help explain how the HexSSP structure is initially learned, an important area for future research. Rather, we build on the premise that grid cells provide an efficient, robust basis for spatial representation, and we develop an SSP encoding that leverages this structure, and embeds it in a plausible neural mechanism.

However, our path integration model, SSP-PI, differs from previous models in its foundation, details, and interpretation. Unlike prior work, we begin with the hypothesis that HRRs provide a framework for compositional neural representation, with SSPs serving as the natural encoding for spatial variables within this framework. By deriving the dynamics of SSPs, we show that continual binding has dynamics equivalent to velocity-controlled oscillators – a core component in OI models. As a result, the SSP-PI model is a hybrid OI and CANN model due to the novel oscillator-stabilizing attractor dynamics.

We view the consistency between SSP-PI and existing OI and CANN-based models as a strength. Previous models have effectively matched neural data and generated testable predictions, and the fact that SSP dynamics lead to a similar integration mechanism supports their role as a framework for continuous encoding in real neural systems. This connection also provides a new interpretation of past OI models as preserving the dynamics of SSP spatial embeddings. Furthermore, SSP-PI extends path integration to $n$-dimensional spaces without any modifications to the network's internal structure or weights, moving beyond traditional 2D or 3D spaces. Our proposed framework also enables us to bind path integration output with discrete representations, allowing for contextual and compositional spatial representations, such as maps.

In SSP-PI, grid cells represent the model's output but do not participate in its recurrent dynamics. Although the model could be adapted to use recurrent populations of grid cells, this approach hurts performance and limits the model's generalizability. While not explored in this thesis, we suspect that grid cells may contribute to error correction or stabilize path integration dynamics through coupling mechanisms, rather than serving as the primary path integration network. Past work has explored this (*e.g.*, Zilli et al., 2010; Mosheiff et al., 2019), but work remains to incorporate such mechanisms into SSP-PI.

Instead of grid cells, velocity-dependent band cells drive path integration in SSP-PI. Neurons with sensitivity to both heading direction and speed have been identified in deeper layers of the MEC. Although these neurons have appeared to have band- or stripe-like spatial tuning in past studies, our results suggest that the 'banded-ness' of velocity-dependent band cells may be challenging to detect in natural environments. We hypothesize that spatial tuning patterns in these neurons may have been previously overlooked. This hypothesis leads to a testable prediction: by recording from these deeper MEC layers in rodents navigating spiral mazes or other environments where heading direction correlates with location, spatial patterns in tuning curves – such as band structures – may be observed, providing empirical support for the SSP-PI model.

In Chapter 4, we explored cognitive map formation and SLAM with SSPs, showing that, by interpreting the output of a path integration network as a representation within a VSA, we can naturally perform operations needed in navigation – such as vector addition of spatial variables and binding spatial locations with discrete features. SSP-SLAM illustrates that VSAs provide a versatile framework capable of integrating probabilistic representations, semantic knowledge, and symbolic representations in spiking systems. These findings highlight the potential of VSAs in SLAM and may serve as a foundation for future research, in both biologically inspired SLAM and robotics applications.

By demonstrating SSP-SLAM on a neuromorphic emulator, we suggest that, with further development, this approach could be deployed on neuromorphic hardware as a low-power SLAM system, which may have applications in space, underwater vehicles, and edge computing, where power efficiency is critical. However, a current limitation is that SSP-SLAM requires semantic input rather than raw sensory data. Deploying SSP-SLAM on hardware and comparing it with existing visual SLAM models requires the integration of networks for feature extraction from sensory input. That being said, ongoing research into image representation with SSPs indicates that such integration is achievable (Penzkofer et al., 2024).

In Chapter 5, we explored biologically plausible model-free RL – using spiking networks that learn online in continuous time, incorporating elements like episodic memory models and stochastic action sampling. We found that the LDN can be used as a memory buffer for RL. Other aspects of this Chapter – including the successor feature model, reward distribution embeddings, and policy representation, learning, and sampling – all leveraged SSPs. In particular, this work made use of the fact that weighted combinations of SSPs can represent probability distributions, functioning similarly to finite-dimensional kernel mean embeddings.

We discuss the potential application of SSPs in reward distribution learning and presented preliminary results. Although this work does not constitute a full distributional RL model or analysis, it is an interesting proposal. Current biologically grounded distributional RL models often use discretized distributions or parameterized quantile functions to represent return distributions, whereas SSP embeddings offer an alternative that enables the same network architecture to represent either successor features or return distributions.

SSP embeddings were also applied to policy representation. Our proposed approach to learn policy embeddings online faced challenges as the variance of the likelihood ratio estimate of the policy gradient grows unbounded in continuous time settings. However, alternative methods from control theory could inform future work on continuous-time policy learning (Munos, 2006; Ainsworth et al., 2021). Despite these challenges, we demonstrated that a spiking recurrent network can successfully sample from these embeddings, showcasing the potential of SSPs for policy representation in spiking RL models.

Lastly, in Chapter 6, we investigated the use SSP and HRR encodings of state in deep RL. We found that while SSPs – both with and without grid-cell-like structure – can be used to encode (often non-spatial) state variables in control tasks, they did not improve performance in many cases. In fact, SSP states reduced performance on tasks with very high-dimensional state spaces. However, using HRRs that combined SSPs encoding agent location with vectors representing discrete variables (such as objects and their properties) significantly improved performance in navigation tasks. This suggests that deep networks can leverage the structured, distributed representations characteristic of VSAs and benefit from these representations in certain settings.

The results in Chapter 6 provided insights into constructing effective VSA embeddings. The control task findings show that not all variables benefit from projection into SSP space, while the navigation task results highlighted the importance of learning the SSP parameters and the importance of the structure of compositional representations. This motivates further research into encoding information into VSAs in order to improve the performance on deep RL tasks. Rather than a one-size-fits-all approach, we suggest that compositional representations should be adaptable, adjusting according to context and task demands. This adaptability could be addressed by hierarchical reinforcement learning (HRL) or other meta-learning methods, opening new avenues for developing state representations in deep RL.

## 7.3 Future work

There are numerous avenues for future research that can extend the work presented in this thesis. While each Chapter includes discussion on future work, here we review some of the proposed research directions that we view as especially promising. Additionally, we outline research opportunities that integrate methods and models from different chapters.

| | Description |
|---|---|
| **Path Integration and SLAM in Non-Spatial Contexts** | The applicability of the processes underlying spatial navigation to a much broader range of non-spatial cognitive functions was a major motivation for this thesis. Now that we have developed models of PI and SLAM with SSPs, we should apply them to non-spatial domains, such as navigating social scenarios. For example, maintaining an estimate of one's position in a social group over time or evaluating relationships along multiple dimensions could leverage similar computational principles. Extending these models to abstract domains will help test the hypothesis that shared neural mechanisms underlie spatial and non-spatial cognition. |
| **Non-stationary SSPs** | While we primarily used SSPs with fixed bandwidth/length-scale parameters, certain tasks might benefit from variable length scales. For example, SSP policy embeddings could have varying spatial or temporal resolution to balance broad generalization in some regions of the state space with the need for fine-grained control in others. This could be achieved by a neural network that outputs length scales or by selectively masking rows of SSPs in the Fourier domain to achieve variable resolution. Exploring these approaches could enhance the flexibility of SSPs. |
| **Grid Cells and Coupling in Path Integration** | Future research could investigate coupling mechanisms that enforce or maintain the HexSSP structure during path integration, enabling corrections to the current SSP-PI model from grid cell populations. For examples, mechanisms from Ji (2014) or Kymn et al. (2024) could be applicable. |
| **Visual SSP-SLAM** | To make SSP-SLAM deployable for practical applications, we should extend the model by integrating visual input, using a network to extract landmark features or identities as well as depth estimates. |
| **Memory Replay and Consolidation for RL** | While we used the LDN memory to compute TD errors incorporating past neural activity estimates, future work could explore a memory network that associates states with LDN memory vectors for replay and consolidation, similar to episodic memory retrieval in biological systems. |
| **Learning SSP Policy Embeddings in Continuous Time** | To address the limitations of our current method for learning SSP policy embeddings in continuous time, we should explore alternatives, informed by the work in Ainsworth et al. (2021) and Jia et al. (2022). |

| | |
|---|---|
| **Distributional RL** | We could expand on the initial work with SSPs in distributional RL, which proposed a method for online learning of an embedding of the distribution of future rewards. This work could be extended by modelling full return distributions using a Bellman updating mechanism that incorporates binding (rather than bundling) of observations and future estimates. Additionally, these embeddings could be used for risk-aware planning or to optimize policies under specific constraints. A natural next step would be to compare these results to existing neural data that support distributional RL hypotheses in the brain (Lowet et al., 2020; Dabney et al., 2020). |
| **SSP-SLAM and RL Integration** | A natural next step is to integrate the spiking SSP-SLAM model with a spiking RL network in a continuous navigation task, where an agent must achieve goals while estimating state. The localization output would represent the agent's pose, while the environment map learned in the associative memory network could contribute a compressed cognitive map as input to the RL agent. |
| **Model-Based RL with SSP Cognitive Maps** | Although this thesis explored both biologically plausible cognitive mapping and RL, it did not investigate the integration of cognitive mapping into RL or model-based RL more broadly – despite its significant relevance to learning in animals. While such exploration was beyond the scope of this work, future research should focus on leveraging SSPs to develop cognitive maps that support model-based RL. For example, this could involve learning an embedding of landmarks bound with locations in a navigation task and providing this as additional input to a value or policy network. Alternatively, it could involve using action sampling and PI to simulate and plan trajectories. |
| **SSP Representations of Actions, Goals, and Policy Embeddings in Deep RL** | Future work could extend SSPs-based representations to encode actions, goals, and policies within deep RL frameworks. This approach could enable more structured and interpretable representations that could be used for goal-directed planning or policy learning. |

| | |
|---|---|
| **Meta-Learning Over HRRs** | A promising direction for future work is the use of meta-learning to learn and adapt the structure of state HRR embeddings based on environmental feedback. With a fixed vocabulary of vectors representing symbols, the meta-learning process would focus on selecting which symbols to include and determining how to combine them. This involves learning whether specific components should be bound or bundled, as well as the order of operations needed to construct task-relevant representations. Techniques such as symbolic regression or gradient-based meta-learning could be employed to optimize these decisions. This would enable a system to learn task-specific representations that are tailored to particular objectives, or more generalized embeddings that support compositional generalization across different tasks. |
| **Spaun in Space** | Spaun is a large-scale functioning brain model built with the SPA. The models and methods developed here could be integrated into future iterations of Spaun, enabling it to navigate. This mobile Spaun could leverage the spatial encoding and RL models for dynamic, online learning and spatial reasoning in complex environments. |

## 7.4 Final remarks

All in all, this work contributes a novel framework for neural spatial encoding and mapping, offering insights into the basis of navigation in the brain and suggesting potential AI applications. By developing models like SSP-PI and SSP-SLAM, we have demonstrated that SSPs are not only useful for encoding spatial location, but can also support cognitive mapping, object-location associations, and the handling of probabilistic information in biologically inspired RL. We have focused on a subset of key navigation-related functions: localization, mapping, and learning from the output of the such functions. Additionally, one of our goals was to develop general-purpose methods for representation and mapping, extending beyond spatial applications. While VSAs have shown promise as a versatile framework for modelling concepts, emotions, and language, and our models were designed with this generality in mind, we have yet to apply our navigation models to non-spatial domains.

Spatial cognition – and cognition more broadly – involves far more than just representation; it encompasses reasoning, model-based learning, self-monitoring, and numerous other processes. By coupling VSAs with neural dynamics to explore spatial representation and navigation in the brain, this work provides a promising framework but is far from a complete picture. Fundamental questions remain unanswered: How are representations learned and flexibly adapted to changing environments? How are they composed and manipulated in context-dependent ways? How do these processes integrate to support the full spectrum of cognitive functions?

This thesis represents not an endpoint but a foundation – laying the groundwork for future

investigations into how VSAs might be enriched and extended to model these more complex and dynamic aspects of cognition, ultimately bridging the gap between abstract frameworks and the richly adaptive nature of biological intelligence.

# Bibliography


Abad Perez, Pablo, Luis M Martinez-Otero, Victor Borrell, and Jorge R Brotons (2024). "Evidence of spatial periodic firing in the subiculum of mice". In: *bioRxiv*, pp. 2024–05.

Abbott, Larry F (1999). "Lapicque's introduction of the integrate-and-fire model neuron (1907)". In: *Brain research bulletin* 50.5-6, pp. 303–304.

Agarwal, Rishabh, Max Schwarzer, Pablo Samuel Castro, Aaron Courville, and Marc G Bellemare (2021). "Deep Reinforcement Learning at the Edge of the Statistical Precipice". In: *Advances in Neural Information Processing Systems*.

Agmon, Haggai and Yoram Burak (2020). "A theory of joint attractor dynamics in the hippocampus and the entorhinal cortex accounts for artificial remapping and grid cell field-to-field variability". In: *Elife* 9, e56894.

Aharon, Gal, Meshi Sadot, and Yossi Yovel (2017). "Bats use path integration rather than acoustic flow to assess flight distance along flyways". In: *Current Biology* 27.23, pp. 3650–3657.

Ainsworth, Samuel, Kendall Lowrey, John Thickstun, Zaid Harchaoui, and Siddhartha Srinivasa (2021). "Faster policy learning with continuous-time gradients". In: *Learning for Dynamics and Control*. PMLR, pp. 1054–1067.

Aminoff, Elissa M, Kestutis Kveraga, and Moshe Bar (2013). "The role of the parahippocampal cortex in cognition". In: *Trends in cognitive sciences* 17.8, pp. 379–390.

Amit, Daniel J and Daniel J Amit (1989). *Modeling brain function: The world of attractor neural networks*. Cambridge university press.

Andersen, Per (2007). *The hippocampus book*. Oxford university press.

Anderson, Greg, Abhinav Verma, Isil Dillig, and Swarat Chaudhuri (2020). "Neurosymbolic reinforcement learning with formally verified exploration". In: *Advances in neural information processing systems* 33, pp. 6172–6183.

Anderson, Michael I and Kathryn J Jeffery (2003). "Heterogeneous modulation of place cell firing by changes in context". In: *Journal of Neuroscience* 23.26, pp. 8827–8835.

Andres, Alain, Esther Villar-Rodriguez, and Javier Del Ser (2022). "An evaluation study of intrinsic motivation techniques applied to reinforcement learning over hard exploration environments". In: *International Cross-Domain Conference for Machine Learning and Knowledge Extraction*. Springer, pp. 201–220.

Anggraini, Dian, Stefan Glasauer, and Klaus Wunderlich (2018). "Neural signatures of reinforcement learning correlate with strategy adoption during spatial navigation". In: *Scientific reports* 8.1, pp. 1–14.

Aronov, Dmitriy, Rhino Nevers, and David W. Tank (2017). "Mapping of a non-spatial dimension by the hippocampal–entorhinal circuit". In: *Nature* 543.7647.

Arora, Nipun, Robert West, Andrew Brook, and Matthew Kelly (2018). "Why the Common Model of the mind needs holographic a-priori categories". In: *Procedia Computer Science* 145, pp. 680–690.

Ashwood, Zoe, Aditi Jha, and Jonathan W Pillow (2022). "Dynamic inverse reinforcement learning for characterizing animal behavior". In: *Advances in neural information processing systems* 35, pp. 29663–29676.

Atwood, Harold L and Shanker Karunanithi (2002). "Diversification of synaptic strength: presynaptic elements". In: *Nature Reviews Neuroscience* 3.7, pp. 497–516.

Azizzadenesheli, Kamyar, Alessandro Lazaric, and Animashree Anandkumar (2016). "Reinforcement learning of pomdps using spectral methods". In: *Conference on Learning Theory*. PMLR, pp. 193–256.

Bacon, Pierre-Luc, Jean Harb, and Doina Precup (2017). "The option-critic architecture". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 31. 1.

Baird, Leemon C (1994). "Reinforcement learning in continuous time: Advantage updating". In: *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94)*. Vol. 4. IEEE, pp. 2448–2453.

Bakker, Bram (2001). "Reinforcement learning with long short-term memory". In: *Advances in neural information processing systems* 14.

Banino, Andrea et al. (2018). "Vector-based navigation using grid-like representations in artificial agents". In: *Nature* 557.7705, pp. 429–433.

Bao, Xiaojun, Eva Gjorgieva, Laura K Shanahan, James D Howard, Thorsten Kahnt, and Jay A Gottfried (2019). "Grid-like neural representations support olfactory navigation of a two-dimensional odor space". In: *Neuron* 102.5, pp. 1066–1075.

Barber, Michael J, John W Clark, and Charles H Anderson (2003). "Neural representation of probabilistic information". In: *Neural Computation* 15.8, pp. 1843–1864.

Barreto, André, Will Dabney, Rémi Munos, Jonathan J Hunt, Tom Schaul, Hado P van Hasselt, and David Silver (2017). "Successor features for transfer in reinforcement learning". In: *Advances in neural information processing systems*, pp. 4055–4065.

Barry, Caswell, Lin Lin Ginzberg, John O'Keefe, and Neil Burgess (2012). "Grid cell firing patterns signal environmental novelty by expansion". In: *Proceedings of the National Academy of Sciences* 109.43, pp. 17687–17692.

Barry, Caswell, Colin Lever, Robin Hayman, Tom Hartley, Stephen Burton, John O'Keefe, Kate Jeffery, and N Burgess (2006). "The boundary vector cell model of place cell firing and spatial memory". In: *Reviews in the Neurosciences* 17.1-2, pp. 71–98.

Bartlett, Madeleine, Nicole Sandra-Yaffa Dumont, Michael P Furlong, and Terrence C Stewart (2022). "Biologically-Plausible Memory for Continuous-Time Reinforcement Learning". In.

Bartlett, Madeleine, Michael Furlong, Terrence C Stewart, and Jeff Orchard (2023a). "Using Vector Symbolic Architectures for Distributed Action Representations in a Spiking Model of the Basal Ganglia". In: *Proceedings of the Annual Meeting of the Cognitive Science Society*. Vol. 46.

Bartlett, Madeleine, Kathryn Simone, ND Dumont, M Furlong, Chris Eliasmith, Jeff Orchard, and T Stewart (2023b). "Improving reinforcement learning with biologically motivated continuous state representations". In: *Proceedings of the 21st International Conference on Cognitive Modeling*.

Barto, Andrew G, Richard S Sutton, and Charles W Anderson (1983). "Neuronlike adaptive elements that can solve difficult learning control problems". In: *IEEE transactions on systems, man, and cybernetics* 5, pp. 834–846.

Bekolay, Trevor, James Bergstra, Eric Hunsberger, Travis DeWolf, Terrence C Stewart, Daniel Rasmussen, Xuan Choo, Aaron Russell Voelker, and Chris Eliasmith (2014). "Nengo: a Python tool for building large-scale functional brain models". In: *Frontiers in neuroinformatics* 7, p. 48.

Bekolay, Trevor, Carter Kolbeck, and Chris Eliasmith (2013). "Simultaneous unsupervised and supervised learning of cognitive functions in biologically plausible spiking neural networks". In: *Proceedings of the Annual Meeting of the Cognitive Science Society*. Vol. 35. 35.

Bellec, Guillaume, Franz Scherr, Anand Subramoney, Elias Hajek, Darjan Salaj, Robert Legenstein, and Wolfgang Maass (2020). "A solution to the learning dilemma for recurrent networks of spiking neurons". In: *Nature communications* 11.1, p. 3625.

Bellemare, Marc G, Will Dabney, and Rémi Munos (2017). "A distributional perspective on reinforcement learning". In: *International conference on machine learning*. PMLR, pp. 449–458.

Bellmund, Jacob LS, Peter Gärdenfors, Edvard I Moser, and Christian F Doeller (2018). "Navigating cognition: Spatial codes for human thinking". In: *Science* 362.6415, eaat6766.

Benhamou, Simon (1997). "Path integration by swimming rats". In: *Animal Behaviour* 54.2, pp. 321–327.

Bernardi, Silvia, Marcus K Benna, Mattia Rigotti, Jérôme Munuera, Stefano Fusi, and C Daniel Salzman (2020). "The geometry of abstraction in the hippocampus and prefrontal cortex". In: *Cell* 183.4, pp. 954–967.

Bersuker, Gennadi, Maribeth Mason, and Karen L Jones (2018). "Neuromorphic computing: The potential for high-performance processing in space". In: *Game Changer*, pp. 1–12.

Bhuyan, Bikram Pratim, Amar Ramdane-Cherif, Ravi Tomar, and TP Singh (2024). "Neuro-symbolic artificial intelligence: a survey". In: *Neural Computing and Applications*, pp. 1–36.

Bingman, Verner P, Gerald E Hough II, Meghan C Kahn, and Jennifer J Siegel (2003). "The homing pigeon hippocampus and space: in search of adaptive specialization". In: *Brain Behavior and Evolution* 62.2, pp. 117–127.

Bircher, Andreas, Mina Kamel, Kostas Alexis, Helen Oleynikova, and Roland Siegwart (2016). "Receding horizon" next-best-view" planner for 3d exploration". In: *2016 IEEE international conference on robotics and automation (ICRA)*. IEEE, pp. 1462–1468.

Blair, Hugh T, Kishan Gupta, and Kechen Zhang (2008). "Conversion of a phase-to a rate-coded position signal by a three-stage model of theta cells, grid cells, and place cells". In: *Hippocampus* 18.12, pp. 1239–1255.

Blouw, Peter, Eugene Solodkin, Paul Thagard, and Chris Eliasmith (2016). "Concepts as semantic pointers: A framework and computational model". In: *Cognitive science* 40.5, pp. 1128–1162.

Bobier, Bruce, Terrence C Stewart, and Chris Eliasmith (2014). "A unifying mechanistic model of selective attention in spiking neurons." In: *PLoS computational biology* 10.6. Ed. by Wolfgang Einhäuser, e1003577. ISSN: 1553-7358. DOI: 10.1371/journal.pcbi.1003577.

Boccara, Charlotte N, Francesca Sargolini, Veslemøy Hult Thoresen, Trygve Solstad, Menno P Witter, Edvard I Moser, and May-Britt Moser (2010). "Grid cells in pre-and parasubiculum". In: *Nature neuroscience* 13.8, pp. 987–994.

Bochner, Salomon (2005). *Harmonic analysis and the theory of probability*. Courier Corporation.

Bono, Jacopo, Sara Zannone, Victor Pedrosa, and Claudia Clopath (2023). "Learning predictive cognitive maps with spiking neurons during behavior and replays". In: *Elife* 12, e80671.

Bowman, Sean L (2022). "Semantic Simultaneous Localization and Mapping". PhD thesis. University of Pennsylvania.

Brady, Timothy F, Talia Konkle, George A Alvarez, and Aude Oliva (2008). "Visual long-term memory has a massive storage capacity for object details". In: *Proceedings of the National Academy of Sciences* 105.38, pp. 14325–14329.

Brellmann, David, David Filliat, and Goran Frehse (2023). "Fourier features in reinforcement learning with neural networks". In: *Transactions on Machine Learning Research*.

Bremer, Alicia and Jeff Orchard (2024). "Improved Cleanup and Decoding of Fractional Power Encodings". In: *arXiv preprint arXiv:2412.00488*.

Brossard, Martin, Silvere Bonnabel, and Axel Barrau (2018). "Invariant Kalman filtering for visual inertial SLAM". In: *2018 21st International Conference on Information Fusion (FUSION)*. IEEE, pp. 2021–2028.

Brown, Thackery I, Robert S Ross, Joseph B Keller, Michael E Hasselmo, and Chantal E Stern (2010). "Which way was I going? Contextual retrieval supports the disambiguation of well learned overlapping navigational routes". In: *Journal of Neuroscience* 30.21, pp. 7414–7422.

Brun, Vegard Heimly, Trygve Solstad, Kirsten Brun Kjelstrup, Marianne Fyhn, Menno P Witter, Edvard I Moser, and May-Britt Moser (2008). "Progressive increase in grid scale from dorsal to ventral medial entorhinal cortex". In: *Hippocampus* 18.12, pp. 1200–1212.

Burak, Yoram, Ted Brookings, and Ila Fiete (2006). "Triangular lattice neurons may implement an advanced numeral system to precisely encode rat position over large ranges". In: *arXiv preprint q-bio/0606005*.

Burak, Yoram and Ila R Fiete (2009). "Accurate path integration in continuous attractor network models of grid cells". In: *PLoS Comput Biol* 5.2, e1000291.

Burgess, Christopher P and Neil Burgess (2014). "Controlling phase noise in oscillatory interference models of grid cell firing". In: *Journal of Neuroscience* 34.18, pp. 6224–6232.

Burgess, Neil (2008). "Grid cells and theta as oscillatory interference: theory and predictions". In: *Hippocampus* 18.12, pp. 1157–1174.

Burgess, Neil, Caswell Barry, and John O'Keefe (2007). "An oscillatory interference model of grid cell firing". In: *Hippocampus* 17.9.

Burgess, Neil, Eleanor A Maguire, and John O'Keefe (2002). "The human hippocampus and spatial and episodic memory". In: *Neuron* 35.4, pp. 625–641.

Burgess, Neil, Michael Recce, and John O'Keefe (1994). "A model of hippocampal function". In: *Neural networks* 7.6-7, pp. 1065–1081.

Bush, Daniel and Neil Burgess (2014). "A hybrid oscillatory interference/continuous attractor network model of grid cell firing". In: *Journal of Neuroscience* 34.14, pp. 5065–5079.

Buzsáki, György (2005). "Theta rhythm of navigation: link between path integration and landmark navigation, episodic and semantic memory". In: *Hippocampus* 15.7, pp. 827–840.

Buzsáki, György and David Tingley (2023). "Cognition from the body-brain partnership: exaptation of memory". In: *Annual review of neuroscience* 46.1, pp. 191–210.

Calton, Jeffrey L and Jeffrey S Taube (2009). "Where am I and how will I get there from here? A role for posterior parietal cortex in the integration of spatial information and route planning". In: *Neurobiology of learning and memory* 91.2, pp. 186–196.

Canto, Cathrin B, Noriko Koganezawa, Prateep Beed, Edvard I Moser, and Menno P Witter (2012). "All layers of medial entorhinal cortex receive presubicular and parasubicular inputs". In: *Journal of Neuroscience* 32.49, pp. 17620–17631.

Cao, Yongqiang, Yang Chen, and Deepak Khosla (2015). "Spiking deep convolutional neural networks for energy-efficient object recognition". In: *International Journal of Computer Vision* 113, pp. 54–66.

Chakravarthy, V Srinivasa, Denny Joseph, and Raju S Bapi (2010). "What do the basal ganglia do? A modeling perspective". In: *Biological cybernetics* 103, pp. 237–253.

Chalmers, Eric, Artur Luczak, and Aaron J Gruber (2016). "Computational properties of the hippocampus increase the efficiency of goal-directed foraging through hierarchical reinforcement learning". In: *Frontiers in computational neuroscience* 10, p. 128.

Chaudhuri, Rishidev, Berk Gerçek, Biraj Pandey, Adrien Peyrache, and Ila Fiete (2019). "The intrinsic attractor manifold and population dynamics of a canonical cognitive circuit across waking and sleep". In: *Nature neuroscience* 22.9, pp. 1512–1520.

Chen, Ding, Peixi Peng, Tiejun Huang, and Yonghong Tian (2024). "Fully spiking actor network with intralayer connections for reinforcement learning". In: *IEEE Transactions on Neural Networks and Learning Systems*.

Chen, Guifen, Daniel Manson, Francesca Cacucci, and Thomas Joseph Wills (2016). "Absence of visual input results in the disruption of grid cell firing in the mouse". In: *Current Biology* 26.17, pp. 2335–2342.

Chen, Xiaoli, Timothy P McNamara, Jonathan W Kelly, and Thomas Wolbers (2017). "Cue combination in human spatial navigation". In: *Cognitive Psychology* 95, pp. 105–144.

Chen, Zhe Sage, Xiaohan Zhang, Xiaoyang Long, and Sheng-Jia Zhang (2022). "Are grid-like representations a component of all perception and cognition?" In: *Frontiers in Neural Circuits* 16, p. 924016.

Chevalier-Boisvert, Maxime, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio (2018a). "Babyai: A platform to study the sample efficiency of grounded language learning". In: *arXiv preprint arXiv:1810.08272*.

Chevalier-Boisvert, Maxime, Lucas Willems, and Suman Pal (2018b). *Minimalistic Gridworld Environment for OpenAI Gym*. `https://github.com/maximecb/gym-minigrid`.

Chilkuri, Narsimha, Eric Hunsberger, Aaron Voelker, Gurshaant Malik, and Chris Eliasmith (2021a). "Language modeling using LMUs: 10x better data efficiency or improved scaling compared to transformers". In: *arXiv preprint arXiv:2110.02402*.

Chilkuri, Narsimha Reddy and Chris Eliasmith (2021b). "Parallelizing legendre memory unit training". In: *International Conference on Machine Learning*. PMLR, pp. 1898–1907.

Choi, Yunho, Kyungjae Lee, and Songhwai Oh (2019). "Distributional deep reinforcement learning with a mixture of Gaussians". In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 9791–9797.

Choo, Feng-Xuan (2018). "Spaun 2.0: Extending the world's largest functional brain model". In.

Choo, Feng-Xuan and Chris Eliasmith (2010). "A spiking neuron model of serial-order recall". In: *Proceedings of the annual meeting of the cognitive science society*. Vol. 32. 32.

Citri, Ami and Robert C Malenka (2008). "Synaptic plasticity: multiple forms, functions, and mechanisms". In: *Neuropsychopharmacology* 33.1, pp. 18–41.

Clarke, Alex and Lorraine K Tyler (2014). "Object-specific semantic coding in human perirhinal cortex". In: *Journal of Neuroscience* 34.14, pp. 4766–4775.

Clarkson, Kenneth L, Shashanka Ubaru, and Elizabeth Yang (2023). "Capacity analysis of vector symbolic architectures". In: *arXiv preprint arXiv:2301.10352*.

Cohen, NJ (1993). *Memory, amnesia and the hippocampal system*. MIT Press.

Conklin, John and Chris Eliasmith (2005). "A controlled attractor network model of path integration in the rat". In: *Journal of computational neuroscience* 18.2, pp. 183–203.

Constantinescu, Alexandra O., Jill X. O'Reilly, and Timothy E. J. Behrens (2016). "Organizing conceptual knowledge in humans with a gridlike code". In: *Science* 352.6292.

Constantinidis, Christos and Torkel Klingberg (2016). "The neuroscience of working memory capacity and training". In: *Nature Reviews Neuroscience* 17.7, pp. 438–449.

Cothi, William de and Caswell Barry (2020). "Neurobiological successor features for spatial navigation". In: *Hippocampus* 30.12, pp. 1347–1355.

Courellis, Hristos S et al. (2024). "Abstract representations emerge in human hippocampal neurons during inference". In: *Nature* 632.8026, pp. 841–849.

Coutureau, Etienne and Georges Di Scala (2009). "Entorhinal cortex and cognition". In: *Progress in Neuro-Psychopharmacology and Biological Psychiatry* 33.5, pp. 753–761.

Covington, Natalie V, Sarah Brown-Schmidt, and Melissa C Duff (2018). "The necessity of the hippocampus for statistical learning". In: *Journal of cognitive neuroscience* 30.5, pp. 680–697.

Cowan, Nelson (2001). "The magical number 4 in short-term memory: A reconsideration of mental storage capacity". In: *Behavioral and brain sciences* 24.1, pp. 87–114.

Cueva, Christopher J and Xue-Xin Wei (2018). "Emergence of grid-like representations by training recurrent neural networks to perform spatial localization". In.

Cunningham, John P and Byron M Yu (2014). "Dimensionality reduction for large-scale neural recordings". In: *Nature neuroscience* 17.11, pp. 1500–1509.

Dabney, Will, Zeb Kurth-Nelson, Naoshige Uchida, Clara Kwon Starkweather, Demis Hassabis, Rémi Munos, and Matthew Botvinick (2020). "A distributional code for value in dopamine-based reinforcement learning". In: *Nature* 577.7792, pp. 671–675.

Dabney, Will, Mark Rowland, Marc Bellemare, and Rémi Munos (2018). "Distributional reinforcement learning with quantile regression". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 32. 1.

Dang, Suogui, Yining Wu, Rui Yan, and Huajin Tang (2021). "Why grid cells function as a metric for space". In: *Neural Networks* 142, pp. 128–137.

Dannenberg, Holger, Hallie Lazaro, Pranav Nambiar, Alec Hoyland, and Michael E Hasselmo (2020). "Effects of visual inputs on neural dynamics for coding of location and running speed in medial entorhinal cortex". In: *Elife* 9, e62500.

Darshan, Ran and Alexander Rivkind (2022). "Learning to represent continuous variables in heterogeneous neural networks". In: *Cell Reports* 39.1, p. 110612.

Davies, Mike, Andreas Wild, Garrick Orchard, Yulia Sandamirskaya, Gabriel A Fonseca Guerra, Prasad Joshi, Philipp Plank, and Sumedh R Risbud (2021). "Advancing neuromorphic computing with loihi: A survey of results and outlook". In: *Proceedings of the IEEE* 109.5, pp. 911–934.

Dayan, Peter (1993). "Improving generalization for temporal difference learning: The successor representation". In: *Neural Computation* 5.4, pp. 613–624.

Deshmukh, Sachin S and James J Knierim (2013). "Influence of local objects on hippocampal representations: Landmark vectors and memory". In: *Hippocampus* 23.4, pp. 253–267.

DeWolf, Travis and Chris Eliasmith (2011). "A spiking neuron model of movement and pre-movement activity in M1". In: *Cognitive and Systems Neuroscience*. Salt Lake City, UT.

DeWolf, Travis, Terrence C Stewart, Jean-Jacques Slotine, and Chris Eliasmith (2016). "A spiking neural model of adaptive arm control". In: *Proceedings of the Royal Society B: Biological Sciences* 283.1843, p. 20162134.

Dezfouli, Amir and Bernard W Balleine (2013). "Actions, action sequences and habits: evidence that goal-directed and habitual action control are hierarchically organized". In: *PLoS computational biology* 9.12, e1003364.

Dhanraj, Varun and Chris Eliasmith (2025). "Improving Rule-based Reasoning in LLMs via Neurosymbolic Representations". In: *arXiv preprint arXiv:2502.01657*.

DiCarlo, James J and David D Cox (2007). "Untangling invariant object recognition". In: *Trends in cognitive sciences* 11.8, pp. 333–341.

Doeller, Christian F, Caswell Barry, and Neil Burgess (2010). "Evidence for grid cells in a human memory network". In: *Nature* 463.7281, pp. 657–661.

Dolgov, Sergey, Dante Kalise, and Karl K Kunisch (2021). "Tensor decomposition methods for high-dimensional Hamilton–Jacobi–Bellman equations". In: *SIAM Journal on Scientific Computing* 43.3, A1625–A1650.

Dordek, Yedidyah, Daniel Soudry, Ron Meir, and Dori Derdikman (2016). "Extracting grid cell characteristics from place cell inputs using non-negative principal component analysis". In: *Elife* 5, e10094.

Dossa, Rousslan Fernand Julien and Takashi Matsubara (2022). "Toward Human Cognition-inspired High-Level Decision Making For Hierarchical Reinforcement Learning Agents". In: *Decision Awareness in Reinforcement Learning Workshop at ICML 2022*.

Doya, Kenji (2000). "Reinforcement learning in continuous time and space". In: *Neural computation* 12.1, pp. 219–245.

Du, Jianzhun, Joseph Futoma, and Finale Doshi-Velez (2020). "Model-based reinforcement learning for semi-markov decision processes with neural odes". In: *Advances in Neural Information Processing Systems* 33, pp. 19805–19816.

Duggins, Peter (2023). "Learning and Decision Making in Social Contexts: Neural and Computational Models". PhD thesis. Waterloo, Canada.

Duggins, Peter and Chris Eliasmith (2024). "A scalable spiking amygdala model that explains fear conditioning, extinction, renewal and generalization". In: *European Journal of Neuroscience*.

Duggins, Peter, Terrence C Stewart, and Chris Eliasmith (2022). "Reinforcement Learning, Social Value Orientation, and Decision Making: Computational Models and Empirical Validation". In: *Proceedings of the Annual Meeting of the Cognitive Science Society*. Vol. 44. 44. URL: https://escholarship.org/uc/item/1b30p4sw.

Duggins, Peter, Terrence C. Stewart, Xuan Choo, and Chris Eliasmith (2017). "Effects of Guanfacine and Phenylephrine on a Spiking Neuron Model of Working Memory". In: *Topics in Cognitive Science*. ISSN: 1756-8757. DOI: 10.1111/tops.12247.

Dumont, Nicole Sandra-Yaffa and Chris Eliasmith (2020). "Accurate representation for spatial cognition using grid cells". In: *42nd Annual Meeting of the Cognitive Science Society*. Toronto, ON: Cognitive Science Society, pp. 2367–2373.

Dumont, Nicole Sandra-Yaffa, P Michael Furlong, Jeff Orchard, and Chris Eliasmith (2023a). "Exploiting semantic information in a spiking neural SLAM system". In: *Frontiers in Neuroscience* 17, p. 1190515.

Dumont, Nicole Sandra-Yaffa, Jeff Orchard, and Chris Eliasmith (2022). "A model of path integration that connects neural and symbolic representation". In: *Proceedings of the Annual Meeting of the Cognitive Science Society*. Vol. 44. 44. Toronto, ON: Cognitive Science Society.

Dumont, Nicole Sandra-Yaffa, Andreas Stöckel, P Michael Furlong, Madeleine Bartlett, Chris Eliasmith, and Terrence C Stewart (2023b). "Biologically-Based Computation: How Neural Details and Dynamics Are Suited for Implementing a Variety of Algorithms". In: *Brain Sciences* 13.2, p. 245.

Eacott, MJ and EA Gaffan (2005). "The roles of perirhinal cortex, postrhinal cortex, and the fornix in memory for objects, contexts, and events in the rat". In: *The Quarterly Journal of Experimental Psychology Section B* 58.3-4b, pp. 202–217.

Eichenbaum, Howard (1992). "The hippocampal system and declarative memory in animals". In: *Journal of Cognitive Neuroscience* 4.3, pp. 217–231.

— (2000). "A cortical–hippocampal system for declarative memory". In: *Nature reviews neuroscience* 1.1, pp. 41–50.

— (2001). “The hippocampus and declarative memory: cognitive mechanisms and neural codes”. In: *Behavioural brain research* 127.1-2, pp. 199–207.
— (2014). “Time cells in the hippocampus: a new dimension for mapping memories”. In: *Nature Reviews Neuroscience* 15.11, pp. 732–744.
— (2017). “Prefrontal–hippocampal interactions in episodic memory”. In: *Nature Reviews Neuroscience* 18.9, pp. 547–558.

Eliasmith, Chris (2005a). “A unified approach to building and controlling spiking attractor networks”. In: *Neural computation* 17.6, pp. 1276–1314.
— (2005b). “Cognition with neurons: A large-scale, biologically realistic model of the Wason task”. In: *Proceedings of the Annual Meeting of the Cognitive Science Society*. Vol. 27. 27.
— (2013). *How to build a brain: A neural architecture for biological cognition*. Oxford University Press.

Eliasmith, Chris and Charles H Anderson (2003). “Neural engineering”. In: *Massachusetts Institue of Technology*.

Eliasmith, Chris, Terrence C Stewart, Xuan Choo, Trevor Bekolay, Travis DeWolf, Yichuan Tang, and Daniel Rasmussen (2012). “A large-scale model of the functioning brain”. In: *science* 338.6111, pp. 1202–1205.

Epstein, Russell and Nancy Kanwisher (1998). “A cortical representation of the local visual environment”. In: *Nature* 392.6676, pp. 598–601.

Ermentrout, Bard and David H Terman (2010). *Mathematical foundations of neuroscience*. Vol. 35. Springer.

Etienne, Ariane S, Roland Maurer, Valérie Boulens, Arik Levy, and Tiffany Rowe (2004). “Resetting the path integrator: a basic condition for route-based navigation”. In: *Journal of Experimental Biology* 207.9, pp. 1491–1508.

Etienne, Ariane S, Roland Maurer, and Valérie Séguinot (1996). “Path integration in mammals and its interaction with visual landmarks”. In: *Journal of experimental Biology* 199.1, pp. 201–209.

Etienne, AS (1987). “The control of short-distance homing in the golden hamster”. In: *Cognitive processes and spatial orientation in animal and man*. Springer, pp. 233–251.

Fang, Ching, Dmitriy Aronov, LF Abbott, and Emily L Mackevicius (2023). “Neural learning rules for generating flexible predictions and computing the successor representation”. In: *elife* 12, e80680.

Feigenbaum, Janet D and Edmund T Rolls (1991). “Allocentric and egocentric spatial information processing in the hippocampal formation of the behaving primate”. In: *Psychobiology* 19.1, pp. 21–40.

Finkelstein, Arseny, Dori Derdikman, Alon Rubin, Jakob N Foerster, Liora Las, and Nachum Ulanovsky (2015). "Three-dimensional head-direction coding in the bat brain". In: *Nature* 517.7533, pp. 159–164.

Fischler-Ruiz, Walter, David G Clark, Narendra R Joshi, Virginia Devi-Chou, Lacey Kitch, Mark Schnitzer, Larry F Abbott, and Richard Axel (2021). "Olfactory landmarks and path integration converge to form a cognitive spatial map". In: *Neuron* 109.24, pp. 4036–4049.

Fon, Edward A and Robert H Edwards (2001). "Molecular mechanisms of neurotransmitter release". In: *Muscle & Nerve: Official Journal of the American Association of Electrodiagnostic Medicine* 24.5, pp. 581–601.

Frady, E Paxon, Spencer J Kent, Bruno A Olshausen, and Friedrich T Sommer (2020). "Resonator networks, 1: An efficient solution for factoring high-dimensional, distributed representations of data structures". In: *Neural computation* 32.12, pp. 2311–2331.

Frady, E Paxon, Denis Kleyko, Christopher J Kymn, Bruno A Olshausen, and Friedrich T Sommer (2022). "Computing on functions using randomized vector representations (in brief)". In: *Proceedings of the 2022 Annual Neuro-Inspired Computational Elements Conference*, pp. 115–122.

Freche, Dominik, Ulrike Pannasch, Nathalie Rouach, and David Holcman (2011). "Synapse geometry and receptor dynamics modulate synaptic strength". In: *PloS one* 6.10, e25122.

Frémaux, Nicolas, Henning Sprekeler, and Wulfram Gerstner (2013). "Reinforcement learning using a continuous time actor-critic framework with spiking neurons". In: *PLoS computational biology* 9.4, e1003024.

Frost, Duncan P, Olaf Kähler, and David W Murray (2016). "Object-aware bundle adjustment for correcting monocular scale drift". In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 4770–4776.

Fu, Justin, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine (2020). "D4rl: Datasets for deep data-driven reinforcement learning". In: *arXiv preprint arXiv:2004.07219*.

Fuhs, Mark C and David S Touretzky (2006). "A spin glass model of path integration in rat medial entorhinal cortex". In: *Journal of Neuroscience* 26.16, pp. 4266–4276.

Fukawa, Ayako, Takahiro Aizawa, Hiroshi Yamakawa, and Ikuko Eguchi Yairi (2020). "Identifying core regions for path integration on medial entorhinal cortex of hippocampal formation". In: *Brain Sciences* 10.1, p. 28.

Funahashi, Shintaro (2017). "Working memory in the prefrontal cortex". In: *Brain sciences* 7.5, p. 49.

Furlong, P Michael and Chris Eliasmith (2023). "Modelling neural probabilistic computation using vector symbolic architectures". In: *Cognitive Neurodynamics*, pp. 1–24.

Furlong, P Michael, Katherine Simone, Nicole Dumont, Madeline Bartlett, Terrence C. Stewart, Jeff Orchard, and Chris Eliasmith (2024). "Biologically-plausible Markov Chain Monte Carlo Sampling from Vector Symbolic Algebra-encoded Distributions". In: *Artificial Neural Networks and Machine Learning – ICANN 2024*. Springer Nature Switzerland.

Furlong, P. Michael and Chris Eliasmith (2022). "Fractional Binding in Vector Symbolic Architectures as Quasi-Probability Statements". In: *44th Annual Meeting of the Cognitive Science Society*. Cognitive Science Society. URL: http://compneuro.uwaterloo.ca/files/publications/furlong.2022.pdf.

Furlong, P. Michael, Terrrence C. Stewart, and Chris Eliasmith (2021). "Fractional Binding in Vector Symbolic Representations for Efficient Mutual Information Exploration". In: *ICRA Workshop: Towards Curious Robots: Modern Approaches for Intrinsically-Motivated Intelligent Behavior*. URL: https://www.researchgate.net/publication/358485911_Fractional_Binding_in_Vector_Symbolic_Representations_for_Efficient_Mutual_Information_Exploration.

Fyhn, Marianne, Torkel Hafting, Alessandro Treves, May-Britt Moser, and Edvard I Moser (2007). "Hippocampal remapping and grid realignment in entorhinal cortex". In: *Nature* 446.7132, pp. 190–194.

El-Gaby, Mohamady et al. (2021). "An emergent neural coactivity code for dynamic memory". In: *Nature neuroscience* 24.5, pp. 694–704.

Gallego, Juan A, Matthew G Perich, Stephanie N Naufel, Christian Ethier, Sara A Solla, and Lee E Miller (2018). "Cortical population activity within a preserved neural manifold underlies multiple motor behaviors". In: *Nature communications* 9.1, p. 4233.

Gao, Ruiqi, Jianwen Xie, Song-Chun Zhu, and Ying Nian Wu (2018). "Learning grid cells as vector representation of self-position coupled with matrix representation of self-motion". In: *arXiv preprint arXiv:1810.05597*.

Gardner, Matthew PH, Geoffrey Schoenbaum, and Samuel J Gershman (2018). "Rethinking dopamine as generalized prediction error". In: *Proceedings of the Royal Society B* 285.1891, p. 20181645.

Gardner, Richard J, Erik Hermansen, Marius Pachitariu, Yoram Burak, Nils A Baas, Benjamin A Dunn, May-Britt Moser, and Edvard I Moser (2022). "Toroidal topology of population activity in grid cells". In: *Nature* 602.7895, pp. 123–128.

Garvert, Mona M, Raymond J Dolan, and Timothy EJ Behrens (2017). "A map of abstract relational knowledge in the human hippocampal–entorhinal cortex". In: *Elife* 6, e17086.

Gayler, Ross W (2004). "Vector symbolic architectures answer Jackendoff's challenges for cognitive neuroscience". In: *arXiv preprint cs/0412059*.

Gayler, Ross W and Simon D Levy (2009). "A distributed basis for analogical mapping". In: *New Frontiers in Analogy Research; Proc. of 2nd Intern. Analogy Conf.* Vol. 9.

Geiller, Tristan, Mohammad Fattahi, June-Seek Choi, and Sébastien Royer (2017). "Place cells are more strongly tied to landmarks in deep than in superficial CA1". In: *Nature communications* 8.1, p. 14531.

George, Dileep, Rajeev V Rikhye, Nishad Gothoskar, J Swaroop Guntupalli, Antoine Dedieu, and Miguel Lázaro-Gredilla (2021). "Clone-structured graph representations enable flexible learning and vicarious evaluation of cognitive maps". In: *Nature communications* 12.1, p. 2392.

George, Tom M, William de Cothi, Kimberly L Stachenfeld, and Caswell Barry (2023). "Rapid learning of predictive maps with STDP and theta phase precession". In: *Elife* 12, e80663.

Georgopoulos, Apostolos P, John F Kalaska, Roberto Caminiti, and Joe T Massey (1982). "On the relations between the direction of two-dimensional arm movements and cell discharge in primate motor cortex". In: *Journal of Neuroscience* 2.11, pp. 1527–1537.

Geromichalos, Dimitrios, Martin Azkarate, Emmanouil Tsardoulias, Levin Gerdes, Loukas Petrou, and Carlos Perez Del Pulgar (2020). "SLAM for autonomous planetary rovers with global localization". In: *Journal of Field Robotics* 37.5, pp. 830–847.

Gershman, Samuel J (2018). "The successor representation: its computational logic and neural substrates". In: *Journal of Neuroscience* 38.33, pp. 7193–7200.

Gershman, Samuel J, John A Assad, Sandeep Robert Datta, Scott W Linderman, Bernardo L Sabatini, Naoshige Uchida, and Linda Wilbrecht (2024). "Explaining dopamine through prediction errors and beyond". In: *Nature Neuroscience*, pp. 1–11.

Gil, Mariana, Mihai Ancau, Magdalene I Schlesiger, Angela Neitz, Kevin Allen, Rodrigo J De Marco, and Hannah Monyer (2018). "Impaired path integration in mice with disrupted grid cell firing". In: *Nature neuroscience* 21.1, pp. 81–91.

Gilchrist, Amanda L, Nelson Cowan, and Moshe Naveh-Benjamin (2008). "Working memory capacity for spoken sentences decreases with adult ageing: Recall of fewer but not smaller chunks in older adults". In: *Memory* 16.7, pp. 773–787.

Ginosar, Gily, Johnatan Aljadeff, Yoram Burak, Haim Sompolinsky, Liora Las, and Nachum Ulanovsky (2021). "Locally ordered representation of 3D space in the entorhinal cortex". In: *Nature* 596.7872, pp. 404–409.

Girardeau, Gabrielle, Karim Benchenane, Sidney I Wiener, György Buzsáki, and Michaël B Zugaro (2009). "Selective suppression of hippocampal ripples impairs spatial memory". In: *Nature neuroscience* 12.10, pp. 1222–1223.

Glad, Ingrid K, Nils Lid Hjort, and Nikolai Ushakov (2007). "Density estimation using the sinc kernel". In: *Preprint statistics* 2, p. 2007.

Glad, Ingrid K, Nils Lid Hjort, and Nikolai G Ushakov (2003). "Correction of density estimators that are not densities". In: *Scandinavian Journal of Statistics* 30.2, pp. 415–427.

Gläscher, Jan, Nathaniel Daw, Peter Dayan, and John P O'Doherty (2010). "States versus rewards: dissociable neural prediction error signals underlying model-based and model-free reinforcement learning". In: *Neuron* 66.4, pp. 585–595.

Gosmann, Jan and Chris Eliasmith (2016). "Optimizing semantic pointer representations for symbol-like processing in spiking neural networks". In: *PloS One* 11.2, e0149928.

— (2019). "Vector-derived transformation binding: An improved binding operation for deep symbol-like processing in neural networks". In: *Neural computation* 31.5, pp. 849–869.

— (2021). "CUE: A unified spiking neuron model of short-term and long-term memory". In: *Psychological Review* 128.1, pp. 104–124. DOI: 10.1037/rev0000250.

Grabska-Barwinska, Agnieszka, Jeff Beck, Alexandre Pouget, and Peter Latham (2013). "Demixing odors-fast inference in olfaction". In: *Advances in Neural Information Processing Systems* 26.

Grieves, Roddy M, Selim Jedidi-Ayoub, Karyna Mishchanchuk, Anyi Liu, Sophie Renaudineau, Éléonore Duvelle, and Kate J Jeffery (2021). "Irregular distribution of grid cell firing fields in rats exploring a 3D volumetric space". In: *Nature neuroscience* 24.11, pp. 1567–1573.

Griffin, Andrea S and Ariane S Etienne (1998). "Updating the path integrator through a visual fix". In: *Psychobiology* 26.3, pp. 240–248.

Grill, Jean-Bastien et al. (2020). "Bootstrap your own latent-a new approach to self-supervised learning". In: *Advances in neural information processing systems* 33, pp. 21271–21284.

Gu, Albert and Tri Dao (2023). "Mamba: Linear-time sequence modeling with selective state spaces". In: *arXiv preprint arXiv:2312.00752*.

Gu, Albert, Karan Goel, and Christopher Re (2022a). "Efficiently Modeling Long Sequences with Structured State Spaces". In: *International Conference on Learning Representations*.

Gu, Albert, Isys Johnson, Aman Timalsina, Atri Rudra, and Christopher Ré (2022b). "How to train your hippo: State space models with generalized orthogonal basis projections". In: *arXiv preprint arXiv:2206.12037*.

Guanella, Alexis, Daniel Kiper, and Paul Verschure (2007). "A model of grid cells based on a twisted torus topology". In: *International journal of neural systems* 17.04, pp. 231–240.

Hafting, Torkel, Marianne Fyhn, Sturla Molden, May-Britt Moser, and Edvard I Moser (2005). "Microstructure of a spatial map in the entorhinal cortex". In: *Nature* 436.7052, pp. 801–806.

Hardcastle, Kiah, Surya Ganguli, and Lisa M Giocomo (2015). "Environmental boundaries as an error correction mechanism for grid cells". In: *Neuron* 86.3, pp. 827–839.

Harootonian, Sevan K, Arne D Ekstrom, and Robert C Wilson (2022). "Combination and competition between path integration and landmark navigation in the estimation of heading direction". In: *PLoS computational biology* 18.2, e1009222.

Hasselmo, Michael E, Lisa M Giocomo, and Eric A Zilli (2007). "Grid cell firing may arise from interference of theta frequency membrane potential oscillations in single neurons". In: *Hippocampus* 17.12, pp. 1252–1271.

Heddes, Mike, Igor Nunes, Tony Givargis, Alexandru Nicolau, and Alex Veidenbaum (2022). "Hyperdimensional Hashing: A Robust and Efficient Dynamic Hash Table". In: *arXiv preprint arXiv:2205.07850*.

Hennequin, Guillaume, Laurence Aitchison, and Máté Lengyel (2014). "Fast sampling-based inference in balanced neuronal networks". In: *Advances in neural information processing systems* 27.

Hodgkin, Alan L and Andrew F Huxley (1952). "A quantitative description of membrane current and its application to conduction and excitation in nerve". In: *The Journal of physiology* 117.4, p. 500.

Hollerman, Jeffrey R and Wolfram Schultz (1998). "Dopamine neurons report an error in the temporal prediction of reward during learning". In: *Nature neuroscience* 1.4, pp. 304–309.

Hopfield, John J (1982). "Neural networks and physical systems with emergent collective computational abilities." In: *Proceedings of the national academy of sciences* 79.8, pp. 2554–2558.

Howard, Lorelei R, Amir Homayoun Javadi, Yichao Yu, Ravi D Mill, Laura C Morrison, Rebecca Knight, Michelle M Loftus, Laura Staskute, and Hugo J Spiers (2014). "The hippocampus and entorhinal cortex encode the path and Euclidean distances to goals during navigation". In: *Current Biology* 24.12, pp. 1331–1340.

Howard, Marc W (2017). "Temporal and spatial context in the mind and brain". In: *Current opinion in behavioral sciences* 17, pp. 14–19.

Høydal, Øyvind Arne, Emilie Ranheim Skytøen, Sebastian Ola Andersson, May-Britt Moser, and Edvard I Moser (2019). "Object-vector coding in the medial entorhinal cortex". In: *Nature* 568.7752, pp. 400–404.

Hsu, Hao-Lun and Miroslav Pajic (2024). "Robust exploration with adversary via Langevin Monte Carlo". In: *6th Annual Learning for Dynamics & Control Conference*. PMLR, pp. 1592–1605.

Hu, Edward J, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen (2021). "Lora: Low-rank adaptation of large language models". In: *arXiv preprint arXiv:2106.09685*.

Hubel, David H and Torsten N Wiesel (1962). "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex". In: *The Journal of physiology* 160.1, p. 106.

Humphries, Mark D (2022). "Basal ganglia: Mechanisms for action selection". In: *Encyclopedia of computational neuroscience*. Springer, pp. 410–415.

Hunsberger, Eric and Chris Eliasmith (2015). "Spiking Deep Networks with LIF Neurons". In: *arXiv:1510.08829*. URL: http://arxiv.org/abs/1510.08829.

Hussaini, Somayeh, Michael Milford, and Tobias Fischer (2022). "Spiking neural networks for visual place recognition via weighted neuronal assignments". In: *IEEE Robotics and Automation Letters* 7.2, pp. 4094–4101.

Inoue, Katsuma, Kohei Nakajima, and Yasuo Kuniyoshi (2020). "Designing spontaneous behavioral switching via chaotic itinerancy". In: *Science advances* 6.46, eabb3989.

Izhikevich, Eugene M (2007). "Solving the distal reward problem through linkage of STDP and dopamine signaling". In: *Cerebral cortex* 17.10, pp. 2443–2452.

Javadi, Amir-Homayoun, Beatrix Emo, Lorelei R Howard, Fiona E Zisch, Yichao Yu, Rebecca Knight, Joao Pinelo Silva, and Hugo J Spiers (2017). "Hippocampal and prefrontal processing of network topology to simulate the future". In: *Nature communications* 8.1, p. 14652.

Jayakumar, Ravikrishnan P, Manu S Madhav, Francesco Savelli, Hugh T Blair, Noah J Cowan, and James J Knierim (2019). "Recalibration of path integration in hippocampal place cells". In: *Nature* 566.7745, pp. 533–537.

Jazayeri, Mehrdad and Srdjan Ostojic (2021). "Interpreting neural computations by examining intrinsic and embedding dimensionality of neural activity". In: *Current opinion in neurobiology* 70, pp. 113–120.

Ji, Xiang (2014). "Generalized Strategies for Path Integration using Neural Oscillators". MA thesis. University of Waterloo.

Jia, Yanwei and Xun Yu Zhou (2022). "Policy gradient and actor-critic learning in continuous time and space: Theory and algorithms". In: *Journal of Machine Learning Research* 23.275, pp. 1–50.

Jin, Mu, Zhihao Ma, Kebing Jin, Hankz Hankui Zhuo, Chen Chen, and Chao Yu (2022). "Creativity of ai: Automatic symbolic option discovery for facilitating deep reinforcement learning". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 36. 6, pp. 7042–7050.

Joel, Daphna, Yael Niv, and Eytan Ruppin (2002). "Actor–critic models of the basal ganglia: New anatomical and computational perspectives". In: *Neural networks* 15.4-6, pp. 535–547.

Jones, Michael N and Douglas JK Mewhort (2007). "Representing word meaning and order information in a composite holographic lexicon." In: *Psychological review* 114.1, p. 1.

Jong, Joost de, Elkan Akyürek, and Hedderik van Rijn (2024). "A Unified Neurocomputational Model of Prospective and Retrospective Timing". In: *Psychological Review*.

Joshi, Aditya, Johan T Halseth, and Pentti Kanerva (2017). "Language geometry using random indexing". In: *Quantum Interaction: 10th International Conference, QI 2016, San Francisco, CA, USA, July 20-22, 2016, Revised Selected Papers 10*. Springer, pp. 265–274.

Kajić, Ivana, Tobias Schröder, Terrence C Stewart, and Paul Thagard (2019a). "The semantic pointer theory of emotion: Integrating physiology, appraisal, and construction". In: *Cognitive Systems Research* 58, pp. 35–53.

— (2019b). "The Semantic Pointer Theory of Emotion: Integrating Physiology, Appraisal, and Construction". In: *Cognitive Systems Research*. ISSN: 1389-0417. DOI: https://doi.org/10.1016/j.cogsys.2019.04.007.

Kamalaruban, Parameswaran, Yu-Ting Huang, Ya-Ping Hsieh, Paul Rolland, Cheng Shi, and Volkan Cevher (2020). "Robust Reinforcement Learning via Adversarial training with Langevin Dynamics". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin. Vol. 33. Curran Associates, Inc., pp. 8127–8138. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/5cb0e249689cd6d8369c4885435a56c2-Paper.pdf.

Kanerva, Pentti (1996). "Binary spatter-coding of ordered k-tuples". In: *International conference on artificial neural networks*. Springer, pp. 869–873.

Kanwisher, Nancy and Galit Yovel (2006). "The fusiform face area: a cortical region specialized for the perception of faces". In: *Philosophical Transactions of the Royal Society B: Biological Sciences* 361.1476, pp. 2109–2128.

Kapushev, Yermek, Anastasia Kishkun, Gonzalo Ferrer, and Evgeny Burnaev (2021). "Random fourier features based slam". In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 6597–6602.

Karamanis, Marios, Davide Zambrano, and Sander Bohté (2018). "Continuous-time spike-based reinforcement learning for working memory tasks". In: *Artificial Neural Networks and Machine Learning–ICANN 2018: 27th International Conference on Artificial Neural Networks, Rhodes, Greece, October 4-7, 2018, Proceedings, Part II 27*. Springer, pp. 250–262.

Kearns, Melissa J, William H Warren, Andrew P Duchon, and Michael J Tarr (2002). "Path integration from optic flow and body senses in a homing task". In: *Perception* 31.3, pp. 349–374.

Keinath, Alexandra T, Russell A Epstein, and Vijay Balasubramanian (2018). "Environmental deformations dynamically shift the grid cell spatial metric". In: *Elife* 7, e38169.

Kelly, Matthew A, Nipun Arora, Robert L West, and David Reitter (2020). "Holographic declarative memory: Distributional semantics as the architecture of memory". In: *Cognitive Science* 44.11, e12904.

Killian, Nathaniel J, Michael J Jutras, and Elizabeth A Buffalo (2012). "A map of visual space in the primate entorhinal cortex". In: *Nature* 491.7426, pp. 761–764.

Kim, Ayoung and Ryan M Eustice (2013). "Real-time visual SLAM for autonomous underwater hull inspection using visual saliency". In: *IEEE Transactions on Robotics* 29.3, pp. 719–733.

Kim, Seijoon, Seongsik Park, Byunggook Na, and Sungroh Yoon (2020a). "Spiking-yolo: spiking neural network for energy-efficient object detection". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 34. 07, pp. 11270–11277.

Kim, Soyoun, Dajung Jung, and Sébastien Royer (2020b). "Place cell maps slowly develop via competitive learning and conjunctive coding in the dentate gyrus". In: *Nature communications* 11.1, p. 4550.

Kleyko, Denis et al. (2022). "Vector symbolic architectures as a computing framework for emerging hardware". In: *Proceedings of the IEEE* 110.10, pp. 1538–1571.

Knierim, James J (2002). "Dynamic interactions between local surface cues, distal landmarks, and intrinsic circuitry in hippocampal place cells". In: *Journal of Neuroscience* 22.14, pp. 6254–6264.

Komer, Brent (2020). "Biologically Inspired Spatial Representation". PhD Thesis. Waterloo, ON: University of Waterloo. URL: https://uwspace.uwaterloo.ca/handle/10012/16430.

Komer, Brent, Terrence C. Stewart, Aaron R. Voelker, and Chris Eliasmith (2019). "A neural representation of continuous space using fractional binding". In: *41st Annual Meeting of the Cognitive Science Society*. Montreal, QC: Cognitive Science Society.

Kommers, Cody, Volkan Ustun, Abram Demski, and Paul Rosenbloom (2015). *Hierarchical Reasoning with Distributed Vector Representations*. Tech. rep. UNIVERSITY OF SOUTHERN CALIFORNIA LOS ANGELES.

Konidaris, George, Sarah Osentoski, and Philip Thomas (2011). "Value function approximation in reinforcement learning using the Fourier basis". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 25. 1, pp. 380–385.

Kreiser, Raphaela, Alpha Renner, Vanessa RC Leite, Baris Serhan, Chiara Bartolozzi, Arren Glover, and Yulia Sandamirskaya (2020a). "An on-chip spiking neural network for estimation of the head pose of the icub robot". In: *Frontiers in Neuroscience* 14, p. 551.

Kreiser, Raphaela, Gabriel Waibel, Nuria Armengol, Alpha Renner, and Yulia Sandamirskaya (2020b). "Error estimation and correction in a spiking neural network for map formation in neuromorphic hardware". In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, pp. 6134–6140.

Krupic, Julija, Marius Bauza, Stephen Burton, Colin Lever, and John O'Keefe (2014). "How environment geometry affects grid cell symmetry and what we can learn from it". In: *Philosophical Transactions of the Royal Society B: Biological Sciences* 369.1635, p. 20130188.

Krupic, Julija, Neil Burgess, and John O'Keefe (2012). "Neural representations of location composed of spatially periodic bands". In: *Science* 337.6096, pp. 853–857.

Kulkarni, Tejas D, Ardavan Saeedi, Simanta Gautam, and Samuel J Gershman (2016). "Deep successor reinforcement learning". In: *arXiv preprint arXiv:1606.02396*.

Kymn, Christopher J, Sonia Mazelet, Anthony Thomas, Denis Kleyko, E Paxon Frady, Friedrich T Sommer, and Bruno A Olshausen (2024). "Binding in hippocampal-entorhinal circuits enables compositionality in cognitive maps". In: *Advances in Neural Information Processing Systems*.

Landajuela, Mikel, Brenden K Petersen, Sookyung Kim, Claudio P Santiago, Ruben Glatt, Nathan Mundhenk, Jacob F Pettit, and Daniel Faissol (2021). "Discovering symbolic policies with deep reinforcement learning". In: *International Conference on Machine Learning*. PMLR, pp. 5979–5989.

Lateef, Fahad and Yassine Ruichek (2019). "Survey on semantic segmentation using deep learning techniques". In: *Neurocomputing* 338, pp. 321–348.

Laube, Ryan and Chris Eliasmith (2024). "QAVSA: Question Answering using Vector Symbolic Algebras". In: *Proceedings of the 9th Workshop on Representation Learning for NLP (RepL4NLP-2024)*, pp. 191–202.

Lee, Rachel S, Ben Engelhard, Ilana B Witten, and Nathaniel D Daw (2022). "A vector reward prediction error model explains dopaminergic heterogeneity". In: *bioRxiv*.

Lee, Sang Wan, Shinsuke Shimojo, and John P O'Doherty (2014). "Neural computations underlying arbitration between model-based and model-free learning". In: *Neuron* 81.3, pp. 687–699.

Lerner, Talia N, Ashley L Holloway, and Jillian L Seiler (2021). "Dopamine, updated: reward prediction error and beyond". In: *Current opinion in neurobiology* 67, pp. 123–130.

Leutgeb, Stefan, Jill K Leutgeb, Alessandro Treves, May-Britt Moser, and Edvard I Moser (2004). "Distinct ensemble codes in hippocampal areas CA3 and CA1". In: *Science*.

Lever, Colin, Stephen Burton, Ali Jeewajee, John O'Keefe, and Neil Burgess (2009). "Boundary vector cells in the subiculum of the hippocampal formation". In: *Journal of Neuroscience* 29.31, pp. 9771–9777.

Levitan, Irwin B and Leonard K Kaczmarek (2015). *The neuron: cell and molecular biology*. Oxford University Press, USA.

Li, Alexander and Deepak Pathak (2021). "Functional regularization for reinforcement learning via learned fourier features". In: *Advances in Neural Information Processing Systems* 34, pp. 19046–19055.

Li, Yiding et al. (2017). "A distinct entorhinal cortex to hippocampal CA1 direct circuit for olfactory associative learning". In: *Nature neuroscience* 20.4, pp. 559–570.

Liang, Zilu, Simeng Wu, Jie Wu, Wen-Xu Wang, Shaozheng Qin, and Chao Liu (2024). "Distance and grid-like codes support the navigation of abstract social space in the human brain". In: *Elife* 12, RP89025.

Libby, Laura A, Arne D Ekstrom, J Daniel Ragland, and Charan Ranganath (2012). "Differential connectivity of perirhinal and parahippocampal cortices within human hippocampal subregions revealed by high-resolution functional imaging". In: *Journal of Neuroscience* 32.19, pp. 6550–6560.

Lin, Dongyan, Ann Zixiang Huang, and Blake Aaron Richards (2023). "Temporal encoding in deep reinforcement learning agents". In: *Scientific Reports* 13.1, p. 22335.

Lin, Long-Ji (1992). "Self-improving reactive agents based on reinforcement learning, planning and teaching". In: *Machine learning* 8, pp. 293–321.

Liu, Can, Ralitsa Todorova, Wenbo Tang, Azahara Oliva, and Antonio Fernandez-Ruiz (2023). "Associative and predictive hippocampal codes support memory-guided behaviors". In: *Science* 382.6668, eadi8237.

Liu, Guisong, Wenjie Deng, Xiurui Xie, Li Huang, and Huajin Tang (2022). "Human-level control through directly trained deep spiking Q-networks". In: *IEEE transactions on cybernetics* 53.11, pp. 7187–7198.

Low, Ryan J, Sam Lewallen, Dmitriy Aronov, Rhino Nevers, and David W Tank (2018). "Probing variability in a cognitive map using manifold inference from neural dynamics". In: *BioRxiv*, p. 418939.

Lowet, Adam S, Qiao Zheng, Sara Matias, Jan Drugowitsch, and Naoshige Uchida (2020). "Distributional reinforcement learning in the brain". In: *Trends in neurosciences* 43.12, pp. 980–997.

Lu, Chris, Yannick Schroecker, Albert Gu, Emilio Parisotto, Jakob Foerster, Satinder Singh, and Feryal Behbahani (2024). "Structured state space models for in-context reinforcement learning". In: *Advances in Neural Information Processing Systems* 36.

Luo, Yudong, Guiliang Liu, Haonan Duan, Oliver Schulte, and Pascal Poupart (2021). "Distributional reinforcement learning with monotonic splines". In: *International Conference on Learning Representations*.

Ma, Chen, Dylan R. Ashley, Junfeng Wen, and Yoshua Bengio (2020). "Universal Successor Features for Transfer Reinforcement Learning". In: *Computing Research Repository (CoRR)* abs/2001.04025. URL: https://arxiv.org/abs/2001.04025.

Ma, Wei Ji, Mehrdad Jazayeri, et al. (2014). "Neural coding of uncertainty and probability". In: *Annual review of neuroscience* 37.1, pp. 205–220.

MacDonald, Christopher J, Kyle Q Lepage, Uri T Eden, and Howard Eichenbaum (2011). "Hippocampal "time cells" bridge the gap in memory for discontiguous events". In: *Neuron* 71.4, pp. 737–749.

MacNeil, David and Chris Eliasmith (2011). "Fine-tuning and the stability of recurrent neural networks". In: *PloS one* 6.9, e22885.

Madarasz, Tamas and Tim Behrens (2019). "Better transfer learning with inferred successor maps". In: *Advances in Neural Information Processing Systems*, pp. 9029–9040.

Marchette, Steven A, Lindsay K Vass, Jack Ryan, and Russell A Epstein (2015). "Outside looking in: landmark generalization in the human navigational system". In: *Journal of Neuroscience* 35.44, pp. 14896–14908.

Mathis, Alexander, Martin B Stemmler, and Andreas VM Herz (2015). "Probable nature of higher-dimensional symmetries underlying mammalian grid-cell activity patterns". In: *Elife* 4, e05979.

Matsumura, Michikazu, Dao-fen Chen, Toshiyuki Sawaguchi, Kisou Kubota, and Eberhard E Fetz (1996). "Synaptic interactions between primate precentral cortex neurons revealed by spike-triggered averaging of intracellular membrane potentials in vivo". In: *Journal of Neuroscience* 16.23, pp. 7757–7767.

McClelland, James L, Bruce L McNaughton, and Randall C O'Reilly (1995). "Why there are complementary learning systems in the hippocampus and neocortex: insights from the successes and failures of connectionist models of learning and memory." In: *Psychological review* 102.3, p. 419.

McCulloch, Warren S and Walter Pitts (1943). "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5.4, pp. 115–133.

McNaughton, Bruce L, Francesco P Battaglia, Ole Jensen, Edvard I Moser, and May-Britt Moser (2006). "Path integration and the neural basis of the 'cognitive map'". In: *Nature Reviews Neuroscience* 7.8, pp. 663–678.

Mehrali-Varjani, Mohsen, Mostafa Shamsi, and Alaeddin Malek (2018). "Solving a class of Hamilton-Jacobi-Bellman equations using pseudospectral methods". In: *Kybernetika* 54.4, pp. 629–647.

Mehta, Mayank R, Michael C Quirk, and Matthew A Wilson (2000). "Experience-dependent asymmetric shape of hippocampal receptive fields". In: *Neuron* 25.3, pp. 707–715.

Mikolov, Tomas, Kai Chen, Greg Corrado, and Jeffrey Dean (2013). "Efficient estimation of word representations in vector space". In: *arXiv preprint arXiv:1301.3781*.

Milford, Michael, Adam Jacobson, Zetao Chen, and Gordon Wyeth (2016). "RatSLAM: Using models of rodent hippocampus for robot navigation and beyond". In: *Robotics Research*. Springer, pp. 467–485.

Milford, Michael J, Gordon F Wyeth, and David Prasser (2004). "RatSLAM: a hippocampal model for simultaneous localization and mapping". In: *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004.* Vol. 1. IEEE, pp. 403–408.

Miller, George A (1956). "The magical number seven, plus or minus two: Some limits on our capacity for processing information." In: *Psychological review* 63.2, p. 81.

Mitchell, Anna S, Rafal Czajkowski, Ningyu Zhang, Kate Jeffery, and Andrew JD Nelson (2018). "Retrosplenial cortex and its role in spatial cognition". In: *Brain and neuroscience advances* 2, p. 2398212818757098.

Mitchell-Heggs, Rufus, Seigfred Prado, Giuseppe P Gava, Mary Ann Go, and Simon R Schultz (2023). "Neural manifold analysis of brain circuit dynamics in health and disease". In: *Journal of computational neuroscience* 51.1, pp. 1–21.

Mitchener, Ludovico, David Tuckey, Matthew Crosby, and Alessandra Russo (2022). "Detect, understand, act: A neuro-symbolic hierarchical reinforcement learning framework". In: *Machine Learning* 111.4, pp. 1523–1549.

Mitrokhin, Anton, Peter Sutor, Douglas Summers-Stay, Cornelia Fermüller, and Yiannis Aloimonos (2020). "Symbolic representation and learning with hyperdimensional computing". In: *Front Robot AI* 7.

Mittelstaedt, Horst and M-L Mittelstaedt (1982). "Homing by path integration". In: *Avian navigation*. Springer, pp. 290–297.

Mittelstaedt, Marie-Luise and Horst Mittelstaedt (2001). "Idiothetic navigation in humans: estimation of path length". In: *Experimental Brain Research* 139.3, pp. 318–332.

Mnih, Volodymyr et al. (2015). "Human-level control through deep reinforcement learning". In: *nature* 518.7540, pp. 529–533.

Mok, Robert M and Bradley C Love (2019). "A non-spatial account of place and grid cells based on clustering models of concept learning". In: *Nature communications* 10.1, p. 5685.

Montague, P Read, Peter Dayan, and Terrence J Sejnowski (1996). "A framework for mesencephalic dopamine systems based on predictive Hebbian learning". In: *Journal of neuroscience* 16.5, pp. 1936–1947.

Montemerlo, Michael, Sebastian Thrun, Daphne Koller, Ben Wegbreit, et al. (2002). "FastSLAM: A factored solution to the simultaneous localization and mapping problem". In: *Aaai/iaai* 593598.

Monti, Jim M, Gillian E Cooke, Patrick D Watson, Michelle W Voss, Arthur F Kramer, and Neal J Cohen (2014). "Relating hippocampus to relational memory processing across domains and delays". In: *Journal of cognitive neuroscience* 27.2, pp. 234–245.

Morere, Philippe and Fabio Ramos (2018). "Bayesian RL for goal-only rewards". In: *Conference on Robot Learning*. PMLR, pp. 386–398.

Mosheiff, Noga and Yoram Burak (2019). "Velocity coupling of grid cell modules enables stable embedding of a low dimensional variable in a high dimensional neural attractor". In: *Elife* 8, e48494.

Mozifian, Melissa, Dieter Fox, David Meger, Fabio Ramos, and Animesh Garg (2022). "Learning Successor Feature Representations to Train Robust Policies for Multi-task Learning". In: *Deep Reinforcement Learning Workshop NeurIPS 2022*.

Muller, Robert U and John L Kubie (1987). "The effects of changes in the environment on the spatial firing of hippocampal complex-spike cells". In: *Journal of Neuroscience* 7.7, pp. 1951–1968.

Muller, Timothy H, James L Butler, Sebastijan Veselic, Bruno Miranda, Joni D Wallis, Peter Dayan, Timothy EJ Behrens, Zeb Kurth-Nelson, and Steven W Kennerley (2024). "Distributional reinforcement learning in prefrontal cortex". In: *Nature Neuroscience* 27.3, pp. 403–408.

Munos, Rémi (2006). "Policy gradient in continuous time". In: *Journal of Machine Learning Research* 7, pp. 771–791.

Murdock Jr, Bennet B (1962). "The serial position effect of free recall." In: *Journal of experimental psychology* 64.5, p. 482.

Murray, Elisabeth A, Steven P Wise, and Kim S Graham (2017). *The Evolution of Memory Systems: ancestors, anatomy, and adaptations*. Oxford University Press.

Nachum, Ofir, Haoran Tang, Xingyu Lu, Shixiang Gu, Honglak Lee, and Sergey Levine (2019). "Why does hierarchy (sometimes) work so well in reinforcement learning?" In: *arXiv preprint arXiv:1909.10618*.

Neftci, Emre O and Bruno B Averbeck (2019). "Reinforcement learning in artificial and biological systems". In: *Nature Machine Intelligence* 1.3, pp. 133–143.

Nguyen-Tang, Thanh, Sunil Gupta, and Svetha Venkatesh (2021). "Distributional reinforcement learning via moment matching". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 10, pp. 9144–9152.

Nieh, Edward H et al. (2021). "Geometry of abstract learned knowledge in the hippocampus". In: *Nature* 595.7865, pp. 80–84.

Niv, Yael (2009). "Reinforcement learning in the brain". In: *Journal of Mathematical Psychology* 53.3, pp. 139–154.

Nyberg, Nils, Éléonore Duvelle, Caswell Barry, and Hugo J Spiers (2022). "Spatial goal coding in the hippocampal formation". In: *Neuron* 110.3, pp. 394–422.

O'Doherty, John, Peter Dayan, Johannes Schultz, Ralf Deichmann, Karl Friston, and Raymond J Dolan (2004). "Dissociable roles of ventral and dorsal striatum in instrumental conditioning". In: *science* 304.5669, pp. 452–454.

O'Keefe, John and Neil Burgess (2005). "Dual phase and rate coding in hippocampal place cells: theoretical significance and relationship to entorhinal grid cells". In: *Hippocampus* 15.7, pp. 853–866.

O'Keefe, John and Jonathan Dostrovsky (1971). "The hippocampus as a spatial map: Preliminary evidence from unit activity in the freely-moving rat." In: *Brain research*.

O'Keefe, John and Lynn Nadel (1978). *The hippocampus as a cognitive map*. Oxford university press.

O'Keefe, John, Lynn Nadel, Simon Keightley, and David Kill (1975). "Fornix lesions selectively abolish place learning in the rat". In: *Experimental neurology* 48.1, pp. 152–166.

Oja, Erkki (1982). "Simplified neuron model as a principal component analyzer". In: *Journal of mathematical biology* 15.3, pp. 267–273.

Ólafsdóttir, H Freyja, Daniel Bush, and Caswell Barry (2018). "The role of hippocampal replay in memory and planning". In: *Current Biology* 28.1, R37–R50.

Olsen, Rosanna K, Sandra N Moses, Lily Riggs, and Jennifer D Ryan (2012). "The hippocampus supports multiple cognitive processes through relational binding and comparison". In: *Frontiers in human neuroscience* 6, p. 146.

Orchard, Jeff (2015). "Oscillator-interference models of path integration do not require theta oscillations". In: *Neural computation* 27.3, pp. 548–560.

Orchard, Jeff, Hao Yang, and Xiang Ji (2013). "Does the entorhinal cortex use the Fourier transform?" In: *Frontiers in computational neuroscience* 7, p. 179.

Osswald, Marc, Sio-Hoi Ieng, Ryad Benosman, and Giacomo Indiveri (2017). "A spiking neural network model of 3D perception for event-based neuromorphic stereo vision systems". In: *Scientific reports* 7.1, p. 40703.

Palomeras, Narcís, Marc Carreras, and Juan Andrade-Cetto (2019). "Active SLAM for autonomous underwater exploration". In: *Remote Sensing* 11.23, p. 2827.

Pan, Zihan, Jibin Wu, Malu Zhang, Haizhou Li, and Yansong Chua (2019). "Neural population coding for effective temporal classification". In: *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE, pp. 1–8.

Pandarinath, Chethan et al. (2018). "Inferring single-trial neural population dynamics using sequential auto-encoders". In: *Nature methods* 15.10, pp. 805–815.

Park, Seongmin A, Douglas S Miller, and Erie D Boorman (2021). "Inferences on a multidimensional social hierarchy use a grid-like code". In: *Nature neuroscience* 24.9, pp. 1292–1301.

Parkin, Alan J (1996). "Human memory: the hippocampus is the key". In: *Current Biology* 6.12, pp. 1583–1585.

Parkinson, Carolyn, Shari Liu, and Thalia Wheatley (2014). "A common cortical metric for spatial, temporal, and social distance". In: *Journal of Neuroscience* 34.5, pp. 1979–1987.

Patai, Eva Zita and Hugo J Spiers (2021). "The versatile wayfinder: prefrontal contributions to spatial navigation". In: *Trends in cognitive sciences* 25.6, pp. 520–533.

Pathak, Deepak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell (2017). "Curiosity-driven exploration by self-supervised prediction". In: *International conference on machine learning*. PMLR, pp. 2778–2787.

Patra, Sunandita, Mark Cavolowsky, Onur Kulaksizoglu, Ruoxi Li, Laura Hiatt, Mark Roberts, and Dana Nau (2022). "A hierarchical goal-biased curriculum for training reinforcement learning". In: *The international FLAIRS conference proceedings*. Vol. 35.

Peng, Jing-Jie, Beate Throm, Maryam Najafian Jazi, Ting-Yun Yen, Hannah Monyer, and Kevin Allen (2023). "Grid cells perform path integration in multiple reference frames during self-motion-based navigation". In: *bioRxiv*, pp. 2023–12.

Pennartz, CMA, R Ito, PFMJ Verschure, FP Battaglia, and TW Robbins (2011). "The hippocampal–striatal axis in learning, prediction and goal-directed behavior". In: *Trends in neurosciences* 34.10, pp. 548–559.

Pennington, Jeffrey, Richard Socher, and Christopher D Manning (2014). "Glove: Global vectors for word representation". In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 1532–1543.

Penzkofer, Anna, Lei Shi, and Andreas Bulling (2024). "VSA4VQA: Scaling A Vector Symbolic Architecture To Visual Question Answering on Natural Images". In: *Proc. 46th Annual Meeting of the Cognitive Science Society (CogSci)*.

Pereira-Obilinovic, Ulises, Johnatan Aljadeff, and Nicolas Brunel (2023). "Forgetting leads to chaos in attractor networks". In: *Physical Review X* 13.1, p. 011009.

Peters-Founshtein, Gregory, Amnon Dafni-Merom, Rotem Monsa, and Shahar Arzy (2024). "Evidence for grid-cell-like activity in the time domain". In: *Neuropsychologia* 198, p. 108878.

Peyrache, Adrien, Marie M Lacroix, Peter C Petersen, and György Buzsáki (2015). "Internally organized mechanisms of the head direction sense". In: *Nature neuroscience* 18.4, pp. 569–575.

Plate, Tony A (1992). "Holographic recurrent networks". In: *Advances in neural information processing systems* 5.

— (1995). "Holographic reduced representations". In: *IEEE Transactions on Neural networks* 6.3, pp. 623–641.

Plebe, Alessio, Arianna Pavone, et al. (2020). "Neural Semantic Pointers in Context." In: *IJCCI*, pp. 447–454.

Potjans, Wiebke, Abigail Morrison, and Markus Diesmann (2009). "A spiking neural network model of an actor-critic learning agent". In: *Neural computation* 21.2, pp. 301–339.

Pouget, Alexandre, Peter Dayan, and Richard Zemel (2000). "Information processing with population codes". In: *Nature Reviews Neuroscience* 1.2, pp. 125–132.

Pritzel, Alexander, Benigno Uria, Sriram Srinivasan, Adria Puigdomenech Badia, Oriol Vinyals, Demis Hassabis, Daan Wierstra, and Charles Blundell (2017). "Neural episodic control". In: *International conference on machine learning*. PMLR, pp. 2827–2836.

Quirk, Gregory J, Robert U Muller, and John L Kubie (1990). "The firing of hippocampal place cells in the dark depends on the rat's recent experience". In: *Journal of Neuroscience* 10.6, pp. 2008–2017.

Rachkovskij, Dmitri A and Ernst M Kussul (2001). "Binding and normalization of binary sparse distributed representations by context-dependent thinning". In: *Neural Computation* 13.2, pp. 411–452.

Rahimi, Ali and Benjamin Recht (2007). "Random features for large-scale kernel machines". In: *Advances in neural information processing systems* 20.

Rajeswaran, Aravind, Kendall Lowrey, Emanuel V Todorov, and Sham M Kakade (2017). "Towards generalization and simplicity in continuous control". In: *Advances in neural information processing systems* 30.

Ranck, James B (1985). "Head direction cells in the deep cell layer of dorsolateral pre-subiculum in freely moving rats". In: *Electrical activity of the archicortex*.

Rasmussen, Daniel and Chris Eliasmith (2011). "A neural model of rule generation in inductive reasoning". In: *Topics in Cognitive Science* 3.1, pp. 140–153.

Rasmussen, Daniel, Aaron Voelker, and Chris Eliasmith (2017). "A neural model of hierarchical reinforcement learning". In: *PloS one* 12.7, e0180234.

Rathi, Nitin, Amogh Agrawal, Chankyu Lee, Adarsh Kumar Kosta, and Kaushik Roy (2021). "Exploring spike-based learning for neuromorphic computing: Prospects and perspectives". In: *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, pp. 902–907.

Reitman, Judith S (1974). "Without surreptitious rehearsal, information in short-term memory decay". In: *Journal of Verbal Learning and Verbal Behavior* 13.4, pp. 365–377.

Rempel-Clower, Nancy L, Stuart M Zola, Larry R Squire, and David G Amaral (1996). "Three cases of enduring memory impairment after bilateral damage limited to the hippocampal formation". In: *Journal of Neuroscience* 16.16, pp. 5233–5255.

Ren, Tongzheng, Tianjun Zhang, Lisa Lee, Joseph E Gonzalez, Dale Schuurmans, and Bo Dai (2022). "Spectral decomposition representation for reinforcement learning". In: *arXiv preprint arXiv:2208.09515*.

Ricker, Timothy J and Nelson Cowan (2010). "Loss of visual working memory within seconds: the combined use of refreshable and non-refreshable features." In: *Journal of Experimental Psychology: Learning, Memory, and Cognition* 36.6, p. 1355.

Rivest, Francois, John F Kalaska, and Yoshua Bengio (2014). "Conditioning and time representation in long short-term memory networks". In: *Biological cybernetics* 108, pp. 23–48.

Rodrıguez, Fernando, J Carlos López, J Pedro Vargas, Yolanda Gómez, Cristina Broglio, and Cosme Salas (2002). "Conservation of spatial memory function in the pallial forebrain of reptiles and ray-finned fishes". In: *Journal of Neuroscience* 22.7, pp. 2894–2903.

Rolls, Edmund T, Robert G Robertson, and Pierre Georges-François (1997). "Spatial view cells in the primate hippocampus". In: *European Journal of Neuroscience* 9.8, pp. 1789–1794.

Rolnick, David, Arun Ahuja, Jonathan Schwarz, Timothy Lillicrap, and Gregory Wayne (2019). "Experience replay for continual learning". In: *Advances in Neural Information Processing Systems* 32.

Rowland, David C, Yasser Roudi, May-Britt Moser, and Edvard I Moser (2016). "Ten years of grid cells". In: *Annual review of neuroscience* 39, pp. 19–40.

Safron, Adam, Ozan Catal, and Tim Verbelen (2022a). "Generalized Simultaneous Localization and Mapping (G-SLAM) as unification framework for natural and artificial intelligences: towards reverse engineering the hippocampal/entorhinal system and principles of high-level cognition". In: *Frontiers in Systems Neuroscience* 16.

Safron, Adam, Ozan Çatal, and Tim Verbelen (2022b). "Generalized Simultaneous Localization and Mapping (G-SLAM) as unification framework for natural and artificial intelligences: towards reverse engineering the hippocampal/entorhinal system and principles of high-level cognition". In: *Frontiers in Systems Neuroscience* 16. ISSN: 1662-5137. DOI: 10.3389/fnsys.2022.787659.

Salas, Cosme, Cristina Broglio, and Fernando Rodríguez (2003). "Evolution of forebrain and spatial cognition in vertebrates: conservation across diversity". In: *Brain Behavior and Evolution* 62.2, pp. 72–82.

Salinas, Emilio and LF Abbott (1994). "Vector reconstruction from firing rates". In: *Journal of computational neuroscience* 1.1, pp. 89–107.

Samsonovich, Alexei and Bruce L McNaughton (1997). "Path integration and cognitive mapping in a continuous attractor neural network model". In: *Journal of Neuroscience* 17.15, pp. 5900–5920.

Sargolini, Francesca, Marianne Fyhn, Torkel Hafting, Bruce L McNaughton, Menno P Witter, May-Britt Moser, and Edvard I Moser (2006). "Conjunctive representation of position, direction, and velocity in entorhinal cortex". In: *Science* 312.5774, pp. 758–762.

Save, Etienne, Ludek Nerad, and Bruno Poucet (2000). "Contribution of multiple sensory information to place field stability in hippocampal place cells". In: *Hippocampus* 10.1, pp. 64–76.

Savelli, Francesco and James J Knierim (2019). "Origin and role of path integration in the cognitive representations of the hippocampus: computational insights into open questions". In: *Journal of Experimental Biology* 222.Suppl_1, jeb188912.

Savin, Cristina and Sophie Denève (2014). "Spatio-temporal representations of uncertainty in spiking neural networks". In: *Advances in Neural Information Processing Systems* 27.

Schaeffer, Rylan, Mikail Khona, and Ila Fiete (2022). "No free lunch from deep learning in neuroscience: A case study through models of the entorhinal-hippocampal circuit". In: *Advances in neural information processing systems* 35, pp. 16052–16067.

Schaeffer, Rylan, Mikail Khona, Tzuhsuan Ma, Cristobal Eyzaguirre, Sanmi Koyejo, and Ila Fiete (2024). "Self-supervised learning of representations for space generates multi-modular grid cells". In: *Advances in Neural Information Processing Systems* 36.

Schapiro, Anna C, Emma Gregory, Barbara Landau, Michael McCloskey, and Nicholas B Turk-Browne (2014). "The necessity of the medial temporal lobe for statistical learning". In: *Journal of cognitive neuroscience* 26.8, pp. 1736–1747.

Schapiro, Anna C, Nicholas B Turk-Browne, Matthew M Botvinick, and Kenneth A Norman (2017). "Complementary learning systems within the hippocampus: a neural network modelling approach to reconciling episodic memory with statistical learning". In: *Philosophical Transactions of the Royal Society B: Biological Sciences* 372.1711, p. 20160049.

Schapiro, Anna C, Nicholas B Turk-Browne, Kenneth A Norman, and Matthew M Botvinick (2016). "Statistical learning of temporal community structure in the hippocampus". In: *Hippocampus* 26.1, pp. 3–8.

Schaul, Tom, John Quan, Ioannis Antonoglou, and David Silver (2015). "Prioritized experience replay". In: *arXiv preprint arXiv:1511.05952*.

Schiller, Daniela, Howard Eichenbaum, Elizabeth A Buffalo, Lila Davachi, David J Foster, Stefan Leutgeb, and Charan Ranganath (2015). "Memory and space: towards an understanding of the cognitive map". In: *Journal of Neuroscience* 35.41, pp. 13904–13911.

Schulman, John, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel (2015). "High-dimensional continuous control using generalized advantage estimation". In: *arXiv preprint arXiv:1506.02438*.

Schultz, Wolfram, Peter Dayan, and P Read Montague (1997). "A neural substrate of prediction and reward". In: *Science* 275.5306, pp. 1593–1599.

Seung, H Sebastian (2003). "Learning in spiking neural networks by reinforcement of stochastic synaptic transmission". In: *Neuron* 40.6, pp. 1063–1073.

Seung, H Sebastian and Haim Sompolinsky (1993). "Simple models for reading neuronal population codes." In: *Proceedings of the national academy of sciences* 90.22, pp. 10749–10753.

Shamir, Maoz and Haim Sompolinsky (2004). "Nonlinear population codes". In: *Neural computation* 16.6, pp. 1105–1136.

Sharpe, Melissa J, Chun Yun Chang, Melissa A Liu, Hannah M Batchelor, Lauren E Mueller, Joshua L Jones, Yael Niv, and Geoffrey Schoenbaum (2017). "Dopamine transients are sufficient and necessary for acquisition of model-based associations". In: *Nature neuroscience* 20.5, pp. 735–742.

Shatz, Carla J (1992). "The developing brain". In: *Scientific American* 267.3, pp. 60–67.

Silva, Andrew and Matthew Gombolay (2021). "Encoding human domain knowledge to warm start reinforcement learning". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 35. 6, pp. 5042–5050.

Silveira, Luan, Felipe Guth, Paulo Drews-Jr, Pedro Ballester, Matheus Machado, Felipe Codevilla, Nelson Duarte-Filho, and Silvia Botelho (2015). "An open-source bio-inspired solution to underwater SLAM". In: *IFAC-PapersOnLine* 48.2, pp. 212–217.

Sim, Robert, Pantelis Elinas, Matt Griffin, James J Little, et al. (2005). "Vision-based SLAM using the Rao-Blackwellised particle filter". In: *IJCAI Workshop on Reasoning with Uncertainty in Robotics*. Vol. 14. 1. Citeseer, pp. 9–16.

Singh, Ray and Chris Eliasmith (2006). "Higher-dimensional neurons explain the tuning and dynamics of working memory cells". In: *Journal of Neuroscience* 26.14, pp. 3667–3678.

Skarda, Christine A and Walter J Freeman (1987). "How brains make chaos in order to make sense of the world". In: *Behavioral and brain sciences* 10.2, pp. 161–173.

Smith, David M and Sheri JY Mizumori (2006). "Hippocampal place cells, context, and episodic memory". In: *Hippocampus* 16.9, pp. 716–729.

Smith, Randall, Matthew Self, and Peter Cheeseman (1990). "Estimating uncertain spatial relationships in robotics". In: *Autonomous robot vehicles*, pp. 167–193.

Sohn, Hansem, Devika Narain, Nicolas Meirhaeghe, and Mehrdad Jazayeri (2019). "Bayesian computation through cortical latent dynamics". In: *Neuron* 103.5, pp. 934–947.

Solstad, Trygve, Charlotte N Boccara, Emilio Kropff, May-Britt Moser, and Edvard I Moser (2008). "Representation of geometric borders in the entorhinal cortex". In: *Science* 322.5909, pp. 1865–1868.

Sorscher, Ben, Gabriel Mel, Surya Ganguli, and Samuel Ocko (2019). "A unified theory for the origin of grid cells through the lens of pattern formation". In: *Advances in Neural Information Processing Systems*.

Sorscher, Ben, Gabriel C Mel, Samuel A Ocko, Lisa M Giocomo, and Surya Ganguli (2023). "A unified theory for the computational and mechanistic origins of grid cells". In: *Neuron* 111.1, pp. 121–137.

Squire, Larry R and Stuart M Zola (1998). "Episodic memory, semantic memory, and amnesia". In: *Hippocampus* 8.3, pp. 205–211.

Sreenivasan, Sameet and Ila Fiete (2011). "Grid cells generate an analog error-correcting code for singularly precise neural computation". In: *Nature neuroscience* 14.10, p. 1330.

Stachenfeld, Kimberly L, Matthew M Botvinick, and Samuel J Gershman (2017). "The hippocampus as a predictive map". In: *Nature neuroscience* 20.11, pp. 1643–1653.

Stachniss, Cyrill, Dirk Hahnel, and Wolfram Burgard (2004). "Exploration with active loop-closing for FastSLAM". In: *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Vol. 2. IEEE, pp. 1505–1510.

Stangl, Matthias, Ingmar Kanitscheider, Martin Riemer, Ila Fiete, and Thomas Wolbers (2020). "Sources of path integration error in young and aging humans". In: *Nature communications* 11.1, p. 2626.

Staudigl, Tobias, Marcin Leszczynski, Joshua Jacobs, Sameer A Sheth, Charles E Schroeder, Ole Jensen, and Christian F Doeller (2018). "Hexadirectional modulation of high-frequency electrophysiological activity in the human anterior medial temporal lobe maps visual space". In: *Current Biology* 28.20, pp. 3325–3329.

Steckel, Jan and Herbert Peremans (2013). "BatSLAM: Simultaneous localization and mapping using biomimetic sonar". In: *PloS one* 8.1, e54076.

Steinberg, Julia and Haim Sompolinsky (2022). "Associative memory of structured knowledge". In: *Scientific Reports* 12.1, p. 21808.

Stenborg, Erik, Carl Toft, and Lars Hammarstrand (2018). "Long-term visual localization using semantically segmented images". In: *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, pp. 6484–6490.

Stewart, Terrence, Feng-Xuan Choo, and Chris Eliasmith (2012). "Spaun: A perception-cognition-action model using spiking neurons". In: *Proceedings of the Annual Meeting of the Cognitive Science Society*. Vol. 34. 34.

— (2014). "Sentence processing in spiking neurons: A biologically plausible left-corner parser". In: *Proceedings of the Annual Meeting of the Cognitive Science Society*. Vol. 36. 36.

Stewart, Terrence, Xuan Choo, and Chris Eliasmith (2010a). "Symbolic reasoning in spiking neurons: A model of the cortex/basal ganglia/thalamus loop". In: *Proceedings of the Annual Meeting of the Cognitive Science Society*. Vol. 32. 32.

Stewart, Terrence C, Xuan Choo, Chris Eliasmith, et al. (2010b). "Dynamic behaviour of a spiking model of action selection in the basal ganglia". In: *Proceedings of the 10th international conference on cognitive modeling*. Citeseer, pp. 235–40.

Stewart, Terrence C, P Michael Furlong, Kathryn Simone, Madeleine Bartlett, and Jeff Orchard (2023). "Novelty Detection, Insect Olfaction, Mismatch Negativity, and the Representation of Probability in the Brain". In: *Proceedings of the 21st International Conference on Cognitive Modeling*.

Stewart, Terrence C, Yichuan Tang, and Chris Eliasmith (2011). "A biologically realistic cleanup memory: Autoassociation in spiking neurons". In: *Cognitive Systems Research* 12.2, pp. 84–92.

Stöckel, Andreas (2022). "Harnessing neural dynamics as a computational resource". PhD thesis. University of Waterloo.

Stoianov, Ivilin Peev, Cyriel MA Pennartz, Carien S Lansink, and Giovani Pezzulo (2018). "Model-based spatial navigation in the hippocampus-ventral striatum circuit: A computational analysis". In: *PLoS computational biology* 14.9, e1006316.

Striedter, Georg F (2016). "Evolution of the hippocampus in reptiles and birds". In: *Journal of Comparative Neurology* 524.3, pp. 496–517.

Sun, Yinqian, Yi Zeng, Feifei Zhao, and Zhuoya Zhao (2023). "Multi-compartment neuron and population encoding improved spiking neural network for deep distributional reinforcement learning". In: *arXiv preprint arXiv:2301.07275*.

Sünderhauf, Niko and Peter Protzel (2012). "Switchable constraints for robust pose graph SLAM". In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, pp. 1879–1884.

Sutherland, Danica J and Jeff Schneider (2015). "On the error of random Fourier features". In: *arXiv preprint arXiv:1506.02785*.

Sutton, Richard S (1988). "Learning to predict by the methods of temporal differences". In: *Machine learning* 3, pp. 9–44.

Sutton, Richard S and Andrew G Barto (2018). *Reinforcement learning: An introduction*. MIT press.

Sutton, Richard S, Doina Precup, and Satinder Singh (1999). "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning". In: *Artificial intelligence* 112.1-2, pp. 181–211.

Takahashi, Yuji K, Hannah M Batchelor, Bing Liu, Akash Khanna, Marisela Morales, and Geoffrey Schoenbaum (2017). "Dopamine neurons respond to errors in the prediction of sensory features of expected rewards". In: *Neuron* 95.6, pp. 1395–1405.

Tallec, Corentin, Léonard Blier, and Yann Ollivier (2019). "Making deep q-learning methods robust to time discretization". In: *International Conference on Machine Learning*. PMLR, pp. 6096–6104.

Tang, Guangzhi, Neelesh Kumar, Raymond Yoo, and Konstantinos Michmizos (2021). "Deep reinforcement learning with population-coded spiking neural network for continuous control". In: *Conference on Robot Learning*. PMLR, pp. 2016–2029.

Tang, Guangzhi and Konstantinos P Michmizos (2018). "Gridbot: An autonomous robot controlled by a spiking neural network mimicking the brain's navigational system". In: *Proceedings of the International Conference on Neuromorphic Systems*, pp. 1–8.

Tang, Guangzhi, Arpit Shah, and Konstantinos P Michmizos (2019). "Spiking neural network on neuromorphic hardware for energy-efficient unidimensional slam". In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 4176–4181.

Tassa, Yuval et al. (2018). "Deepmind control suite". In: *arXiv preprint arXiv:1801.00690*.

Taube, Jeffrey S, Robert U Muller, and James B Ranck (1990). "Head-direction cells recorded from the postsubiculum in freely moving rats. I. Description and quantitative analysis". In: *Journal of Neuroscience* 10.2, pp. 420–435.

Tavares, Rita Morais, Avi Mendelsohn, Yael Grossman, Christian Hamilton Williams, Matthew Shapiro, Yaacov Trope, and Daniela Schiller (2015). "A map for social navigation in the human brain". In: *Neuron* 87.1, pp. 231–243.

Tay, Yi et al. (2020). "Long range arena: A benchmark for efficient transformers". In: *arXiv preprint arXiv:2011.04006*.

Teyler, Timothy J and Pascal DiScenna (1986). "The hippocampal memory indexing theory." In: *Behavioral neuroscience* 100.2, p. 147.

Teyler, Timothy J and Jerry W Rudy (2007). "The hippocampal indexing theory and episodic memory: updating the index". In: *Hippocampus* 17.12, pp. 1158–1169.

Thakur, Chetan Singh et al. (2018). "Large-scale neuromorphic spiking array processors: A quest to mimic the brain". In: *Frontiers in neuroscience* 12, p. 891.

Thrun, Sebastian and Michael Montemerlo (2006). "The graph SLAM algorithm with applications to large-scale mapping of urban structures". In: *The International Journal of Robotics Research* 25.5-6, pp. 403–429.

Tolman, Edward C (1948). "Cognitive maps in rats and men." In: *Psychological review* 55.4, p. 189.

Tolman, Edward Chace and Charles H Honzik (1930). "Introduction and removal of reward, and maze performance in rats." In: *University of California publications in psychology*.

Touati, Ahmed and Yann Ollivier (2021). "Learning one representation to optimize all rewards". In: *Advances in Neural Information Processing Systems* 34, pp. 13–23.

Touati, Ahmed, Jérémy Rapin, and Yann Ollivier (2022). "Does Zero-Shot Reinforcement Learning Exist?" In: *arXiv preprint arXiv:2209.14935*.

Trivedi, Dweep, Jesse Zhang, Shao-Hua Sun, and Joseph J Lim (2021). "Learning to synthesize programs as interpretable and generalizable policies". In: *Advances in neural information processing systems* 34, pp. 25146–25163.

Trujillo, Oliver and Chris Eliasmith (2014). "A spiking-neuron model of memory encoding and replay in hippocampus". In: *BMC Neuroscience* 15.Suppl 1, P166.

Tsodyks, Misha (1999). "Attractor neural network models of spatial maps in hippocampus". In: *Hippocampus* 9.4, pp. 481–489.

Tulving, Endel and Donald M Thomson (1973). "Encoding specificity and retrieval processes in episodic memory." In: *Psychological review* 80.5, p. 352.

Umbach, Gray, Pranish Kantak, Joshua Jacobs, Michael Kahana, Brad E Pfeiffer, Michael Sperling, and Bradley Lega (2020). "Time cells in the human hippocampus and entorhinal cortex support episodic memory". In: *Proceedings of the National Academy of Sciences* 117.45, pp. 28463–28474.

Vágó, Lajos and Balázs B Ujfalussy (2018). "Robust and efficient coding with grid cells". In: *PLoS computational biology* 14.1, e1005922.

Vaidyanathan, Natarajan and Chris Eliasmith (2019). "Spiking neuron model of motor control with adaptation to visuomotor rotation". In: *49th Annual Meeting of the Society of Neuroscience*. Chicago, IL: Society of Neuroscience.

Vargha-Khadem, Faraneh, David G Gadian, Kate E Watkins, Alan Connelly, Wim Van Paesschen, and Mortimer Mishkin (1997). "Differential effects of early hippocampal pathology on episodic and semantic memory". In: *Science* 277.5324, pp. 376–380.

Vasilaki, Eleni, Nicolas Frémaux, Robert Urbanczik, Walter Senn, and Wulfram Gerstner (2009). "Spike-based reinforcement learning in continuous state and action space: when policy gradient methods fail". In: *PLoS computational biology* 5.12, e1000586.

Voelker, Aaron, Ivana Kajić, and Chris Eliasmith (2019). "Legendre memory units: Continuous-time representation in recurrent neural networks". In: *Advances in neural information processing systems* 32.

Voelker, Aaron R (2020). "A short letter on the dot product between rotated Fourier transforms". In: *arXiv preprint arXiv:2007.13462*.

Voelker, Aaron R, Peter Blouw, Xuan Choo, Nicole Sandra-Yaffa Dumont, Terrence C Stewart, and Chris Eliasmith (2021). "Simulating and Predicting Dynamical Systems With Spatial Semantic Pointers". In: *Neural Computation* 33.8, pp. 2033–2067.

Voelker, Aaron R and Chris Eliasmith (2018). "Improving spiking dynamical networks: Accurate delays, higher-order synapses, and time cells". In: *Neural computation* 30.3, pp. 569–609.

Voelker, Aaron R, Jan Gosmann, and Terrence C Stewart (2017). "Efficiently sampling vectors and coordinates from the n-sphere and n-ball". In: *Centre for Theoretical Neuroscience-Technical Report* 1.

Voelker, Aaron Russell (2019). "Dynamical systems in spiking neuromorphic hardware". In.

Voelker, Aaron Russell, Eric Crawford, and Chris Eliasmith (2014). "Learning large-scale heteroassociative memories in spiking neurons". In: *Unconventional Computation and Natural Computation* 7, p. 2014.

Wang, Haoran, Thaleia Zariphopoulou, and Xun Yu Zhou (2020). "Reinforcement learning in continuous time and space: A stochastic control approach". In: *The Journal of Machine Learning Research* 21.1, pp. 8145–8178.

Wang, Ranxiao Frances (2016). "Building a cognitive map by assembling multiple path integration systems". In: *Psychonomic Bulletin & Review* 23, pp. 692–702.

Wang, Raymond and Louis Kang (2022). "Multiple bumps can enhance robustness to noise in continuous attractor networks". In: *PLoS Computational Biology* 18.10, e1010547.

Wei, Xue-Xin, Jason Prentice, and Vijay Balasubramanian (2015). "A principle of economy predicts the functional architecture of grid cells". In: *Elife* 4, e08362.

Weilbächer, Regina A and Sebastian Gluth (2016). "The interplay of hippocampus and ventromedial prefrontal cortex in memory-based decision making". In: *Brain sciences* 7.1, p. 4.

Welday, Adam C, I Gary Shlifer, Matthew L Bloom, Kechen Zhang, and Hugh T Blair (2011). "Cosine directional tuning of theta cell burst frequencies: evidence for spatial coding by oscillatory interference". In: *Journal of Neuroscience* 31.45, pp. 16157–16176.

Wenliang, Li Kevin, Gregoire Deletang, Matthew Aitchison, Marcus Hutter, Anian Ruoss, Arthur Gretton, and Mark Rowland (2024). "Distributional Bellman Operators over Mean Embeddings". In: *Proceedings of the 41st International Conference on Machine Learning*. Ed. by Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp. Vol. 235. Proceedings of Machine Learning Research. PMLR, pp. 52839–52868.

Whittington, James CR, Timothy H Muller, Shirley Mark, Guifen Chen, Caswell Barry, Neil Burgess, and Timothy EJ Behrens (2020). "The Tolman-Eichenbaum machine: unifying space and relational memory through generalization in the hippocampal formation". In: *Cell* 183.5, pp. 1249–1263.

Willems, Lucas (2023). *RL Starter Files*. `https://github.com/lcswillems/rl-starter-files`.

Witter, Menno P, Thanh P Doan, Bente Jacobsen, Eirik S Nilssen, and Shinya Ohara (2017). "Architecture of the entorhinal cortex a review of entorhinal anatomy in rodents with some comparative notes". In: *Frontiers in systems neuroscience* 11, p. 46.

Witter, Menno P and Edvard I Moser (2006). “Spatial representation and the architecture of the entorhinal cortex”. In: *Trends in neurosciences* 29.12, pp. 671–678.

Xiao, Zhuocheng, Kevin Lin, and Jean-Marc Fellous (2020). “Conjunctive reward–place coding properties of dorsal distal CA1 hippocampus cells”. In: *Biological cybernetics* 114.2, pp. 285–301.

Xiong, Wenhan, Thien Hoang, and William Yang Wang (2017). “DeepPath: A Reinforcement Learning Method for Knowledge Graph Reasoning”. In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing (EMNLP 2017)*. Copenhagen, Denmark: ACL.

Xu, Dehong, Ruiqi Gao, Wenhao Zhang, Xue-Xin Wei, and Ying Nian Wu (2022). “Conformal Isometry of Lie Group Representation in Recurrent Network of Grid Cells”. In: *NeurIPS 2022 Workshop on Symmetry and Geometry in Neural Representations*. URL: https://openreview.net/forum?id=FszPdSkvGjz.

Yakopcic, Chris, Nayim Rahman, Tanvir Atahary, Tarek M Taha, and Scott Douglass (2020). “Solving constraint satisfaction problems using the loihi spiking neuromorphic processor”. In: *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, pp. 1079–1084.

Yan, Hui, Xinle Liu, Hong Huo, and Tao Fang (2019). “Mechanisms of reward-modulated stdp and winner-take-all in bayesian spiking decision-making circuit”. In: *Neural Information Processing: 26th International Conference, ICONIP 2019, Sydney, NSW, Australia, December 12–15, 2019, Proceedings, Part III 26*. Springer, pp. 162–172.

Yan, Yexin et al. (2021). “Comparing Loihi with a SpiNNaker 2 prototype on low-latency keyword spotting and adaptive robotic control”. In: *Neuromorphic Computing and Engineering* 1.1, p. 014002.

Yartsev, Michael M and Nachum Ulanovsky (2013). “Representation of three-dimensional space in the hippocampus of flying bats”. In: *Science* 340.6130, pp. 367–372.

Yartsev, Michael M, Menno P Witter, and Nachum Ulanovsky (2011). “Grid cells without theta oscillations in the entorhinal cortex of bats”. In: *Nature* 479.7371, pp. 103–107.

Yildiz, Cagatay, Markus Heinonen, and Harri Lähdesmäki (2021). “Continuous-time model-based reinforcement learning”. In: *International Conference on Machine Learning*. PMLR, pp. 12009–12018.

Yu, Byron M, John P Cunningham, Gopal Santhanam, Stephen Ryu, Krishna V Shenoy, and Maneesh Sahani (2008). “Gaussian-process factor analysis for low-dimensional single-trial analysis of neural population activity”. In: *Advances in neural information processing systems* 21.

Yu, Changmin, Timothy EJ Behrens, and Neil Burgess (2021). “Prediction and generalisation over directed actions by grid cells”. In: *ICLR Proceedings 2021*.

Yu, Changmin, Neil Burgess, Maneesh Sahani, and Samuel J Gershman (2024). "Successor-Predecessor Intrinsic Exploration". In: *Advances in Neural Information Processing Systems* 36.

Yu, Fangwen, Jianga Shang, Youjian Hu, and Michael Milford (2019). "NeuroSLAM: a brain-inspired SLAM system for 3D environments". In: *Biological Cybernetics* 113.5, pp. 515–545.

Yuan, Mengwen, Xi Wu, Rui Yan, and Huajin Tang (2019). "Reinforcement learning in spiking neural networks with stochastic and deterministic synapses". In: *Neural computation* 31.12, pp. 2368–2389.

Zaidel, Yuval, Albert Shalumov, Alex Volinski, Lazar Supic, and Elishai Ezra Tsur (2021). "Neuromorphic NEF-based inverse kinematics and PID control". In: *Frontiers in neurorobotics* 15, p. 631159.

Zambrano, Davide, Pieter R Roelfsema, and Sander Bohte (2021). "Learning continuous-time working memory tasks with on-policy neural reinforcement learning". In: *Neurocomputing* 461, pp. 635–656.

Zambrano, Davide, Pieter R Roelfsema, and Sander M Bohte (2015). "Continuous-time on-policy neural reinforcement learning of working memory tasks". In: *2015 International Joint Conference on Neural Networks (IJCNN)*. IEEE, pp. 1–8.

Zeng, Taiping and Bailu Si (2017). "Cognitive mapping based on conjunctive representations of space and movement". In: *Frontiers in neurorobotics* 11, p. 61.

Zhang, Huanqiu, P Dylan Rich, Albert K Lee, and Tatyana O Sharpee (2023). "Hippocampal spatial representations exhibit a hyperbolic geometry that expands with experience". In: *Nature Neuroscience* 26.1, pp. 131–139.

Zhang, Jingwei, Jost Tobias Springenberg, Joschka Boedecker, and Wolfram Burgard (2017). "Deep reinforcement learning with successor features for navigation across similar environments". In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, pp. 2371–2378.

Zhang, Kechen (1996). "Representation of spatial orientation by the intrinsic dynamics of the head-direction cell ensemble: a theory". In: *Journal of Neuroscience* 16.6, pp. 2112–2126.

Zhang, Kechen and Terrence J Sejnowski (1999). "Neuronal tuning: To sharpen or broaden?" In: *Neural computation* 11.1, pp. 75–84.

Zhang, Xiaohan, Lu Liu, Guodong Long, Jing Jiang, and Shenquan Liu (2021). "Episodic memory governs choices: An rnn-based reinforcement learning model for decision-making task". In: *Neural Networks* 134, pp. 1–10.

Zhao, Mintao and William H Warren (2015). "How you get there from here: Interaction of visual landmarks and path integration in human navigation". In: *Psychological science* 26.6, pp. 915–924.

Zhou, Mo, Jiequn Han, and Jianfeng Lu (2021). "Actor-critic method for high dimensional static Hamilton–Jacobi–Bellman partial differential equations based on neural networks". In: *SIAM Journal on Scientific Computing* 43.6, A4043–A4066.

Zilli, Eric A (2012). "Models of grid cell spatial firing published 2005–2011". In: *Frontiers in neural circuits* 6, p. 16.

Zilli, Eric A and Michael E Hasselmo (2010). "Coupled noisy spiking neurons as velocity-controlled oscillators in a model of grid cell spatial firing". In: *Journal of Neuroscience* 30.41, pp. 13850–13860.

Zilli, Eric A, Motoharu Yoshida, Babak Tahvildari, Lisa M Giocomo, and Michael E Hasselmo (2009). "Evaluation of the oscillatory interference model of grid cell firing through analysis and measured period variance of some biological oscillators". In: *PLoS computational biology* 5.11, e1000573.

# Appendix A

# Supplementary detail

## A.1 SSP dynamics

Recall that derivative of an Spatial Semantic Pointer (SSP) representing the coordinates $\boldsymbol{x}(t) \in \mathcal{R}^n$ is

$$\dot{\phi}(\boldsymbol{x}(t)) = \phi(\boldsymbol{x}(t)) \circledast \ln \phi(\dot{\boldsymbol{x}}(t)), \tag{A.1}$$

where the logarithm of an SSP is defined as

$$\ln \phi(\boldsymbol{x}) = \mathcal{F}^{-1}\left\{ i\Theta \frac{\boldsymbol{x}}{\ell} \right\} = \sum_{j=1}^{n} \frac{x_j}{\ell_j} \mathcal{F}^{-1}\left\{ i\boldsymbol{\theta}_j \right\}. \tag{A.2}$$

### *A.1.1 Stability analysis of attractor dynamics*

The nonlinear velocity controlled oscillator (VCO) is described by

$$\frac{d}{dt}\begin{bmatrix} \mathrm{Re}\,\mathcal{F}\left\{\phi(\boldsymbol{x}(t))\right\}_j \\ \mathrm{Im}\,\mathcal{F}\left\{\phi(\boldsymbol{x}(t))\right\}_j \end{bmatrix} = \begin{bmatrix} -\omega_j(t)\mathrm{Im}\,\mathcal{F}\left\{\phi(\boldsymbol{x}(t))\right\}_j + \frac{1-r^2}{r}\mathrm{Re}\,\mathcal{F}\left\{\phi(\boldsymbol{x}(t))\right\}_j \\ \omega_j(t)\mathrm{Re}\,\mathcal{F}\left\{\phi(\boldsymbol{x}(t))\right\}_j + \frac{1-r^2}{r}\mathrm{Im}\,\mathcal{F}\left\{\phi(\boldsymbol{x}(t))\right\}_j \end{bmatrix}, \quad \text{where } r \equiv |\mathcal{F}\{\phi(\boldsymbol{x}(t))\}_j|. \tag{A.3}$$

The dynamical system can alternatively be written as

$$\frac{dr}{dt} = \frac{1-r^2}{r}, \quad \frac{d\theta_j}{dt} = \omega_j(t) \tag{A.4}$$

where $\theta_j$ is the phase of the VCO. It is clear that $r = 1$ is a stable equilibrium as $\frac{dr}{dt} = 0$ at $r = 1, \frac{dr}{dt} > 0$ when $r < 1$, and $\frac{dr}{dt} < 0$ when $r > 1$.

### *A.1.2 Higher-Order SSP dynamics*

The $m > 1$ order derivative of an SSP, in terms of it's lower-order derivatives, is

$$\frac{d^m\phi}{dt^m}(\boldsymbol{x}(t)) = \sum_{n=0}^{m-1} \binom{m-1}{n} \frac{d^n\phi}{dt^n}(\boldsymbol{x}(t)) \circledast \ln \phi \left( \frac{d^{(m-n)}\boldsymbol{x}}{dt^{(m-n)}}(t) \right) \tag{A.5}$$

For example, the second-order derivative is

$$\ddot{\phi}(\boldsymbol{x}(t)) = \phi(\boldsymbol{x}(t)) \circledast \ln \phi(\ddot{\boldsymbol{x}}(t)) + \dot{\phi}(\boldsymbol{x}(t)) \circledast \ln \phi(\dot{\boldsymbol{x}}(t)) \tag{A.6}$$

One can also obtain a formula for higher-order SSP derivatives in terms derivatives of the variables it represents. For example,

$$\dddot{\phi}(\boldsymbol{x}(t)) = \phi(\boldsymbol{x}(t)) \circledast \left(\ln \phi(\ddot{\boldsymbol{x}}(t)) + \ln \phi(\dot{\boldsymbol{x}}(t))^2\right). \tag{A.7}$$

# Appendix B

# Extended methods and experiments

## B.1 Non-stationary SSPs and Recursive SSP Dynamics

Recursive Spatial Semantic Pointers (rSSPs) were introduced in Voelker (2019) as a means to overcome the representational limitations of traditional Spatial Semantic Pointers (SSPs). Here we shall describe these representations, extend this work by deriving the dynamics of rSSPs, and briefly discuss challenges associated with modelling these dynamics within a neural system.

One key observation is that the SSP manifold, defined as $\{\phi(\boldsymbol{x}) : \boldsymbol{x} \in \mathbb{R}^n\}$, is not identical to $\mathbb{R}^d$. Rather, SSPs are confined to a lower-dimensional subspace of $\mathbb{R}^d$, as demonstrated by Voelker (2019). This constraint arises because the encoding function $\phi(\boldsymbol{x})$ maps the continuous variable $\boldsymbol{x}$ into a structured representation that resides on a non-linear manifold. Such structure is advantageous for certain applications but imposes limitations in others, particularly in terms of expressiveness and generalization.

Furthermore, the kernel defined by $\phi(\boldsymbol{x})$, given as $k(\boldsymbol{x}, \boldsymbol{x}') = \phi(\boldsymbol{x}) \cdot \phi(\boldsymbol{x}')$, is isotropic and symmetric. While this property ensures that spatial relationships are encoded consistently across all dimensions, it also introduces a stationarity that may not be desirable in tasks requiring context-dependent or anisotropic representations. For example, in non-stationary environments or scenarios where spatial relationships evolve dynamically, such isotropic kernels might fail to capture the nuanced dependencies between variables.

Examples of non-stationary kernels include those that vary with position or time – for example, position-dependent length scales or neural kernels that adapt dynamically based on learned weights. These kernels enable greater flexibility in encoding, allowing the system to emphasize specific features or dimensions depending on task demands.

An rSSP representing the variable $\boldsymbol{x} \in \mathbb{R}^N$ is given by

$$\phi(\boldsymbol{x}) = \mathcal{F}^{-1}\left\{ e^{i\Theta(\boldsymbol{x})\frac{\boldsymbol{x}}{\ell(\boldsymbol{x})}} \right\}. \tag{B.1}$$

The parameters of the SSP projection, $\Theta$ and $\ell$, are now functions of the variable $\boldsymbol{x}$. Essentially, the mapping adapts in some way over space. These function can be hand-set or learned in some fashion. Voelker (2019) uses rSSP embeddings to create kernels that approximate fractal sets.

The dynamics of these embeddings are given by:

$$\dot{\phi}(\boldsymbol{x}(t)) = \phi(\boldsymbol{x}(t)) \circledast \mathcal{F}^{-1}\left\{ i\dot{\Theta}(\boldsymbol{x}(t))\text{diag}\left(\frac{1}{\ell(\boldsymbol{x}(t))}\right)\boldsymbol{x}(t) + i\Theta(\boldsymbol{x}(t))\text{diag}\left(\frac{1}{\ell(\boldsymbol{x}(t))}\right)\dot{\boldsymbol{x}}(t) \right. \quad \text{(B.2)}$$

$$\left. + i\Theta(\boldsymbol{x}(t))\text{diag}\left(\frac{-\dot{\ell}(\boldsymbol{x}(t))}{\ell(\boldsymbol{x}(t))}\right)\boldsymbol{x}(t) \right\}. \quad \text{(B.3)}$$

The dynamics are still continuous binding – *i.e.*, oscillators in the Fourier domain. However, now the frequency consists of three terms, one that depends on velocity and two that depend directly on the underlying variable $\boldsymbol{x}(t)$ – information that we have so far assumed is not available to a neural system.

**Non-stationary length scale** The dynamics can be simplified slightly by considering a subset of possible rSSPs. In particular, ones in which $\Theta(\boldsymbol{x}(t)) = \Theta$ – *i.e.*, the only part of the projection that varies over space are the length scale parameters, $\ell(\boldsymbol{x}(t))$. This corresponds to a kernel with a fixed shape but varying width and, thus, spatial resolution. The dynamics in this case are given by

$$\dot{\phi}(\boldsymbol{x}(t)) = \phi(\boldsymbol{x}(t)) \circledast \mathcal{F}^{-1}\left\{ i\Theta\left(\text{diag}\left(\frac{-\dot{\ell}(\boldsymbol{x}(t))}{\ell(\boldsymbol{x}(t))}\right)\frac{\boldsymbol{x}(t)}{\ell(\boldsymbol{x}(t))} + \frac{\dot{\boldsymbol{x}}(t)}{\ell(\boldsymbol{x}(t))}\right)\right\} \quad \text{(B.4)}$$

$$= \phi(\boldsymbol{x}(t)) \circledast \left(\ln\phi(\dot{\boldsymbol{x}}(t)) + \ln\phi\left(\frac{-\dot{\ell}(\boldsymbol{x}(t))}{\ell(\boldsymbol{x}(t))}\boldsymbol{x}(t)\right)\right), \quad \text{(B.5)}$$

which still depends on the variable $\boldsymbol{x}(t)$. This could be estimated from $\phi(\boldsymbol{x}(t))$ via decoding. However, the idea that a biological system would decode back to the $\boldsymbol{x}$ space rather than stay in the space of embeddings is questionable.

## B.2 MiniGrid experiments

In this appendix, we describe other experiments that were run on the MiniGrid tasks discussed in detail in Section 6.4. All of the results here use much lower-dimensional SSP embeddings – only 303 trainable parameters – compared to the results in Section 6.4.2. Figure B.1 shows results where the SSP size is the only difference from those in 6.6. While the Holographic Reduced Representation (HRR) view embedding remains the best overall, SSP-based pose and slot-filler embeddings experienced more significant performance degradation.

We also compared using learned versus fixed phase matrices (Figure B.2). The trainable version generally does better, indicating moderate sensitivity to the phase matrix configuration. It is important to note that in the fixed matrix scenario, the SSP length scale vector remains trainable, allowing the model to adjust spatial resolution and kernel width, though not the kernel's shape. Interestingly, one task demonstrated superior performance with a fixed matrix,

potentially because the grid cell structure (*i.e.*, HexSSP) may not be maintained when the phase matrix is trainable.

Experiments using HRR embeddings without SSP pose information showed worse performance on several tasks compared to embeddings that included pose (Figure B.3). However, on some tasks, the absence of pose information did not significantly affect performance. This indicates that the utility of pose information in HRR embeddings may be task-dependent, with some environments being more reliant on explicit pose representations.

Figure B.4 shows results when tanh activation functions (as opposed to the ReLU activation used in the experiments presented in the main text) are used throughout the network. The image-based representation performed better on most tasks. Conversely, SSP and HRR embeddings experienced slight performance declines across most tasks. Despite this, the HRR view embedding still achieved the best sample efficiency and final returns on the majority of tasks, though its dominance was less consistent compared to the ReLU-based experiments.

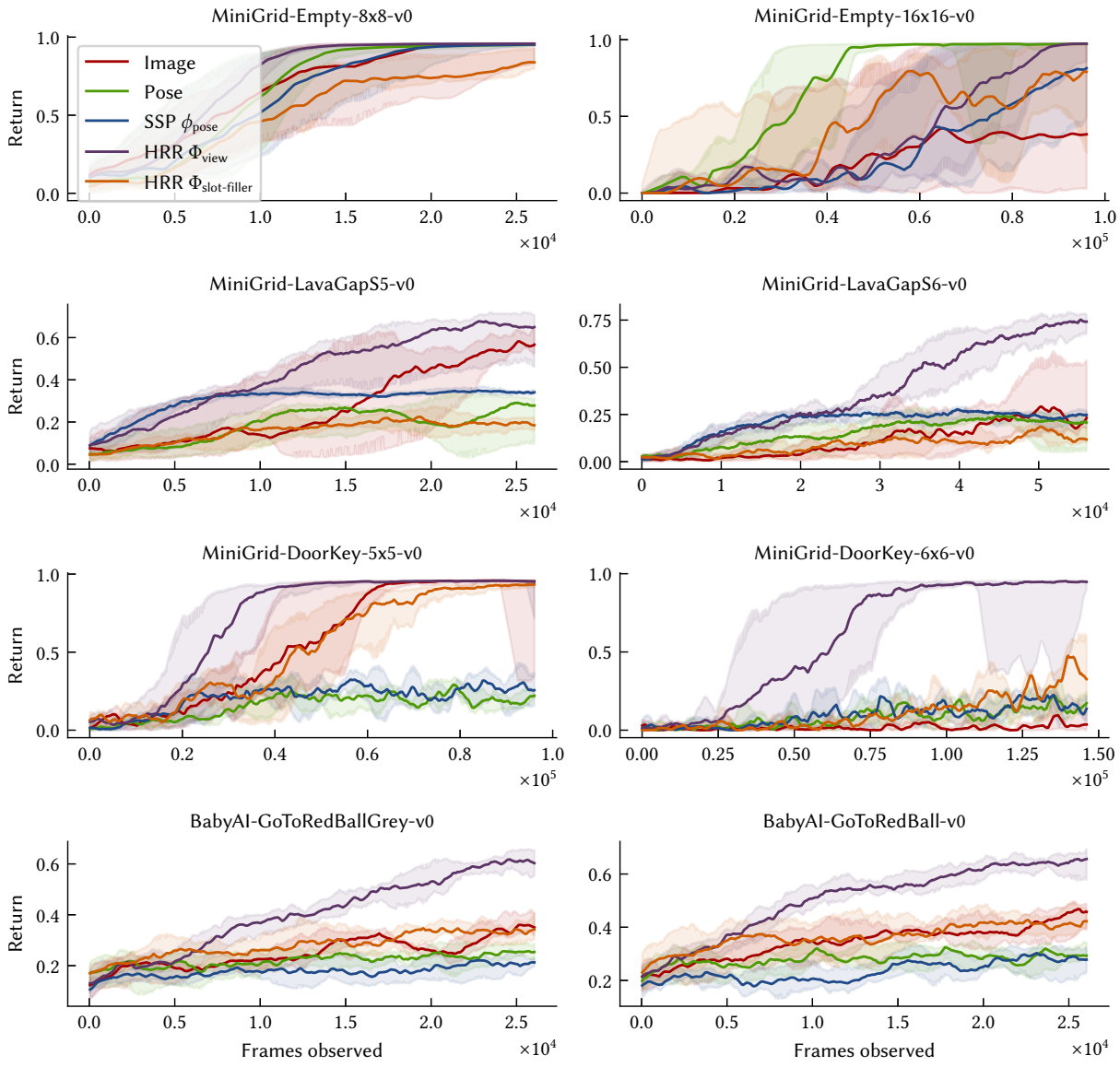



**Figure B.1:** Sample efficiency curves for navigation benchmarks with lower-dimensional SSP and HRR embeddings.

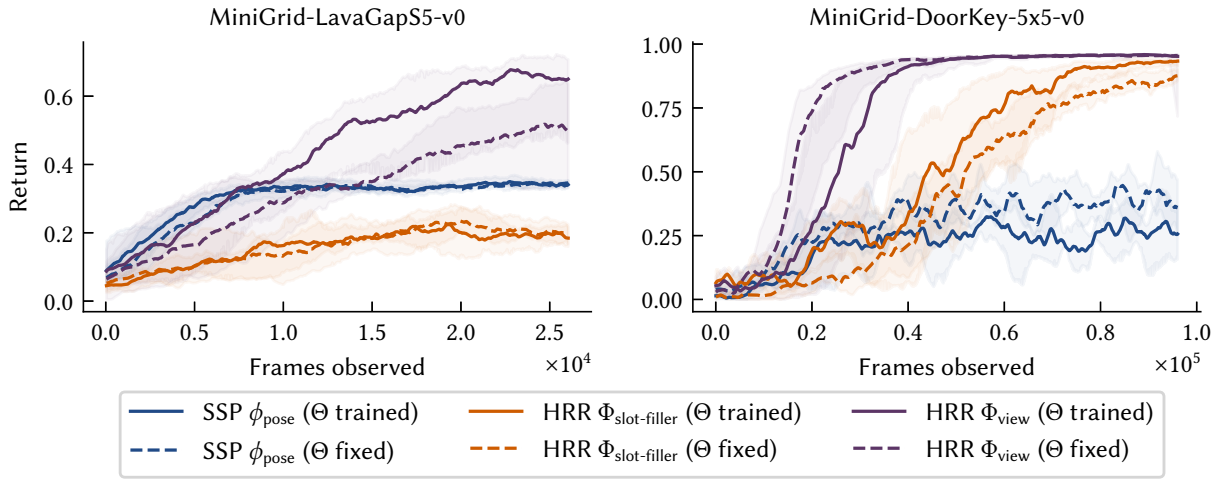



**Figure B.2:** The interquartile mean (IQM) of episodic reward over five seeds for the SSP and HRR feature embedding methods, with the training of the SSP phase matrix Θ toggled. The versions that learn the matrix Θ are what is used in the other results.

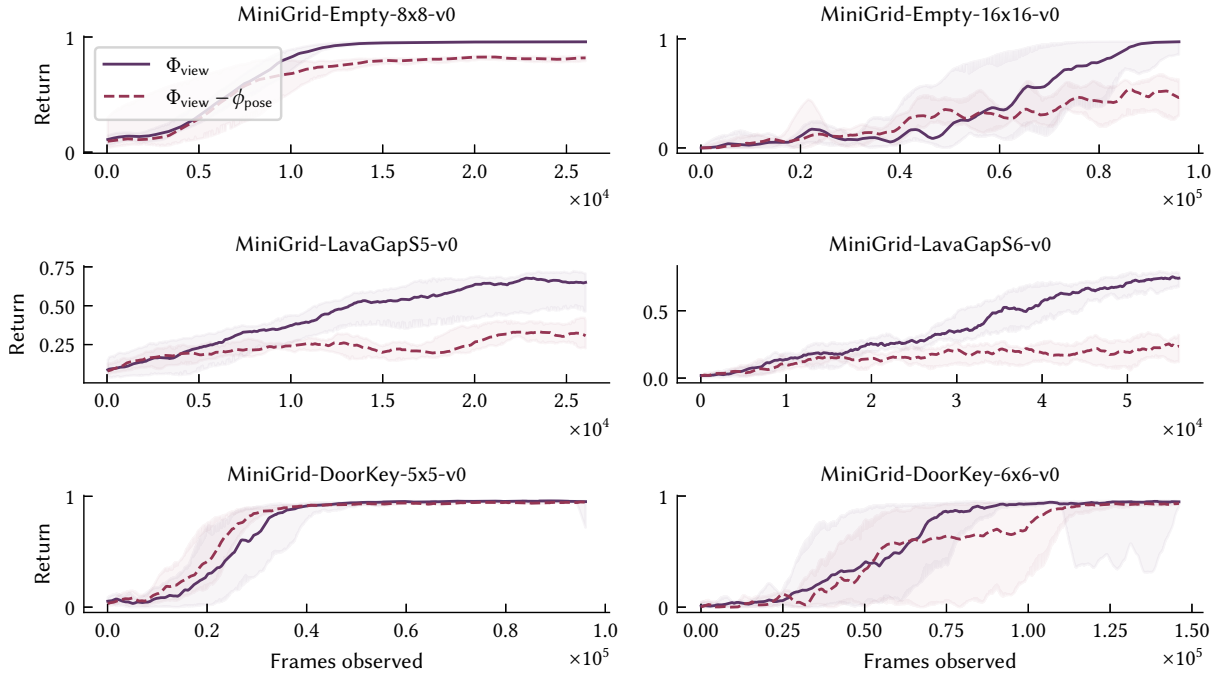



**Figure B.3:** The IQM of episodic reward over five seeds, along with stratified bootstrap 95% confidence intervals, versus the number of observations for HRR view embedding with, $\Phi_{\text{view}}$, and without pose information, $\Phi_{\text{view}} - \phi_{\text{pose}}$.

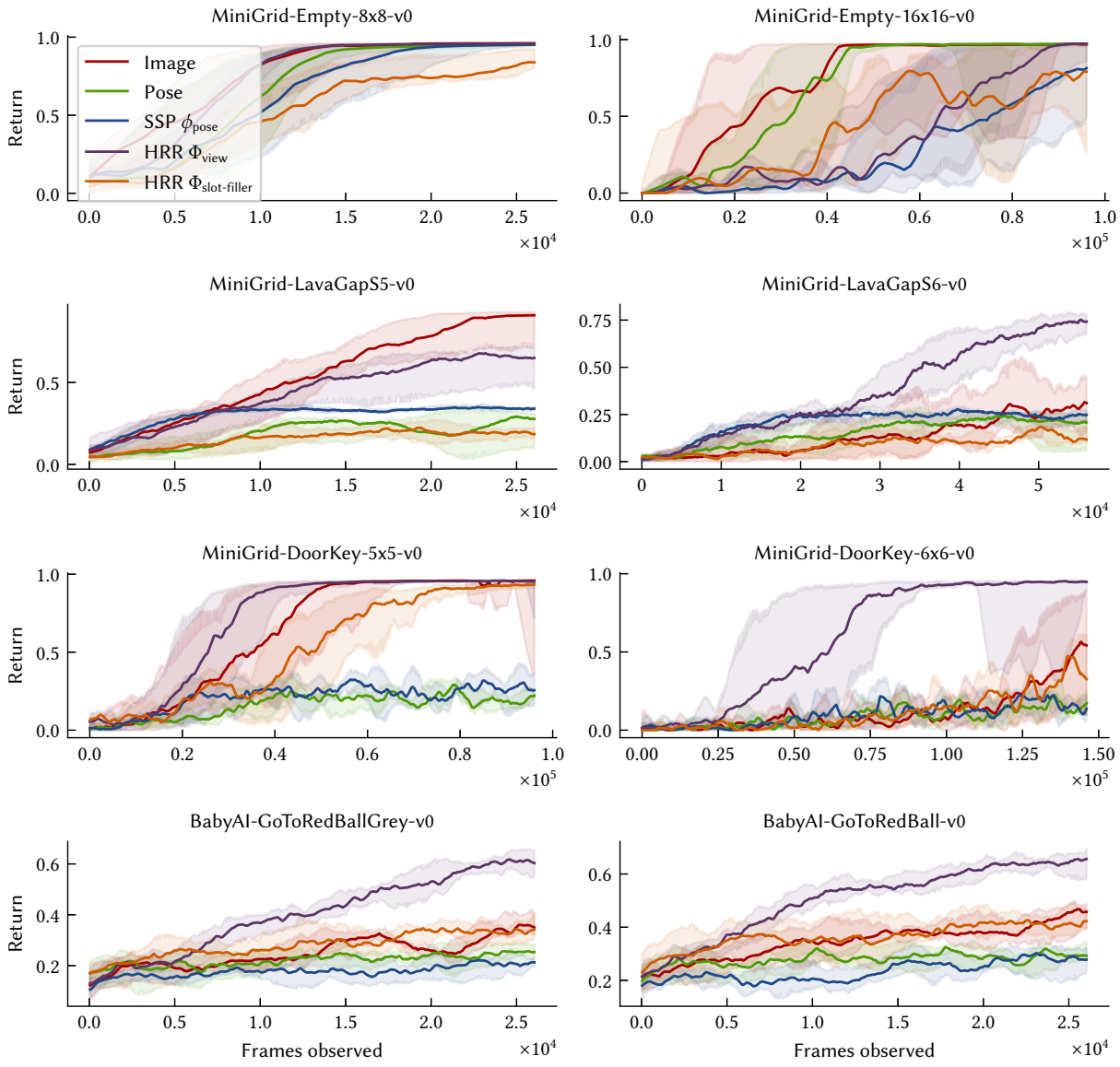



**Figure B.4:** Sample efficiency curves for navigation benchmarks using tanh activation functions in the network.

# Appendix C

# Experiment parameters

These sections include tables with model parameters for experiments in Chapters 2-6. Details on the experimental setup and model architecture are given in the main text. Code for all experiments will be made available at `https://github.com/nsdumont/PhD-thesis`. All spiking neural network (SNN) models presented in this thesis were built and simulated using Nengo (Bekolay et al., 2014), an open source brain simulation software package. In our experiments, we use the default settings (Table C.1) unless otherwise stated.

**Table C.1:** Default parameters in Nengo, used to construct spiking neural network models.

| Parameter | Value (or formula) |
|---|---|
| Neuron model | Leaky integrate-and-fire (LIF) |
| Synaptic filtering model | Low-pass filter |
| Synaptic time constant [s] | 0.005 |
| Maximum firing rates [Hz] | Uniform(200, 400) |
| Intercepts | Uniform(-1.0, 0.9) |
| Decoder optimization method | Least-squares with L2 regularization |
| Decoder regularization $\lambda$ | 0.1 |
| Simulation $\Delta t$ [s] | 0.001 |

**Table C.3:** Hyperparameters used to generate the path integration results shown in Section 3.4.6 and Section 3.4.6. The values listed under Figure 3.15 refer to those used for the grid cell model.

| Parameter | 3.10A | 3.10B | 3.11 | 3.12 | 3.14 | 3.15 |
|---|---|---|---|---|---|---|
| SSP size $d$ | 97 | 97 | 55 | 211 | 55 | 55 |
| HexSSP | True | True | True | True | True | True |
| Num. of simplex scales | 4 | 4 | 3 | 7 | 3 | 3 |
| Num. of simplex rotations | 4 | 4 | 3 | 5 | 3 | 3 |
| SSP length scale $\ell$ | 0.3 | 0.3 | 0.5 | 0.3 | 0.3 | 0.3 |
| VCO state size | 3 | 3 | 3 | 3 | 3 | 9 |
| Num. of neurons per VCO | 1000 | 1000 | 300 | 800 | 1000 | 3000 |
| Total num. of neurons | 48000 | 48000 | 8100 | 21600 | 27000 | 27000 |
| Recurrent synaptic constant $\tau_{syn}$ [s] | 0.05 | 0.05 | 0.05 | 0.05 | 0.05 | 0.05 |
| Trajectory dimensionality | 2 | 3 | 2 | 2 | 2 | 2 |
| Trajectory length [s] | 120 | 120 | 60 | 60 | 50 | 60 |
| Num. of trials | 10 | 10 | 1 | 1 | 10 | 10 |

**Table C.2:** Hyperparameters used to generate the results in Section 3.3.3.

| Parameter | Value | Range |
|---|---|---|
| Direct | | |
| Intercepts | $\mathcal{U}(-1.0, 0.9)$ | |
| Sparse RandSSP | | |
| Size $d$ | 151 | $[25, 55, 97, 151, 217, 295]$ |
| Length scale | 0.1 | |
| Intercepts | $\mathcal{U}(0.25, 0.7)$ | |
| Dense RandSSP | | |
| Size $d$ | 151 | $[25, 55, 97, 151, 217, 295]$ |
| Length scale | 0.1 | |
| Intercepts | $\mathcal{U}(-0.05, 0.2)$ | |
| HexSSP | | |
| Size $d$ | 151 | $[25, 55, 97, 151, 217, 295]$ |
| Number of simplex scales | 5 | $[2, 3, 4, 5, 6, 7]$ |
| Number of simplex rotations | 5 | $[2, 3, 4, 5, 6, 7]$ |
| Length scale | 0.1 | |
| Intercepts | $\mathcal{U}(-0.02, 0.02)$ | |
| Neuron model | LIF rate approximation | |
| Number of neurons | 151 | $[10, 32, 100, 316, 1000, 3162]$ |
| Variable range | $\pm$ 1.0 | $\pm[1.0, 3.25, 5.5, 7.75, 10.0]$ |
| Number of sample points $n_x$ | 10000 | |
| Gaussian noise $\sigma$ | $10^{-3}$ | |
| Number of trials | 10 | |

**Table C.4:** Hyperparameters used to generate the results in Figure 3.13. The SSP size, number of neurons per VCO, and maximum trajectory frequency were varied (values under 'Range') in these experiments. While one parameter was varied, the others remained fixed – these fixed values are listed under 'Fixed' along with the values of parameters that did not vary.

| Parameter | Fixed | Range |
|---|---|---|
| SSP size $d$ | 55 | 7-295 |
| HexSSP | True | |
| Number of simplex scales | 3 | 1-7 |
| Number of simplex rotations | 3 | 1-7 |
| SSP length scale $\ell$ | 0.3 | |
| Number of neurons per VCO | 800 | 100-1000 |
| Total number of neurons | 21600 | 100-147000 |
| Trajectory length [s] | 60 | |
| Trajectory generation | Band-lim. | |
| Maximum frequency [Hz] | 0.1 | 0.01-0.4 |
| Number of trials | 10 | |

**Table C.5:** The hyperparameters used for localization accuracy experiments with SSP-SLAM in Sections 4.4.2 and 4.4.3, *i.e.*, the results in Figures 4.8 to 4.10, and Table 4.1.

| Parameter | Value |
|---|---|
| SSP size $d$ | 97 |
| HexSSP | True |
| Num. of simplex scales | 4 (3) |
| Num. of simplex rotations | 4 (4) |
| SSP length scale $\ell$ | 0.3 |
| Number of neurons | 93080 |
| Per VCO | 1000 |
| Full SSP-PI | 49000 |
| Grid cells | 1000 |
| Landmark perception | 19600 |
| Environment map | 2910 |
| Loop closure | 19600 |
| View radius of agent | 0.2 m |
| Recurrent $\tau_{syn}$ [s] | 0.05 |
| Voja learning rate | $5 \times 10^{-4}$ |
| PES learning rate | $1 \times 10^{-3}$ |
| Environment bounds [m] | [-1,1] |
| Number of landmarks | 100 |
| Trajectory dimensionality | 2 (3) |
| Trajectory length [s] | 500 (250) |
| Trajectory generation | Band-limited |
| Maximum frequency [Hz] | 0.1 |
| Number of trials | 10 |

**Table C.6:** The hyperparameters used in Section 4.4.4 for SSP-SLAM in Nengo Loihi, *i.e.*, the results in Figure 4.11.

| Parameter | Value |
|---|---|
| SSP size $d$ | 55 |
| HexSSP | True |
| Num. of simplex scales | 3 |
| Num. of simplex rotations | 3 |
| SSP length scale $\ell$ | 0.3 |
| Number of neurons | 20400 |
| Per VCO | 300 |
| Full SSP-PI | 8100 |
| Grid cells | N/A |
| Landmark perception | 5600 |
| Environment map | 1100 |
| Loop closure | 5600 |
| View radius of agent | 0.2 m |
| Recurrent $\tau_{syn}$ [s] | 0.05 |
| Voja learning rate | N/A |
| PES learning rate | $1 \times 10^{-3}$ |
| Environment bounds [m] | [-1,1] |
| Number of landmarks | 100 |
| Trajectory dimensionality | 2 |
| Trajectory length [s] | 60 |
| Trajectory generation | Band-limited |
| Maximum frequency [Hz] | 0.1 |
| Number of trials | 10 |

**Table C.7:** The hyperparameters used for mapping example with SSP-SLAM in Section 4.4.1, *i.e.*, the results in Figures 4.6 and 4.7.

| Parameter | Value |
|---|---|
| SSP size $d$ | 241 |
| HexSSP | True |
| Num. of simplex scales | 8 |
| Num. of simplex rotations | 5 |
| SSP length scale $\ell$ | 0.3 |
| Number of neurons | |
| Per VCO | 500 |
| Full SSP-PI | |
| Grid cells | 1000 |
| Landmark perception module | |
| Environment map module | 4820 |
| Loop closure module | |
| View radius of agent | 0.2 m |
| Recurrent synaptic constant $\tau_{syn}$ [s] | 0.05 |
| Voja learning rate | $5 \times 10^{-4}$ |
| PES learning rate | $1 \times 10^{-3}$ |
| Environment bounds [m] | [-1,1] |
| Number of landmarks | 3 object, 2 walls |
| Trajectory dimensionality | 2 |
| Trajectory length [s] | 60 |

**Table C.8:** Hyperparameters for the value learning in a T-maze example (Figure 5.8) in Section 5.3.4.

| Parameter | ET-TD($\lambda$) | LDN-TD(0) | LDN-TD($\theta$) | LDN-TD($\lambda$) |
|---|---|---|---|---|
| State representation | RandSSP | RandSSP | RandSSP | RandSSP |
| SSP size $d$ | 201 | 201 | 201 | 201 |
| SSP length scale $\ell$ | 0.2 | 0.2 | 0.2 | 0.2 |
| Number of neurons | | | | |
| State population | 800 | 800 | 800 | 800 |
| Value population | 800 | 800 | 800 | 800 |
| LDNs (per variable) | 200 | 200 | 200 | 200 |
| TD update rule | Eq. (5.36) | Eq. (5.36) | Eq. (5.42) | Eq. (5.69) |
| Learning rate | $5 \times 10^{-3}$ | $5 \times 10^{-3}$ | $5 \times 10^{-3}$ | $5 \times 10^{-3}$ |
| LDN sizes $q$ | | | | |
| State activity $q_n$ | 1 | 2 | 8 | 8 |
| Reward $q_r$ | 2 | 2 | 8 | 8 |
| Value function $q_v$ | 2 | 2 | 8 | 8 |
| LDN window $\theta$ [s] | 0.1 | 0.1 | 1/6 | 6 |
| Discount $\gamma$ | 0.9 | 0.9 | 0.9 | 0.9 |
| Lambda $\lambda$ | 0.8 | N/A | N/A | 0.8 |
| Number of trials | 10 | 10 | 10 | 10 |

**Table C.9:** Hyperparameters for the value learning in a trace conditioning task example (Figure 5.9) in Section 5.3.4.

| Parameter | LDN-TD(0) | LDN-TD($\theta$) | LDN-TD($\lambda$) |
|---|---|---|---|
| State representation | LDN | LDN | LDN |
| State LDN size $q_s$ | 6 | 6 | 6 |
| State LDN window $\theta$ [s] | 10 | 10 | 10 |
| Number of neurons | | | |
| State population | 800 | 800 | 800 |
| Value population | 400 | 400 | 400 |
| LDNs | 200 | 200 | 200 |
| TD update rule | Eq. (5.36) | Eq. (5.42) | Eq. (5.69) |
| Learning rate | $5 \times 10^{-5}$ | $5 \times 10^{-5}$ | $5 \times 10^{-5}$ |
| Other LDN sizes $q$ | | | |
| State activity $q_n$ | 8 | 8 | 8 |
| Reward $q_r$ | 8 | 8 | |
| Value function $q_v$ | 8 | 8 | 8 |
| Other LDN window $\theta$ [s] | 8 | 8 | 8 |
| Discount $\gamma$ | 0.95 | 0.95 | 0.95 |
| Lambda $\lambda$ | N/A | N/A | 0.8 |

**Table C.10:** Hyperparameters for both the successor feature learning in a hairpin maze example (Figure 5.11) in Section 5.4.1 and the two goals example (Figure 5.12) in Section 5.4.1.

| Parameter | Value |
|---|---|
| State representation | HexSSP |
| SSP size $d$ | 337 |
| SSP length scale $\ell$ | 0.2 |
| Number of neurons | |
| State population | 800 |
| Reward population | 800 |
| SF population | 800 |
| LDNs | 200 |
| TD update rule | Eq. (5.69) |
| SF learning rate | $4 \times 10^{-3}$ |
| Reward learning rate | $1 \times 10^{-3}$ |
| LDN sizes $q$ | |
| State $q_s$ | 8 |
| State activity $q_n$ | 8 |
| SF function $q_{\text{SF}}$ | 8 |
| LDN window $\theta$ [s] | 10 |
| Discount $\gamma$ | 0.9 |
| Lambda $\lambda$ | 0.9 |

**Table C.11:** Hyperparameters for the variable magnitude reward example (Figure 5.13) in Section 5.4.2. The reward values were sampled from Categorical($[0.1, 0.3, 1.2, 2.5, 5, 10, 20]$), with all values having equal probability.

| Parameter | Value |
|---|---|
| State representation | RandSSP |
| SSP size $d$ | 1001 |
| SSP length scale $\ell$ | 1.0 |
| Number of neurons | |
| State population | 50 |
| Reward population | 3000 |
| Reward distribution | 6000 |
| LDNs | 200 |
| TD update rule | Eq. (5.36) |
| Learning rate | $1 \times 10^{-5}$ |
| LDN sizes $q$ | |
| State activity $q_n$ | 1 |
| Reward $q_r$ | 2 |
| Reward distribution $q_Z$ | 2 |
| LDN window $\theta$ [s] | 1.0 |
| Discount $\gamma$ | 0.5 |

**Table C.12:** Hyperparameters for the Gaussian mixture sampling example (Figure 5.15) in Section 5.5.2.

| Parameter | Value |
|---|---|
| State representation | RandSSP |
| SSP size $d$ | 1024 |
| SSP length scale $\ell$ | 1.6 |
| Number of neurons | 2000 |
| Recurrent $\tau_{syn}$ [s] | 0.1 |
| Sampler rate $\alpha$ | $1.3 \times 10^{-2}$ |
| Sampler noise standard dev. $\sigma$ | $4.7 \times 10^{-2}$ |

**Table C.13:** Hyperparameters for the bandit task example (Figure 5.16) in Section 5.5.

| Parameter | Value |
|---|---|
| Action representation | Random SPs |
| Action SP size $d$ | 100 |
| Number of neurons | |
| State population | 10 |
| Policy population | 500 |
| Sampler distribution | 6000 |
| Sampler rate $\alpha$ | 5.0 |
| Sampler standard dev. $\sigma$ | 5.5 |
| Sampler $\tau_{syn}$ [s] | 0.1 |
| Policy learning rate | $2 \times 10^{-4}$ |
| Num. of actions | 10 |
| Action means | $m_i \sim \text{Unif}(-1, 1)$ |
| Action std. | $\sigma_i \sim \text{Unif}(0, 0.05)$ |

**Table C.14:** Hyperparameters for the water maze example (Figure 5.17) in Section 5.5.

| Parameter | Value |
|---|---|
| State representation | RandSSP |
| State SSP size $d$ | 55 |
| State SSP length scale $\ell$ | 0.2 |
| Action representation | RandSSP |
| Action SSP size $d$ | 601 |
| Action SSP length scale $\ell$ | 0.005 |
| Number of neurons | |
| State population | 165 |
| Policy population | 3005 |
| Sampler population | 5000 |
| Value population | 500 |
| LDNs | 200 |
| Sampler rate $\alpha$ | 10 |
| Sampler standard dev. $\sigma$ | 150 |
| Sampler $\tau_{syn}$ [s] | |
| Policy learning rate | $5 \times 10^{-4}$ |
| Value learning rate | $1 \times 10^{-3}$ |
| LDN sizes $q$ | |
| State activity $q_n$ | 8 |
| Reward $q_r$ | 8 |
| Value $q_v$ | 8 |
| LDN window $\theta$ [s] | 0.1 |
| Discount $\gamma$ | 0.9 |

**Table C.15:** Hyperparameters for the DeepMind Control Suite tasks (6.2) in Section 6.3.2. Most values are taken from Li et al. (2021).

| Parameter | Value |
|---|---|
| Algorithm | Soft Actor-Critic (SAC) |
| Codebase | Adapted from Li et al. (2021) |
| Optimizer | Adam |
| Adam parameters $(\beta_1, \beta_2)$ | $(0.9, 0.999)$ |
| Discount $\gamma$ | 0.99 |
| Batch size | 1024 |
| Target smoothing coefficient | 0.005 |
| Reward scale | Auto-tuned |
| Actor learning rate | $1 \times 10^{-4}$ |
| Critic learning rate | $1 \times 10^{-4}$ |
| Reward scale learning rate | $1 \times 10^{-4}$ |
| Number of warmup steps | 5000 |
| Number of hidden layers | 3 for state baseline,<br>2 for RFFs and SSPs |
| Hidden size | 1014 |
| Hidden nonlinearity | ReLU |
| SSP and RFF size $d$ | 1024 |
| Initial RFF standard dev$^{\cdot}$ | 0.001 |
| Initial SSP length scale $\ell$ | 0.001 |

**Table C.16:** Hyperparameters for the MiniGrid tasks (6.6) in Section 6.4.2.

| Parameter | Value |
|---|---|
| Algorithm | Advantage Actor-Critic (A2C)<br>with Generalized Advantage Estimation (GAE) |
| Codebase | Adapted from Willems (2023) |
| Optimizer | RMSProp |
| RMSProp $\alpha$ | 0.99 |
| RMSProp $\varepsilon$ | $10^{-8}$ |
| Max. gradient norm | 10 |
| Num. concurrent environment processes | 5 |
| Num. of frames processes before update | 5 |
| Discount $\gamma$ | 0.99 |
| Lambda $\lambda$ | 0.95 |
| Entropy regularization coefficient | 0.001 |
| Actor hidden size | 64 |
| Critic hidden size | 64 |
| Actor learning rate | $1 \times 10^{-3}$ |
| Critic learning rate | $1 \times 10^{-3}$ |
| Hidden nonlinearity | ReLU |
| SSP size $d$ | 1353 |
| Initial SSP length scale $\ell$ | 1.0 |