Biologically Inspired Spatial Representation

by

Brent Komer

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Systems Design Engineering

Waterloo, Ontario, Canada, 2020

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:	Caswell Barry Professor, Dept. of Cell and Developmental Biology, University College London
Supervisor:	Chris Eliasmith Professor, Dept. of Systems Design Engineering, University of Waterloo
Internal Member:	Bryan Tripp Associate Professor, Dept. of Systems Design Engineering, University of Waterloo
Internal-External Member:	Jeff Orchard Associate Professor, David R. Cheriton School of Computer Science, University of Waterloo
Internal-External Member:	Steven Waslander Associate Professor, Dept. of Mechanical and Mechatronics Engineering, University of Waterloo

Author's Declaration

I hereby declare that I am the sole author of this 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.

Abstract

In this thesis I explore a biologically inspired method of encoding continuous space within a population of neurons. This method provides an extension to the Semantic Pointer Architecture (SPA) to encompass Semantic Pointers with real-valued spatial content in addition to symbol-like representations. I demonstrate how these Spatial Semantic Pointers (SSPs) can be used to generate cognitive maps containing objects at various locations. A series of operations are defined that can retrieve objects or locations from the encoded map as well as manipulate the contents of the memory. These capabilities are all implemented by a network of spiking neurons.

I explore the topology of the SSP vector space and show how it preserves metric information while compressing all coordinates to unit length vectors. This allows a limitless spatial extent to be represented in a finite region. Neurons encoding space represented in this manner have firing fields similar to entorhinal grid cells.

Beyond constructing biologically plausible models of spatial cognition, SSPs are applied to the domain of machine learning. I demonstrate how replacing traditional spatial encoding mechanisms with SSPs can improve performance on networks trained to compute a navigational policy. In addition, SSPs are also effective for training a network to localize within an environment based on sensor measurements as well as perform path integration. To demonstrate a practical, integrated system using SSPs, I combine a goal driven navigational policy with the localization network and cognitive map representation to produce an agent that can navigate to semantically defined goals.

In addition to spatial tasks, the SSP encoding is applied to a more general class of machine learning problems involving arbitrary continuous signals. Results on a collection of 122 benchmark datasets across a variety of domains indicate that neural networks trained with SSP encoding outperform commonly used methods for the majority of the datasets.

Overall, the experiments in this thesis demonstrate the importance of exploring new kinds of representations within neural networks and how they shape the kinds of functions that can be effectively computed. They provide an example of how insights regarding how the brain may encode information can inspire new ways of designing artificial neural networks.

Acknowledgements

Writing this thesis would not have been possible without the support of many people. I would like to thank my supervisor, Chris Eliasmith, for all of his support and guidance throughout the years. I could not have asked for a better mentor and the outstanding work he has put into cultivating a lab environment that is welcoming, fun, and challenging is incredible. I can not overstate how fortunate I am to have been a part of the Computational Neuroscience Research Group.

A huge thanks to all members of the CNRG lab, especially those I had the pleasure of living with; Xuan, Pete, Jan, and Andreas. My time here has been amazing and I have learned so much. I really appreciate all of the wonderful friends I have met over the years. In particular I would like to thank Eric De Val for his enthusiasm and humour keeping me sane and Julia Chernushevich for continually pushing me out of my comfort zone.

Last but certainly not least, I want to thank my family for always being there and the immense support they have given me throughout all aspects of my life. I am extremely grateful for my parents, Barb and Bill, for all that they have taught me and the help they have provided, as well as for my siblings, Will, Rob, Matt, Andrew and Leah. A special thanks goes out to Sally Siu for her endless encouragement and patience.

Dedication

To Sally, the love of my life, who has been here through it all

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

The world around us is vast and complex. To understand ourselves and our surroundings we must encode facets of this infinite world within our finite brains. One prominent feature of our world is that everything we see around us exists at locations in space. It is important to be able to reason about space for survival. An animal searching for food will need to remember how to get to promising areas as well as remember places it has already unsuccessfully checked. Learning shortcuts in an environment, navigating around dangerous areas, and remembering the way back to shelter are all abilities that rely on an internal representation of the vast outside world.

This internal representation must be very efficient as there is simply too much information available to encode it all accurately. As a result, such a representation must capture some features accurately and ignore or minimize others. The internal representation shapes the class of functions that can be computed with it, and in turn, the kinds of behaviours that can be expressed. Animals have been subject to millions of years of evolution and, because of the importance of spatial information, their brains almost certainly have developed very effective spatial encoding strategies in this time.

Putting aside biological considerations, spatial reasoning is also important for robotic applications, such as self driving cars or automated search and rescue missions. These systems need to use sensory information to construct a map of the environment, estimate their location within this map, and use this information to plan a trajectory to a given goal. A growing area of research focuses on applying deep learning to solve these problems [118, 187]. Given the overlap in problem domains, and the prominence of neural network architectures in both kinds of system, it would be unsurprising if leveraging the structure of biological representations may inform the design of better artificial systems. In addition,

algorithms inspired by the brain and implementable with spiking neural networks allows the capability to take advantage of certain kinds of neuromorphic hardware [114, 122, 123, 105]. This hardware is modelled after the brain, where all communication between neurons is facilitated through discrete spike events that occur at any continuous point in time. Communication is asynchronous allowing processing to be massively parallel. Neuromorphic hardware is capable of performing computations at a fraction of the energy cost of traditional computing paradigms [71, 16].

The goal of this thesis is two-fold, to help understand how spatial cognition could be realized in a biological brain, and to apply what is learned from biology to artificial systems. As such, this work is situated between computational neuroscience and machine learning. Our journey begins in the realm of computational neuroscience, taking inspiration from studies of the brain regions implicated in spatial cognition, as well as computational models of higher-level reasoning. With this background in hand I propose a mathematical foundation of a new kind of spatial representation. I initially explore this representation in the context of biologically inspired models of spatial cognition and eventually consider it for applications to machine learning for spatial tasks. I then consider non-spatial tasks in machine learning, which is motivated by evidence that the areas of the brain previously thought to be devoted to reasoning purely about physical space are much more general [44, 171, 170].

Many connections between neuroscience and machine learning have already been found [99, 109]. Methods for reinforcement learning modelled after the brain (specifically basal ganglia and frontal cortex) have been shown to have advantages over standard approaches [146]. One of the most striking connections is the similarity between Convolutional Neural Networks (CNNs) and the primate visual system. CNNs are multi-layered neural networks trained on images, and the features learned at different depths strongly correlate to the neural responses at different levels in the human visual system [190]. There are many more connections to explore and new pathways to take. My hope is that this work charts one additional fruitful course through this research landscape.

1.1 Outline

The next chapter (Chapter 2) contains background material as it relates to this thesis. It is divided into three main sections. The first (Section 2.1) covers findings in the neuroscience literature as they relate to spatial cognition. This includes regions of the brain involved, cell types that have been found to have spatially tuned response characteristics, and tasks used to measure spatial cognition. The second (Section 2.2) provides background on the

computational modelling methods that will be used and expanded upon in this work. The third (Section 2.3) gives an overview of the relevant machine learning techniques that will be used throughout this work.

Chapter 3 introduces Spatial Semantic Pointers (SSPs), a core method for spatial representation used in this work. The motivation and mathematical formulation are presented, including examples of the types of spatial operations that can be performed. This chapter also describes methods for generating SSPs with different properties depending on the application.

Chapter 4 describes a set of useful spatial operations and a series of experiments evaluating SSP performance on those operations. This chapter illustrates the ability of SSP representations to be implemented in spiking neural networks. Examples include a model of hierarchical spatial encoding, mental imagery, and association learning. Relations between SSPs and spatially selective neurons in the brain are also explored.

Chapter 5 provides a series of experiments comparing the effectiveness of SSPs to other spatial encodings typically used in machine learning on a variety of tasks. These tasks define sub-components that are integrated into a single navigation system. This system is capable of navigating a series of environments to the location of a desired goal object. The agent is able to localize using visual cues and path integrate using both self motion and vision. Given a semantically provided goal object, it is able to retrieve the location of the object from memory and choose movement actions based on the estimated location of itself and the goal object. Experiments are provided demonstrating the effectiveness of this system.

Chapter 6 explores the use of SSPs as a general encoding method that can be applied to more than just spatial information. Neural networks are trained with arbitrary continuous features being encoded as input. Experiments are conducted on a diverse suite of classification and regression benchmark datasets. This chapter also examines properties of SSPs and other spatial encodings to provide insight into how neural networks learn to use the representation.

Chapter 7 concludes the thesis with a summary of contributions and a discussion with potential areas of future work.

Chapter 2

Background

2.1 Spatial Cognition

Spatial cognition deals with the ability to reason about spatial relationships. This includes relatively simple abilities such as estimating distances or understanding the arrangement of objects in front of you. It also includes spatial memory, such as remembering where you parked your car or being able to mentally picture the layout of the neighbourhood where you grew up. Both spatial memory and spatial reasoning combine to allow complex navigational behaviour, such as finding your way around the Psychology building. Familiar visual cues can help situate your location in a mental map being constructed on-the-fly. Upon entering a previously visited environment, you can navigate more quickly based on past experience as well as infer novel shortcuts.

Converging evidence points to structures in the medial temporal lobe region being important for processing spatial information [65, 22]. In particular, the hippocampus, entorhinal cortex, and retrosplenial cortex are important for allocentric spatial information, and the posterior parietal cortex is implicated in processing egocentric information [25].

Neural recordings from animals performing spatial tasks give insight into the particular functions different regions could be performing. A broad graphical overview of the connectivity and cell types involved is shown in Figure 2.1. The individual classes of neurons implicated in processing spatial information are described in more detail below. The majority of the data comes from studies in rodents, but advances in imaging techniques are helping to bridge the gap to humans.

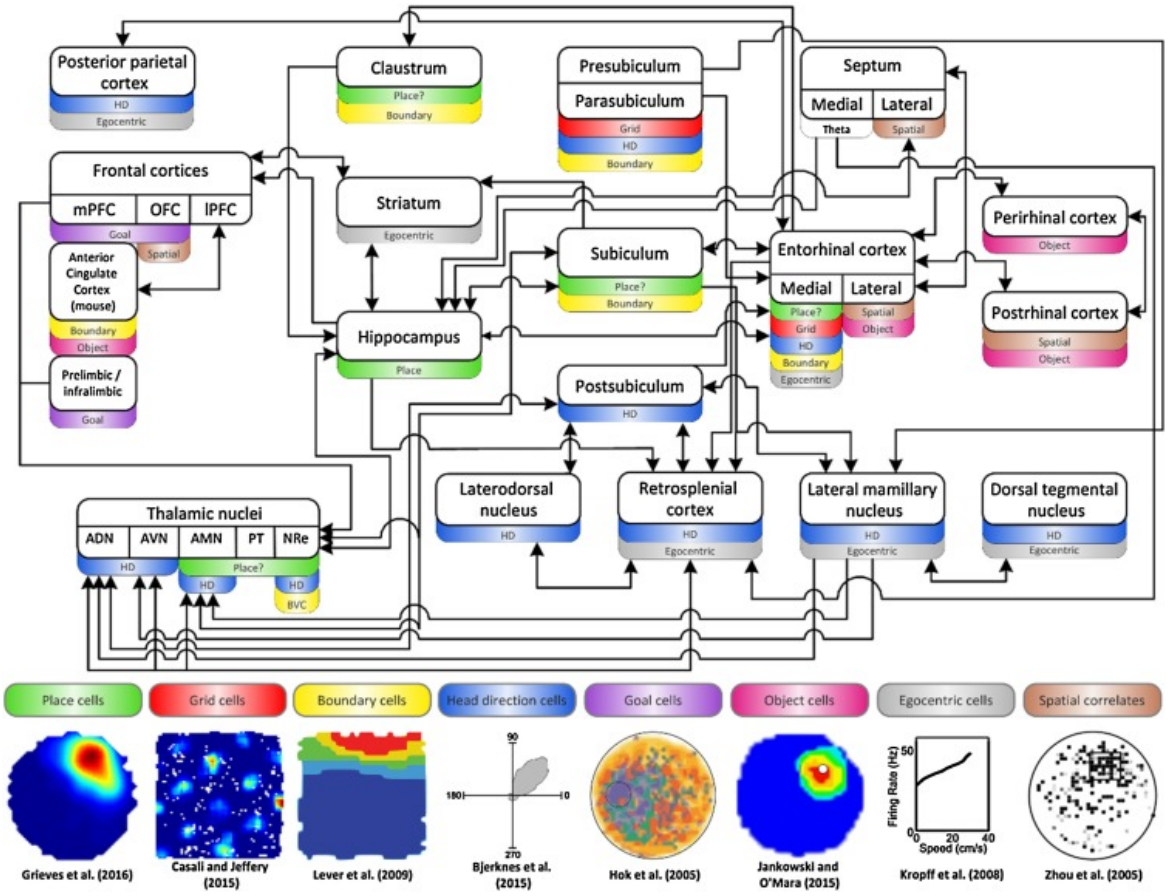


Figure 2.1: **Brain Regions Involved in Spatial Cognition**. This diagram summarizes decades of work analyzing connectivity between brain regions and the associated spatially selective cells found within each region. Image from [65]. Reprinted with permission from Elsevier.

Place Cells

Place cells are a type of pyramidal neuron found within the hippocampus that fire when the animal is within a particular region of space. This region of space is known as the cell's 'place field'. Place cells were first discovered by John O'Keefe and Jonothan Dostrovsky in 1971 [128] and most studies on these type of cells have been carried out in rodents. There is also evidence for place cells in bats [151], monkeys [108, 111, 76], and humans [45]. Place fields are formed shortly after entering a new environment [188] and can be stable for months when revisiting the same area [172]. In small environments, each place cell tends to only have one corresponding place field, while in larger environments cells have been observed to have multiple fields [134]. There is no apparent pattern or topography to the place fields (the field location can not be estimated from the recording location in the brain) [127]. When the environment is modified, many place cells will remap to new locations with no clear structure guiding this remapping [121].

Fields are directional when the environment is limited (e.g., a long hallway), but independent of direction when it is open [112, 120]. The shape of a place field can be non-circular and vary with changes in the environment [126]. There are also cells that have been observed in monkeys that are very similar to place cells but instead of being active when the animal is in a particular location they are active when the animal is looking at a particular location. These cells have been called Spatial View Cells [149, 56].

Place cells demonstrate both pattern completion and pattern separation qualities [119, 148]. Pattern completion is the ability to retrieve a complete memory based on partial information. Pattern separation is the ability to differentiate very similar memories as distinct. Place cell remapping has been hypothesized to play a role in memory formation by facilitating pattern separation [30].

Grid Cells

Grid cells are similar to place cells except that instead of having one or a few place fields they have many place fields arranged in a regular hexagonal grid. These cells were first discovered in 2005 by Edvard and May-Britt Moser [69]. Grid cells have been primarily studied in the medial entorhinal cortex (MEC), but are also found in the presubiculum (PrS) and parasubiculum (PaS) [18]. Grid cells next to each other tend to show the same grid spacing and orientation, but the grid vertices are displaced by random offsets [157]. Cells recorded from a separate electrode at a distance from one another show different grid spacings. The size of the firing field is correlated with the location of the cell, with cells located more ventrally having larger firing fields and a greater spacing between the

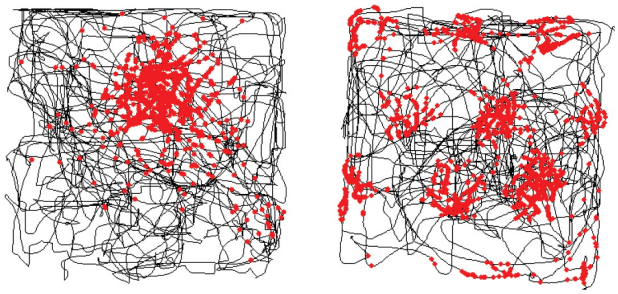


Figure 2.2: **Place and Grid Cells**. A rodent is placed in a square arena and can freely move around while neurons are being recorded. In black is the trajectory of the rodent. Every time the neuron of interest fires, a red dot is placed at the animal's current location. Shown on the left is a neuron that preferentially fires at a specific location in space and is therefore classified as a Place Cell. Shown on the right is a neuron that fires in multiple regions organized as a regular hexagonal grid and as such is classified as a Grid Cell. Image from [119]. Reprinted with permission from Annual Review of Neuroscience.

vertices while cells in the dorsal area have smaller firing fields that are closer together [69]. Grid cells are organized into a finite number of discrete modules, where cells within a given module all share similar scales and grid orientations [163]. Grid cells belonging to the same module, as opposed to different modules, show similarity in their re-scaling properties in response to changes in the environment. Figure 2.2 (a) depicts the spatial firing of a place cell while a rodent forages in a square environment. Figure 2.2 (b) depicts the spatial firing of a grid cell in a similar environment.

More recently, additional non-grid spatially periodic cells have been discovered in MEC and PaS. These include neurons with firing fields that form periodic stripes at particular orientations, known as band cells [98], as well as spatially periodic cells that form irregular grids [97]. The PaS contains a greater proportion of spatially periodic cells than the MEC, although the MEC contains a greater proportion of hexagonally symmetric cells [98].

Head Direction Cells

Head direction cells are neurons that preferentially fire when an animal's head is facing a particular direction in allocentric space. They were first discovered in 1984 by James Ranck [142]. Each cell has a preferred direction that fires maximally, with firing rates decreasing around 45 degrees away from the preferred direction [168]. The preferred directions are uniformly distributed across the 360 degree space [169]. Even when an animal is sleeping

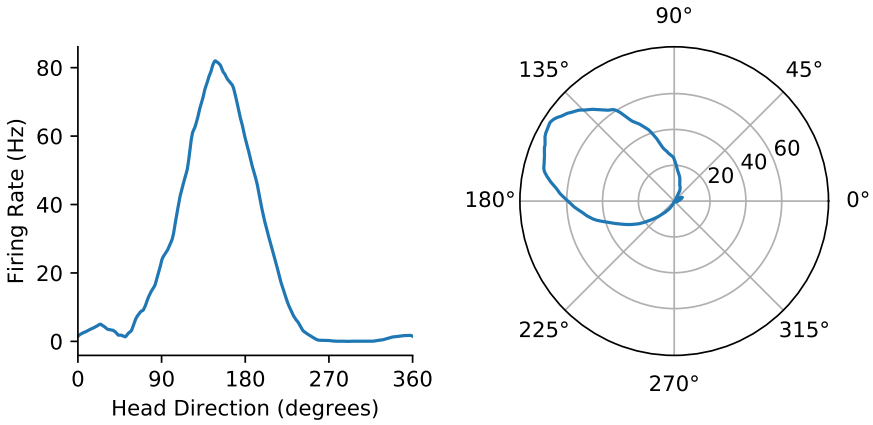


Figure 2.3: **Head Direction Cell Activity**. Example of what the firing characteristics of a typical head direction cell might look like. Shown on the left is the firing rate of the neuron as a function of orientation in a fixed reference frame. The same data is visualized with a polar plot on the right. The maximum firing rate occurs at approximately 150 degrees, which is considered this neuron's preferred direction.

the firing activity of the head direction cells remain consistent with one another. The activity drifts around but only represents a single direction at a given time [138]. A visual depiction of the activity of these cells is shown in Figure 2.3.

Boundary Cells

Boundary cells preferentially fire when the animal is at a particular distance and direction from an obstacle. These cells have been found primarily in the entorhinal cortex [160] and subiculum [104].

More recently, some cells have been found that respond to all boundaries rather than a particular location [78, 186, 79]. These have been identified in the anterior claustrum [78], anterior cingulate cortex [186], and nucleus reuiniens [79]. In these same regions there are cells that are only active if there are no nearby boundaries.

Boundary cells are sensitive to more than just perception, they fire based on self-motion cues and the memory of a boundary [104, 144]. If an animal turns its head so that it no longer sees the boundary, the cell will still fire, maintaining the representation of the boundary that is out of view.

Neural responses of boundary cells recorded in rodents are illustrated in Figure 2.4.

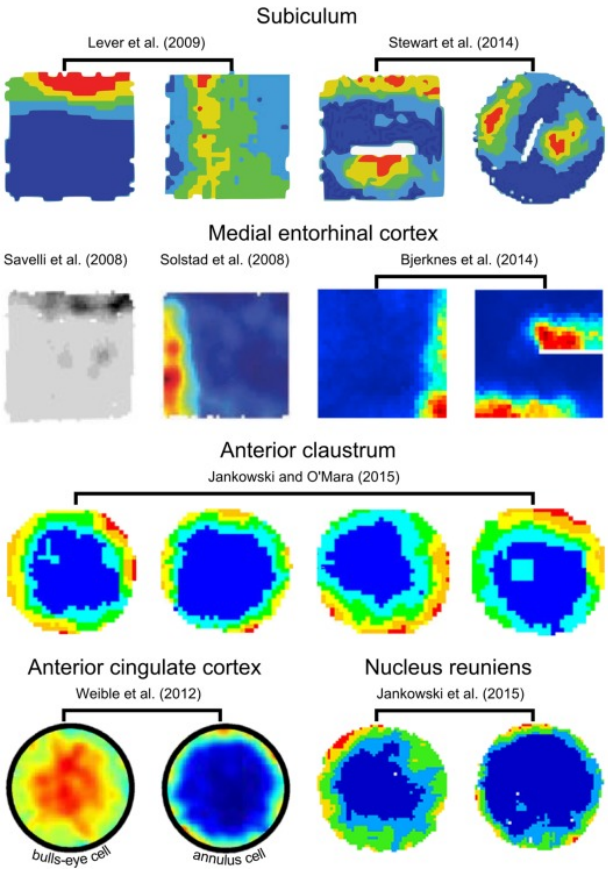


Figure 2.4: **Boundary Cells**. These images are created by discretizing the environment into spatial bins and colouring each bin based on the firing rate of a particular neuron when the animal is in that space. Blue corresponds to little to no activity, and red corresponds to the maximum observed firing rate. Particular cells will fire based on the presence of an obstacle in a particular distance and direction from the animal (top set of 8). Some neurons will fire based on the distance to a wall at any orientation (bottom set of 8). Image from [65]. Reprinted with permission from Elsevier.

Speed Cells

An animal's position changes over time as it navigates, so a measure of self-motion is an important component for updating a representation of position. Speed cells have been discovered in the MEC and are theorized to play a strong role in self-localization [96]. These neurons are sensitive to the speed of the animal, regardless of the direction. Combined with the head direction cells found in the same brain region, a velocity vector can be estimated.

Goal Cells

Studies in human navigation have identified neural activity correlated to distance to a goal [7] and direction to a goal [27]. Neural recordings in bats have also isolated neurons sensitive to the distance or direction to a goal [38]. Some neurons represent both distance and direction to form a vector representation to the goal, including occluded locations that must be maintained in memory [156].

2.2 Computational Neuroscience

Many decades of experimental work is being continuously integrated to paint an ever improving picture of the neural circuitry involved in spatial cognition. A complementary research goal is to construct models derived from these experimental results to test theoretical claims regarding the functions involved and to make testable predictions to guide future experiments. This section highlights individual models that capture aspects of spatial cognition, followed by more general techniques for constructing biologically plausible neural models of various functions.

2.2.1 Models of Spatial Cognition

An important spatial ability for many animals is to be able to find their way back to where they started from. One way to accomplish this is through keeping an up-to-date representation of the current position relative to some origin by integrating self-motion over time. This process is known as path integration. When returning to the start of a trajectory, the shortest direction can be easily inferred from the starting and current locations in a consistent frame of reference, instead of following the exact, potentially winding path the animal took to get to the current location in the first place.

There is strong evidence that grid cells are a key component in path integration as mice with disrupted grid cell firing show impaired path integration ability. This was found through experiments conducted on mice genetically modified to specifically disrupt grid cell firing while leaving head direction and place cells intact [57].

It is widely believed that all of the cell types described in Section 2.1 are involved in a path integration circuit although the exact mechanisms are not known. Many computational models have been proposed to attempt to explain the data observed as well as make predictions for future experiments. Models of path integration can be broadly divided into two categories; attractor models and oscillatory interference models.

Attractor Models

An attractor is any system where evolution of the state space over time tends towards a smaller subspace. This subspace could be a single point, a set of points, or any continuous manifold. A network of neurons can form an attractor when the connectivity causes the dynamics of the state space to settle in a particular region or to form a particular temporal pattern. Often a continuous attractor is approximated by a discrete set of attractor points organized along the manifold of interest. In the limit as the number of neurons grows to infinity, this attractor becomes continuous.

Early attractor models focused on modelling the head direction system [159, 195]. Here neurons are arranged in a ring, with the location on the ring corresponding to the directional tuning. Neurons with similar preferred directions are more strongly connected to each other, while neurons with very different preferred directions inhibit each other. At steady state, this connectivity allows a stable bump of activity to form somewhere in the ring. When the connections between neurons are made slightly asymmetric, this bump of activity will move around the ring in proportion to that asymmetry. In these models this asymmetry is controlled through head motion cells.

The idea of a ring of neurons can be extended to a 2D sheet to represent spatial location [154]. To eliminate boundary effects, this sheet can be connected across the boundaries in a torus topology. Instead of explicitly arranging neurons in a sheet, populations of neurons can be connected such that the state space they represent is a 2D surface with a means of moving along this surface [31].

With the discovery of grid cells these 2D models were further modified to account for the hexagonal structure in the neural activity by using a twisted torus topology [113, 66].

Oscillator Interference Models

Another class of models posits that path integration can be achieved through the interference patterns of various oscillators. These ideas originated with the discovery of phase precession in place cells [129] and later in grid cells as well [68]. When an animal is within the place field of a neuron, the neuron will periodically emit short bursts of spikes. These bursts are in phase with the theta oscillation, which is a large amplitude oscillation in the local field potential that occurs during a variety of voluntary behaviours, including running [189]. This oscillation is typically 7-12Hz in rodents [129] but appears to be more varied and slower (2-8Hz) in humans [193]. As the animal moves through the field, the phase at which the neuron spikes becomes progressively earlier in the theta cycle, to the point where when the animal is in the center of the field, firing is anti-phase to theta [129]. One mechanism that can explain this effect is the interference of two membrane-potential oscillators of different frequencies affecting the same neuron [129, 102]. The phase precession effect can arise when one oscillator has a relatively constant frequency while the other is modulated by velocity. This effect can also be explained by injecting theta into an attractor model [31].

Early models inspired by grid cells use sets of three speed-modulated oscillators sensitive to speed in directions oriented 60 degrees apart [23, 58]. Each of these oscillations interact with a baseline oscillation (assumed to come from theta). When a threshold is applied to the interference, hexagonal grid-like patterns emerge, reminiscent of entorhinal grid cells.

Another approach is to use populations of interconnected neurons to form each oscillator [198]. In this model, many velocity controlled oscillators and one baseline oscillator insensitive to velocity are used. Each grid cell generally receives input from three of these oscillators and the resulting firing pattern is spatially periodic. Velocity controlled oscillator models can be reformulated in terms of Fourier theory and implemented with populations of spiking neurons [132].

Although many models make use of theta oscillations, it has been shown that these oscillations are not required for path integration [131]. Recordings from bats show grid cell activity despite no characteristic theta oscillations [49]. Place cells still showed synchronization to this non-rhythmic field potential, indicating that a phase code is still preserved across species.

RatSLAM

Moving more into the domain of engineering, the RatSLAM [116] system takes inspiration from the hippocampus to perform simultaneous localization and mapping (SLAM). A main

component of RatSLAM is the 'pose cell' network which is used to represent the agent's state. Every node corresponds to a different position and heading direction (x, y, θ). The spatial locations of the nodes are defined in a torus configuration (i.e. opposite faces are connected to each other) so that every node has the same number of neighbors and there are no edges. Connection weights between nodes follow a difference of Gaussians shape, with nearby nodes having positive connections and further nodes having negative connections. This forms attractor dynamics, where a small cluster of nearby nodes will remain active and suppress activity outside of the cluster. Connection strengths are modulated by velocity to allow a stable bump of activity to move throughout the network. When an agent moves through an environment, the activation of particular pose cells will repeat periodically, similar to the firing patterns observed in grid cells. Another component of RatSLAM is 'view cells'. These are a group of nodes that each represent unique visual input and can be thought of as analogous to place cells. When a visual stimulus is encountered that is sufficiently different than any previous stimuli, a new view cell is created and associated to the currently most active pose cell. If the visual stimulus closely matches a previously existing view cell, energy will be injected into the pose cell network around the pose cell associated to this view cell. This algorithm has been implemented to run in real time on a robot.

While building individual models of various neural circuits is useful, it is also important to have more general techniques for constructing and scaling neural models. The following section describes a framework for building large scale spiking neural networks from a mathematical description of the functions involved.

2.2.2 The Neural Engineering Framework

The Neural Engineering Framework (NEF) [47] is a method for building functional biologically plausible neural systems based on three principles (representation, transformation, and dynamics). Within this framework arbitrary vectors can be represented by the activity of neural ensembles. This is done through a nonlinear encoding mechanism carried out by the neuron tuning curves, and a weighted linear decoding of the neurons' responses to retrieve an approximation of the vector being encoded. A different choice of weights on the decoding can allow arbitrary transformations to occur from the original input. Importantly, all of this can be simulated with spiking neurons and takes place in a dynamical system where timing effects and filters across connections play a role in the system behaviour.

The software package Nengo [12] implements the principles of the NEF and provides a Python interface for constructing complex neural models. Nengo supports a variety of

underlying neuron models but the most commonly used is the Leaky Integrate-and-Fire (LIF) neuron [24].

Encoding in the NEF is computed using:

$$a_i = G_i(\alpha_i x \cdot e_i + J_i^{\text{bias}}) \tag{2.1}$$

where a_i is defined as the instantaneous firing rate or 'activity' of the ith neuron, x is the state variable being encoded, and e_i is the encoder or 'preferred direction' of the neuron. Each neuron also has a gain α_i and bias J_i^{bias}. A non-linearity G_i corresponding to the spiking or non-spiking neuron model is applied to the current. For a LIF neuron, this function is described by:

$$G(J) = \frac{1}{\tau_{\text{ref}} - \tau_{\text{RC}} \ln\left(1 - \frac{1}{J}\right)} \tag{2.2}$$

where τ_{ref} is the time constant of the refractory period, τ_{RC} is the time constant of the cell membrane, and J is the current being injected into the cell.

The state variable can be estimated from the activities of neurons by performing a weighted linear decoding:

$$\hat{x} = \sum_{i=0}^{N} a_i d_i \tag{2.3}$$

where N is the number of neurons in the ensemble, d_i is the decoder for the ith neuron and a_i is the activity of that neuron as described by Equation 2.1. Alternatively, any function of x can be approximated using an alternative set of linear decoders, denoted as d' in:

$$\hat{f}(x) = \sum_{i=0}^{N} a_i d_i' \tag{2.4}$$

A full connection weight matrix can be computed between two populations of neurons by taking the outer product of the decoders of the pre-population and the scaled encoders of the post-population:

$$W_{i,j} = d_i' \cdot \alpha_i \cdot e_i \tag{2.5}$$

where $W_{i,j}$ is the connection weight between the i-th neuron in the pre-population and the j-th neuron in the post-population.

Decoders are obtained using an L_2 regularized least squares optimization on a set of state vectors and their corresponding neural activities:

$$\mathbf{D} = (\mathbf{A}\mathbf{A}^T + k\sigma^2\mathbf{I})^{-1}\mathbf{A}\mathbf{X}^T \tag{2.6}$$

where $\mathbf{D} \in \mathbb{R}^{n \times d}$ is a matrix containing the d-dimensional encoders for each of the n neurons, $\mathbf{A} \in \mathbb{R}^{n \times k}$ is a matrix containing the activity of each neuron for each of the k sample points (calculated using Equation 2.1), σ is the standard deviation of Gaussian noise to account for, $\mathbf{I}$ is an $n \times n$ identity matrix, and $\mathbf{X} \in \mathbb{R}^{d \times k}$ is a matrix containing state vector inputs for every sample point.

In addition to connectivity between neurons, there are also important temporal dynamics that arise from the synapses between neurons. These synapses can be modelled as a low-pass filter on the incoming spikes to the neuron. A commonly used model of a synaptic filter in Nengo is a 1st order exponential low-pass filter:

$$h(t) = \frac{e^{-t/\tau}}{\tau} \tag{2.7}$$

where the synaptic time constant τ can be chosen based on measurements from the brain region being modelled.

The 3rd principle of the NEF posits that neural representations can be treated as control theoretic state variables, allowing the neural dynamics to be analyzed using control theory. Consider a standard Linear-Time-Invariant (LTI) system describing the rate of change of a state variable x given the current state and an input u:

$$\dot{x}(t) = \mathbf{A}x(t) + \mathbf{B}u(t) \tag{2.8}$$

An equivalent neural LTI system can be constructed using a recurrently connected network with a synaptic filter $h(t)$:

$$x(t) = h(t) * (\mathbf{A}'x(t) + \mathbf{B}'u(t)) \tag{2.9}$$

Using the synapse model from Equation 2.7 with a synaptic time constant of τ the new control matrices are as follows:

$$\begin{aligned} \mathbf{A}' &= \tau\mathbf{A} + \mathbf{I} \\ \mathbf{B}' &= \tau\mathbf{B} \end{aligned} \tag{2.10}$$

These matrices can be used in conjunction with the encoders and decoders to solve for the connection weight matrices for the input and recurrent connections:

$$W_{in} = \mathbf{D}' \cdot \mathbf{B}' \cdot \mathbf{E}$$
$$W_{rec} = \mathbf{D} \cdot \mathbf{A}' \cdot \mathbf{E} \tag{2.11}$$

where $W_{in} \in \mathbb{R}^{n_u \times n_x}$ is the weight matrix for the input connection, $W_{rec} \in \mathbb{R}^{n_x \times n_x}$ is the weight matrix for the recurrent connection, $\mathbf{D}' \in \mathbb{R}^{n_u \times d_u}$ are the decoders from the input population, $\mathbf{D} \in \mathbb{R}^{n_x \times d_x}$ are the decoders from the recurrent population, $\mathbf{E} \in \mathbb{R}^{d_x \times n_x}$ are the scaled encoders for the recurrent population. d_u and d_x are the dimensionalities of the state space of the input and recurrent populations respectively, and n_u and n_x are the number of neurons in those populations.

2.2.3 The Semantic Pointer Architecture

The Semantic Pointer Architecture (SPA) [46] is based on the NEF and describes a series of modules and methods for constructing large-scale cognitive models with spiking neurons. It proposes methods in which populations of neurons can represent what is referred to as a 'semantic pointer', a type of representation that is geared to supporting complex cognitive function. The name 'semantic pointer' stems from the fact that these representations contain semantic information, but can also be used as a 'pointer' to additional information not embedded in the representation itself. Semantic pointers are vectors within a high-dimensional vector space and are generated by using a compression operator. Similarity between representations can be measured by the angle between the vectors. Individual semantic pointers can be 'dereferenced' (or 'decompressed') to reconstruct compressed information such as a high level sequence of actions, a low level motor plan, or a visual image.

The SPA includes a particular Vector Symbolic Architecture (VSA), which allows symbol-like processing to be implemented on a set of vectors. All VSAs must define three operations that can be applied to a pair of vectors. One operation is a similarity function, which gives a measure of how similar two vectors are, and is often interpreted as how semantically similar they are. The second is a superposition function, which combines two vectors into a single new vector representing both original vectors simultaneously. Lastly, there is a binding function, which combines two vectors into a single new vector with a distinct semantic meaning.

The SPA can use any VSA with superposition and binding operations that preserve the dimensionality. However, the most commonly used VSA algebra for the SPA is based on Holographic Reduced Representation (HRR) [140]. Similarity between two vectors is

measured by their cosine distance (angle between them, smaller angle being more similar). Superposition is performed by simply summing the vectors together. The binding method is circular convolution. The main differences between the SPA implementation and standard HRRs are: 1) All elements are implemented in (typically spiking) neurons, 2) Normalization is often approximated or implemented with neural saturation in the SPA (as opposed to ideal normalization in HRRs), 3) Clean up is computed through a matching and winner-take-all or other neural mechanism (rather than via an ideal max function). In short, the SPA tends to use a particular approximation to HRR operations. Nevertheless, algebraic characterizations of manipulations involving circular convolution like those used in HRRs are useful for understanding the computations.

An alternative algebra that has been used with the SPA is Vector-Derived Transformation Binding (VTB) [64]. This method has shown improvement over circular convolution when representing deep structures with spiking neurons but has the disadvantage that the binding operation is non-commutative.

Circular Convolution

Circular convolution is an operation that can be applied to two vectors in order to bind their contents together. The circled asterisk symbol ($\circledast$) is commonly used to denote circular convolution and this will be the symbol used in this thesis as well. The discrete circular convolution for two d-dimensional vectors is described by Equation 2.12.

$$(v \circledast w)_i = \sum_{j=0}^{d-1} v_j w_{(i-j) \mod d} \tag{2.12}$$

Each element in the output vector is the dot product of one input vector with a shifted and reversed version of the other input vector. This operation can also be viewed as a multiplication between a vector and a matrix, where the vector is one input, and the matrix is a circulant matrix constructed from the second input. This interpretation is depicted in Equation 2.13.

$$(v \circledast w) = v \cdot \begin{bmatrix} w_0 & w_{d-1} & w_{d-2} & \dots & w_2 & w_1 \\ w_1 & w_0 & w_{d-1} & \dots & w_3 & w_2 \\ w_2 & w_1 & w_0 & \dots & w_4 & w_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ w_{d-2} & w_{d-3} & w_{d-4} & \dots & w_0 & w_{d-1} \\ w_{d-1} & w_{d-2} & w_{d-3} & \dots & w_1 & w_0 \end{bmatrix} \tag{2.13}$$

When two vectors are circularly convolved together, the resulting vector is highly dissimilar to both input vectors (in terms of normalized cosine similarity).

The operation is also approximately reversible. When convolving two input vectors to produce an output, the output can be convolved with an inverse of either of the inputs to produce the other input. The approximate inverse of a vector for use in circular convolution is defined as:

$$v^{-1} := (v_0, v_{d-1}, v_{d-2}, ..., v_1)^T \tag{2.14}$$

This inverse is cheap to compute as it is a simple permutation of the elements. The first element is kept the same and the order of all subsequent elements is reversed. The inverse of the vector is also equivalent to the first row of the circulant matrix that would be constructed to perform convolution with the vector.

In the special case where the vector is 'unitary' the approximate inverse is equal to the true inverse. A unitary vector is defined as a vector that preserves the dot product under binding. A vector x is unitary if it satisfies:

$$(w \circledast x) \cdot (v \circledast x) = w \cdot v \tag{2.15}$$

Circular convolution can be implemented extremely efficiently in a neural network [63]. Convolution in the time domain is multiplication in the frequency domain. In this manner, circular convolution between two vectors can be computed by taking the Discrete Fourier Transform (DFT) of each vector, multiplying the coefficients together, and computing the inverse DFT to obtain the output. Both the DFT and inverse DFT are linear operations and can be implemented via the connection weights between ensembles of neurons. The multiplication of the Fourier coefficients of the two inputs is a non-linear operation but can still be implemented efficiently using the NEF [61].

Constructing Neural Cognitive Models

How concepts are represented in memory plays an important role for the types of cognitive computations that can be performed. For example, if you were to play with some toy blocks, one of which is a green circle, the other a blue square, how would you encode these objects in memory? Simply encoding the individual concepts; 'green', 'circle', 'blue', and 'square' you would have lost information as to how these concepts were organized. Was the circle green, was it blue, or was it a mix of the two? From simply summing the vectors corresponding to each of these concepts together, there is no way to know. This is often referred to as the 'binding problem' [176].

VSAs offer a solution to this problem by constructing higher level concepts by binding properties together. For instance, the example below depicts a memory containing a green circle and a blue square using circular convolution:

$$M = \text{GREEN} \circledast \text{CIRCLE} + \text{BLUE} \circledast \text{SQUARE}. \tag{2.16}$$

Each symbol is chosen to be a random unit vector in a high dimensional space (e.g., 512 dimensions). M corresponds to the memory, which will have a high dot product with the vectors $\text{GREEN} \circledast \text{CIRCLE}$ and $\text{BLUE} \circledast \text{SQUARE}$, indicating that both of these concepts are stored in this memory. A query can be computed on the memory by binding with the inverse of one of the properties or objects. For example, binding with the inverse of the vector for circle essentially asks the question 'what colour is the circle?':

$$\begin{aligned} M \circledast \text{CIRCLE}^{-1} &\approx \text{GREEN} + \text{BLUE} \circledast \text{SQUARE} \circledast \text{CIRCLE}^{-1} \\ M \circledast \text{CIRCLE}^{-1} &\approx \text{GREEN} + \text{noise} \end{aligned} \tag{2.17}$$

The vector $\text{BLUE} \circledast \text{SQUARE} \circledast \text{CIRCLE}^{-1}$ does not correspond to anything meaningful (i.e., anything assumed to be stored in a clean up memory) and can thus be treated as noise. Despite the noise, the output will be highly similar to GREEN in terms of dot product. The output will also be dissimilar to all of the other objects and properties (due to these vectors existing in a high dimensional space and having been randomly chosen).

In addition to a neural implementation of a VSA, the SPA contains many components useful for constructing cognitive models. A stored representation will degrade after repeated operations as the noise accumulates. A common way to prevent this is to use an autoassociative memory (often referred to as a cleanup memory) to transform a noisy representation into a clean one [165]. Instead of just recovering the input, heteroassociative memories can also be learned to map from one semantic pointer to another or to a different representation [180]. Networks modelled from the circuitry of the basal ganglia can be used to route information between different cortical areas and perform action selection [164].

From these basic building blocks more complicated models can be constructed. Examples include neural models of mental imagery [145], memory [26, 177, 82], language [34, 17], and emotion [83]. One of the most prominent neural models using the SPA demonstrating the scaling capability of this approach is the Semantic Pointer Architecture Unified Network (SPAUN) [48]. This brain model contains 2.5 million neurons and is capable of performing 8 diverse tasks with instructions given visually and output performed with a simulated arm. Recently this model has been expanded to 6 million neurons, with a more sophisticated vision and motor system, and the ability to perform simple instruction following [28].

2.3 Machine Learning

In contrast to methods focused on building models of human and animal cognition, as explored in the last two sections, many methods in machine learning are developed to perform specific functions efficiently. In short, researchers in machine learning are less concerned with how biological systems work, and more concerned with solving particular computational problems. More specifically, machine learning is a subfield of artificial intelligence that focuses on constructing computational models that improve as they are given more data. Often these models contain a series of parameters that are updated each time a new data point is received. The current field of machine learning can be broadly divided into three categories: supervised learning, unsupervised learning, and reinforcement learning.

At a high level, supervised learning is a method for approximating an unknown function given the outputs of that function. This function is typically one that would solve some task, such as giving a label to an image, steering a car to avoid a pedestrian, or choosing a song that would make a listener happy. The definition is extremely broad and essentially includes anything that takes an input and produces an output. The 'true function' can be difficult to define, but there must be ways of generating individual data points that are consistent with this function (e.g., images and desired labels). A model can be defined with a set of parameters such that changing the parameters will change the output of the model to a given input. The goal is to select the parameters of the model such that it best approximates the true underlying function. Supervised learning deals with systematic ways of changing these parameters (learning) to improve the approximation given enough pairs of inputs and corresponding outputs. An ideal model will not only make accurate predictions on data it was trained on, but also generalize to new data it has not seen before.

Supervised learning can be further divided into classification and regression. Classification is the process of categorizing an input into one of several possible classes (e.g., based on sensor measurements, will it be sunny, rainy, or cloudy tomorrow?). Regression involves predicting a continuous value from an input (e.g., what will the temperature be, how many millimeters of rain?).

Unsupervised learning is the process of learning a useful function from data that has no explicit labels. One such example is clustering, where a series of inputs are grouped based on similar features. Insight can be gained about the data by observing the clusters that are found (e.g., clustering of songs may result in new sub-genres being discovered). Another instance of unsupervised learning is training autoencoders [94]. Here the goal is to simply reconstruct the input from a modified version of the input, such as a noisy input [178] or lower dimensional embedding [94]. The result is a function that can clean up noise or compute a compression and corresponding decompression.

Reinforcement learning deals with learning a behaviour to maximize rewards over time. At each time step the agent observes information from the world and must choose an action to perform. After performing the action the agent may or may not receive a scalar reward. This reward can be positive, or negative. The agent will continue to observe, produce actions, and receive rewards either indefinitely (continual task) or until a terminal state is reached (episodic task). The agent does not inherently know whether the reward it just received is due to the last action it took, or due to a series of state transitions it made in the past. In this manner it is different from supervised learning since the correct output is not known, but it is not quite unsupervised since a reward signal is given to guide behaviour.

Typically the reinforcement learning problem is divided into two related sub-problems; learning a policy to map from states to actions, and learning a value function to assign a value to a given state under a particular policy.

2.3.1 Artificial Neural Networks

Artificial Neural Networks (ANNs) have been proven to be universal function approximators, meaning that ANNs can compute any computable function [36, 107]. With enough neurons, appropriate connection weights between each neuron, and a nonpolynomial activation function, any continuous function can be approximated [103]. While this is good news for ANN researchers, it does not solve the problem of learning the network parameters. Furthermore, real-world applications always have restrictions on computational and memory resources, so efficient computation of particular functions is often key. Unsurprisingly, then, some functions are easier to approximate than others given the current techniques known to the machine learning community.

The most common way of parameterizing a feed-forward neural network is to select a number of hidden layers (depth), the number of neurons within each hidden layer (width), and a differentiable non-linear activation function to use between the layers. The trainable parameters of the network become the weights between neurons in successive layers (including the input and output) and biases added to each neuron in a hidden layer. Techniques have also been used to learn the number of neurons in each layer [5], parameters of the activation function [3], and the topology of the network itself [162].

Training

The parameters (weights and biases) of the network are updated systematically based on the difference between the network's response and the true desired response. Backpropa-

gation [152] is a successful method for computing the gradients of each parameter based on a chosen loss function. Knowing the gradient means knowing how a small change in parameter values will affect the loss. It is important for every operation in the network to be differentiable so that the gradient computation can be propagated from the loss on the output all of the way back to the input weights.

There are many different optimization techniques to update the parameters based on the gradient. While the gradient gives a direction in parameter space to move locally, it does not give information on how far to move in that direction before the slope of the space changes enough that this direction no longer results in improved performance. Some optimizers make use of higher order derivative terms to get a better estimate of the curvature of the space and are able to make more precice updates, but often at a much higher computational cost [11]. Typically most optimizers are run iteratively with a learning rate parameter determining how far of a step to take in the direction of the gradient each iteration. Updates are often performed on smaller batches of data rather than waiting to evaluate the whole dataset. This can help speed up the learning process by providing more frequent updates. Often there are many similarities between datapoints, so including them all on each update can result in redundant information. Commonly used optimization algorithms include stochastic gradient descent [147], RMSProp [175], and Adam [86].

Regularization

Even though neural networks are universal function approximators, that does not guarantee that they will approximate the function you want them to learn well, with finite resources. Training is conducted on a finite dataset, and there can be infinitely many functions that would perfectly match the data. The goal is to create a network capable of generalizing to new data that has not been seen yet, so that the machine may provide an accurate prediction without human assistance. One way to estimate how well a network will generalize is to run it on a set of data with known labels that has not been used in training. If the performance on the training set is good but the performance on the test set is poor, this indicates that the model is overfit to the training set. Essentially the parameters of the model are tuned in such a way that they memorize just the data samples seen during training at the cost of ability to generalize.

One way to combat overfitting is to incorporate regularization. This can take many forms, one of the simplest is to incorporate an early stopping mechanism in the training. An additional set of data called a validation set is held out of the training set. Periodically during training the model is evaluated on the validation set. At some point the performance

on the validation set will begin to plateau or become worse, while the performance on the training set continues to improve. This indicates that further training from this point is only overfitting to the training set and that training should stop.

Another form of regularization is to reduce the size of the network. With less parameters it becomes more difficult to fit the data without generalization capability, as the function learned will be more smooth and many desired functions for real world applications tend to be smooth. A penalty on the magnitude of the weights can also help match this smoothness assumption [192]. Training with dropout [75] is another form of regularization that involves setting the activations of a random subset of neurons to zero during training. Each iteration a different subset is affected, forcing the network to learn a more redundant representation in order to achieve good performance. This has the effect of preventing specific neurons or groups of neurons from focusing on memorizing particular data points, allowing a more general function to be learned.

2.3.2 Representation

An important branch of machine learning is representation learning (also called feature learning). Both supervised and unsupervised methods can be used to learn a representation. The activations within the hidden layers of a neural network can be considered to be a representation of the input. Some representations are more useful than others when computing particular functions. For a classification task, if representations of the same class are more similar to each other than they are to members of another class it becomes easier to learn a classification boundary.

To illustrate the importance of representation, consider the following scenario. You have a partial chess board with four pieces, two white knights and two black knights. The goal is to swap the positions of the black and white pieces using the least number of moves possible. No two pieces are allowed to occupy the same square and all moves must be legal chess moves for a knight. The starting positions and available squares are depicted in Figure 2.5.

From observing the problem as is, it can be difficult to form a solution. For the average person it may take many steps of trial and error to solve the problem, and once solved there is no easy way of knowing whether the solution uses the least number of moves.

A simple change in representation can make this problem much easier to solve. Instead of a standard arrangement of the squares in a chessboard, a graph can be constructed based on which squares can be accessed from one another via a single knight move. This graphical representation is depicted in Figure 2.6. With this new way of framing the

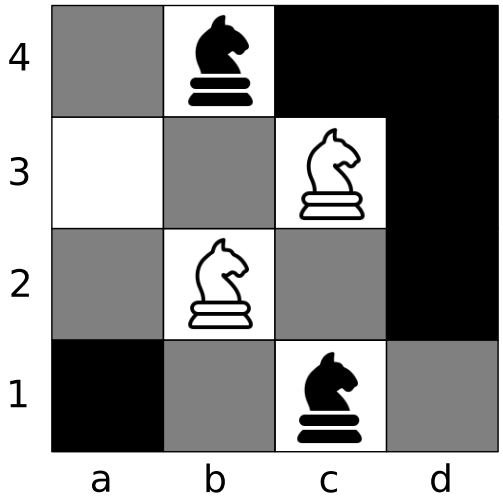


Figure 2.5: **Knight Movement Problem**. The goal is to swap the positions of the black and white knights in the minimum number of moves. Solid black squares are considered inaccessible.

problem, possible solutions become immediately apparent. One optimal solution is as follows:

Move the white knight from c3 to b1 to make space for the black knights to get through. Move the black knight along the path from c1 to d1, move the white knight from b2 to the black knight original position of c1. Now the black knight on d1 can move into b2. The black knight on b4 can move clockwise to its final position on c3, and the white knight on b1 can continue clockwise to its final position on b4. This solution is visualized in Figure 2.7.

While no new information has been added to the problem, the system that is trying to solve it (the human) is able to do so much faster. In fact, there can even be a loss of information when transforming to this graph representation (without labels on the squares, there are multiple chess board configurations that would produce the same graph structure).

In a machine learning context, an analogous situation often arises. An algorithm may have trouble solving a particular task, but with a change in representation the same algorithm may succeed where it had previously struggled [67, 133, 141, 14]. Many techniques can be used to generate representations with different properties.

The structure of the network can guide the types of representations learned. A network with a smaller layer following a larger layer (bottleneck) will enforce a lower dimensional

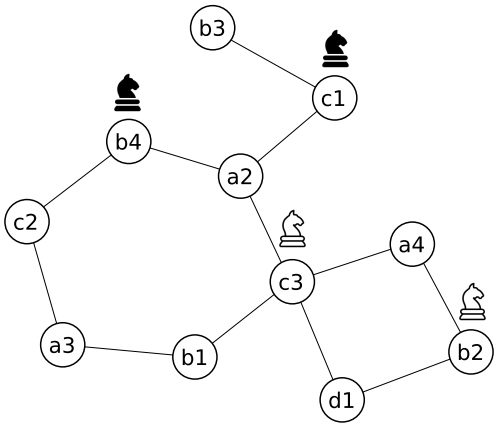


Figure 2.6: **Knight Movement Problem Graph Representation**. An edge between two nodes indicates that a transition is possible with one move.

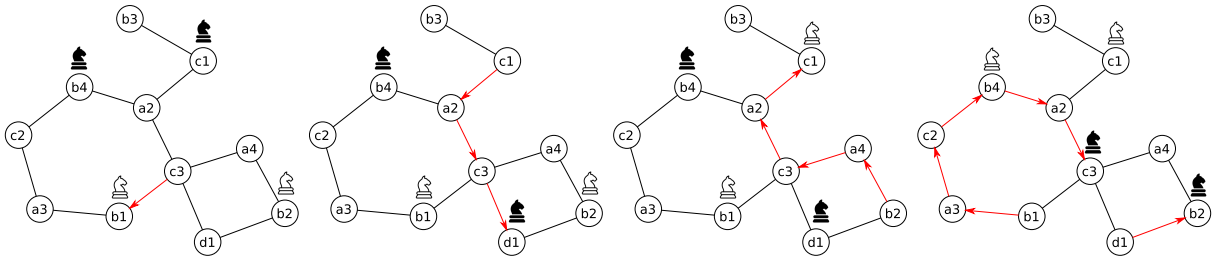


Figure 2.7: **Knight Movement Solution**. One possible set of moves to solve the problem in the minimum number of moves (14). Easier to visualize and verify correctness of a solution in the graph representation.

representation. A convolutional neural network [95] encourages spatial invariance through weight sharing, which affects the types of features learned.

Regularization techniques can be used to control the type of representation learned. A penalty based on the L_1 norm of the weights encourages sparse representations to be learned [174]. Autoencoders can be trained to perform a nonlinear Principal Components Analysis (PCA) on the input data [94]. By training each node of the hidden layer successively on the residuals (error) of the reconstruction from the previous nodes, the final encoding will be encouraged to learn distinct factors to represent with each node. Variational Autoencoders are trained to represent inputs as a distribution over a latent space [87]. By applying regularization to encourage the learned distribution to be standard normal during training, the final space can easily be sampled to generate new data points. Generative Adversarial Networks are another technique for learning a normal distribution to generate realistic data [60].

When working with written text, it is often useful to learn word embedding representations. These are representations of words as high dimensional vectors, allowing relationships between words to be quantified by operations on these vectors. Methods for learning these representations from a text corpus include GloVe [137] and Word2Vec [115]. Similarity between words can be estimated by their distance, and relationships between pairs of words can be estimated by their vector difference if embedding representations are learned appropriately [115]. Another way to develop word embeddings is to measure human behaviour during a free association task, giving an estimate of how concepts are related in the brain. Results from extensive studies on humans have been compiled into the association matrices of USF Norms [124].

In the spatial domain, SLAM algorithms can be used to generate a map of the environment which can then be used by other subsystems to choose actions to execute. Often this map is an occupancy grid labelling each point as an obstacle or free space with a particular probability, but could also be a higher level abstraction such as a graph with free space regions as nodes and edges as pathways between them. Many machine learning techniques have been applied to SLAM, including learning spatial structure through sensor prediction [100], unsupervised loop detection [54], and learning to plan on generated maps [194].

For a spatial representation it is useful to be able to preserve metric information such as distance and direction between represented quantities. A sparse representation of locations can help a network learn location dependent functions but at the same time a smooth representation encourages generalization to nearby locations. There are often trade-offs between different representations and a good representation is one which can balance those trade-offs for a desired task.

Chapter 3

Methods

This chapter describes the various methods of representing spatial information that will be used throughout this thesis. The following section is based on the work from [92] where Spatial Semantic Pointers (SSPs) are first introduced.

3.1 Spatial Semantic Pointers

3.1.1 Motivation

There is evidence for a wide variety of types of mental representation. Some mental representations are well-described using *discrete* structures (e.g., graphs, trees, lists, and so on). Others are well-described using *continuous* structures (e.g., images, maps, surfaces, and so on). Here we propose a kind of mental representation of continuous structures that is amenable to neural implementation.

Recently, there have been several proposals for how neural networks can represent discrete structures. One family of approaches, called Vector Symbolic Architectures (VSAs), defines algebras over high-dimensional vector spaces, and uses those algebras to encode such structures. VSAs have been used to characterize a variety of cognitive behaviours, including analogical reasoning [139], language processing [81], and concept encoding [35]. Most VSAs are defined over continuous vector spaces, including Multiply Add Permute [MAP; 55], Holographic Reduced Representations [HRR; 140], and Vector-derived Transformation Binding [VTB; 62]. When VSAs are used to model cognitive behaviours, they essentially

define methods for characterizing continuous vectors as both continuous slots and continuous fillers and define a method of binding fillers to slots.

In this work, we will use the Semantic Pointer Architecture [SPA; 46], which proposes a means of neurally implementing VSAs for explaining cognitive behaviour in biologically plausible spiking networks. This architecture uses aspects of VSAs for cognitive representation, but the SPA also addresses visual processing, motor control, memory, decision making, and cognitive control in ways that do not use VSAs. However, all of these elements of the SPA use representations called semantic pointers (SPs), which result from compressing and decompressing information in cortex. As a result, we can think of VSA algebras as proposing a family of operators that are well-suited for certain cognitive tasks.

However, as with most uses of VSAs, in past work the SPA addresses cognitive tasks with a focus on representations of discrete structures (i.e., discrete slots in a represented structure). Here we propose a method for encoding cognitive structures over continuous spaces. We call this kind of representation "spatial semantic pointers" (SSPs). In this section we propose and examine in some detail a specific kind of SSP implemented using a particular "fractional binding" operator to encode real-valued quantities – although a variety of other operators can be analogously defined.

The following sections of this thesis provide a mathematical definition of SSPs, and show how SSPs can provide a natural means of generating and manipulating representations that are useful for spatial cognition. We identify desiderata for spatial representation that are useful for cognitive explanations. We then implement this representation both mathematically and neurally, and perform simulation experiments to demonstrate that it has a variety of useful properties, including: being able to query a memory for its spatial or non-spatial contents, representing multiple objects and locations simultaneously, spatially transforming memory contents without decoding them, and representing regions of space of various shapes and sizes. The choice of VSA and binding operator used in this work allows the representation and various transformations to be implemented efficiently by a spiking neural network.

3.1.2 Representation

Our proposed representation generalizes the notion of vector binding to continuous spaces. By analogy to fractional powers defining the multiplication of reals, we define fractional bindings for vectors in a vector space. To explain, let us first consider binding a vector to itself a discrete number of times. That is, let $k \in \mathbb{N}$ be a natural number, $B \in \mathbb{R}^d$ be a fixed d-dimensional vector (i.e., semantic pointer), and $\circledast$ be a binding operator. We can

repeatedly bind B with itself $k-1$ times[1] as follows:

$$B^k = \underbrace{B \circledast B \circledast \ldots \circledast B}_{B \text{ appears } k \text{ times}}. \tag{3.1}$$

This representation has been used in several cognitive models, for instance, to encode the position (k) in a list in serial working memory [26]. We propose to generalize this to continuous quantities (as opposed to discrete lists, for example) by permitting k to be real. Allowing a real k means that the resulting vector B^k encodes a continuous quantity. Most VSA operators can be interpreted in this manner (including MAP [55], VTB [62], and HRR [140]), but not all (e.g., spatter codes [84]).

In the specific case of the SPA, we take the binding operator to be circular convolution (as proposed by Plate) and the fixed d-dimensional vectors to be semantic pointers chosen from the unit sphere. We then define our fractional binding operation by expressing equation 3.1 in the complex domain:

$$B^k = \mathcal{F}^{-1}\left\{\mathcal{F}\{B\}^k\right\}, \quad k \in \mathbb{R}, \tag{3.2}$$

where $\mathcal{F}\{\cdot\}$ is the Fourier transform, and $\mathcal{F}\{B\}^k$ is an element-wise exponentiation of a complex vector—analogous to exponentiation using fractional powers (e.g., $b^{2.5}$)—permitting k to be real.[2] In the present thesis, we use unitary vectors for B due to the fact that their length does not change with multiple bindings, and their inverse is equal to their approximate inverse (see below).

This definition comes with many useful algebraic properties analogous to the relationship between multiplication and exponentiation (e.g., $b^{2.5}b^{1.5} = b^4$), in particular:

$$B^{k_1} \circledast B^{k_2} = B^{k_1+k_2}, \quad k_1, k_2 \in \mathbb{R}. \tag{3.3}$$

In essence, fractional binding is to circular convolution as exponentiation is to multiplication. We exploit equation 3.3 to perform semantically meaningful operations (e.g., shifting space) in our experiments.

Next, we extend this representation to multiple dimensions, which is the focus of our experiments in this work. In general, we can represent points in $\mathbb{R}^n$ by repeating equation 3.2, n times, using a different base semantic pointer for each represented dimension (i.e., for each axis), and then binding all of the resulting vectors together. For $n = 2$, we

[1] When $k = 0$ we get the identity vector corresponding to $\circledast$.

[2] For natural k, equations 3.1 and 3.2 are mathematically equivalent.

think of the representation as encoding a continuous 2-D spatial representation (e.g., the location of objects on a map). In this case, the SSP that represents the point (x, y) is defined as the vector resulting from the function:

$$S(x, y) = X^x \circledast Y^y, \tag{3.4}$$

where X and Y are fixed semantic pointers, x and y are reals, and we are using fractional binding as defined by equation 3.2.

Similarly, the SSP that represents a continuous region (e.g., a solid rectangle), specified by some infinite set of 2-D points R, is defined as:

$$S(R) = \int_{(x,y)\in R} X^x \circledast Y^y \, dx \, dy. \tag{3.5}$$

There are efficient ways to compute equations 3.4 and 3.5 with spiking neurons using the Neural Engineering Framework [NEF; 47]. We use a publicly-available implementation in several of our results in the next chapter.

A useful way of visualizing an SSP is to look at its similarity to other SSPs. Similarity is defined as the cosine distance between two vectors. For vectors of unit length, this is equivalent to the dot product. In the 1-D case, the dot product can be computed with the axis vector raised to a set of exponents covering the range of interest. This visualization is shown for two SSPs in Figure 3.1 (Left). Similarly, for a SSP representing a 2-D quantity, the dot product can be computed with vectors representing positions spaced by Δx and Δy to tile the space and produce a heatmap as shown in Figure 3.1 (Right). The region SSP can be visualized with a similarity heatmap as well, as shown in Figure 3.2.

To represent a single object occupying some location or region, we bind its semantic pointer representation, OBJ, with the SSP from equation 3.4 or 3.5, respectively:

$$M = OBJ \circledast S. \tag{3.6}$$

In general, to represent a set of m labelled objects together in the same memory, we can use superposition:

$$M = \sum_{i=1}^{m} OBJ_i \circledast S_i, \tag{3.7}$$

with a distinct semantic pointer OBJ_i tagging each object.

Given a representation like that in equation 3.7, we can query it in a number of ways. For example, to determine what object is at location (x, y) we can compute:

$$M \circledast (X^x \circledast Y^y)^{-1} = M \circledast X^{-x} \circledast Y^{-y}. \tag{3.8}$$

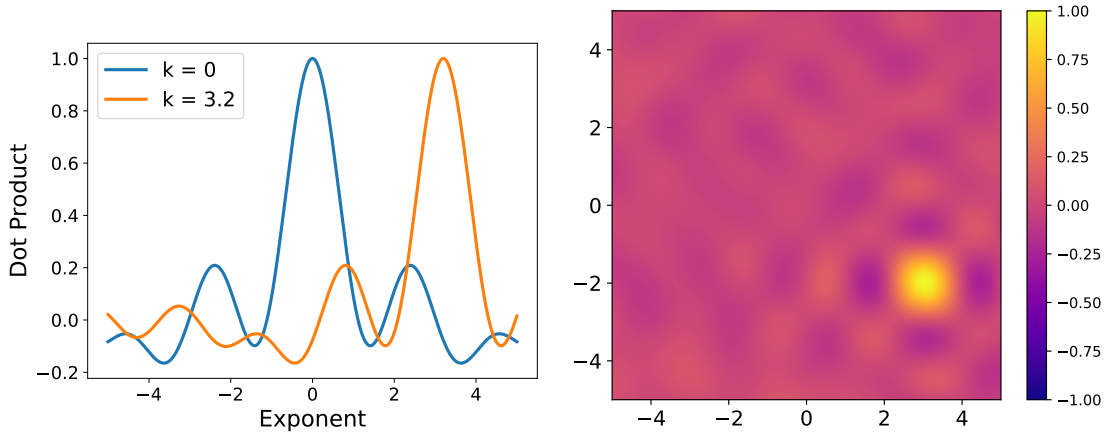


Figure 3.1: **Spatial Semantic Pointer Similarity**. **Left**: Visualization of 1-D SSPs. k is the value of the exponent of the vector of interest, the x-axis on the plot shows the value of the exponent for the vector that the dot product is performed with. **Right**: Visualization of a 2-D SSP representing the location (3, -2). The two axes on the plot depict the values of the two exponents of the vector the dot product is performed with. The colour corresponds to the value of the dot product.

By the properties of binding and superposition, the resulting vector will have highest cosine similarity (i.e., dot product) with the object at (x, y).[3] Note that the inverse used in equation 3.8 is approximate, but choosing X and Y to be unitary vectors guarantees it is equal to the true inverse.

We can construct a heatmap of representations defined by equation 3.7, to visualize a decoding of the objects back into the original continuous space. For instance, for $m = 2$ (i.e., two represented objects), taking the dot product of $M \circledast OBJ_i^{-1}$ (M is from equation 3.7) with vectors representing positions spaced by Δx and Δy to tile the 2-D space, provides the visualization of Figure 3.3.

In summary, fractional binding provides a scheme for encoding a set of n-dimensional points into a d-dimensional SSP. This comes with an algebra for operating on these SSPs in meaningful ways (e.g., querying, shifting, and so on). When combined with the methods of the SPA, we can spatially manipulate collections of objects in a spiking neural implementation, as detailed in the experiments in the next chapter.

[3]This assumes d is sufficiently large, relative to m, as is typical for VSAs.

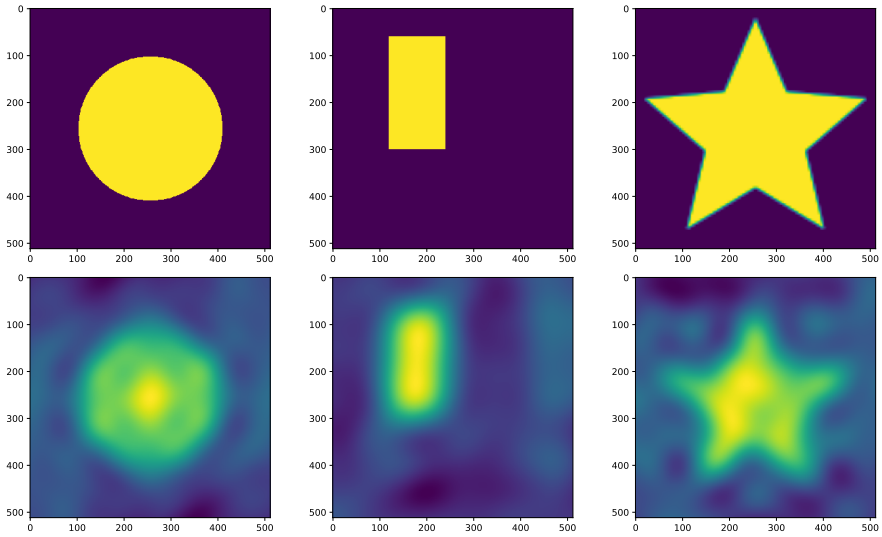


Figure 3.2: **Region SSP Visualization**. **Top Row**: Shape of the region to integrate over. The space is discretized into 512x512 pixels for an approximate computation of the integral in Equation 3.5. **Bottom Row**: Heatmaps for the corresponding region SSPs. Each SSP is a 512 dimensional vector.

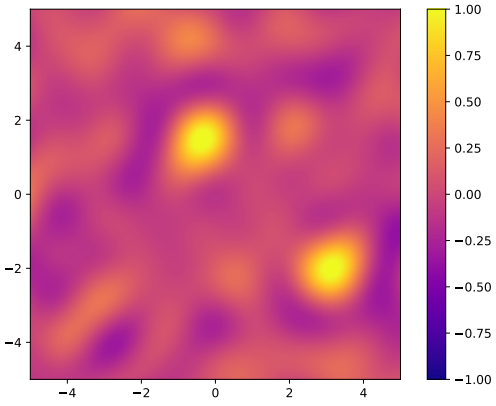


Figure 3.3: **Heatmap Visualizing Two Objects**. Shown is a heatmap of the representation of two objects at different locations, as specified by equation 3.7. This plot is the sum of the decoding for each object.

3.1.3 Example queries

To illustrate the application of representing objects at continuous spatial locations using SSPs we demonstrate a variety of example queries in Figure 3.4. A set of animals (objects) at random spatial locations are encoded into an SSP using equation 3.7 as shown in Figure 3.4a. This is accomplished by binding the SP representing each object with the SSP corresponding to its location, summing these values together, and then normalizing the result.

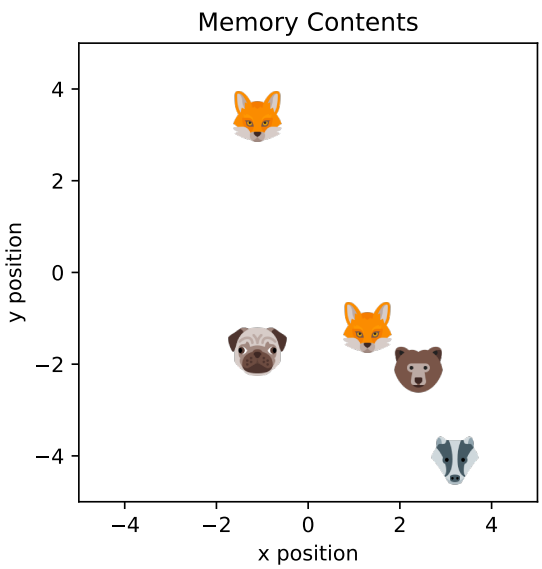
Various queries can be made with this representation. Figure 3.4c (top) shows the results of asking for the locations of different objects, decoded as a heatmap. If the object exists at more than one location, the resulting SSP will be highly similar to all of these locations (image on the far left). If the object does not exist at any location, the resulting SSP will not be similar to any location on the heatmap (image on the far right). Figure 3.4c (bottom) shows the reverse is possible too: given a location, find out which objects are at that location. If there are no objects at the queried location, the result can be treated as noise and will not be similar to any object in the vocabulary (as shown in the far right).

Location queries can also be extended to regions of space, as shown in Figure 3.4b. If the region encompasses multiple objects, all objects should be returned, as depicted by the bar charts at the bottom. The region semantic pointers themselves are a single vector that is formed by integrating over the spatial semantic pointers within the region and normalizing the result, as described by equation 3.5. This process creates a vector that has a high dot product (i.e., similarity) with all vectors within a particular area while having a low dot product with vectors outside the region. Two example represented regions are illustrated in the heatmaps at the top of the figure. It is important to note that due to the normalization, the dot product with the region vector and a single point within the region will decrease as the area of the region increases. The consequence of this fact is that the optimal threshold for detection is a function of the area.

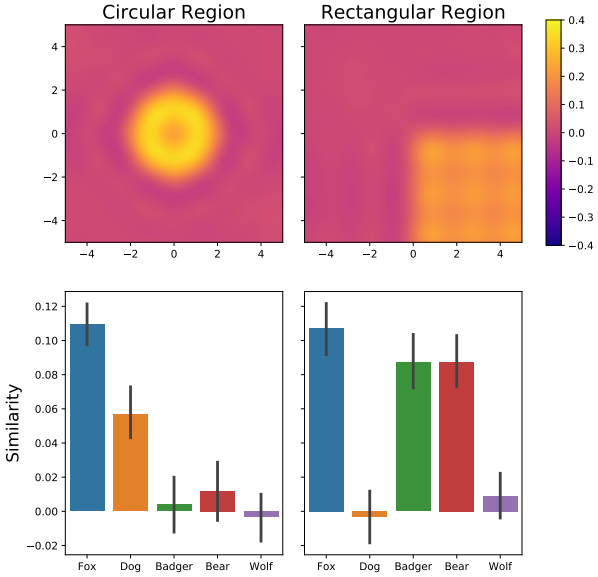
3.2 Shape of the Unitary Vector Space

The unitary vectors live in a very interesting subspace of all possible vectors. This section will explore the shape of this space and illustrate how it can very naturally be mapped to physical space in the real world.

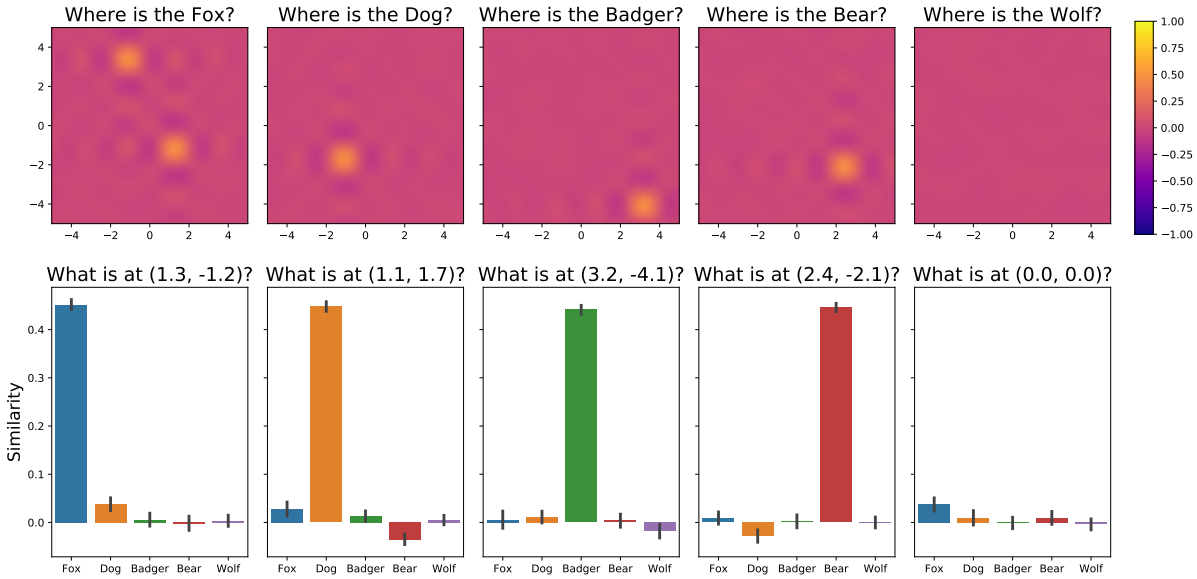
A unitary vector is any vector where its corresponding circulant matrix is an orthonormal matrix (all rows are orthogonal vectors with a magnitude of 1). This ensures that



(a) Memory contents



(b) Region queries



(c) Location and item queries

Figure 3.4: **Example Queries of Items, Locations, and Regions**. **a)** An example memory encoded into an SSP. **b)** Region queries applied to the memory in (a). **c)** Object (top) and location (bottom) queries applied to the memory in (a).

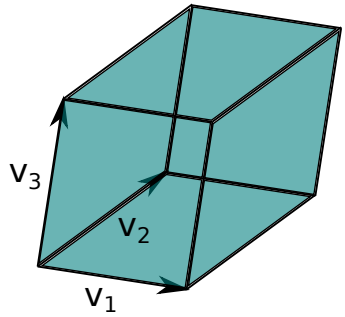


Figure 3.5: **Geometric Interpretation of Determinant**. The magnitude of the determinant of a matrix can be expressed as the volume of the parallelotope formed by the vectors from the rows of that matrix.

multiplication by this matrix is a pure rotation and does not involve any scaling. The determinant of this matrix will also have a magnitude of 1. If the determinant does not have a magnitude of 1, successive multiplications will either shrink the magnitude of the resulting vector to 0 or have it grow to infinity. The determinant of a matrix can be expressed geometrically as the volume of the parallelotope (n dimensional generalization of the parallelogram)[33] formed by the vectors from the rows of that matrix. This relation is visualized in Figure 3.5.

Since unitary vectors have a magnitude of 1, for the determinant to have a magnitude of 1, every row in the circulant matrix must be orthogonal. This can be observed from the fact that the area of a parallelogram is the length of the base multiplied by the height and if the base and the connected side do not form a 90 degree angle, the height will be less than the length of the side. Consequently, the parallelotope for a set of unitary vectors in a circulant matrix is always a unit hypercube.

The constraint of a unitary vector forming an orthonormal circulant matrix can be used to define the subspace where these vectors reside. Let $v = [v_1, v_2...v_d]$ be a unitary vector in d dimensions. For the circulant matrix to be orthonormal, the following relations must be true:

1. The dot product between any two vectors in the matrix must be zero.
2. The magnitude of each vector must be one.

The condition that the dot product of any two unique vectors is zero can be expressed by:

$$\begin{bmatrix} v_{1+i} & v_{2+i} & \dots & v_{d+i} \end{bmatrix} \cdot \begin{bmatrix} v_{1+j} & v_{2+j} & \dots & v_{d+j} \end{bmatrix} = 0 \tag{3.9}$$

This relation holds for all integer values of i and j between 0 and $d-1$ and $i \neq j$ (also defining v_{d+n} as equivalent to v_n).

The condition that the magnitude of each vector must be one (i.e., the sum of the squares of all elements is 1), is expressed by:

$$\sum_{i=1}^{d} v_i^2 = 1 \tag{3.10}$$

Summing the dot products of just the first vector and each other vector gives the following relation:

$$\sum_{\substack{i=1 \\ i \neq j}}^{d} \sum_{j=1}^{d} v_i v_j = 0 \tag{3.11}$$

Adding Equation 3.10 and Equation 3.11 reveals an interesting relationship between each element of a unitary vector.

$$\begin{aligned} \sum_{i=1}^{d} v_i^2 + \sum_{\substack{i=1 \\ i \neq j}}^{d} \sum_{j=1}^{d} v_i v_j &= 1 \\ \sum_{i=1}^{d} \sum_{j=1}^{d} v_i v_j &= 1 \\ \sum_{i=1}^{d} v_i \sum_{j=1}^{d} v_j &= 1 \\ \left(\sum_{i=1}^{d} v_i \right)^2 &= 1 \\ \sum_{i=1}^{d} v_i &= \pm\sqrt{1} \end{aligned} \tag{3.12}$$

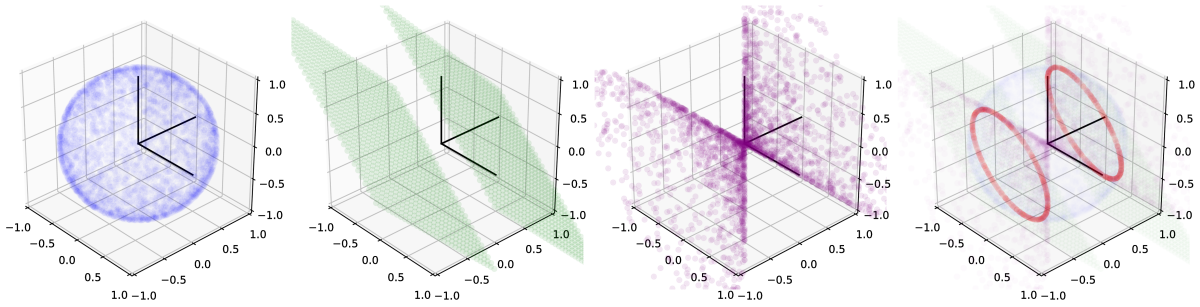


Figure 3.6: **Unitary Vectors in 3D**. The constraint $x^2 + y^2 + z^2 = 1$ is shown in blue. The constraint $x + y + z = \pm 1$ is shown in green. The constraint $xy + yz + zx = 0$ is shown in purple. The space where all of these constraints intersect is shown in red. This intersection contains all possible unitary vectors in three dimensions.

The sum of the individual elements in a unitary vector must equal to $+1$ or -1. This constraint corresponds to these unitary vectors lying on one of two hyperplanes, each with a normal vector $[1, 1...1]$ and offset from the origin by $\pm[1/d, 1/d...1/d]$. Since the magnitude of each unitary vector is 1, they are also constrained to lie on a unit hypersphere. The intersection of a hyperplane and a hypersphere is a hypersphere of one lower dimension. This is easiest to visualize in three dimensions, where the intersection of a plane and a sphere will form a circle. Figure 3.6 shows the space of all possible unitary vectors in 3D.

In general, orthogonality of the circulant matrix imposes $\lfloor d/2 \rfloor$ constraints on the space of unitary vectors (half of the dimensionality, rounded down). This corresponds to the number of unique equations formed by the dot product between the first row and all other rows being zero (Equation 3.9). There are half as many equations as rows due to the symmetry of the circulant matrix. Each equation forms a hypersurface, all of which intersect the unit hypersphere. Note that Equation 3.12 is a linear combination of the other constraint equations, so does not impose an additional constraint itself. Every unique intersection of hypersurfaces forms a hypersurface of one less dimension. For a space of dimensionality d, this means the dimensionality of the surface remaining after taking into account all unique constraints is:

$$d_{surface} = d - 1 - \lfloor d/2 \rfloor = \lfloor (d-1)/2 \rfloor \tag{3.13}$$

This same conclusion on the dimensionality of the unitary vector surface can be reached from looking at the degrees of freedom in the Fourier coefficients. Each coefficient is a complex number with a magnitude and phase. The magnitudes are all restricted to be 1,

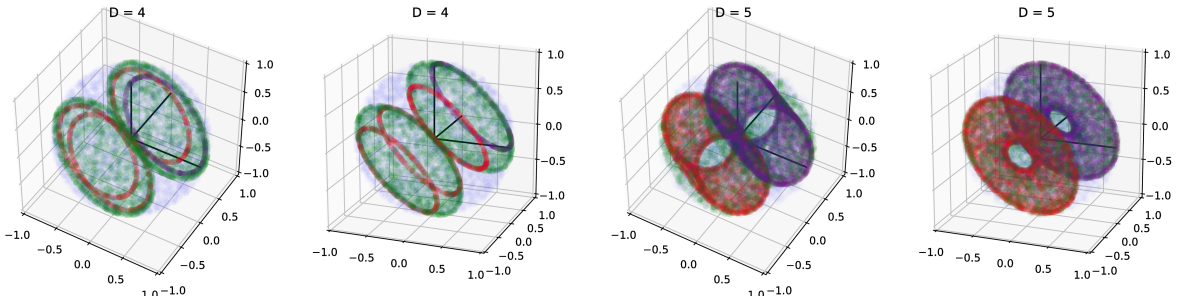


Figure 3.7: **Unitary Vectors in 4D and 5D**. Shown is the projection of the space into 3D (two views of the same data are shown for each dimensionality). The constraint $\sum_{i=1}^{d} v_i^2 = 1$ is shown in blue. The constraint $\sum_{i=1}^{d} v_i = \pm 1$ is shown in green. Unitary vectors with a positive constant and Nyquist coefficient are shown in purple. All other unitary vectors are shown in red. In 4D each unitary region is a circle, in 5D each region is a torus.

so the only degree of freedom for each element is the phase. Another constraint is that the inverse DFT of the Fourier coefficients must result in a real vector (i.e. no imaginary part). This constrains the coefficient for the constant term to be real (± 1). In addition, the phase of the negative frequency terms must be the complex conjugate of the corresponding positive frequency term. This constraint removes half of the remaining degrees of freedom. For the special case of even dimensionality, there is an additional term that represents the sum of the positive and negative Nyquist frequencies [32]. This term must be real for the result of the inverse DFT to be real, effectively removing one continuous degree of freedom when the dimensionality is even. Overall this results in $\lfloor (d-1)/2 \rfloor$ degrees of freedom in the frequency domain, matching the dimensionality derived from the spatial domain. It is important to note that this is the *continuous* dimensionality of the space. The sign of the constant coefficient adds an additional binary dimension, and the sign of the Nyquist coefficient adds another binary dimension for vectors with even dimensionality. This can be observed in the disjoint spaces of unitary vectors in Figure 3.6 and 3.7.

Only one of these disjoint spaces is useful for SSPs, the one in which the constant and Nyquist coefficients are positive. If one of those components is negative, each application of self convolution will flip the sign, causing the result to jump back and forth between two regions. This becomes a problem when fractional binding is used, as partial convolutions will leave the result between the two spaces (the resulting vector will have both real and imaginary parts, and when the imaginary part is discarded, it will no longer be unitary). Figure 3.8 shows this effect in 3 dimensions. Fractional binding in the 'good' part of the

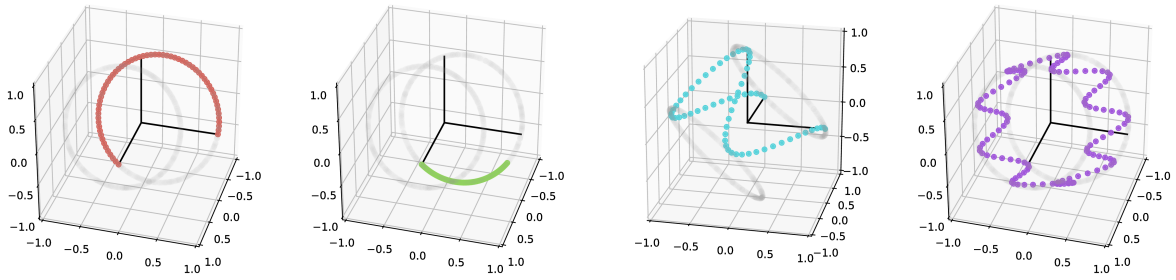


Figure 3.8: **Unitary Traversal in 3D**. Paths covered by the fractional binding of various unitary vectors. The first three plots show X^k for $k \in [0, 2]$, the final plot shows X^k for $k \in [0, 16]$. Where $\mathcal{F}\{X\} = [c_0, c_1, c_2]$. c_0 is the constant Fourier coefficient, c_1 is the positive frequency coefficient, and c_2 is the negative frequency coefficient, set to be the complex conjugate of c_1. **Red**: $c_0 = 1$, $c_1 = e^{i\frac{\pi}{3}}$. **Green**: $c_0 = 1$, $c_1 = e^{-i\frac{\pi}{8}}$. **Teal**: $c_0 = -1$, $c_1 = e^{i\frac{\pi}{3}}$. **Purple**: $c_0 = -1$, $c_1 = e^{i\frac{\pi}{8}}$.

space always results in a vector in the same space, while fractional binding from the other space creates a trajectory that follows a sinusoid between the two spaces. Another way of understanding this phenomenon is that the 'partial convolutions' of fractional binding draws from a continuous set of orthonormal matrices between the origin and the target rotation. If the vector being raised to a non-integer power is not in the 'good' part of the space, the orthonormal rotation matrix used will contain complex numbers with non-zero imaginary components, as it exists somewhere between the unitary regions of the complex space that intersects the reals.

The effect of circularly convolving two vectors together is an element-wise multiplication of their Fourier coefficients. Since the magnitude of every coefficient is 1, the result is a shift in the phase of each coefficient. The current phase corresponds to the location of the vector, which in 3D space is along a circle. When the phase is shifted by 2π, the result will be equivalent to the original vector. In the general case, the shape of the space where the unitary vectors exist will be an n-torus, where $n = \lfloor (d-1)/2 \rfloor$. This can be seen most easily in the frequency domain where the possible values of each Fourier coefficient traces out a circle. The Cartesian product of n circles forms an n-torus [185]. In fact, the circle itself is defined as a 1-torus.

Any individual point on this torus can be used to define a direction and magnitude in that direction. The direction is defined by the vector formed from the phases of each positive frequency Fourier coefficient, and the magnitude is the length of that vector. Examples of trajectories in a fixed direction along the torus in higher dimensions are

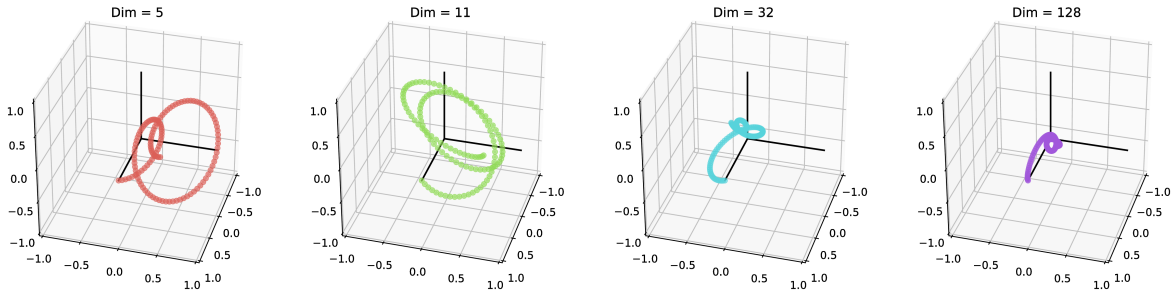


Figure 3.9: **Unitary Traversal in Higher Dimensions**. Each plot shows X^0 to X^5 where X is a random unitary vector in the 'good' subspace. The trajectories appear to be coiling around themselves due to being projected to 3D for viewing. The trajectories have constant curvature and do not self-intersect in their true space. In higher dimensions, the trajectories appear to be smaller because the component magnitudes tend to be smaller.

shown in Figure 3.9.

The origin of this space is the point $[1, 0, ..., 0]$. This vector defines no magnitude or direction, analogous to the origin in standard Cartesian space. The circulant matrix formed by this vector is the identity matrix, which leaves the original vector unchanged when multiplied. Any SSP raised to the power of 0 will result in this origin point (analogous to any vector in standard Cartesian space multiplied by 0 will return the origin point).

One interesting property of all of the good unitary vectors lying on the surface of a hypersphere is that the mean of any number of unique unitary vectors will itself not be a unitary vector. This is due to the fact that the mean must lie within the hypersphere, and will no longer be on the surface.

3.2.1 Representing Cartesian Space

The n-torus that the unitary vectors reside on is a generalization of a special kind of torus known as a Clifford torus [181, 88]. A Clifford torus resides in $\mathbb{R}^4$ and is the Cartesian product of two circles with their own independent embedding spaces in $\mathbb{R}^2$. It can also be derived using complex coordinate spaces using two circles in separate instances of $\mathbb{C}^1$ forming a torus in $\mathbb{C}^2$, which is topologically equivalent to $\mathbb{R}^4$ [10]. This can be generalized to an n dimensional torus by taking the Cartesian product of n circles of the form $S_k^1 = \{e^{i\phi_k} | 0 \leq \phi_k < 2\pi\}$.

$$\prod_{k=1}^{n} S_k^1 = (e^{i\phi_1}, e^{i\phi_2}, ..., e^{i\phi_n}) \tag{3.14}$$

This matches the space of unitary vectors, where $e^{i\phi_k}$ is the value of the kth Fourier coefficient of the DFT of the unitary vector. This proves that the space is topologically a torus, but in order to prove that it is specifically a Clifford torus it needs to be shown that each circle exists in its own independent embedding space. In other words, the plane that contains each circle must be orthogonal to all of the other planes.

One way to prove this is to show that any vector that lies in one circle has a dot product of zero with all other vectors that span any other circle. For an N dimensional space, there will be $\lfloor (N-1)/2 \rfloor$ circles denoted S_k^1. Each of these circles is defined by the Fourier coefficients of an N dimensional vector all set to 1 except for the coefficients at index k and $N+1-k$; for $1 \leq k \leq \lfloor (N-1)/2 \rfloor$. The coefficient at index k is set to $e^{i2\pi\phi_k}$ and the coefficient at index $N+1-k$ is set to be the complex conjugate, $e^{-i2\pi\phi_k}$, where $0 < \phi_k < 2\pi$. Let u_k be an N dimensional vector that exists in the plane of the circle S_k^1. The objective is to show that:

$$u_a \cdot u_b = 0 \qquad \forall a \neq b \tag{3.15}$$

To do this we need to define an equation that represents vectors in the plane of each circle. The vector difference of any two unique points within a plane will produce a vector that lies in that plane. Any point in the circle S_k^1 can be defined by the inverse discrete Fourier transform:

$$u_{k_n} = \frac{1}{N} \sum_{p=0}^{N-1} X_p \cdot e^{i2\pi pn/N} \qquad n = 0, 1, \ldots, N-1 \tag{3.16}$$

where the Fourier coefficients X_p ($p = 0, 1, \ldots N-1$) are defined as described above. The point $o = [1, 0, ..., 0]$ exists in every circle, as its Fourier transform produces $[1, 1, ..., 1]$. Subtracting this point from Equation 3.16 will produce a formula for all directions that lie in the plane containing the kth circle. The relation we need to prove can be expressed as follows:

$$\sum_{n=0}^{N-1} \left(\frac{1}{N} \sum_{p=0}^{N-1} X_p \cdot e^{i2\pi pn/N} - o_n \right) \left(\frac{1}{N} \sum_{q=0}^{N-1} Y_q \cdot e^{i2\pi qn/N} - o_n \right) = 0 \tag{3.17}$$

where X_p are the Fourier coefficients of u_a and Y_q are the Fourier coefficients of u_b. Every element of o_n is 0 except for $o_0 = 1$ so that term can be extracted out of the sum. When

$n = 0$ the complex exponential term evaluates to 1 and can be removed. The $1/N$ can be factored out of the sum and removed as the equation equates to 0:

$$\sum_{n=1}^{N-1}\left(\sum_{p=0}^{N-1} X_p \cdot e^{i2\pi pn/N}\right)\left(\sum_{q=0}^{N-1} Y_q \cdot e^{i2\pi qn/N}\right) + \left(\sum_{p=0}^{N-1} X_p - 1\right)\left(\sum_{q=0}^{N-1} Y_q - 1\right) = 0 \quad (3.18)$$

Focusing for now on just one interior sum, we notice that it is close to a geometric series. By adding and subtracting the misaligned terms, it can be refactored as follows:

$$\left(\sum_{p=0}^{N-1} X_p \cdot e^{i2\pi pn/N}\right) = \frac{1-(e^{i2\pi n/N})^N}{1-e^{i2\pi n/N}} + X_a \cdot e^{i2\pi an/N} + X_a^* \cdot e^{i2\pi(N-a)n/N} - e^{i2\pi an/N} - e^{i2\pi(N-a)n/N} \quad (3.19)$$

where X_a is $e^{i2\pi\phi_a}$ and X_a^* is the complex conjugate, $e^{-i2\pi\phi_a}$.

The term $e^{i2\pi m}$ is equal to 1 for all integer values of m, making the numerator in the fraction equate to 0. The denominator is non-zero since n/N cannot be an integer, because $1 \leq n \leq N-1$. This leads to the simplification:

$$\left(\sum_{p=0}^{N-1} X_p \cdot e^{i2\pi pn/N}\right) = (X_a - 1) \cdot e^{i2\pi an/N} + (X_a^* - 1) \cdot e^{-i2\pi an/N} \quad (3.20)$$

The same process can be repeated for the analogous term containing Y_q. The following simplification can be made for the terms brought outside of the main sum from Equation 3.18, since the majority of the terms within the sum will be zero:

$$\left(\sum_{p=0}^{N-1} X_p - 1\right)\left(\sum_{q=0}^{N-1} Y_q - 1\right) = (X_a - 1 + X_a^* - 1)(Y_b - 1 + Y_b^* - 1) \quad (3.21)$$

Combining the simplifications of Equation 3.20 and 3.21 yields:

$$\sum_{n=1}^{N-1}\left((X_a - 1) \cdot e^{i2\pi an/N} + (X_a^* - 1) \cdot e^{-i2\pi an/N}\right)\left((Y_b - 1) \cdot e^{i2\pi bn/N} + (Y_b^* - 1) \cdot e^{-i2\pi bn/N}\right) + (X_a - 1)(Y_b - 1) + (X_a^* - 1)(Y_b - 1) + (X_a - 1)(Y_b^* - 1) + (X_a^* - 1)(Y_b^* - 1) = 0 \quad (3.22)$$

Rearranging the terms within the sum, it is clear that the terms outside the sum can be brought in as they correspond to the components where $e^0 = 1$. The sum can then be

broken apart into four distinct terms:

$$\begin{aligned}
&\sum_{n=0}^{N-1}\left((X_a-1)(Y_b-1)\cdot e^{i2\pi n(a+b)/N}\right)+\\
&\sum_{n=0}^{N-1}\left((X_a^*-1)(Y_b-1)\cdot e^{i2\pi n(b-a)/N}\right)+\\
&\sum_{n=0}^{N-1}\left((X_a-1)(Y_b^*-1)\cdot e^{i2\pi n(a-b)/N}\right)+\\
&\sum_{n=0}^{N-1}\left((X_a^*-1)(Y_b^*-1)\cdot e^{-i2\pi n(a+b)/N}\right)=0
\end{aligned} \tag{3.23}$$

Each of these terms forms a geometric series which allows them to be simplified. The X_a and Y_b terms within the sums are constant, so they can be factored out:

$$\begin{aligned}
&(X_a-1)(Y_b-1)\left(\frac{1-(e^{i2\pi(a+b)/N})^N}{1-e^{i2\pi(a+b)/N}}\right)+\\
&(X_a^*-1)(Y_b-1)\left(\frac{1-(e^{i2\pi(b-a)/N})^N}{1-e^{i2\pi(b-a)/N}}\right)+\\
&(X_a-1)(Y_b^*-1)\left(\frac{1-(e^{i2\pi(a-b)/N})^N}{1-e^{i2\pi(a-b)/N}}\right)+\\
&(X_a^*-1)(Y_b^*-1)\left(\frac{1-(e^{-i2\pi(a+b)/N})^N}{1-e^{-i2\pi(a+b)/N}}\right)=0
\end{aligned} \tag{3.24}$$

The terms in every numerator have an integer multiple of 2π in the exponent, so they evaluate to zero. For the denominators to be non-zero, the terms $\frac{a+b}{N}$, $\frac{b+a}{N}$, $\frac{a-b}{N}$, and $\frac{-a-b}{N}$ must all be non-integer. Since $a \leq N/2$ and $b \leq N/2$ and $a \neq b$, their sum must be strictly less than N. Similarly, the absolute value of their difference and negative sum must be strictly less than N. Since $a \neq b$, their difference cannot be 0. Therefore all denominators will be non-zero, and the entire left side of the equation will evaluate to 0. This proves that all circles S_k^1 exist in an independent $\mathbb{R}_k^2$, meaning their Cartesian product is a Clifford n-torus.

An extremely useful property of the Clifford torus is that the space is flat (has zero Gaussian curvature); this means that the geometry on the surface is equivalent to Euclidean geometry [185]. In other words, all of the standard properties of Euclidean geometry will

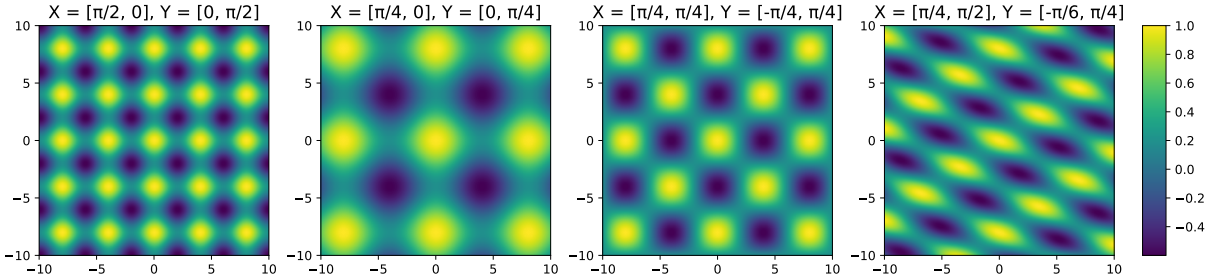


Figure 3.10: **Similarity Heatmap of 2D Axes**. Each plot shows the value of the dot product of $X^x \circledast Y^y$ with $X^0 \circledast Y^0$ with different values of ϕ_k. 5D axis vectors are used, resulting in two parameters required to fully define each vector.

hold locally (e.g. parallel lines will never intersect, the sum of angles in a triangle will be π radians, etc). As a result, using a SSP representation that lies on such a torus maps well to our standard Cartesian system and is thus a natural, though high-dimensional, way to represent space.

3.2.2 2D Subspace

To represent a 2D coordinate as an SSP, a 2D subsurface of the n-torus of unitary vectors will be used. Each axis vector of the SSP representation defines a 1D curve and a scaling along that curve. Two axis vectors will define a 2-torus based on the Cartesian product of their curves, fully embedded in the n-torus. Axis vectors must have a dimensionality of at least 5D to represent 2D space. In this case there is only one possible surface to use, although the mapping between this surface and the 2D coordinates it represents can be arbitrarily rotated and scaled. An illustration of different mappings based on the choice of axis vectors is shown in Figure 3.10. Views of the surface formed by the first mapping from that figure are shown in Figure 3.11. An axis vector can be fully defined by its choice of positive frequency Fourier coefficients, e^{ϕ_k}. With $-\pi < \phi_k < \pi$, movement of one unit in the 2D direction that the axis represents corresponds to moving ϕ_k radians along circle S^1_k. In other words, a smaller magnitude of ϕ corresponds to a shorter distance along the torus, and a larger magnitude a longer distance. The sign determines the direction.

In 7 or more dimensions, there are multiple choices of what 2-torus subspace to use. As the dimensionality increases, random choices of axis vectors will tend to be nearly orthogonal. In addition, the distance required to move in order to return to the same point in the manifold increases dramatically. Heatmaps from random axis vectors of different

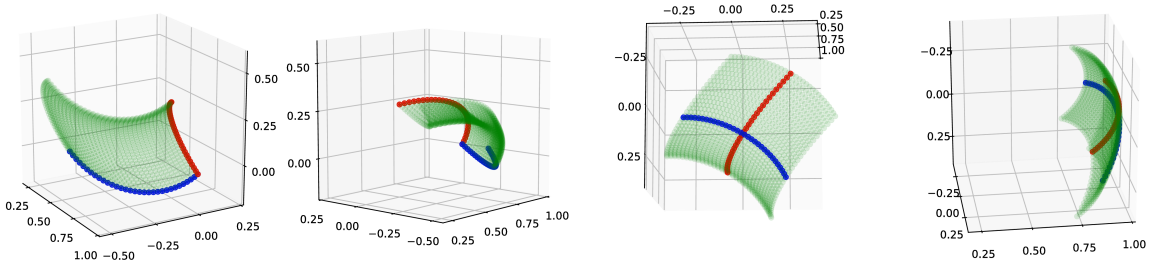


Figure 3.11: **2D Surface**. Locations represented as 5D SSPs. Different views of the space projected into 3D. The x-axis of the space being represented is shown in blue and the y-axis is shown in red. The two plots on the left show the encoding of locations between 0 and 1. The two plots on the right show the encoding of locations between -0.5 and 0.5.

dimensionalities are shown in Figure 3.12.

3.2.3 Controlling Periodicity

Travelling in any direction on a torus from one location will eventually return back to the same location. The distance travelled before the same location is reached is dependent on which direction is taken. Figure 3.13 shows a visual representation of this fact.

The values of ϕ_k for an axis vector define the distance along each orthogonal circle that the representation moves for each circular convolution. Taken together, this will form a particular distance and direction of travel within the entire space. The number of circular convolutions required is dependent on the smallest period T such that $\phi_k T = 2\pi N$ for all k, where N is a non-zero integer. In this manner the values of ϕ_k can be selected to achieve a desired periodicity. Examples of axis vectors with specified periodicity are shown in Figure 3.14. Note that if the ratio of two ϕ_k values is irrational, the representation will never truly repeat (although limits of machine precision make truly irrational phases impossible to implement). While they will in theory never repeat, for all practical purposes they will eventually be close enough to the original vector that the difference is indistinguishable from noise.

For some applications it may be useful to control how soon the representation repeats. For example, if the metric of interest is periodic in nature (such as the direction an animal is looking), then periodicity can be used to ensure that states that should be treated the same are represented the same (e.g. 0 degrees and 360 degrees). For many applications,

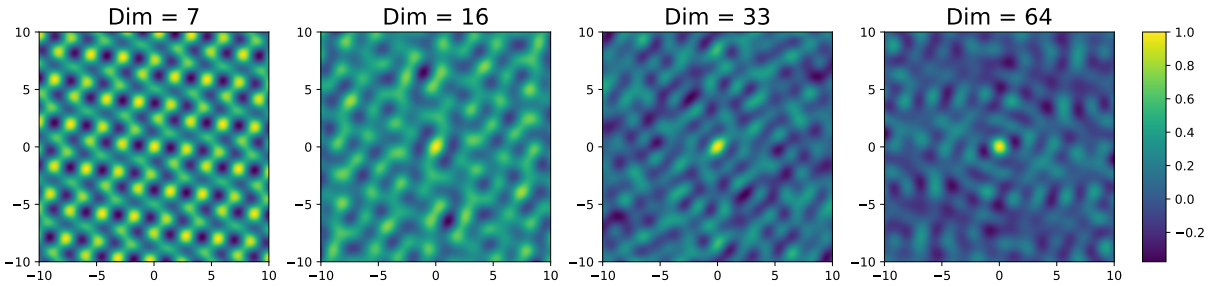


Figure 3.12: **Similarity Heatmap of Random 2D Axes**. Each plot shows the value of the dot product of $X^x \circledast Y^y$ with $X^0 \circledast Y^0$. Values of ϕ_k are drawn uniformly at random from $-\pi < \phi_k < \pi$ for each axis. As the dimensionality increases, the chance that the curves corresponding to each axis vector are orthogonal increases. The distance required to move in order to return to the same point in the manifold also increases.

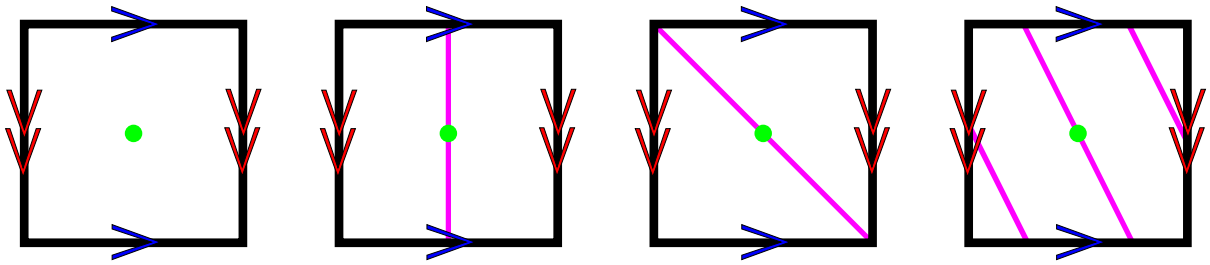


Figure 3.13: **Distance to Return**. Flat depiction of a torus. Shown in magenta on each of the three rightmost figures is a single straight line that passes through a point in green. Opposite edges are connected. The length of a line from any point until it returns to the same point depends on its orientation.

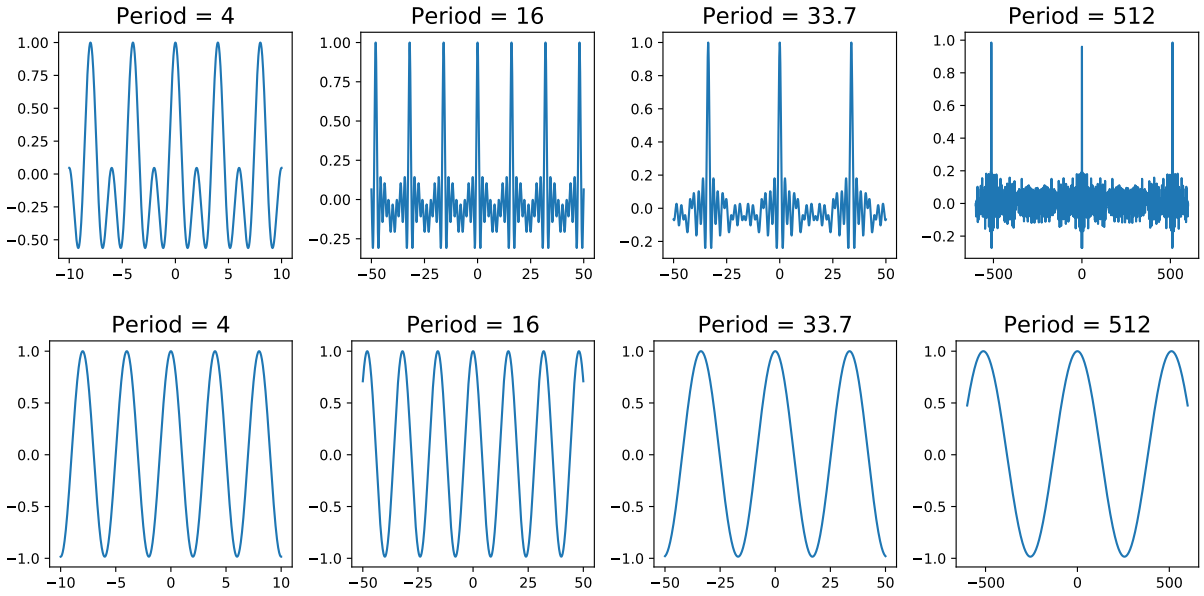


Figure 3.14: **Periodicity of Axis Vectors**. Each plot shows the value of the dot product of X^x with X^0. Values of ϕ_k are chosen such that all $\phi_k \cdot x$ are an integer multiple of 2π only when x is an integer multiple of the desired period. 256D axis vectors are used for all cases. The period can be any real number greater than or equal to 2. **Top**: ϕ_k drawn uniformly at random from the values $\frac{2\pi n}{T}$ where n is a non-zero integer that satisfies $|2n| \leq T$. **Bottom**: ϕ_k drawn from $\pm\frac{2\pi}{T}$

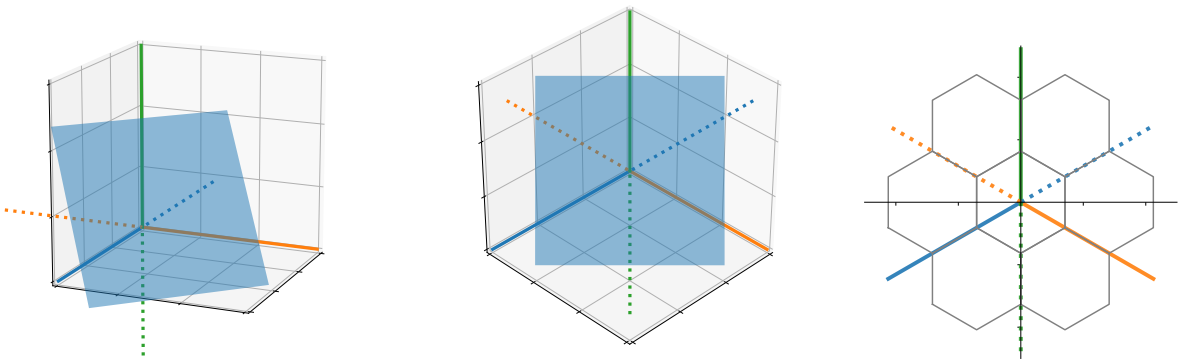


Figure 3.15: **Projection to Hexagonal Coordinates**. Illustration of transforming coordinates in 3D to hexagonal coordinates in 2D. **Left**: Plane in 3D. **Middle**: Perspective aligned to the normal vector of the plane. **Right**: Coordinate system of the plane with principal axes of the 3D system overlaid.

repetition is not desirable. One way of selecting phase offsets to minimize repetition is to choose them randomly.

A mix of periodic and effectively aperiodic axes can be used in the same representation, allowing some dimensions to wrap around while others do not. This is one way of representing 2-D position and heading direction simultaneously in a single semantic pointer.

3.2.4 Hexagonal Coordinate System

Inspired by the hexagonal structure of grid cell activity, another way of representing a 2D coordinate is as a 3D coordinate on a hexagonal grid. Here each of the dimensions is corresponding to moving in the direction of one of the three principal axes of a hexagonal coordinate system.

One way to interpret a hexagonal coordinate system is as a standard 3D coordinate system projected onto a 2D plane. By choosing the plane to have a normal vector of $\begin{bmatrix}1 & 1 & 1\end{bmatrix}$, movement along any of the principal axes in the 3D system corresponds to moving along the 120 degree apart principal axes in the hexagonal coordinate system. A visualization of this relation is shown in Figure 3.15.

In the original SSP formulation, two high-dimensional vectors, X and Y, are chosen to define the encoding. In the modified formulation for a hexagonal coordinate system, three high-dimensional unitary vectors, X, Y, and Z, are used. The convolutional exponents

for each of these vectors are determined by transforming a given 2D coordinate into a 3D coordinate constrained to lie on a 2D plane. This transformation is carried out by multiplying the 2D coordinate by a 2×3 matrix. This matrix is formed by stacking two orthogonal unit vectors that lie on the plane. The hexagonal SSP formulation is:

$$\begin{aligned} S_3(x,y) &= X^{x'} \circledast Y^{y'} \circledast Z^{z'} \\ \begin{bmatrix} x' & y' & z' \end{bmatrix} &= \begin{bmatrix} x & y \end{bmatrix} A_{2\times 3} \end{aligned} \tag{3.25}$$

It is important to note that there is no additional computational cost associated with defining the representation using three axis vectors, as an equivalent two axis vector system can be constructed and used instead. The axis vectors of this equivalent system are defined as:

$$\begin{aligned} S_2(x,y) &= X_2^{vx} \circledast Y_2^{vy} \\ X_2 &= S_3(v^{-1}, 0) \\ Y_2 &= S_3(0, v^{-1}) \end{aligned} \tag{3.26}$$

where v is a scaling factor between the two and three axis systems. These vectors can be thought of as spanning a 2D subspace of the 3D space being represented. Since the original three axis vectors are chosen randomly, this operation can also be interpreted as a change in the distribution where possible axis vectors are sampled from and the new X and Y vectors are no longer independent.

The scaling factor between the two systems can be calculated using the geometric relations depicted in Figure 3.16. The x and y axes of the 2D system are shown in blue and orange, with the plane that they form shaded in light blue. The normal vector to this plane is shown in green. The x, y, and z axes of the 3D system are shown in purple, brown, and pink. From the cube in Figure 3.16 (Middle), the length of the diagonal on a face is $\sqrt{2}$ which leads to $\alpha = \tan^{-1}(1/\sqrt{2})$. This angle is the same as the angle between the y axis in 2D space and the z axis in 3D space, as can be seen from rotating the perspective to Figure 3.16 (Right). This leads to $v = 1/\cos\alpha = \sqrt{3/2}$. This scaling factor is required in Equation 3.26 because moving one unit in the 2D system corresponds to moving v units in the 3D system.

Another way to arrive at the same result is to compute the singular values of the matrix formed by stacking three 2D unit vectors corresponding to the principal axes of a hexagonal coordinate system. Each of these axes will be $2\pi/3$ radians apart:

$$A = \begin{bmatrix} \cos(\theta) & \cos(\theta + 2\pi/3) & \cos(\theta + 4\pi/3) \\ \sin(\theta) & \sin(\theta + 2\pi/3) & \sin(\theta + 4\pi/3) \end{bmatrix} \tag{3.27}$$

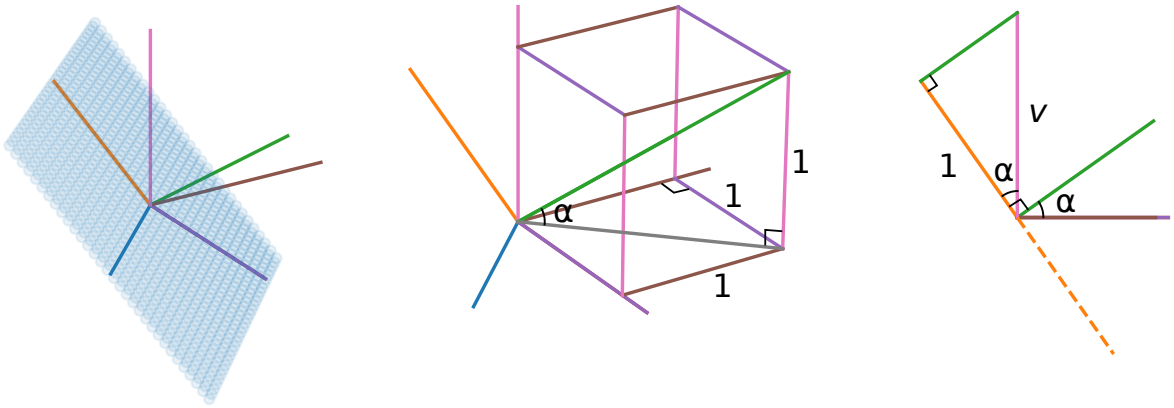


Figure 3.16: **Axis Transformation**. Geometry of the axis transformation for hexagonal coordinates. v is the scaling between the two systems. All lines of the same colour are parallel.

where θ determines the orientation of the hexagonal coordinate system. Solving for the singular values gives:

$$\sqrt{AA^T} = \sqrt{(3/2)} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{3.28}$$

where all singular values are equal to v as calculated above. In general, when mapping N equally spaced principal axes from a 2D system to an orthogonal N-D system, the scaling will be:

$$v = \left(\sum_{n=0}^{N-1} \cos^2 \left(\theta + 2\pi n/N \right) \right)^{1/2} = \sqrt{N/2} \qquad \forall N > 2 \tag{3.29}$$

A proof of this relation can be found at [155].

Heatmaps from a low dimensional hexagonal SSP system of various scales and orientations is shown in Figure 3.17. In these examples, each of the three axes are from an independent circle S^1_k with the same value of ϕ for each (i.e. equal scaling in all directions). Projecting onto a 2-torus from the 3-torus using Equation 3.26 produces hexagonal symmetry in the space. Visualizations of the encoded space for $\phi_k = \pi/2$ are shown in Figure 3.18 and 3.19.

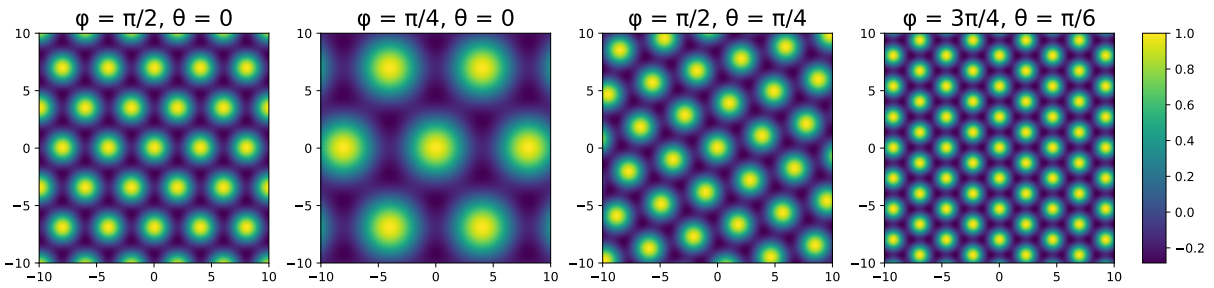


Figure 3.17: **Similarity Heatmap of Hexagonal Axes**. Each plot shows the value of the dot product of $X^x \circledast Y^y$ with $X^0 \circledast Y^0$ with different values of ϕ and θ. Three 7D unitary vectors are transformed to a 2D hexagonal coordinate system using Equation 3.26 with a rotation of θ in the coordinate transformation matrix. These three unitary vectors are set to orthogonal directions with a magnitude of ϕ.

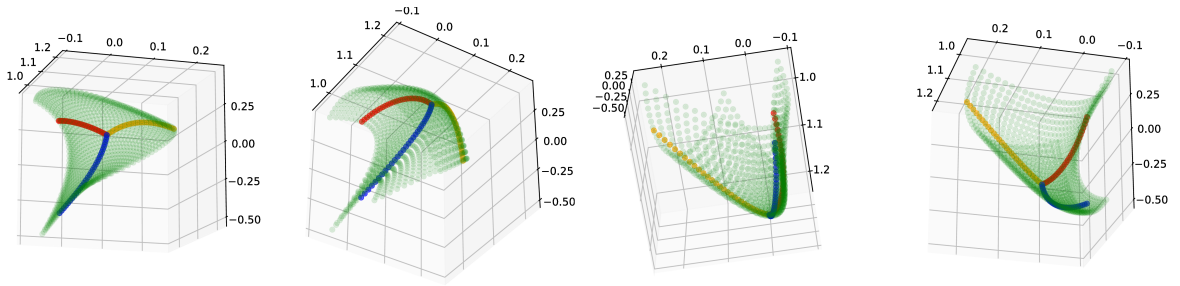


Figure 3.18: **Surface of Hexagonal Torus**. The primary axes of the hexagonal coordinate system are shown in red, blue, and yellow. Shown in green is the encodings of a patch of space with x and y coordinates between -0.5 and 0.5. Each plot shows a different view of the same 7D data.

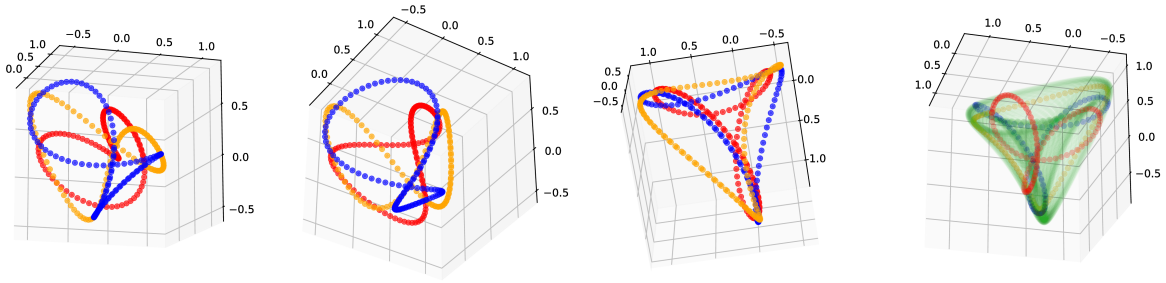


Figure 3.19: **Primary Axes of Hexagonal Torus**. Different views of the same data from 7D SSPs. Each colour corresponds to one primary axis of a hexagonal coordinate system. The green overlay on the far right image is the surface of the entire space.

SSP Formulation Comparison

One way to measure the effectiveness of an encoding method is to compare the representation of a location to all other locations. Ideally if the representations are similar, the locations should be similar (i.e., nearby) and if the representations are different, the corresponding locations should be different. Similarity is measured as the cosine distance between two vectors. A comparison of the similarity heatmaps generated from the square and hexagonal approaches encoding the coordinate (0, 0) is shown in Figure 3.20.

In the original formulation there are clear noise patterns aligned with the axes, visible in both a single choice of axis vectors as well as averaging across many choices. For the hexagonal formulation, there is less distinct structure in the noise pattern and only when averaged across many choices of axis vectors does a faint hexagonally symmetric pattern arise. When comparing the heatmaps to an ideal Gaussian fit, the original formulation results in an RMSE of 0.043 while the hexagonal formulation has an RMSE of 0.019. A 1D slice of this Gaussian fit is shown in Figure 3.21.

Another way to visualize this difference is to calculate the average deviation of the similarity score from 0 as a function of radial distance from the origin. The hexagonal projection case smoothly approaches the noise floor while the original method has larger oscillations before settling out. These findings are shown in Figure 3.22.

Relation to Grid Cells

SSPs and grid cells have many features in common. The most striking is the periodic nature of both representations. Another is the multiple scales at which this periodicity

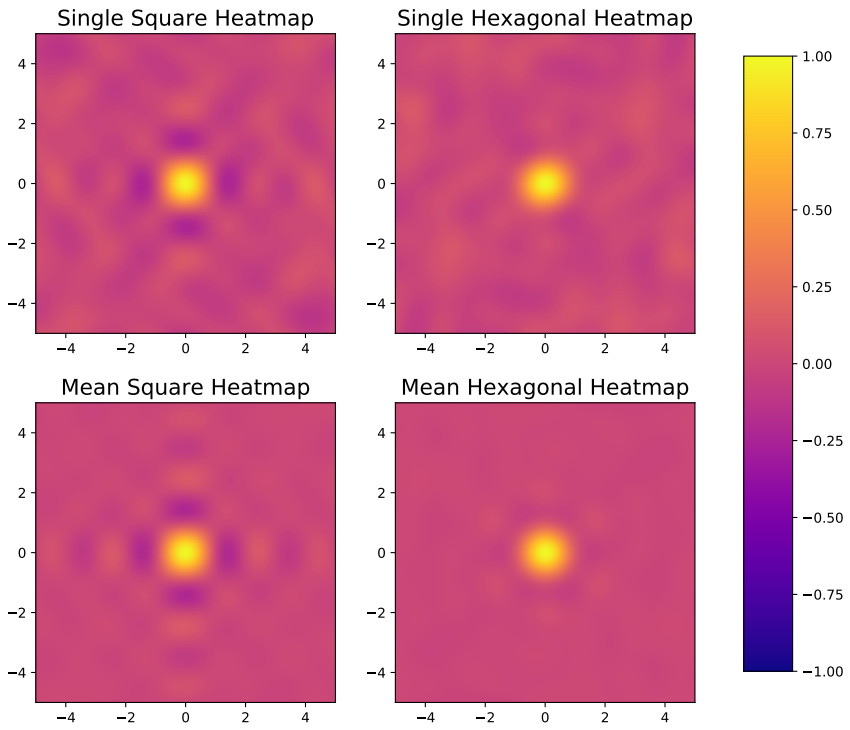


Figure 3.20: **Similarity Heatmap of Different SSP Forumalations**. **Left**: Original SSP formulation with two axis vectors 90 degrees apart. **Right**: Hexagonal SSP formulation with three axis vectors 120 degrees apart. **Top**: Heatmaps for a single set of random axis vectors. **Bottom**: Mean of 25 heatmaps for different random axis vectors.

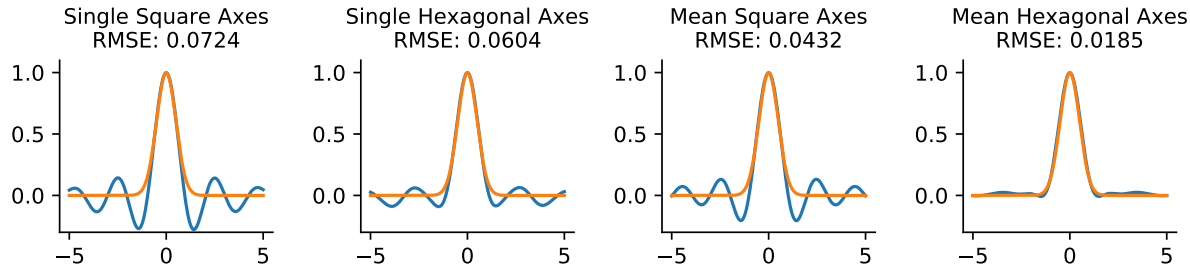


Figure 3.21: **Gaussian Fit**. Matching the shape of the SSP similarity map (blue) to a Gaussian function (orange). A Gaussian with $\sigma = 1/\sqrt{2}$ was the closest fit in all cases. All axis vectors are 256D. The mean is across 32 randomly generated axis vectors. The two plots on the left depict a single sample of square axes and hexagonal axes. RMSE is calculated across the 2D space, discretized into 256x256 bins. Shown is a 1D slice of that space.

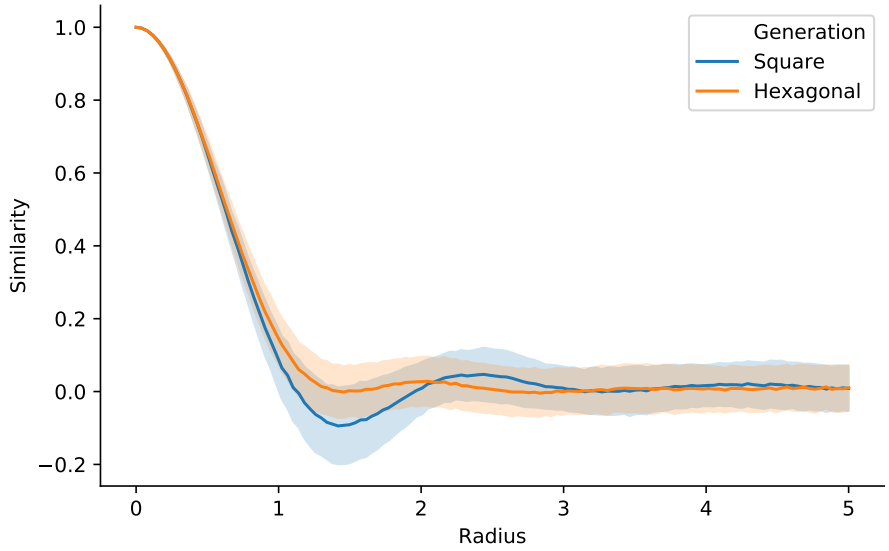


Figure 3.22: **Similarity Deviation**. Shown as a function of radial distance from the origin. Calculated with 256D vectors generated as Hexagonal or Square SSPs. Shaded is the confidence interval of the mean.

occurs. Since all circles S^1_k that form the unitary vector space for SSPs are independent, any grouping of 3 circles can be used to construct a hexagonal torus that is orthogonal to the rest of the space. In this manner, for unitary vectors of dimensionality d, the space can be decomposed into $\lfloor \lfloor (d-1)/2 \rfloor /3 \rfloor$ orthogonal hexagonal tori. Each hexagonal torus can be defined to have a different orientation and scale (e.g., for 256D vectors, there would be 42 different orientations and scales). Any neuron with a preferred direction (encoder) aligned to the vector corresponding to a position on one of the tori will have a set of hexagonal place fields aligned to the orientation and scale of that torus. More detail on constructing spiking neural networks that represent SSPs that exhibit spatial tuning properties found in the brain is described in Section 4.4.

Theoretical work has shown that grid cells are extremely efficient at encoding spatial information relative to a place cell code [161, 110]. An SSP implemented using neurons with grid cell tuning properties can take advantage of this efficiency.

Chapter 4

Neural Experiments

This chapter focuses on implementing spatial functions in spiking neural networks with SSPs. The first section describes a set of operations that are important for any spatial representation to be able to perform, along with experiments demonstrating the effectiveness of spiking neural networks implementing these operations. The next two sections present models of tasks demonstrating two ways in which SSPs can be used for spatial memory. The final section describes how an SSP representation is very naturally implemented by populations of neurons with spatial tuning properties found in the brain.

4.1 Spatial Operations

For a spatial representation to be useful it must be amenable to spatial operations that can manipulate the representation to perform a function. A set of nine desirable operations as well as performance on those operations is shown in Table 4.1. The experiments performed to obtain these results are described below. Each experiment is conducted with both mathematically ideal operations and through a spiking neural network implementing the underlying function. Doing so provides a demonstration of how easy or difficult it is to implement the relevant operation in a biologically plausible substrate.

Query a single object

A query for the location of an object can be performed as follows:

$$S = M \circledast OBJ^{-1}. \tag{4.1}$$

where M is the memory vector, OBJ is a vector corresponding to the object of interest, and S is the SSP representing the location of the object. Accuracy is computed by decoding this high-dimensional vector, S, into the 2-D coordinate it represents and comparing it to the true location.

Query a missing object

Given a memory containing objects, query an object that does not exist (using equation 4.1). The correct behaviour is a result that is highly dissimilar to all locations within the domain of interest. This is determined by the dot product of S and every SSP being less than 0.1.

Query a duplicate object

Given a memory containing many objects with some duplicates, query an object that appears twice. The correct behaviour is to return a spatial semantic pointer that represents both locations of this object.

Query a location

Use equation 3.8 with the location for one of the objects in memory. The correct behaviour is to return a semantic pointer for the object at that location.

Query a region

On each trial a circular region is created with a radius between 1 and 3 units and centered at a random location. An SSP is constructed for this region using equation 3.5. The inverse of this SSP is convolved with the memory to obtain a semantic pointer representing all objects in the region. Accuracy is computed by adding the number of objects correctly detected in the region to the number of objects correctly not detected from outside the region and then dividing by the total number of objects in the memory.

Shift a single object in a group

Moving a single object within a group can be accomplished by adding the object of interest convolved with a vector that is the difference between the start and end positions, as shown

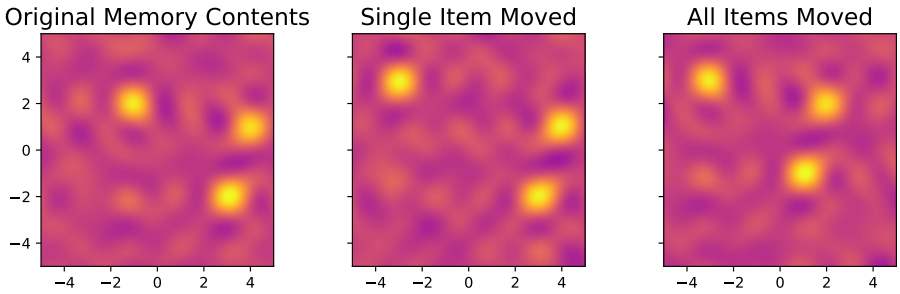


Figure 4.1: **Shifting Objects in Memory**. **Left**: The original memory. **Middle**: Shifting the top left object. **Right**: Shifting all three objects.

in equation 4.2. Accuracy is reported for all objects as well as just the object that was moved.

$$\Delta M = OBJ \circledast \Delta S. \tag{4.2}$$

Shift a whole group

The memory is convolved with an SSP that corresponds to a random displacement, which leverages the property of equation 3.3. An object query is then performed for each object in the memory and the result is considered correct if it moved by the displacement amount. A heatmap visualizing the result of the two shifting operations is shown in Figure 4.1 for a group of three identical objects.

Readout the (x, y) location from an SSP

For the non-neural case, location is extracted from the maximum point in the heatmap. In spiking neurons, a heteroassociative memory is optimized to map from a 512-dimensional SSP to a 2-D location.

Construct an SSP from the (x, y) location

This can be computed directly from equation 3.4. For the experiment using spiking neurons each axis is first computed separately and then convolved together. Correctness is measured by extracting the maximum point in the heatmap of the resulting SSP and comparing this to the true location.

Desiderata	Accuracy	
	Non-Neural	Neural
Query single object	99.1%	95.7%
Query missing object	99.4%	96.7%
Query location	97.3%	94.7%
Query duplicate object	97.4%	95.3%
Query region	90.4%	73.5%
Shift single object in group (all objects)	75.7%	67.3%
Shift single object in group (moved object)	100.0%	100.0%
Shift whole group	97.8%	96.7%
Readout (x, y) location from SSP	100.0%	94.1%
Construct SSP from (x, y) location	100.0%	99.0%

Table 4.1: **Experiments for the desiderata for metric representations of space**. Accuracy is calculated using SSP representations containing 2 to 24 items. When the output is an item, it is considered correct when the output vector is more similar to the vector for the correct item than to any of the 48 other items in the vocabulary. When the output is a location, it is considered correct when the result is within 0.5 units of the true location (locations chosen randomly across a 10 by 10 unit domain).

4.1.1 Results

All spiking neural network experiments used leaky integrate-and-fire (LIF) neurons and the NEF to implement the necessary transformations. In all trials 50 neurons were used per dimension to represent the memory and to compute circular convolutions.

The results of the experiments for each of the desideratum are shown in Table 4.1. As can be seen from the table, the SSP representation is able to address the desiderata quite well, both in purely mathematical and neural implementations. The worst performance is evident in the shifting of a single object in a group. Specifically, the accuracy of the representation for the objects that were not shifted decreases, while the accuracy for the shifted object increases. This is due to normalization effects making the moved object be re-encoded with a larger relative magnitude than the rest of the items. Using a scaling factor proportional to the number of items in the memory mitigates this effect (improves accuracy from 75.7% to 97.8%), but in general the number of items within a memory is not always known without first retrieving items from memory, and equation 4.2 is agnostic to the other contents of the memory.

To better characterize the capacity of a single memory using this representation we

performed queries on memories with progressively larger numbers of items encoded (see Figure 4.2). The shape of the curve is very similar for both location and object queries since the decrease in the dot product is mostly a result of the normalization of the memory to a unit vector. The standard deviation for the dot product of two random vectors in a unit hypersphere is $\sqrt{1/D}$ where D is the dimensionality of the space. The mean is zero, so for 512 dimensions this results in a 3-sigma threshold similarity of 0.133. SSPs that represent coordinates within a finite domain will span a smaller subspace of the hypersphere, so their threshold will be a little higher. Specifically, we estimated the threshold by generating 10,000 random SSPs from a 10×10 2-D domain and computing the dot product between every pair. The mean is approximately zero, and three standard deviations is 0.154. Consequently 99.7% of queries will be above this value for items actually in the memory. The accuracy plots show the importance of dimensionality on accuracy of decoding memories.

4.1.2 Hierarchical Spatial Semantic Pointers

As more semantic pointers are represented by a population of neurons, it becomes more difficult to retrieve any semantic pointer accurately. One solution to this problem is to encode the relations represented by these semantic pointers hierarchically [46]. This can be done by grouping related objects together in their own 'sub-map' and having a single semantic pointer represent this 'sub-map'. This can be seen as analogous to the notion of 'chunking' in the psychology literature, where items are grouped together and treated as a single item to improve recall [59]. A heteroassociative memory can be used to map the pointer representing the group to the detailed contents of the group itself. This technique of hierarchical encoding is capable of representing human-scale language as demonstrated by [34]. The authors were able to successfully encode and decode the main lexical relations in WordNet, a database which contains over 100,000 terms [117], in a compact spiking neural network. The large-scale heteroassociative memories required for this kind of model can also be learned over time [180]. One important design choice in this work was to use unitary vectors for 'role' vectors when encoding complex sentence structures. This improved the accuracy in which sentence constituents could be extracted [35]. In the case of SSPs, all locations are already unitary vectors.

One example of a hierarchical spatial structure of different scales is depicted in Figure 4.3. Here somebody familiar with the cities listed on the map would likely know the general layout of landmarks within the city, as well as structures of neighbourhoods and the interior layout of some of the buildings. It would be impractical to encode the relative layout of rooms in a building on Waterloo campus to the layout of a house in London or

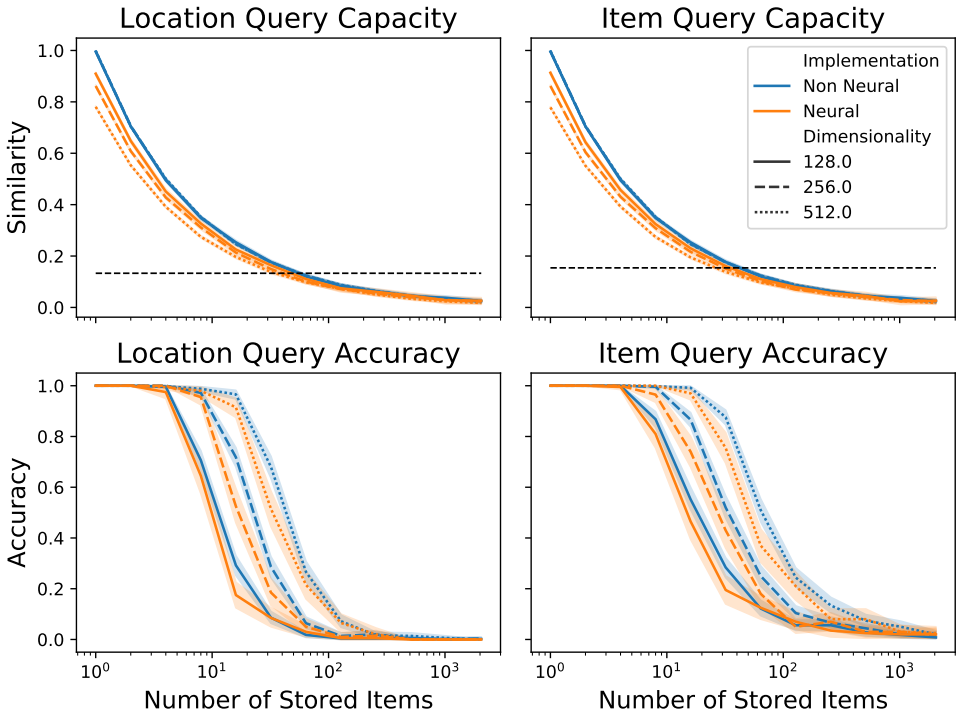


Figure 4.2: **Memory Capacity and Accuracy**. Plotted as a function of the number of items in the SSP for ideal and neural implementations while varying the dimensionality. Top panels show the item and location capacity (dotted line is the 3-sigma similarity threshold). Bottom panels show the item and location accuracy.

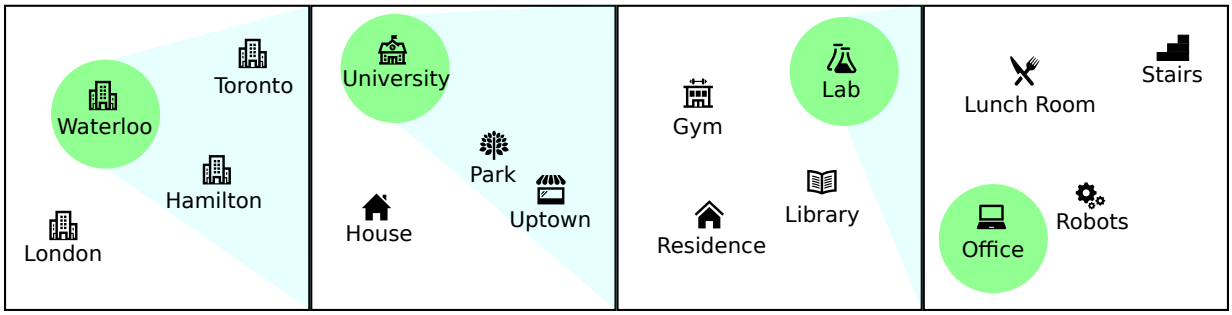


Figure 4.3: **Hierarchical Spatial Structure**: Spatial relations can be naturally organized hierarchically. Items on a higher level map are associated to the spatial arrangement of items in a lower level map.

an apartment in Toronto. These layouts are naturally better remembered as sub-maps, hierarchically arranged at appropriate scales. This is not to say that you could not deduce the relative layout of buildings across cities, but it would require a significant amount of mental effort akin to traversing up and down spatial hierarchies to align very different maps. The less traversal required, the easier the task, but any relationships across the hierarchy can still be retrieved.

With each sub-map only responsible for maintaining locations of several items, retrieval from the sub-map can be highly accurate. A comparison between the standard approach and a hierarchical approach is shown in Figure 4.4. RMSE is computed from the decoded 2D coordinate of the resulting SSP. Item locations are chosen randomly within a 10 by 10 square area. Both cases are implemented using a spiking neural network that retrieves the location of a given item as an SSP. For the standard case, a single memory containing bound pairs of all items and locations is convolved with the inverse of the queued item. For the hierarchical case, sets of items (four for this experiment) are grouped together to form a higher-level semantic pointer. Sets of four of these higher-level semantic pointers are grouped together to form the next level. This process repeats until there is only a single semantic pointer that represents the entire hierarchy.

The method of constructing hierarchical semantic pointers is similar to [35] where a distinction is made between a fixed ID semantic pointer representing a concept (e.g., University of Waterloo campus) and a more flexible semantic pointer representing the contents of that concept (e.g. properties of the buildings on campus, which may be updated over time). This allows the contents of the concept to change without needing to change every reference to the concept. A heteroassociative memory is used to map between an ID semantic pointer and a content semantic pointer. The semantic pointers in the hierarchy

are structured as follows:

$$\mathrm{OBJ}_{k,n} = \sum_{i=1}^{m} \mathrm{ID}_{k+1,n\times m+i} \circledast \mathrm{S}_{k+1,n\times m+i} + \sum_{j=1}^{k-1} L_j \circledast F_{j,k,n} \tag{4.3}$$

where $\mathrm{OBJ}_{k,n}$ is the content semantic pointer for the n-th object at the k-th level in the hierarchy, m is the number of objects grouped together (4 in this experiment), $\mathrm{ID}_{k+1,n\times m+i}$ is the ID semantic pointer for an object one level lower and $\mathrm{S}_{k+1,n\times m+i}$ is the SSP corresponding to its spatial location, L_j is a semantic pointer ID of the j-th level (e.g., city), and $F_{j,k,n}$ is a semantic pointer corresponding to the value of that level for the current object (e.g., Waterloo). The first sum in the equation corresponds to information further down in the hierarchy (more specific), and the second sum corresponds to information higher up (more general).

The hierarchical query is performed as follows:

$$\mathrm{P} \approx H(H(\mathrm{Q}) \circledast \mathrm{L}^{-1}) \circledast \mathrm{Q}^{-1} \tag{4.4}$$

where Q is an ID semantic pointer for the queried object, P is an SSP respresenting the location of that object, L is semantic pointer corresponding to the level in the hierarchy that contains the object (e.g., province, city, neighbourhood, etc), H is a heteroassociative memory that maps from an ID semantic pointer to a content semantic pointer containing more detailed information.

4.2 Mental Imagery

A classic experiment on spatial cognition is mental map traversal [93]. In the standard setting, a human subject is given a map to study with various landmarks at different locations. An example of what this map might look like is shown in Figure 4.5. After the subject has had sufficient time to memorize the layout, the map is taken away and they are asked to mentally visualize the map. They are told to 'zoom in' on a particular landmark so that their mental image does not contain the whole map, but just that part of it in detail. They are then asked to move their mental view to another landmark on the map. From the behavioural data observed, there is a clear linear relationship between the time it takes a subject to imagine the new landmark and the physical distance between the landmarks on the map. The main conclusion of this study is that mental imagery preserves metric spatial information.

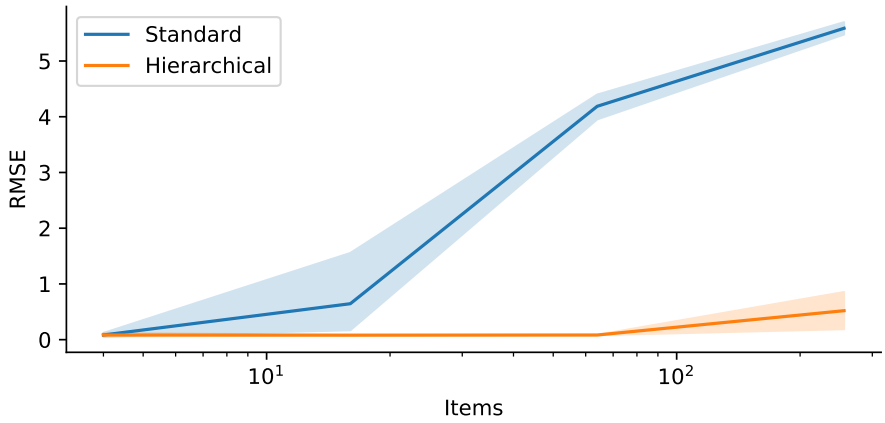


Figure 4.4: **Hierarchical Representation**: Performance of a spiking neural network retrieving the location of a given item from either a standard representation or a hierarchical representation. Locations are chosen randomly from a 10×10 space.

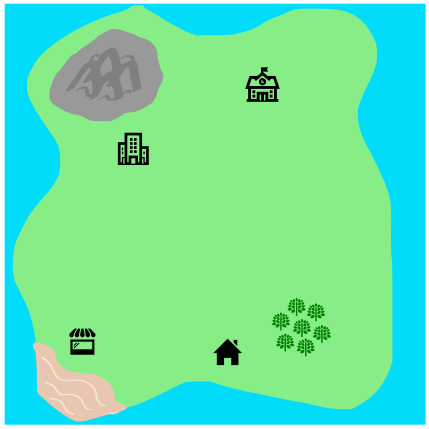


Figure 4.5: **Example Map to Learn**. Simple map with various landmarks at different locations. Each landmark is unique and subjects must learn their positions relative to one another.

4.2.1 Model

A spiking neural model of this task is constructed using Nengo. A diagram of the network structure is shown in Figure 4.6. This network is similar to the SSP image scanning network presented in [106] with a focus on facilitating quantitative comparisons to human data. In this experiment there are 7 distinct objects on each map (locations chosen randomly). Each trial begins with the Memory population of neurons representing a semantic pointer that contains each of these 7 objects bound to the SSP for their locations. The Current Location population is set to be the SSP of one of these objects (chosen randomly), and the Queried Object population is set to be the semantic pointer corresponding to the cued object to move towards. At this point the simulation of the trial begins, and it ends once the Imagined Object population represents a semantic pointer that is sufficiently close to the Queried Object. An estimate of the goal location is retrieved from memory by circularly convolving the contents of the memory with the inverse of the queried object. The direction is then computed from the current location towards the goal location. This direction is represented as an SSP encoding a point on a circle with a fixed radius (0.125 in the data shown). The direction SSP is convolved with the current location SSP, and then connected back to the current location. This causes the representation of the current location to move over time based on the direction. The Current Location population is divided into 6D sub-populations that each represent a different orthogonal hexagonal sub-torus of the SSP space. Encoders of each sub-population are chosen to be only aligned with points on the manifold that correspond to valid SSPs. Each sub-population is recurrently connected to form an attractor network.

In this manner, a potentially noisy state vector will be mapped to the closest point on this attractor surface, effectively cleaning up noise in the SSP. The Imagined Object population contains the output of the Memory being convolved with the inverse of the Current Location, corresponding to a mental image of the objects at the current location during the scanning process. For the simulations, 512D semantic pointers are used and each neural population contains 50 spiking LIF neurons per dimension.

4.2.2 Results

Timing data from the model is compared to timing data measured from humans. The human data used comes from the original paper describing the experiment [93]. When recording reaction times in an experiment with humans or other animals, part of the delay will be due to the mental processes involved in solving the task, but there will also be a delay due to encoding the instructions and producing a response. For example, if someone

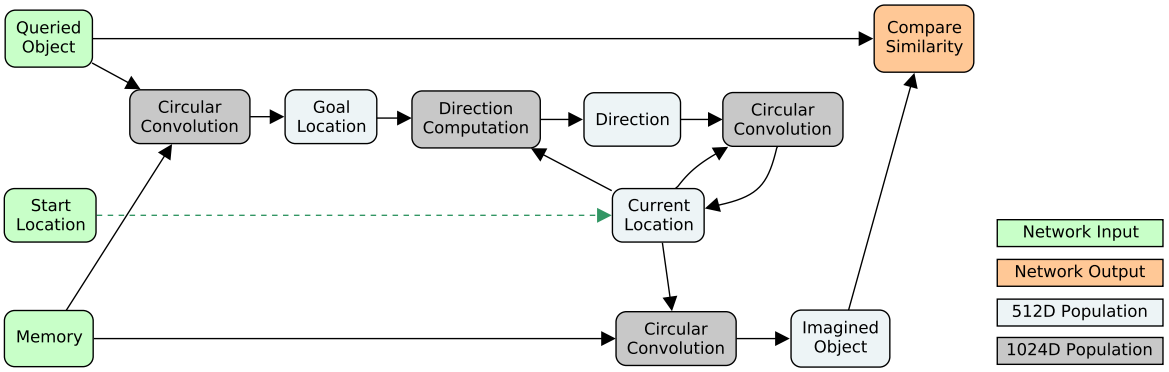


Figure 4.6: **Mental Image Scanning Model**. Diagram of the model that performs the image scanning task. On each trial the queried object as well as a memory for the layout of all objects is provided to the network as input. At the beginning of the trial, the current location is set to be the location of an object that is not the goal. Each time-step the queried object is compared to the readout of the imagined object population. If the dot product is above a threshold (0.45) then the trial ends and the time is recorded.

is told to press a button based on what they see on a screen, their brain needs to first take time to process the visual input, then make a decision based on the task, and then send a motor command to their hand to carry out the actual movement that pushes the button that they chose.

Computational models such as the one presented here often focus on just modelling the mental processes involved in the task itself rather than the other peripherals required to perform the task in real life. When comparing timing data between model and experiment, this difference needs to be taken into account. A common assumption is a fixed time delay added to the response of the model. In this case, a reasonable delay can be computed from the human data itself. Looking at the trend-line in the human data at a distance of 0cm, the corresponding reaction time is 0.96s. Since there is no task related computation required when the distance is 0cm (task is already solved), the only delay will be from beginning the task and issuing a response. This fixed offset of 0.96s is added to the model output to account for this delay.

One free parameter in this model is the mapping of centimeters onto the unit-less values of the binding exponent for the SSP. While this value can be tuned to improve the fit to the human data, it can only provide a linear degree of freedom and the final value used makes a prediction of the scaling used for this task which can be tested in similar experiments. In Figure 4.7, a mapping of 4.2cm to 1 unit in the binding exponent is used to produce the

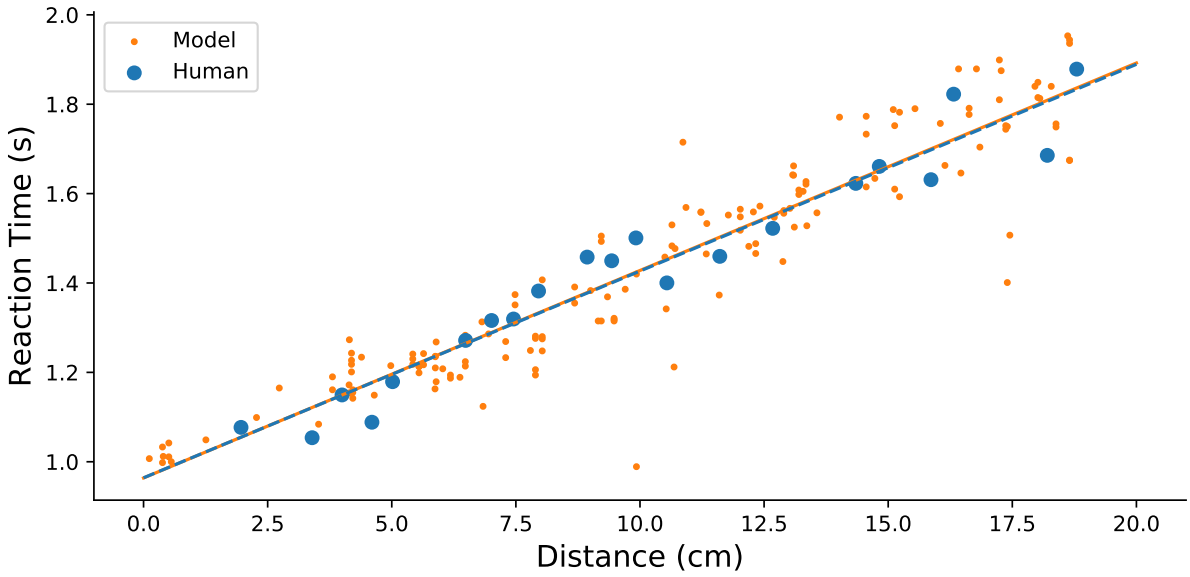


Figure 4.7: **Model and Human Performance**. Simulated time for the model to complete the task compared to reaction time on the task for humans. The human data is from the original experiment [93]. The model data has an r^2 of 0.88 and the human data has an r^2 of 0.97.

model data. The same linear relationship between mental distance and processing time can be observed in both datasets.

4.3 Learning Associations

The task we have chosen to test the ability of SSPs to be used for spatial associations is based on the rodent version of the cued paired associate task [37]. Here a rat is trained to make associations between a cued flavour and the location in a square arena where food pellets of that flavour are located. The food can be in one of many sand wells that cover the arena and the rat must dig in the sand to get to the food. In this manner there is no visual indication of the location of the food, the rat must either try every location or remember the correct location based on the cue. Each unique flavour cue corresponds to one fixed location in the environment.

This experiment typically consists of two phases. The first is a sample phase, where on

each trial only a single sand well is uncovered. When the rat digs in that well it receives food of the associated flavour. After the sample phase, a test phase is conducted where all sand wells are accessible and the rat is cued with a particular flavour at the beginning of the trial. The rat must go to the location associated with that flavour to find food, as all other sand wells will not contain food.

The model we construct to perform this task explores a different way in which the locations of objects can be remembered using SSPs. Instead of a compressed vector representation, the associations can also be stored in the connection weights of a heteroassociative memory. The distinction here is a subtle one, as an associative mapping between two representations is computed in both cases. In the present model, the activity of a population of neurons represents a particular attribute (such as flavour) and the connection weights from this population to another perform a specific associative mapping (e.g., compute the location of any given flavour). For the compressed vector representations used in previous models, circular convolution provides a general associative mapping but requires two inputs. The first is the associations themselves (semantic pointer containing bound pairs of flavour and location) and the second is particular attribute (e.g., flavour) that is being mapped to a different attribute (e.g., location).

4.3.1 Model

A spiking neural model of this task is constructed using Nengo. The structure of the model is shown in Figure 4.8. There are three main branches in this model. The first (green in the diagram) learns to associate flavours with locations, so that given a flavour cue, the model will produce a representation of the location where this flavour was experienced. The second (blue in the diagram) learns to associate locations to flavours, so that a location cue will elicit the flavour that was experienced there. This second branch is not required for the task, but is useful for verifying that the association can be learned both ways. The third (orange in the diagram) is a set of neural populations that implement a simple control system. This controller computes the error between the desired position and the current position in the environment and uses this error to produce a velocity output. The environment block contains a simple simulated world for this task. It consists of a square arena (10×10 units) with randomly placed sand wells and a velocity controlled agent. The only input to the environment is a 2D velocity signal. The outputs are a 2D position and a 7D flavour signal. The flavour sensor is activated when the agent is within a 0.1 radius of a sand well, and the value is injected into the Current Flavour population. The 2D location is converted into a 25D SSP and injected into the Current Location population. Each neural population contains 50 spiking LIF neurons per dimension.

The Task Control node routes various information depending on the phase of the task, shown in Table 4.2. During the sample phase, the working memory populations contain sensory information, and associations are learned between flavours and locations. During the test phase, the working memory populations contain cued information, and the associated output is represented in the recall populations. The Recalled Location is decoded to a 2D position and routed to Desired Position, causing the agent to move towards the sand well where it experienced the cued flavour.

Each working memory population is connected to an association population. This connection is simply a communication channel but with an unsupervised learning rule applied. The learning rule is a vectorized version of the Oja rule [125]. It is implemented in Nengo as the Voja [180] learning rule:

$$\Delta \mathbf{e}_i = \kappa a_i (\mathbf{x} - \mathbf{e}_i) \tag{4.5}$$

where e_i is the encoder of neuron i, a_i is the filtered activity of that neuron, κ is a scalar learning rate, and $\mathbf{x}$ is the input vector encoded by these neurons. The effect is that encoders of the post-population align to more strongly match inputs of the pre-population. The encoders are initially set to random points on a unit hypersphere and the intercepts for these encoders are set to 0.5 (i.e., their firing threshold is reached when the dot product of the input $\mathbf{x}$ and the encoder is greater than 0.5).

The association population is connected to the recall population. The connection weights are initialized randomly, but are modulated by a Prescribed Error Sensitivity (PES) [13] learning rule:

$$\Delta w_{i,j} = \kappa \alpha_j \mathbf{e}_j \cdot \mathbf{E} a_i \tag{4.6}$$

where $\Delta w_{i,j}$ is the change in the connection weight from pre-neuron i to post-neuron j, κ is a scalar learning rate, $\mathbf{e_j}$ is the encoder of a post-neuron and α_j is the scaling on that encoder, a_i is the activity of a pre-neuron, and $\mathbf{E}$ is the vector quantity to minimize.

This connection is learned in a supervised fashion using information from an error population. This error represents the difference between the recall population and the currently experienced sensory information. While the learning rule is active, the weights are updated to minimize this error. In the model these learning rules are active whenever any flavour signal is being sent from the environment.

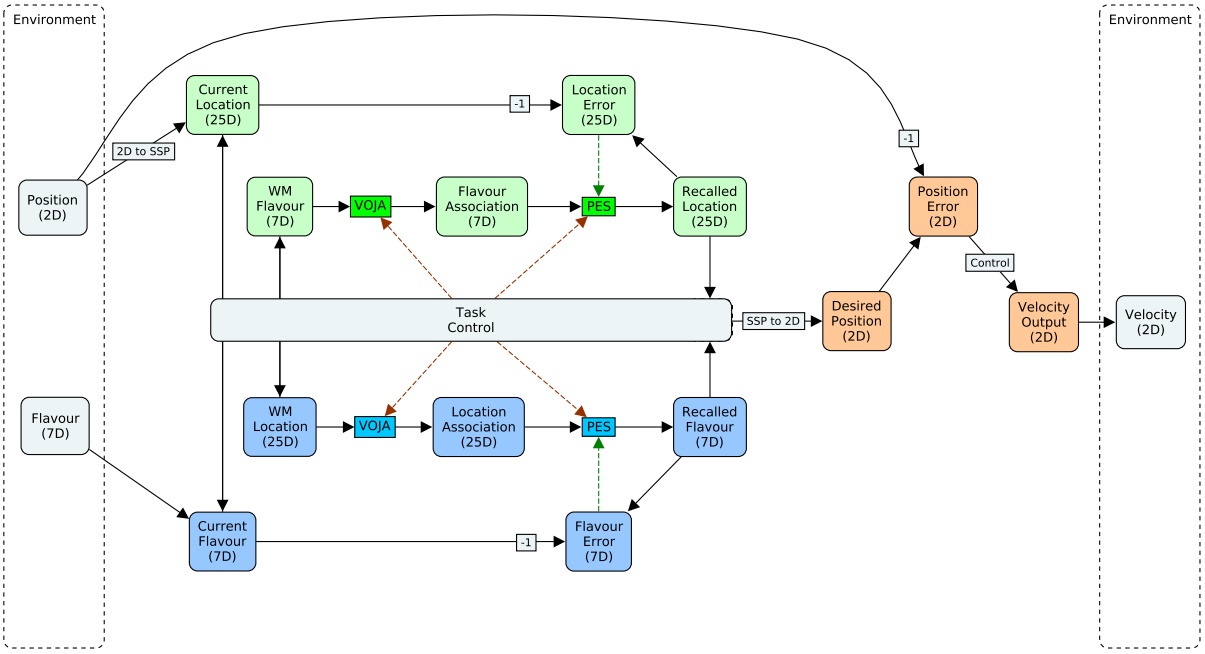


Figure 4.8: **Place-Flavour Association Model**. Diagram of the model that performs the association task. Coloured rounded boxes are populations of neurons. The dimensionality of the signal being represented for each population is shown in the parentheses. The green dotted lines indicate which population provides the error signal for the PES learning rule on a connection. The red dotted lines indicate inhibition of learning rules when learning is turned off. The Environment node represents the simulated world, where the state contains a 2D position signal as well as a 7D flavour signal which is active when the agent is near a flavour well. The position is updated over time based on a 2D velocity signal. The Task Control node routes incoming information to different outputs depending on the phase of the task, shown in Table 4.2.

Output	Input	
	Sample	**Test**
Desired Position	Open Sand Well	Recalled Location
WM Location	Current Location	Recalled Location
WM Flavour	Current Flavour	Cued Flavour
Learning	On	Off

Table 4.2: **Task Control Routing**. Inputs to the Task Control node are routed to different outputs depending on the phase of the task (sample or test). Additionally, learning rules are active during the sample phase but are inhibited during the test phase.

4.3.2 Results

The simulation is conducted in two phases similar to the animal experiment. During the sample phase, each location is visited once in succession, with the agent spending 2 seconds at each location while receiving the corresponding flavour signal. During the test phase, the agent is cued with a random flavour signal which drives the motor system to move the agent to the corresponding location. The model is able to successfully learn this task from a single trial of each place-flavour pair, matching observed behaviour in rodents [37].

When an association is learned between a location and a flavour, a small set of neurons become highly active whenever the agent is in that location and silent elsewhere. This is caused by the Voja learning rule aligning encoders to be more sensitive to the current location. Since location is represented by an SSP, further away locations are nearly orthogonal, causing even less activity outside the place field in these neurons after learning. This alignment will only happen to neurons that are spiking at this location. Neurons that are not driven enough to spike will not undergo any change. This place cell learning effect is shown in Figure 4.9. The encoders of this population (Location Association) are initialized randomly on the unit hypersphere in 25 dimensions. Some of these neurons will be more sensitive to particular regions of space than others, with the shape often being irregular and disjoint (top row of Figure 4.9). Those that are sensitive enough to spike while the agent is experiencing a flavour will become more strongly tuned to the associated location, further increasing their sensitivity to this location and forming a clear place field (bottom right of Figure 4.9).

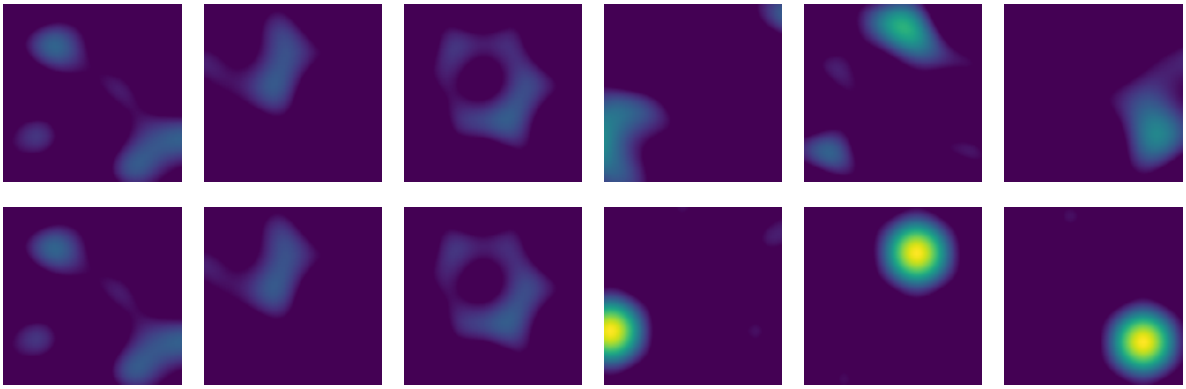


Figure 4.9: **Place Cell Learning**. The heatmaps are generated by taking the dot product of each neuron's encoder with the SSP representation of a grid of locations covering the space. Intensity is the value of the dot product above a threshold of 0.25. Each column is a different neuron from the population (Location Association). The top row is taken from when the network is first initialized. The bottom row is after learning the place-flavour associations.

4.4 Spatially Tuned Neurons

To further demonstrate the biological relevance of SSPs, in this section we show that SSPs are naturally implemented in a spiking neural network using neurons with tuning properties matching those found in the hippocampus and medial entorhinal cortex. We do this by choosing the encoders (i.e., preferred direction vectors) of individual cells to reflect observed tuning of these neurons. The encoder of a neuron determines which part of the state space it is sensitive to. Specifically, the larger the dot product between the encoder and the state space vector, the more the neuron will fire.

An SSP of dimensionality d can be organized into $n = \lfloor \lfloor (d-1)/2 \rfloor /3 \rfloor$ orthogonal hexagonal tori. The orientation of the principal axes and the scale of each torus can be configured using Equation 3.26. One way to think of this surface is as a separate set of n 2D manifolds (hexagonal tori) where a single location has a correspondence to a single point on each manifold. A fixed translation of the true 2-D locations results in a fixed translation on each of the tori, where the distance and direction of this translation is dependent on the orientation and scale on the particular torus. Due to the tight correspondence between points on these different manifolds, the full space they can cover is still a 2-D surface.

Each torus exists in a 6-D space orthogonal to all other tori, so there exists a coordinate frame where the simple concatenation of the coordinates for each of the n points results in the point on this larger 2-D surface. Setting an encoder in this coordinate frame to align with a position on just one torus with zeros everywhere else will make the neuron only be sensitive to the orientation and scale of that torus, as the dot product will be zero with the $n-1$ other points. This will produce grid cell firing patterns. The phase of the grid cell will depend on which point on the surface is chosen. Band cells can be created in a similar manner. Instead of aligning the encoder with a point on the surface, it is aligned with one of the three principal axes. Neurons resembling place cells can be created by aligning the encoder across a multitude of the tori. The firing field will still technically be periodic in this case, but the distance required to travel to repeat would be the lowest common multiple of all of the scales. A neuron can also fire at multiple distinct places by aligning the encoder to be the midpoint of those places.

Firing patterns of individual neurons in an ensemble representing the location of an agent are shown in Figure 4.10. Each image corresponds to a single neuron. In grey are the trajectories of the agent. When the neuron of interest spikes, a red dot is placed on the image at the agent's current location. Another method for generating an SSP and corresponding encoders to obtain similar results is detailed in [41]. In that work the problem is formulated in terms of sets of 3 plane waves in the frequency domain oriented 120 degrees apart. Each sub-ensemble contains plane waves at a different frequency and global rotation. This ends up being a frequency domain formulation of a hexagonal torus.

The orientation of the grid cells or band cells within a particular sub-toroid of the SSP manifold must align with the principal axes that defined that toroid. If each toroid has a unique scale, this means that all neurons that represent a particular scale will have the same set of three orientations, 120 degrees apart. This matches observed data from rodents where grid cells exist at discrete scales and within a particular scale all firing fields are aligned to the same orientations [163].

Place Cell Remapping

Place cells are known to be re-used across different environments and exhibit random place field remapping [121, 52, 80]. Neurons that have a place field in one environment can also have a place field in a different environment, and the relative distances between the place fields of two neurons will differ across environments. There are also cells that will fire in one environment, but not another.

This remapping phenomenon can be modelled using SSPs. One way to represent dif-

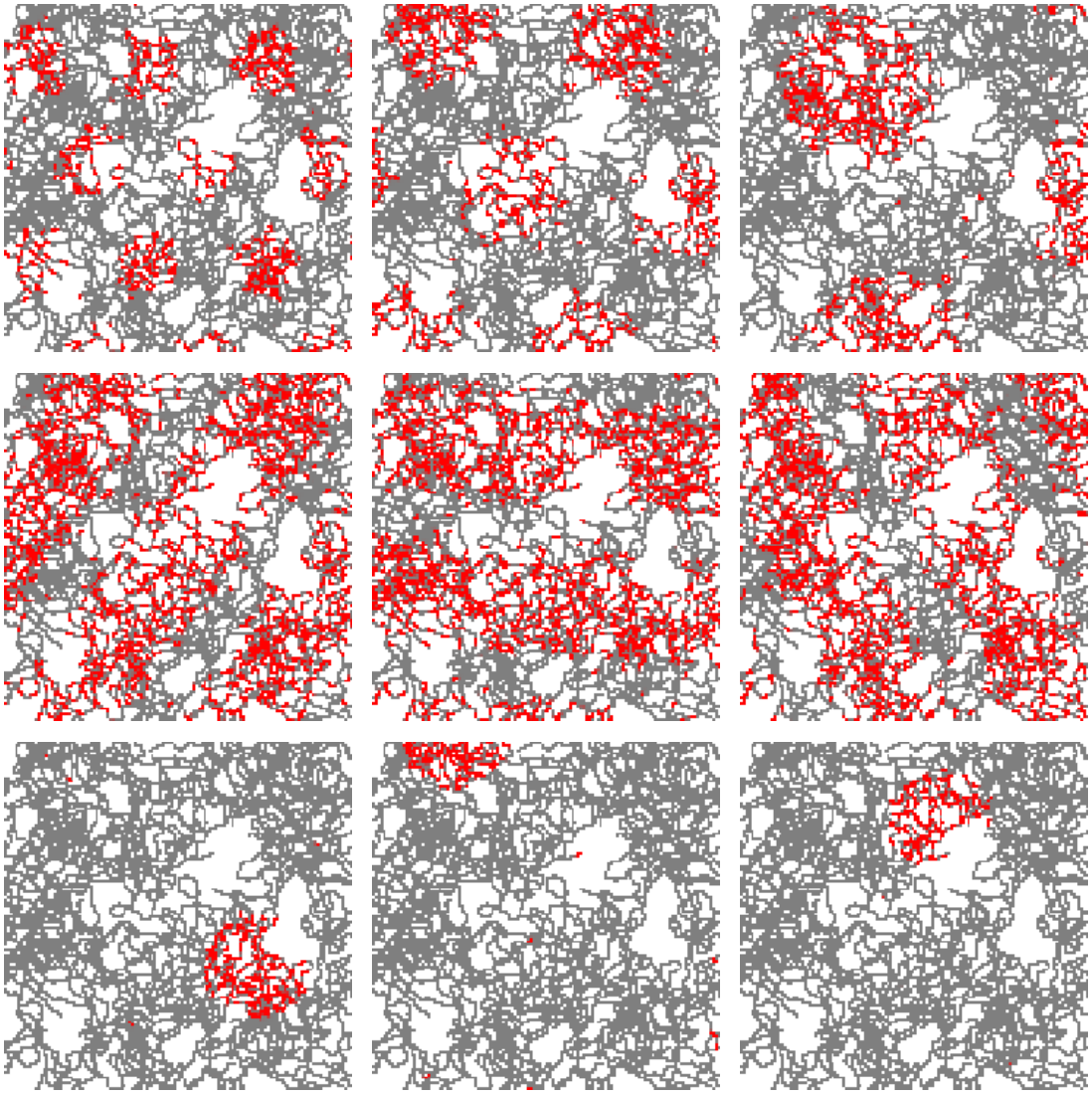


Figure 4.10: **Spatial Tuning of SSP Neurons**. **Top**: Neurons with grid cell-like response. **Middle**: Neurons with band cell-like response. **Bottom**: Neurons with place cell-like response.

ferent environments is with a different set of axis vectors to define the space. The axis vectors could be determined based on features of the environment that distinguish it from other environments. A neuron tuned to a particular direction in a high-dimensional space may be sensitive to one location using one axis vector system while simultaneously being sensitive to a different location under a different system. In this manner no connection weights need to be changed for this neuron to act as a place cell in both environments. This is in line with the neuroscience literature that suggests that place fields are present upon first exposure to a new environment (they appear within a few minutes and are stable for several days), indicating that learning is either extremely rapid or that no connections need to be changed [70].

An experiment is conducted demonstrating place cell remapping using a 512D SSP representation. There are 10 different environments, each with a set of randomly generated axis vectors. A population of 2,560 neurons represent the location of an agent. For each neuron, the preferred direction in the state space (encoder) is set to be the average of 10 different SSPs, one from each environment with locations randomly chosen. For each SSP chosen, there is a chance that the location it represents is outside of the bounds of the environment, to capture the phenomena that some place cells do not show activation in every environment [121]. Every neuron is recorded while the simulated agent explores each environment. The trajectory through the environment is set to be a space filling Hilbert curve so that all regions of the environment are explored in an efficient manner. The spatial tuning of a subset of neurons is shown in Figure 4.11. Despite the encoders being defined to span multiple environments, within an individual environment the neurons maintain a single stable place field. The same neuron can have place fields in the other environments and the remapping is random (i.e., the relative location between two place fields in one environment is not consistent with the relative location between the same two place fields in a different environment).

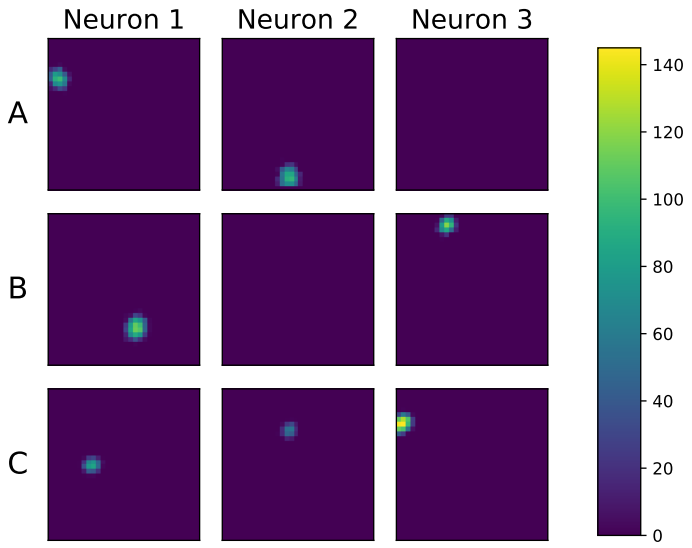


Figure 4.11: **Place Cell Remapping**. Shown are the firing rates (Hz) for three neurons in three different environments. The images are created by dividing the space into 40 evenly sized bins in the x and y directions (total 1600 bins). For every neuron and every bin, the average firing rate is computed for that neuron when the agent's position is within the bin. A separate experiment is run for every environment using the same population of neurons. In every experiment the population of neurons represents an SSP, but each environment is defined by a unique set of axis vectors.

Chapter 5

Navigation System

This chapter introduces components of a navigation system and provides experiments demonstrating the performance of neural networks implementing each component. Each component performs a different task (localization, path integration, memory retrieval, and action selection) and they are combined together to form an integrated system capable of navigating to semantically defined goals.

5.1 Other Spatial Encodings

To demonstrate the effectiveness of the methods used here, we have identified several standard methods to provide benchmark comparisons. Listed below are several alternative encoding methods, chosen based on prevalence of use in machine learning tasks. Experiments training neural networks for spatial tasks in the upcoming sections will be performed with these methods in addition to SSPs.

2D Coordinates

This method simply uses the 2D coordinates directly with no additional encoding mechanism. It provides a benchmark for all other methods to compare against.

One-Hot Encoding

For one-hot encoding, the environment is discretized into a grid. Every element has the value of 0 except for the element which corresponds to the closest grid point of the 2D

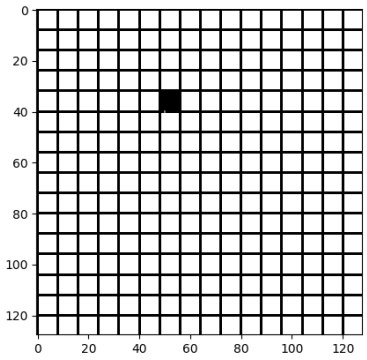


Figure 5.1: **One-Hot Encoding Example**: In this example the 2D space is discretized into 16 bins in the x and y directions, for a total of 256 bins. A location is represented by a 256D vector, where all elements are 0 (depicted as white squares) except for the element that covers the location of interest which is 1 (depicted as a black square).

location; this element has a value of 1. A visualization of this encoding is depicted in Figure 5.1. This encoding provides an extremely sparse representation of the input.

One-hot encoding is used in many applications, including mapping environments [72], DNA motif prediction [29, 85], and text encoding [196, 197].

Tile Coding

Tile Coding [4, 167] is a natural extension to one-hot encoding that helps with generalization. The environment is discretized into a series of overlapping grids, with each grid offset by a different amount. For each grid, the element closest to the encoded location is set to 1, and all other elements are set to 0. An illustrative example is shown in Figure 5.2. Due to the different offsets of the grids, a more precise location can be decoded from the combination of grids than from any individual grid. The offsets also enable any function learned from this representation to be able to generalize to nearby spaces. This is due to the similarity of encoding at nearby locations.

The parameters involved in this encoding are the number of tilings, the number of bins in each tiling, and the amount of space that each tiling spans. The total dimensionality is $n_{tilings} \cdot n_{bins_x} \cdot n_{bins_y}$.

Tile Coding has been used in many applications, most notably in the domain of reinforcement learning [167]. Example applications include hierarchical control of traffic signals

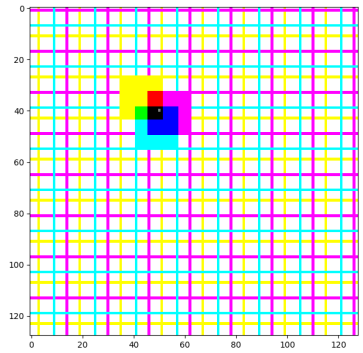


Figure 5.2: **Tile Coding Example**: Three overlapping discretizations of the environment are shown in different colours. The shaded square corresponds to the element of each grid containing the location being represented. The resolution of the encoded position is based on the size of the region where all shaded squares overlap. For 3 tilings with 9 bins in the x and y directions, a 243D vector is used.

[2], camera modelling for 3D sensing [184], robot soccer [166, 89] and multiagent learning [183, 90].

Radial Basis Functions

This encoding mechanism applies a series of 2D functions with varying center locations to the 2D coordinate. Any radial function (i.e., output is based on the distance to the center) can be used, but a Gaussian is the standard choice. The centers are distributed across a 2D domain of interest. Some number, N, of Gaussians are chosen, resulting in a N-D encoding for every point. The standard deviation of each Gaussian can either be fixed for each function or follow a chosen distribution. This can be seen as analogous to a place cell code, where each cell has a preferred location (center) and a receptive field (radius). An illustration of this encoding is shown in Figure 5.3.

There are many different ways to choose the locations of the centers for each Gaussian. One simple method is to evenly tile the space by placing the centers at nodes on a 2D grid. This ensures that the space is fully covered, but with the disadvantage that the number of centers cannot be arbitrary (e.g., to evenly cover a square environment, the number of centers must be selected as a perfect square). Another method that gets around this restriction is to select the centers uniformly at random across the 2D domain of interest. The disadvantage here is that the density of centers tends to not be evenly distributed, with clumps of many centers in some regions and very few in others. A solution to this problem

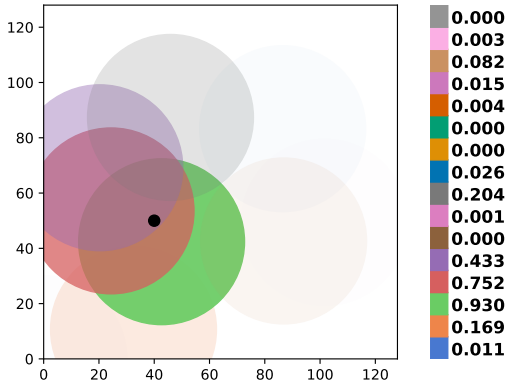


Figure 5.3: **Radial Basis Function Example**: The black dot corresponds to the location being encoded. Each coloured circle represents one dimension of a 16D encoding. The center of the circle is the center of the 2D Gaussian and the radius of the circle is the sigma value for that Gaussian. The opacity corresponds to the activation of each function.

is to choose centers based on locations along a space-filling curve, such as the Hilbert Curve [73]. Since the 2D space can be approximated by a 1D curve, samples can be chosen in 1D and interpreted as a distance along the curve. A 1D region can be subdivided into any number of equal subregions, allowing any number of centers to be chosen. Here the center locations are evenly spaced across a Hilbert curve of order 6 with small amounts of Gaussian noise added to each location to provide some randomness. A comparison of the locations of 2D points generated from a uniform distribution and a Hilbert curve is shown in Figure 5.4.

RBF encoding is used in many applications, including learning spatial relationships between objects [150], facial recognition [50, 9], and learning actions from motor primitives [173].

Legendre Polynomials

This method is inspired by Functional-Link Networks [133] where the dimensionality of the input is expanded by mapping it through a set of functions before being applied to a neural network. One commonly used class of functions in these networks is the Legendre polynomials [153]. These are a set of orthogonal polynomials, the first few orders of which are shown in Figure 5.5. The constant term (order 0) is not used for the encoding as it is invariant to the input, and bias parameters are already present in the neural networks used.

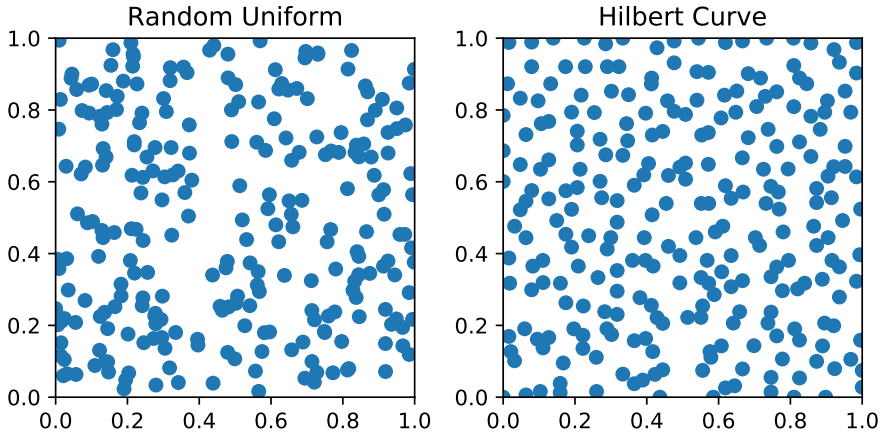


Figure 5.4: **RBF Center Distributions**

For a location input, the x and y coordinates are considered features and are each mapped through a set of Legendre polynomials up to some maximum order. The input coordinates are scaled by the bounds of the space to be between -1 and 1, ensuring the output is also within that range. The outputs from each polynomial are concatenated together to form the encoding.

Networks using a Legendre polynomial encoding have been used for structural system identification [153], chaotic time series prediction [182], and encoding sliding windows of time in a recurrent neural network [179].

Learned Encoding

This method learns an N dimensional encoding using backpropagation. An additional fully connected layer is added to the beginning of the network. This additional layer is trained simultaneously with the rest of the network on whatever task is being performed. If encoding is required for multiple inputs to the network, the weights are shared across all of those inputs, to ensure a consistent encoding is used.

Random Linear Transformation

In addition to these common encoding mechanisms, two random methods are used as well, to construct a baseline. For this method the 2D coordinate is multiplied by a fixed

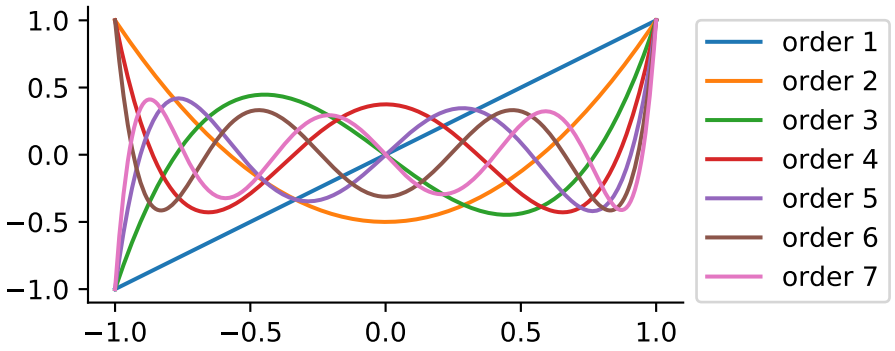


Figure 5.5: **Legendre Polynomials**

randomly generated $2 \times N$ matrix to produce an N dimensional vector. The purpose of this encoding is to provide a comparison where the information is essentially unchanged but the dimensionality matches the other encodings.

Random Mapping

This method is meant to give a baseline level of performance as well as measure the tendency of the training method to overfit. The original 2D coordinate is randomly mapped to an N dimensional vector. This mapping is done by treating the bits in the floating point representation of the 2D coordinate as a seed to a random number generator that will produce an N dimensional unit vector. This process effectively treats every 2D point as unique with no relation between the encodings of nearby points. This violates the smoothness assumption made by most machine learning algorithms so any function learned using this representation is not expected to generalize. It will however give a reasonable worst case to anchor the results of the other encoding mechanisms against.

5.1.1 Decoding

It is useful to be able to reconstruct the 2D coordinate given an encoding. One efficient and general way to approximately reconstruct the encoded position is to use a lookup table as the decoder. Given the assumptions of a) smoothness between nearby representations and b) uniqueness of encoding (i.e., different coordinates should map to different representations), a lookup table for decoding can be constructed. For any encoding, this table is generated by computing that encoding for every 2D coordinate in a dense grid over a

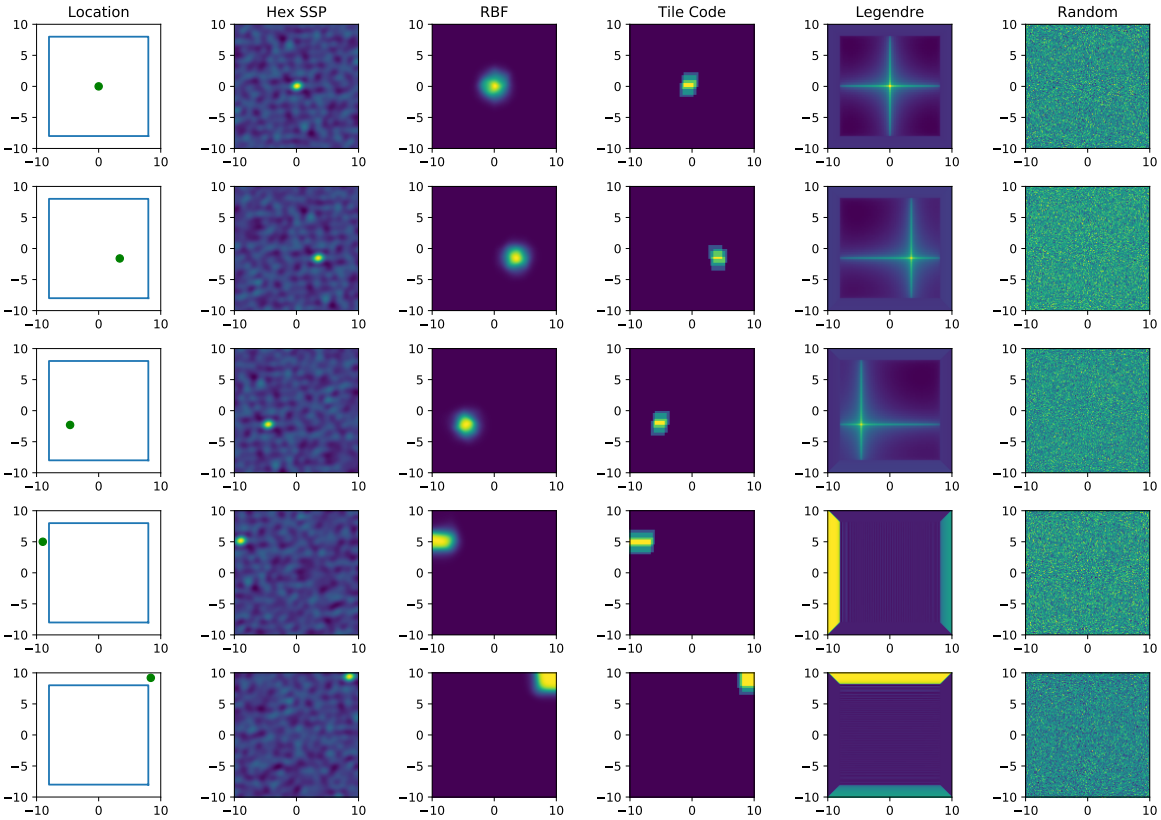


Figure 5.6: **Similarity of Encoding**. Each row depicts a different location being encoded. The location is indicated by the green dot in the far left column. The blue square indicates the bounds of the encoding if one is required. Each other column displays the similarity heatmaps for a different encoding method on each location.

region of interest. The encoding vectors produced are then normalized so that computing the dot product is equivalent to the cosine similarity. To approximately decode any 2D coordinate from a given representation vector, a tensor dot product is computed between the representation vector and the table. The output is an $N \times N$ matrix, where the index of maximum value can be used as an index into the coordinate table to find the approximate 2D coordinate.

One advantage of SSPs over the majority of other methods is no explicit domain limit needs to be specified. Any real number may be encoded as an SSP. Other encodings, such as tile-coding, can only represent values that fall within fixed bins, with the bin size and

location determined by specified boundaries. Any value outside the boundary can either not be represented, or is instead represented by the closest tile on the boundary, meaning everything in a particular direction outside of the boundary is mapped to the exact same representation. This effect is depicted in Figure 5.6, where the similarity of an encoded location is shown compared to all other locations. Some locations depicted are outside of the defined bounds of the encoding.

The following experiment tests the accuracy of decoding from each representation based on the size of the domain and the level of noise applied. For methods that require a boundary, the boundary is set to the edges of the space, so that all coordinates sampled will be within the boundary. The SSP representation used remains the same in all conditions. The only difference is that the decoding method only considers values within the domain. The ground truth decoding in all cases is conducted through a nearest neighbour match of a dense tiling of the space (0.0625 units apart). Input coordinates are sampled from this tiling. If the noisy coordinate is still a closest match with the input coordinate, the point is considered to be decoded correctly. If it matches with a different unit on the grid, the error is based on the distance of the best match to the input coordinate. Results are shown in Figure 5.7. The increase in error for one-hot encoding and tile-coding for larger spaces is due to the fact that their resolution decreases as the space becomes larger. Due to the fixed dimensionality constraint of the experiment, each tile in a larger space occupies a larger area. A similar effect occurs for the RBF encoding. As the space gets larger, the widths of the Gaussians must get larger to compensate, effectively degrading the precision of the representation. If the widths do not get larger, the accuracy degrades even faster with the size of the space, as regions of the space can no longer be covered with a fixed dimensionality. With very wide Gaussians the slope is shallow, so small perturbations due to noise will cause a larger change in the position estimate. The discrete methods (one-hot and tile-code) are highly resilient to noise, but offer only a finite level of precision, based on the dimensionality of the encoding and the size of the space. The continuous methods are highly precise when there is no noise, but performance degrades more quickly as noise is added. The RBF and Legendre methods become more sensitive to noise as the domain increases, while the error in the SSP method remains constant through this domain increase. For the smallest spaces RBF and Legendre offer the best performance, but SSP quickly dominates when scaling is taken into account.

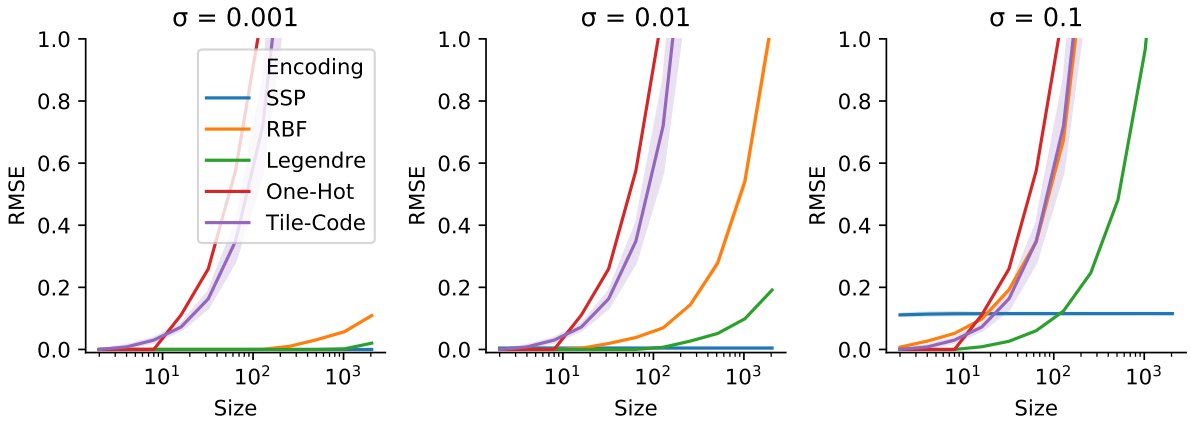


Figure 5.7: **Noise Resilience in Decoding**. Decoding error as a function of noise level and size of the environment. Normally distributed noise is applied to each encoding with a mean of zero and a variance of σ. RMSE is computed on the difference of the original coordinates and the decoding from the noisy representation. The size is the side length of a square environment. Continuous encodings (SSP, RBF, and Legendre) are 256D. Discrete encodings (One Hot and Tile Code) are 4096D.

5.2 Navigation

This experiment involves generating a policy for use in a navigation task and is an extension of the work published in [91]. The objective is to train a network to generate the correct action to take for any state in an environment, where the correct action is defined as the direction the agent should move in order to get to a goal location the fastest. The environments are defined over 2D space and contain regions that are either free space or obstacles. The agent can only move through free space and collides with all obstacles.

The network receives three inputs; the encoding of the current location, the encoding of the goal location, and a context vector corresponding to the current environment. The output of the network is a 2D vector corresponding to the direction to move from the current location to get to the goal. The network is a feed-forward neural network trained with back-propagation with the Adam optimizer. A diagram of the network architecture is shown in Figure 5.8. Note that since two locations are being encoded, when the 'learned encoding' method is used the weights and biases are shared across the two encoded locations. This ensures that a single encoding function is learned.

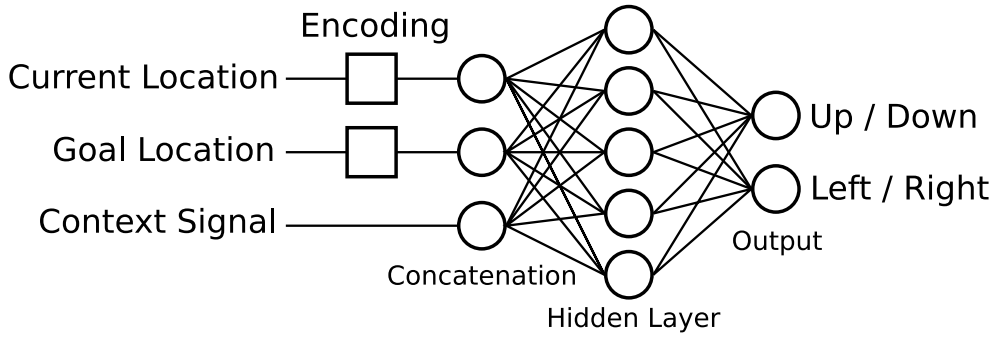


Figure 5.8: **Navigation Task Diagram**. Neural network that performs the navigation task. Current Location and Goal Location are each 2D inputs that are mapped through an encoding function. The encoded inputs are concatenated with a context signal unique to each environment, forming the input of a multi-layer feed-forward neural network. After one or more hidden layers, a 2D output layer represents the predicted direction to move to reach the goal from the current location.

5.2.1 Optimal Policy

In order to facilitate supervised training, an optimal policy needs to be computed to supply the training signal. The policy is defined as an action to take for every point in the space. The space is divided into a 64×64 grid so that a tabular policy may be used. Each cell in the table contains a 2D value corresponding to which direction the agent should move from that location to get to the goal. The optimal policy for a given goal location is computed using a modified version of Dijkstra's [39] search algorithm and is described in Algorithm 1.

This algorithm produces reasonable looking policies, but still can exhibit 'wall-hugging' behaviour that would not be ideal for an agent interacting with the real world. To remedy this problem, an energy function is superimposed on the policy to bias the solution to keep some distance from the walls. This is accomplished by adding a 2D Gaussian to each node with a wall, where the height of the Gaussian corresponds to an outward direction vector (the direction is always outward from the center, and the magnitude is proportional to the value of the Gaussian at each point). This gives an additional flow field that is summed with the policy flow field to create a policy biased to avoid walls. A visual depiction is shown in Figure 5.9.

The policy is conditional on the goal, so a different flow field is computed for every goal location in the dataset.

Algorithm 1 Optimal Policy

Let *nodes* be a discrete set of locations tiling the space
Let *expanded* be an initially empty list of nodes sorted by distance
for each *node* in *nodes* **do**
    node-dist $= \infty$
    node-dir $= (0, 0)$
end for
Start with the node for the goal, set its distance to 0 and place it on a *queue*
while *queue* is not empty **do**
    Pop the next element in *queue* as *current-node*
    Add *current-node* to *expanded*
    for each *adj-node* adjacent to *current-node* **do**
        if *adj-node* is not a wall and *adj-node-dist* $>$ *current-node-dist* $+ 1$ **then**
            Add *adj-node* to *queue*
            for each *exp-node* in *expanded* **do**
                if There exists a non-occluded straight line from *adj-node* to *exp-node* **then**
                    Set *adj-node-dir* to be the direction of the line
                    Set *adj-node-dist* to be the length of the line plus *exp-node-dist*
                    break
                end if
            end for
        end if
    end for
end while

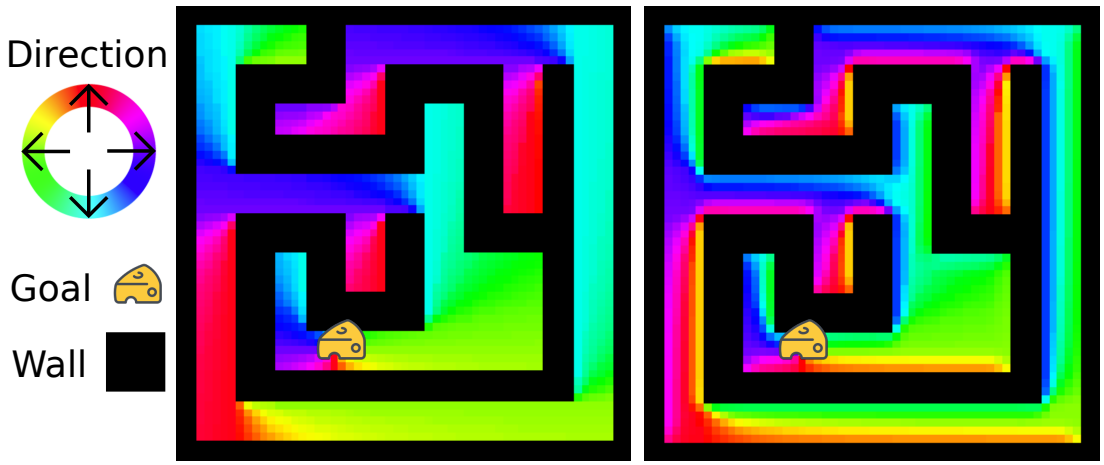


Figure 5.9: **Ground Truth Optimal Policy**. The colour of each pixel corresponds to the optimal direction to move from that location to get to the goal shown. **Left**: Result of the shortest path algorithm. **Right**: Shortest path algorithm with the additional objective of avoiding walls.

5.2.2 Dataset Generation

A series of environments are randomly generated, across two different styles. The first style is labelled 'blocks' and consists of a mainly open world with various clusters of obstacles placed within. The second style is labelled 'maze' and consists of narrow corridors that form a maze-like structure (see Figure 5.12 for an example of each environment style). In both cases there is always an unimpeded path from any free space to any other free space. If during the generation of the environment any regions of isolated space form, the smallest region is fully filled in with obstacles.

The environments are defined over a coarse occupancy grid of 13×13 cells. Although the obstacles are only defined at integer locations, the agent as well as the start and goal locations can occupy any continuous free spatial location within the environment.

A total of 100 unique environments are generated. Due to the computational complexity of generating an optimal action for a given location, the policies are computed across a discretization of the environment, allowing previous results to speed up computation of future results. For each environment, 100 goal locations are sampled from the free space locations on a 64 by 64 grid. For each of these goals, the optimal policy is computed for every free space in the grid. The first 75 goals in each environment are used in the training set, and the final 25 goals only appear in the test set.

5.2.3 Performance

One way to determine how close the learned policy is to the desired optimal policy is to use the Root Mean Squared Error (RMSE) of the output. Since the policy output is a 2D direction vector there are a variety of ways to compute this error. One way would be simply compute RMSE on the output and target vectors directly. An issue with this method is that the magnitude of the vector will play a large role in the error. For this task the direction of output is all that matters for determining success. One way to remove any error caused by magnitude discrepencies is to normalize the output before computing the RMSE. This gives a more relevant result, but there is still room for improvement. The 2D output can instead be converted into the 1D angle in radians that it represents. The error can now be defined as the angular distance between the output and the target. One point on a circle can be reached from another by either moving clockwise or counterclockwise, and the distance in general is different depending on which way is chosen. For this metric we are always interested in the smaller of the two angles. A general way to obtain the smallest angular distance between two angles is shown in Equation 5.1.

$$\theta_e = \min(|\theta_p - \theta_t - 2\pi|, |\theta_p - \theta_t|, |\theta_p - \theta_t + 2\pi|) \tag{5.1}$$

Where θ_e is the error, θ_p is the predicted angle, and θ_t is the target angle. Both the predicted and target angle are constrained to be between $-\pi$ and π. This equation enforces the error to be within the same range. This equation can be vectorized in Python to compute the error for all data points efficiently. The Angular RMSE metric used in experiments follows this error calculation and the result is reported in radians.

5.2.4 Hyperparameter Analysis

One downside to using deep learning approaches to train a network is that there are often many hyperparameters that influence the final performance. Although the focus of this work is not on the training of these networks, care must be taken so that the hyperparameters used produce reasonable results. Therefore, a hyperparameter analysis is conducted on four parameters: the number of hidden layers, the number of units in each layer, the batch size, and the learning rate. Results are shown in Figure 5.10. Overall the best performance was observed for a learning rate of 0.001. The batch size had very little effect on the final result. Performance improved with the number of hidden units and hidden layers, with the best results obtained for the largest values tested.

Increasing the hidden size and number of layers further will likely continue to improve performance as long as more regularization is applied to account for increased model complexity (e.g., more training samples). Since the goal is to compare relative performance of different methods rather than absolute performance, model complexity and training set size are limited. This limit allows models to be trained faster and more models to be run in parallel, resulting in more experiments that can be performed in a given time frame. All encoding methods showed a similar dependence on these hyperparameters, so a comparison can be performed at any fixed hyperparameter setting.

For the SSP encoding methods the scaling from environment coordinates to the exponent of the axis vectors can affect performance. Similarly, the width of the Gaussian functions used in the RBF encoding are important. The other encodings used can be fully defined based on the size of the environment and the dimensionality of the encoding. A parameter sweep from 0.1 to 1.5 is conducted across these two parameters. The minimum loss is observed for an SSP scaling of 0.5 and for a σ of 0.75 for the Gaussian basis. These values will be the defaults for all experiments.

5.2.5 Results

Experiments were conducted across a variety of spatial encodings. Three SSP variants are used; the standard formulation with two randomly chosen axis vectors, the hexagonal formulation with three randomly chosen axis vectors, and a version using a single axis vector encoding x and y separately and concatenating the result together (as opposed to circularly convolving them together). Results are shown in Figure 5.11. This plot and all other bar charts in this thesis will use the standard of displaying the 95% confidence interval of the mean, and all t-tests for statistical significance will use the standard of four stars for $p < 0.0001$, three stars for $p < 0.001$ two stars for $p < 0.01$, and one star for $p < 0.05$. Visualization of the learned policy compared to the optimal policy is shown in Figure 5.12. In each case the dimensionality of the encoding is set to 256 (except for the 2D encoding methods where it remains at 2) and the network size and training hyperparameters are held constant across runs. The context signal is set to a 256D random unit vector unique to each maze layout. When concatenated with the 256D encoding of the current location and 256D encoding of the goal location, the total size of the input vector is 768 (or 260 for the 2D encoding methods). Two hidden layers with 2048 neurons each are used. A ReLU activation function is used before each hidden layer. The output is a linear readout from the last hidden layer.

The network is implemented in PyTorch [135]. It is trained using the Adam optimizer [86] with a learning rate of 0.001 for 50 epochs on a training dataset of 100,000 data

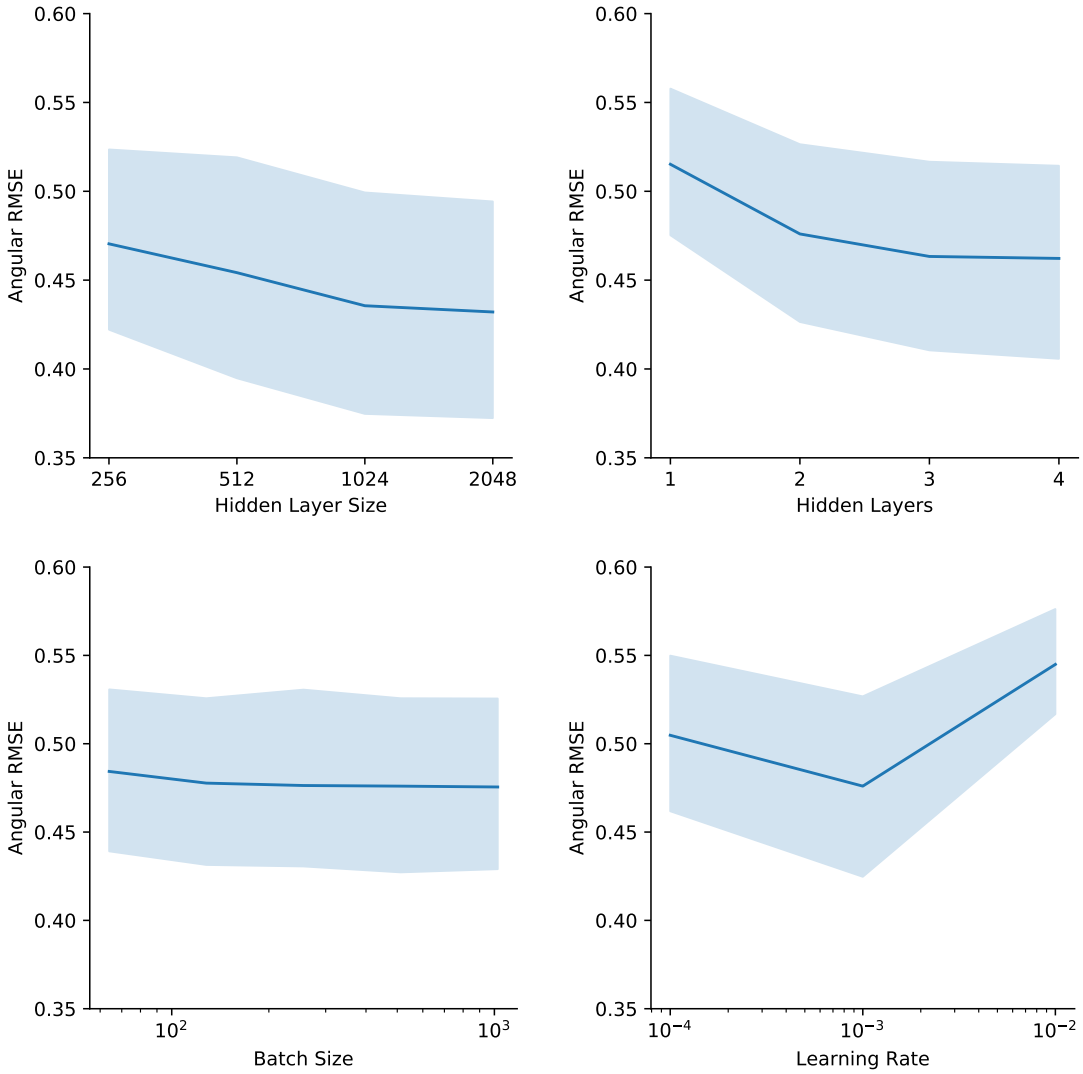


Figure 5.10: **Network Hyperparameter Analysis**. Results shown are averaged across experiments with SSP, Hex-SSP, RBF, Tile-Code, and One-Hot encodings. In each case the other hyperparameters are held constant while the hyperparameter of interest is varied. The default values used are a two layer network, a hidden layer size of 256, a batch size of 512, and a learning rate of 0.001.

points. Initial experiments were run for 500 epochs, but performance on a validation dataset plateaued by 50 epochs, indicating that any additional training would only result in overfitting. All results shown are from a mutually exclusive test dataset of 100,000 data points. The test dataset is set up in such a way that any tuple of (start location, goal location, context) that appears in the training set cannot appear in the test set.

The SSP methods outperform all other methods, with the hexagonal formulation consistently achieving the best score. The independent SSP method, which encodes x and y as separate 1D SSPs does not perform as well as the other two, indicating that combining the coordinates into a single encoding defined across the 2D surface is beneficial for this task. Despite worse performance compared to the other SSP methods, the independent encoding is still competitive with the other top encodings.

For a policy trained on 25 unique mazes, Hex SSP produced an RMSE of 0.4298 (SD 0.0048), SSP 0.4365 (SD 0.0035), Ind SSP 0.4388 (SD 0.0038), and the next best encoding is RBF with an RMSE of 0.4400 (SD 0.0037).

5.2.6 Capacity

The number of unique environments that a single model is able to successfully learn to navigate is a measure of its capacity.

To test the capacity afforded by different encodings, a model is trained on a varying number of environments to see at which point the performance of the model breaks down. Results are shown in Figure 5.13.

The rate at which the task becomes more difficult as the number of environments to learn increases appears to be similar for all encoding methods. This indicates that none of the encoding methods inherently provide an advantage over the others in terms of learned policy capacity. Any limitation will be dependent on the neural network used, the number of samples in the dataset, and the environment context vector (held constant in all experiments).

Interestingly, even the random encoding method showed some dependence on the number of environments to learn. Even though the train and test sets are set up such that no tuple of (start, goal, context) is allowed to appear in both datasets, it is possible that due to random chance some samples partially overlap. For example, consider the training sample (start = bottom left, goal = anywhere, context = environment A, target = up and to the right) and the test sample (start = bottom left, goal = anywhere, context = environment B, target = up and to the right). Although the random encoding has no way

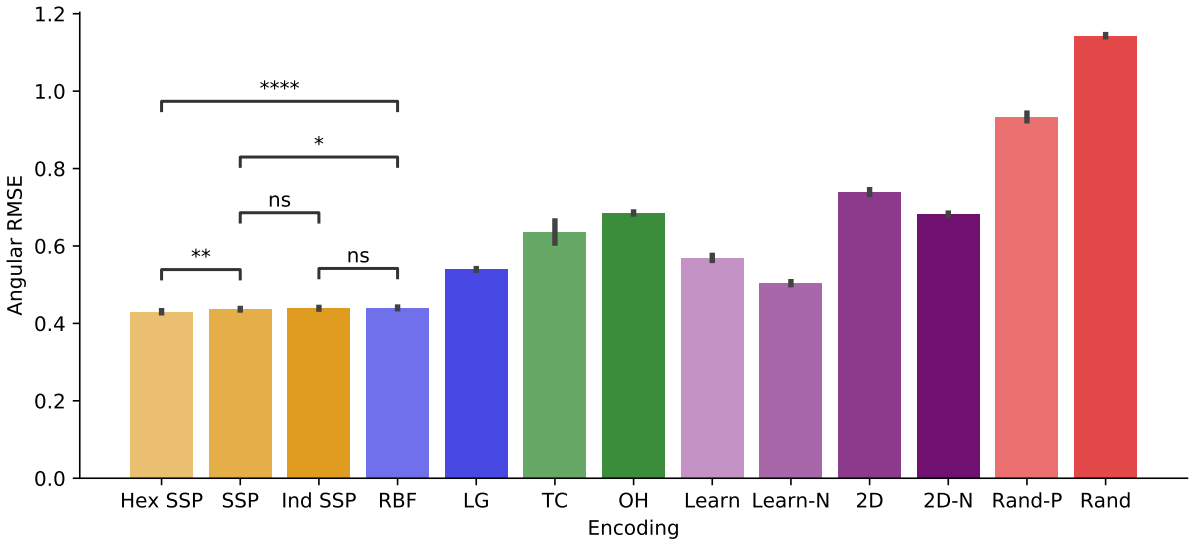


Figure 5.11: **Learned Policy Performance**. Test set performance of models trained with different encoding methods to compute a navigational goal-directed policy. Descriptions of the encoding methods used are as follows: **Hex SSP**: Encoded using random axis vectors for a hexagonal coordinate system. **SSP**: Encoded using random axis vectors for a standard 2D coordinate system. **Ind SSP**: x and y represented as separate 1D SSPs and concatenated together. **RBF**: Gaussian radial basis functions. **LG**: Legendre polynomials. **TC**: Tile-Coding. **OH**: One-Hot encoding. **Learn**: Learned encoding. **Learn-N**: Learned encoding with inputs normalized. **2D**: Directly using 2D inputs. **2D-N**: Directly using normalized 2D inputs. **Rand-P**: Fixed linear transformation to project 2D input into the encoding dimension. **Rand**: Random mapping of 2D input into the encoding dimension.

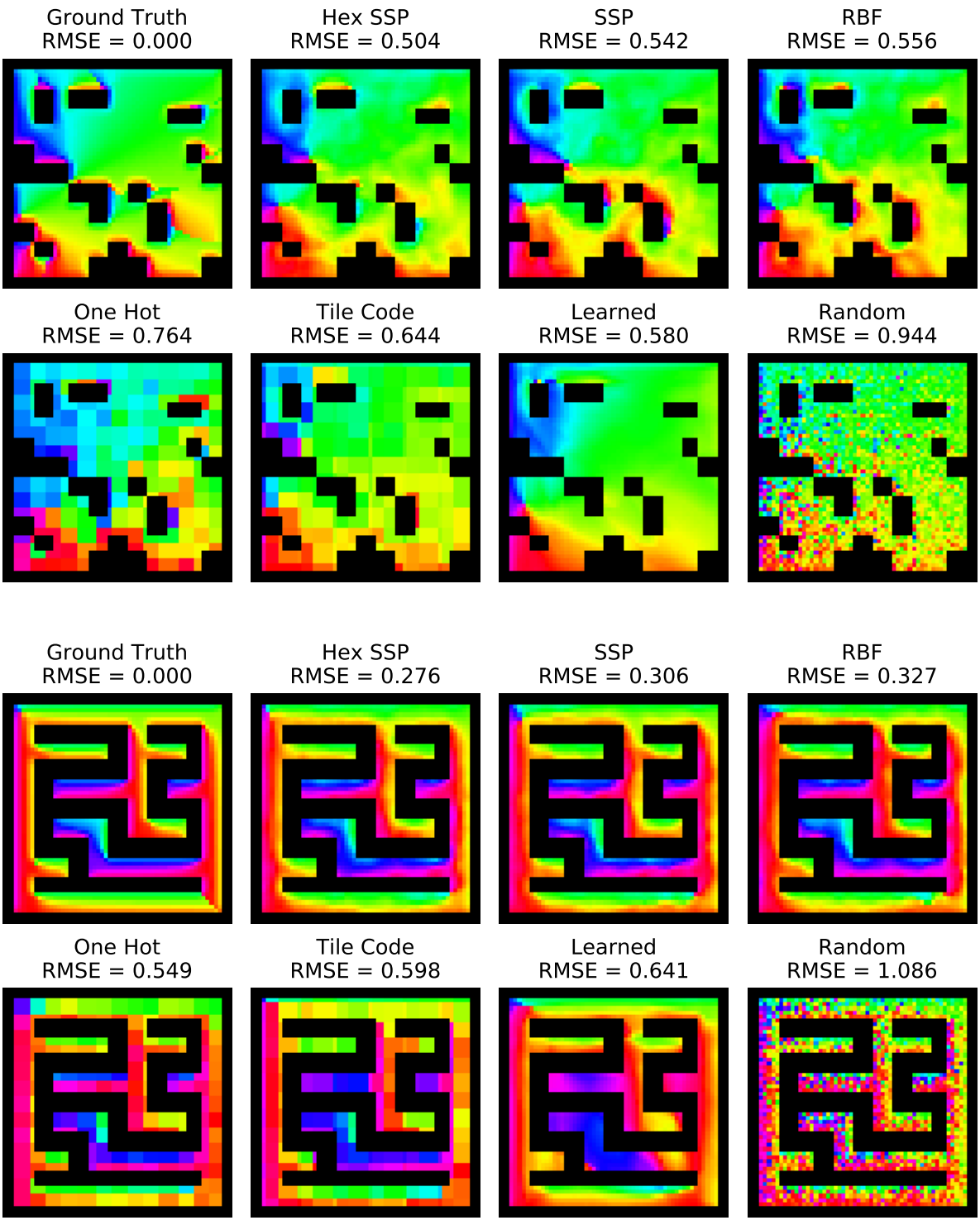


Figure 5.12: **Learned Policy Visualization**. Each pixel is coloured based on the output of the policy for that location as input. For each environment the goal is set to be the same location.

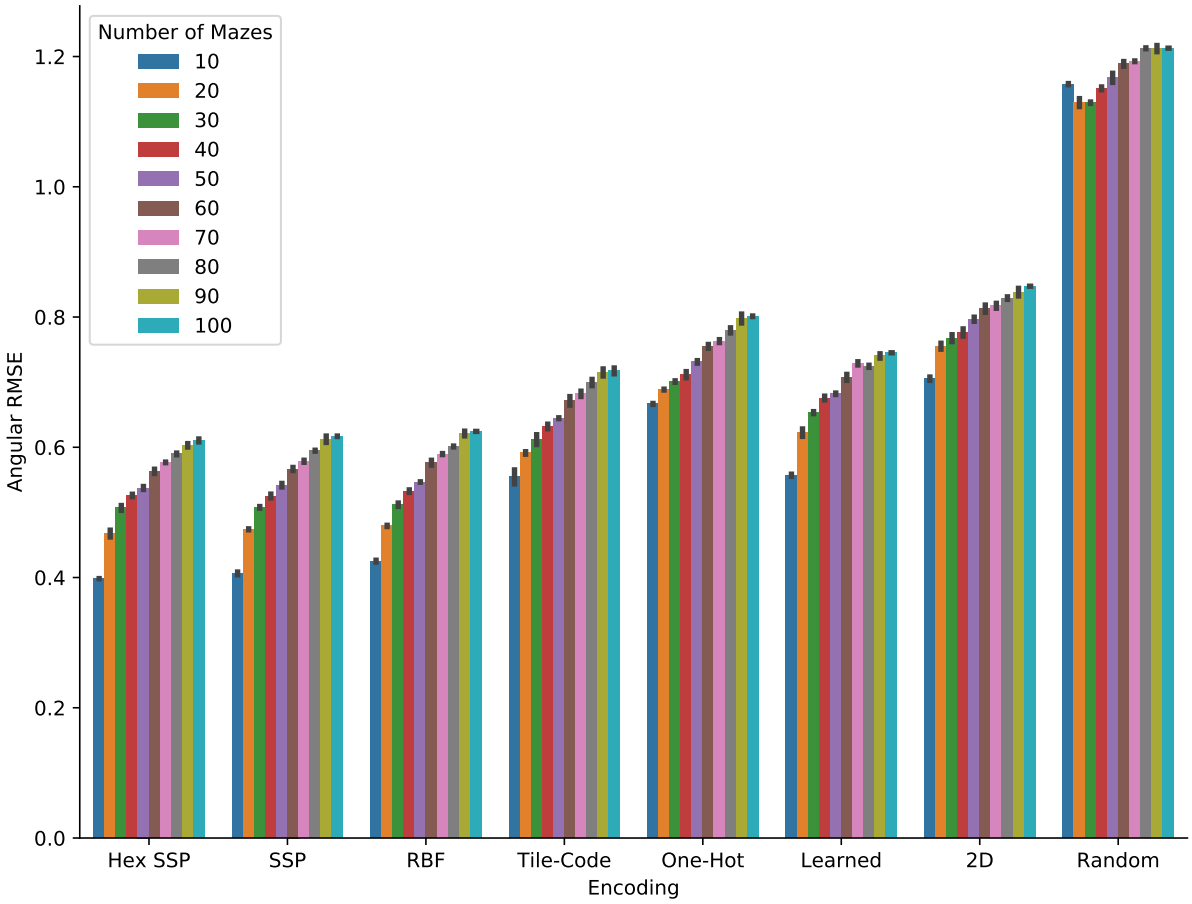


Figure 5.13: **Learned Policy Capacity**. Performance of a learned navigation policy based on spatial encoding and the number of different environments the network must learn. Each encoding is 256D (with the exception of the 2D coordinates directly case). The neural network is trained with Adam (learning rate of 0.001) for 50 epochs on a dataset of 100,000 training samples. There is a single layer with 256 hidden units and ReLU activations between the input layer and hidden layer. Each experiment was performed 5 times with different random number seeds.

of understanding where the start and goal are relative to each other, during training it can learn to associate a particular location (e.g. bottom left) with an output direction that is on average correct with that location (e.g. move up and to the right) regardless of where the goal is located or what the layout of the environment is. While the task is extremely difficult, the network is still able to learn *something*. This 'best output on average' becomes more difficult as more environments are added because it is less likely that the samples partially overlap when the possible space to draw samples becomes larger and only a fixed number of samples are drawn.

5.2.7 Generalizability

Environment Type

With a larger network and enough training data, it is possible for the performance of the 'learned' encoding method to match that of the SSP encoding. The question arises as to whether this learned encoding generalizes to other environments as well as the fixed SSP encoding. To test this, an encoding is first learned by training on one type of environment. After sufficient performance is achieved, the weights and biases of the learned encoding layer are saved and then loaded into a new model to be trained on a different environment. The parameters of the encoding layer are frozen (i.e. not allowed to be updated with gradient descent), while the hidden layer can still be learned. The new model now has the exact same amount of trainable parameters as the SSP model. There are two training conditions for the encoding parameters, one trained on 'blocks' environments and one trained on 'maze' environments. The encoding trained on each of those conditions is then used to train a policy on a different set of 'blocks' or 'maze' environments. Results are shown in Table 5.1, along with the SSP method for comparison.

In general the scores are better on the maze environments than the blocks. This may seem counter-intuitive as the maze environment seems more difficult to solve than the more open blocks environment. The reason for this is that the RMSE is measured based on the deviation from the optimal policy and open environments permit more subtle changes in what the best direction to move is based on the goal. For a corridor in a maze, there are essentially only two directions to move, so learning to choose correctly based on the goal becomes much easier. For the blocks environment, a subtle shift in the goal location could make the optimal path take a different direction around an obstacle than it did before. A policy that goes the longer way to get to the goal will still reach the goal, but the RMSE measured will be higher.

Training	Testing	RMSE
Blocks	Blocks	0.472 (0.022 SD)
Blocks	Maze	0.531 (0.014 SD)
Maze	Blocks	0.461 (0.021 SD)
Maze	Maze	0.443 (0.012 SD)
SSP	Blocks	0.407 (0.004 SD)
SSP	Maze	0.360 (0.006 SD)

Table 5.1: **Generalizability Across Environment Layouts**. Encoding performance based on the style of the environment. Encoding parameters are learned on the training environment and hidden layer parameters are learned on the testing environment. Results shown are for a policy that operates on 10 unique environments.

The encoding learned from training on blocks environments performed best when used on blocks environments, and the encoding learned from training on maze environments performed best when used on maze environments. This is not too surprising, but it illustrates the fact that this learned encoding is dependent on the distribution of data it is trained on. There is no incentive for the encoding to be useful in a larger class of problems; any ability to generalize will have to come from similarities in the data across tasks or specific regularization techniques applied. SSPs on the other hand are designed to represent spatial information with no specific task in mind. In this sense they are not biased by a particular dataset or task, allowing the encoding to be used more generally. For this particular case, SSPs produced 11.7 % less error on blocks environments and 18.7 % less error on maze environments compared to the learned encoding method.

Encoding Bounds

A natural advantage of using a SSP encoding of space is there are no explicitly defined limits of the space that need be represented. When using a method such as Radial Basis Functions or Tile-Coding, boundaries must be chosen for the space to be represented. Any region outside of the boundary cannot be encoded effectively without changing the representation to include additional basis functions or tiles. In contrast, SSPs can represent any location and the dimensionality of the SSP does not have to increase to incorporate more space.

To illustrate this point, experiments were conducted where the encoding is designed for a particular region of space, but the network learns from a larger region that goes outside of these bounds. In this experiment the encodings were designed for a space of half the width

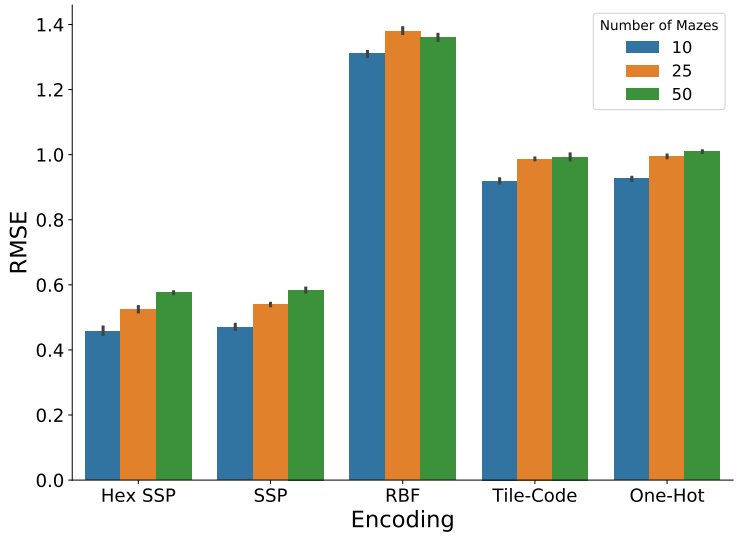


Figure 5.14: **Performance Outside of Defined Bounds**. Encoding methods are trained on a larger space than they are designed for.

and half the height of the entire maze. For SSPs, the only available parameter to change is the scaling of the coordinates used as the convolutional power. The scaling that is used in all previous experiments was increased by a factor of two, to indicate being chosen for a smaller space (i.e., moving one unit with this new scaling is equivalent to moving two units with the old scaling).

As can be seen in Figure 5.14, the SSP encoding methods outperform all other methods by a large margin.

5.2.8 Large Environments

This section explores the effectiveness of the different encoding methods on significantly larger single environments.

Computing the exact optimal policy for an arbitrary maze layout becomes computationally intractable when the environment is large. One way to overcome this issue is to construct the larger environment out of smaller known environments from the previous experiments instead of creating an entirely new larger one. The 100 environments can be tiled into a 10 by 10 grid to create a single environment that is 100 times larger. This method can leverage the fact that the optimal policy is already known for all start and goal

locations within each tile. This experiment is similar to the previous task except instead of re-using coordinates with a different context across sub-environments, all coordinates are unique to a particular sub-environment. The uniqueness of the coordinates provides the context implicitly, so no context signal needs to be provided. Adding the context signal back does little to help performance. Results are shown in Figure 5.16. A visualization of the policy on a particular maze is shown in Figure 5.17.

Another variation of the larger environment used in these experiments allows start and end positions to be anywhere. This is achieved by creating passageways between neighboring tiles. Every tile is connected to at least one neighbor and can have up to four connections. An additional constraint is used, ensuring that there exists a path from any free space location to any other free space location. This forms a hierarchical maze, as shown in Figure 5.15. A ground truth policy is computed for this higher level maze which is used to determine which passageway to use to exit the current lower level maze to get to a goal in any other lower level maze. This hierarchical structure allows the computation of the ground truth optimal action from any location to any other location to be tractable. A training dataset is generated using 500,000 samples along with a test set of the same size. Results are shown in Figure 5.18.

The overall performance of every method is much lower compared to the smaller environment case. This is to be expected as the task is much more difficult and meant to push each method toward its limit. A significant amount of generalization is required to solve this task. There are over 10^{10} possible start and goal locations, which is many orders of magnitude larger than the training set. The SSP encoding methods perform the best overall, with a larger margin over the other methods than in the smaller environment case.

The experiments were performed both with and without a context signal being given for the current sub-environment. The context signal had very little effect, suggesting that being at a different location is context enough for the network to learn how to behave.

5.2.9 Sub-Dimension Grouping

This experiment explores how the subdivision of the unitary vector space into orthogonal tori affects performance of the policy network. The parameter of interest is the sub-dimension τ which corresponds to dividing a N-dimensional space into $T = \lfloor \lfloor (N-1)/2 \rfloor / \tau \rfloor$ orthogonal τ-tori. Each τ-torus is projected onto a 2-torus by mapping the orthonormal unit vectors of the τ-dimensional coordinate system to evenly spaced points on the unit circle in 2D. Each torus is defined by a single ϕ value that is used for each of its τ dimensions. Values of ϕ as well as the angle of the projection (rotation offset of the unit circle points

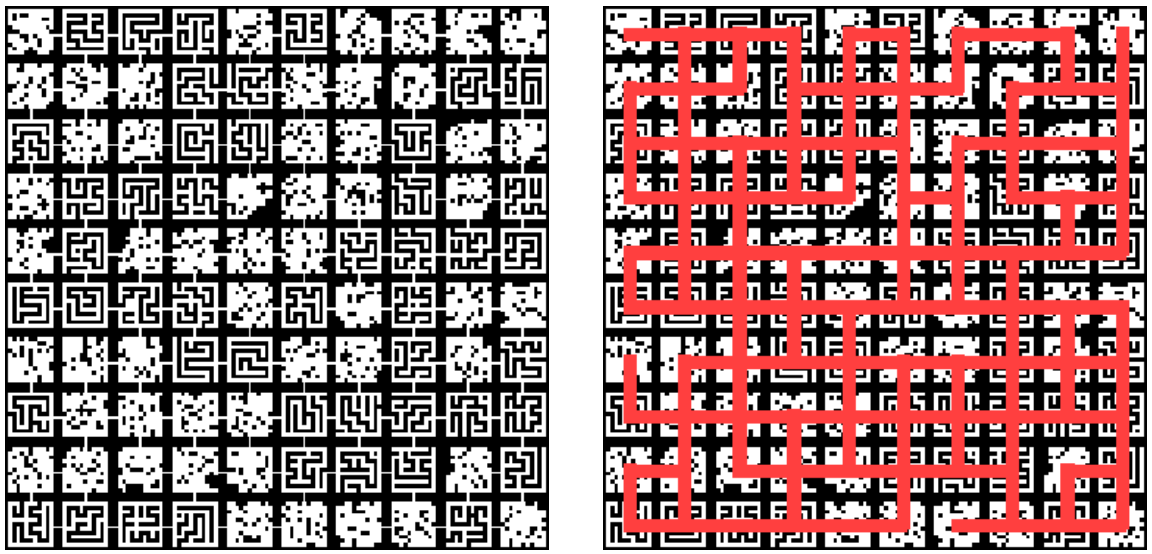


Figure 5.15: **Large Environment**. Arranging 100 smaller environments into a 10 by 10 grid to form one larger environment. Pathways between neighbouring environments are created if there are no obstacles at the midpoint of the walls in both environments. The paths in the higher level maze formed by these connections are highlighted on the right.

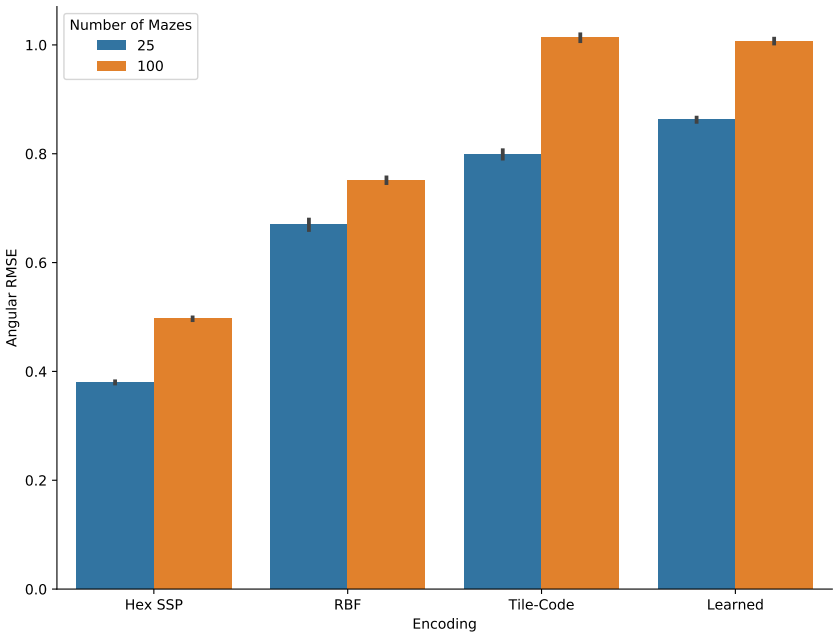


Figure 5.16: **Large Isolated Environment Results**. Two environment sizes are tested, one containing 5×5 and the other 10×10 smaller sub-environments. Start and goal can only appear within the same sub-environment. No context signal is given, but sub-environment coordinates do not overlap. Encoding dimensionality is 1024 and the network hidden size is 2048. Each variation is repeated 10 times with a different random number seed.

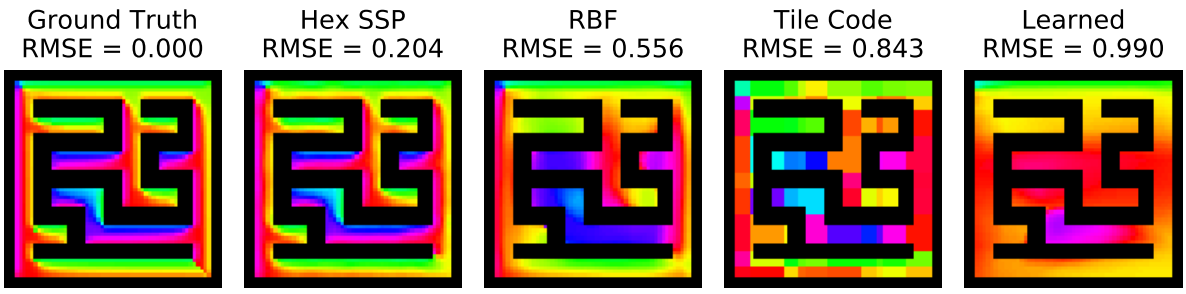


Figure 5.17: **Policy Visualization for Large Isolated Environment**

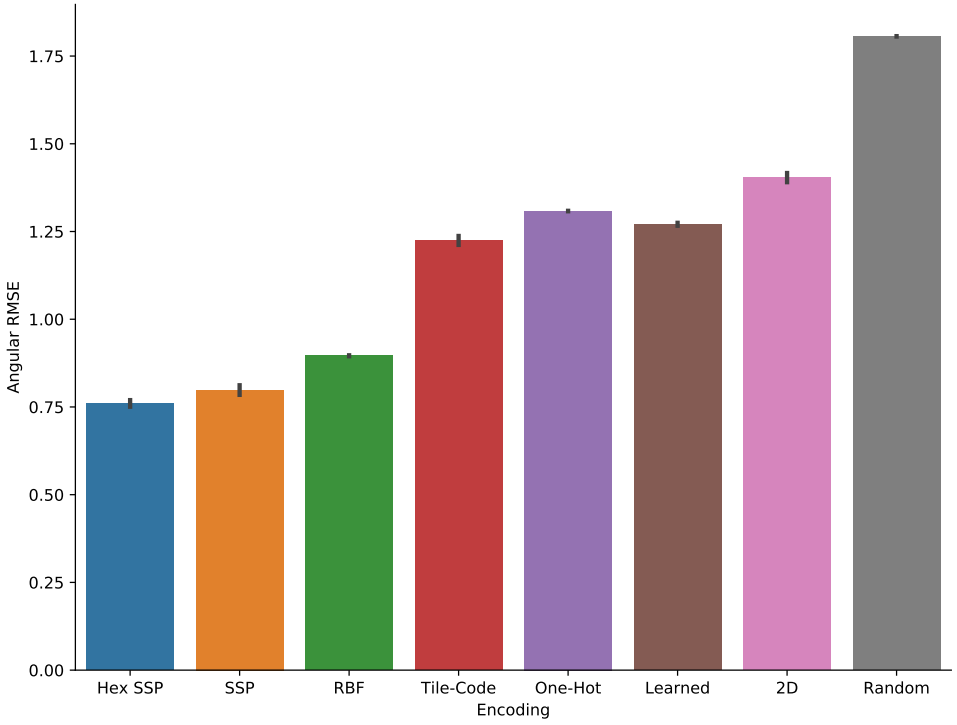


Figure 5.18: **Large Connected Environment Results**. Experiments were run on a hierarchical maze of size 130 by 130 units, discretized into 640 by 640 cells. The start and goal location can appear in any free space cell. Reported are results on a testing dataset consisting only of samples that did not appear in the training dataset. Encoding dimensionality is 256 and the network hidden size is 1024. Each variation is repeated 10 times with a different random number seed.

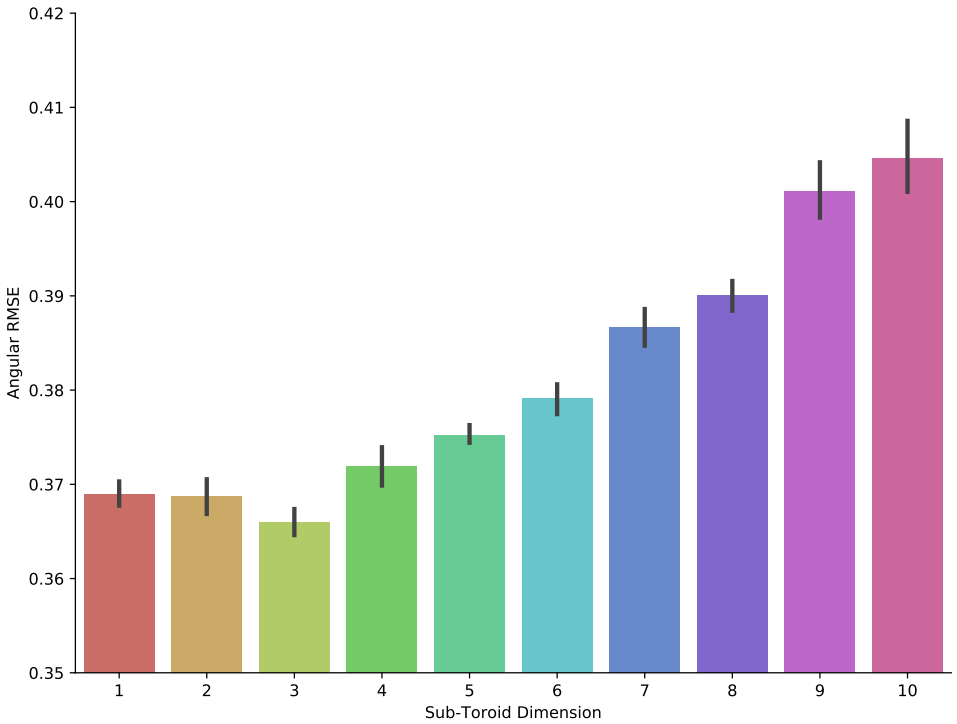


Figure 5.19: **Effect of Sub-Toroid Dimension**. Results of training a policy on a 25 maze task. A 1024D SSP encoding is used where sub-toroid dimensionality is varied. Training is performed on a dataset with 250,000 samples for 100 epochs with the Adam optimizer at a learning rate of 0.001. Network hidden size is set to 2048 with a dropout fraction of 0.1. Each condition is trained 10 times with different random number seeds.

in 2D) are chosen randomly for each τ-torus. For the special case of $\tau = 2$ the mapping is simply a random rotation of the 2D coordinate frame. Results are shown in Figure 5.19 for $1 \leq \tau \leq 10$.

Interestingly, the variant of the encoding which performed the best is $\tau = 3$ which corresponds to a subdivision of the space into hexagonal tori. This is the same SSP formulation that enables grid-cell-like firing patterns when implemented in a spiking neural network. The representation that most closely matches the observed data from animals also provides the best encoding for an artificial neural network to learn this navigation task.

5.2.10 Spiking Neural Network

The navigational policy can be implemented as a spiking neural network. Although spikes are non-differentiable, a smoothed rate based approximation of the neural response function that is differentiable can be used during training [77]. The software package NengoDL [143] contains useful tools for applying deep learning techniques to training spiking neural networks. This package allows a model specified in Nengo to be trained with backpropagation by converting the neurons in the network to a LIF rate approximation during training, and back to spiking for testing.

A feedforward network with 2 layers and 2048 neurons per layer is used. The time constant of the synapses between the spiking neurons is 0.001. This network is trained on 5,000,000 samples of (start, goal, context) across 10 unique environments. An L_2 penalty is applied to the network weights during training with a factor of 10^{-5} applied to this penalty when combined with the mean square error loss of the network output. A 256D SSP encoding is used with a sub-toroid dimensionality of 3. Each sample has Gaussian noise with $\sigma = 0.2$ added to all coordinates before they are encoded, and additional Gaussian noise with $\sigma = 0.01$ added to the encoded coordinate. Training is conducted for 100 epochs using the Adam optimizer with a learning rate of 0.001.

Visualization of network performance is shown in Figure 5.20. The output is the 2D direction decoded from spikes filtered by a synapse with a time constant of 100ms over a 30ms period. The angular RMSE across all environments is 0.2714, compared to 0.2165 for the non-spiking case with the same training parameters. This difference in performance is expected as training is conducted with the non-spiking approximation of each neuron. Both methods obtain an effective policy for solving the navigation task.

5.2.11 Reinforcement Learning

To demonstrate that these navigational policies can be learned in an online fashion, an experiment is conducted using reinforcement learning. Due to the additional computational costs associated with current approaches to reinforcement learning, analysis is restricted to a single environment. There is no theoretical reason why the results would not scale to multiple environments, but it would likely require more training to achieve the same performance. For this task, the start and goal locations can be set to any position that is free space within the environment.

The state space consists of two SSPs concatentated together, one for the agent's current location, and one for the goal location. No context signal is given as training occurs within

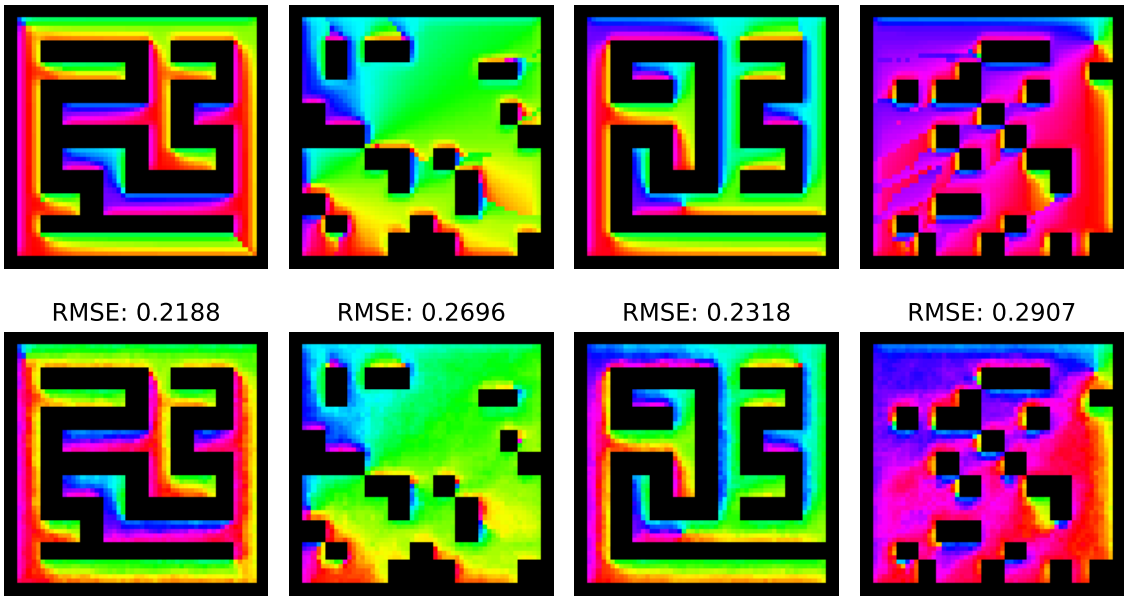


Figure 5.20: **Spiking Policy**. Visualization of spiking neural network policy performance on four unique environments. **Top**: Ground truth. Colour of each pixel indicates the optimal action for that location. **Bottom**: Policy output. Angular RMSE for the particular goal in each environment is shown above the image.

a single environment and this signal will always be static. The action space is a 2D vector corresponding to the velocity of the agent in the x and y directions. Each time-step the agent can move a maximum of 0.1 units in any direction. The environment is a square of 10 units by 10 units with interior obstacles.

Training

The agent is given a reward of +1 whenever it reaches a goal, and a reward of -0.001 for every time-step in which it is not at the goal. This small negative reward each step encourages the agent to learn shorter paths to the goal. The task is episodic, with a maximum of 100 time-steps per episode.

A randomly initialized agent is highly unlikely to stumble across a goal that is far away. If the goal is never reached during an episode, not much can be learned from that data. Training episodes will effectively be wasted until one in which the goal is close enough is used. To speed up training, a curriculum learning approach has been applied [15]. The distance between the start location and the goal location provides an estimate of the difficulty of solving the task. Instead of a random difficulty each episode, the difficulty of the task is gradually increased over the course of training. This allows early policies (i.e. closer to random policies) to be trained on tasks that they can solve. Once the easier task is learned to a sufficient degree, the policy is in a regime where the more difficult tasks become easier to learn. For this task the maximum distance that a goal location can be from the start location is 1 unit when training begins and increases by 1 unit every 50,000 training steps.

Another common method for improving training is reward shaping [101]. The implementation of this technique used here is as follows: instead of just receiving a positive reward when the goal is reached, positive reward is given the closer the agent gets to the goal. Even if the goal is never reached, a policy that gets close to the goal will receive higher reward than one which ends up further from the goal. This helps to guide the space of policies explored during training to be in a more promising regime. For this task the agent receives an additional positive reward each step that reduces the Euclidean distance to the goal and a negative reward for each step that increases this distance. These pseudo-rewards are balanced such that moving towards the goal and then back by the same amount produces a net zero return. This prevents undesired behaviours from being learned, such as circling the goal or moving back and forth near the goal instead of moving to the goal to finish the episode.

The reinforcement learning algorithm used in this experiment is Twin Delayed DDPG

(TD3) [51]. This algorithm has been shown to achieve state of the art performance on many continuous control tasks. A TensorFlow [1] implementation from Stable Baselines [74] is used. The networks for the actor and two critics each consist of two layers with 2048 units and ReLU activation functions.

Results

The average return at various stages during training is shown in Figure 5.21. The return for a given episode will only be positive if the goal was found, and the closer the value is to +1 the quicker it was found. Both reward shaping and curriculum learning help when training the network and the best results are achieved when applying both methods. For some runs the return peaks earlier in training and then begins to decay. This is a common issue that can arise when a combination of function approximation, off-policy learning, and bootstrapping are used [167], as is the case for the TD3 algorithm. This effect is less pronounced in the reward shaping conditions, likely due to the ease at which an accurate value function can be learned with this additional reward.

5.3 Learning SSP Parameters

So far SSPs have been used to provide a fixed encoding. This section explores how the encoding can be learned instead by modifying the parameters that define the transformation. The standard definition is to use a set of P axis vectors of size N (where typically $N \gg P$) to form the mapping $f\colon \mathbb{R}^P \to \mathbb{R}^N$:

$$f(\mathbf{x}) = \mathcal{F}^{-1}\left\{\prod_{p=0}^{P-1} \mathcal{F}\left\{X_p\right\}^{x_p}\right\} \tag{5.2}$$

where X_p is an axis vector and x_p is the corresponding element of the input feature. Each Fourier coefficient of X_p is raised to the power of x_p and coefficients across axis vectors are multiplied together in an element-wise fashion (Hadamard product).

This transformation is defined by the axis vectors, which can be parameterized by the exponents of their Fourier components, ϕ_k. This parameterization is defined by Equation 5.3:

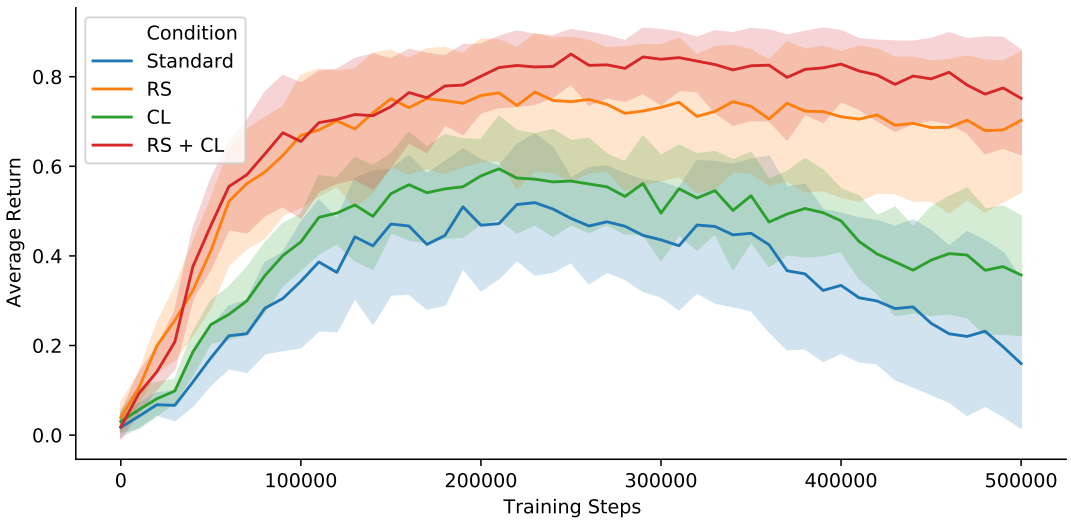


Figure 5.21: **Average Return During Training**. Four conditions are used, standard training on the task, training with reward shaping (RS), training with curriculum learning (CL), and using both modifications. Each condition is repeated 10 times. The average return is calculated every 10,000 training steps across 100 evaluations of a deterministic policy. Each evaluation uses the full space for start and goal locations regardless of difficulty in the curriculum and does not include pseudo-reward in the return shown.

$$v_n = \frac{1}{N} \sum_{k=0}^{N-1} \left(e^{i2\pi kn/N} \cdot \prod_{p=0}^{P-1} e^{i\phi_{p,k} x_p} \right) \tag{5.3}$$

where v_n is the n-th element of the output vector of size N, x_p is the p-th element of the input vector of size P, $\phi_{p,k}$ is the exponent of the k-th Fourier coefficient of the p-th axis vector. For the output to be a unitary vector defining an SSP, $\phi_0 = 0$, $\phi_k = -\phi_{N-k}$, and for even N, $\phi_{N/2} = 0$. These restrictions result in each axis vector having $T = \lfloor (N-1)/2 \rfloor$ parameters. In general, for a P-dimensional coordinate being mapped to an N dimensional space, there will be $P \times T$ parameters. Applying these restrictions, the equation becomes:

$$v_n = \frac{1}{N} + \frac{1}{N} \sum_{k=1}^{T} \left(e^{i2\pi kn/N + \sum_{p=0}^{P-1} \phi_{p,k} x_p} + e^{-i2\pi kn/N - \sum_{p=0}^{P-1} \phi_{p,k} x_p} \right) \tag{5.4}$$

with an extra $+\frac{(-1)^n}{N}$ term for even N. This equation can be further simplified using the identity $\cos\theta = (e^{i\theta} + e^{-i\theta})/2$:

$$v_n = \frac{1}{N} + \frac{(-1)^n}{N}((N-1) \bmod 2) + \frac{2}{N} \sum_{k=1}^{T} \cos \left(2\pi kn/N + \sum_{p=0}^{P-1} \phi_{p,k} x_p \right) \tag{5.5}$$

This formulation contains only real numbers and can be computed efficiently with vectorized operations, allowing it to easily be integrated into deep learning systems. In order to backpropagate through this function, the Jacobian needs to be defined (partial derivatives of the outputs with respect to the inputs). This is a matrix of the form:

$$J = \begin{bmatrix} \frac{\partial v_0}{\partial x_0} & \cdots & \frac{\partial v_0}{\partial x_{P-1}} & \frac{\partial v_0}{\partial \phi_{0,1}} & \cdots & \frac{\partial v_0}{\partial \phi_{0,T}} & \cdots & \frac{\partial v_0}{\partial \phi_{P-1,1}} & \cdots & \frac{\partial v_0}{\partial \phi_{P-1,T}} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \cdots & \vdots & \ddots & \vdots \\ \frac{\partial v_N}{\partial x_0} & \cdots & \frac{\partial v_N}{\partial x_{P-1}} & \frac{\partial v_N}{\partial \phi_{0,1}} & \cdots & \frac{\partial v_N}{\partial \phi_{0,T}} & \cdots & \frac{\partial v_N}{\partial \phi_{P-1,1}} & \cdots & \frac{\partial v_N}{\partial \phi_{P-1,T}} \end{bmatrix} \tag{5.6}$$

The partial derivatives with respect to the inputs are:

$$\frac{\partial v_n}{\partial x_p} = \frac{-2}{N} \sum_{k=1}^{T} \phi_{p,k} \sin \left(2\pi kn/N + \sum_{q=0}^{P-1} \phi_{q,k} x_q \right) \tag{5.7}$$

The partial derivatives with respect to the parameters are:

$$\frac{\partial v_n}{\partial \phi_{p,k}} = \frac{-2x_p}{N} \sin\left(2\pi kn/N + \sum_{q=0}^{P-1} \phi_{q,k} x_q\right) \tag{5.8}$$

With the Jacobian defined, the SSP function can be integrated as a learnable component in any ANN.

Experiments are conducted on the policy learning task where the SSP encoding is learnable along with the parameters of a hidden layer. The networks are trained for 100 epochs on a dataset of 250,000 samples across 25 unique environments. A dropout fraction of 0.1 is applied to the hidden layer during training and an L_2 penalty is applied to the weights to prevent overfitting. The encoding dimension, context vector, and hidden size is set to 256. Results are shown alongside fixed encodings and a standard learned encoding in Figure 5.22. Learning the SSP encoding parameters provides improvement over learning a standard encoding layer (weights, biases, with ReLU activation). There are also fewer parameters involved in this transformation:

$$\begin{aligned} p_{fc} &= (n_{k-1} + 1) \cdot n_k \\ p_{ssp} &= (n_{k-1}) \cdot \lfloor (n_k - 1)/2 \rfloor \end{aligned} \tag{5.9}$$

where n_k is the number of neurons in the current layer, n_{k-1} is the number of neurons in the previous layer, p_{fc} is the number of parameters for a fully connected layer, and p_{ssp} is the number of parameters for an SSP layer. The learned SSP encoding also provides improved performance over a randomly generated SSP and a randomly generated SSP with grid-cell-like structure. This improvement is only seen if appropriate regularization is applied during training, as the learned SSP method can easily overfit to the training data. Figure 5.23 shows the results for all four methods with and without regularization applied. The learned SSP method performs disproportionately worse than the other methods on the test set without regularization, but performs best overall with it. Two variants of regularization are applied to the learned SSP method. The first applies the L_2 penalty only to the weights of the network, the second also applies an L_2 penalty to the ϕ parameters as well. The first method is shown in the plot, the second method resulted in a slightly higher mean RMSE (.414 vs 0.418) but the difference between the two methods is not significant ($p = 0.062$).

There are also interesting differences in the structure of the SSPs that are learned in this task. Heatmaps of the SSPs learned with and without regularization are compared to two fixed methods in Figure 5.24. Qualitatively, the random fixed SSP (row A) and the learned SSP without regularization (row C) are quite similar. These encodings also obtain similar performance on the test set. Another way of visualizing an SSP is through histograms of

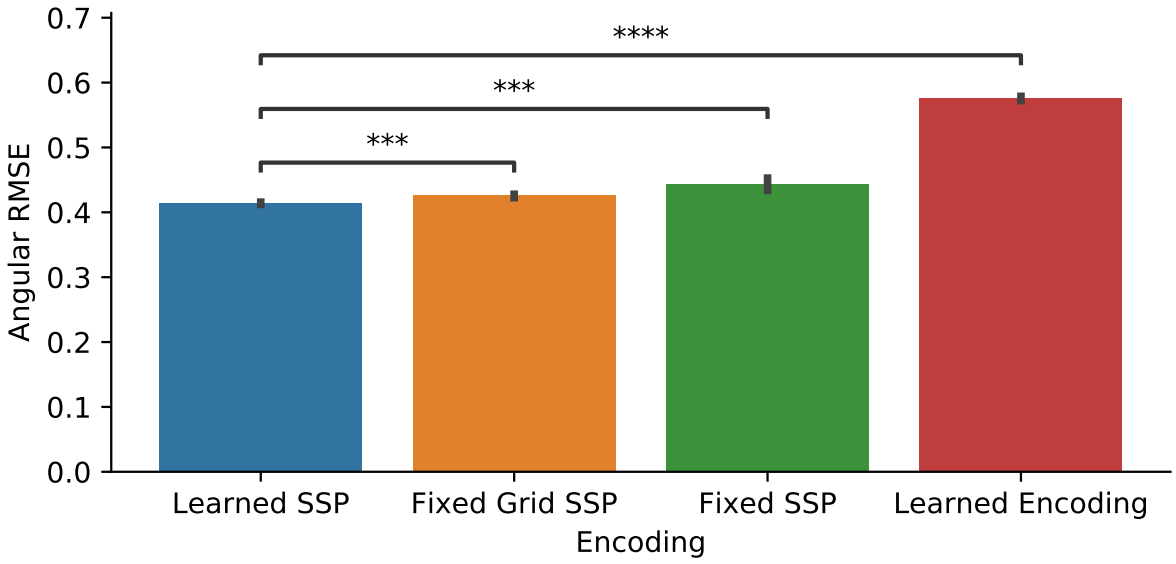


Figure 5.22: **Learning SSP Parameters**. Test set performance of a network with learnable SSP parameters compared to fixed SSPs and learned encoding weights. Each network uses a 256D encoding.

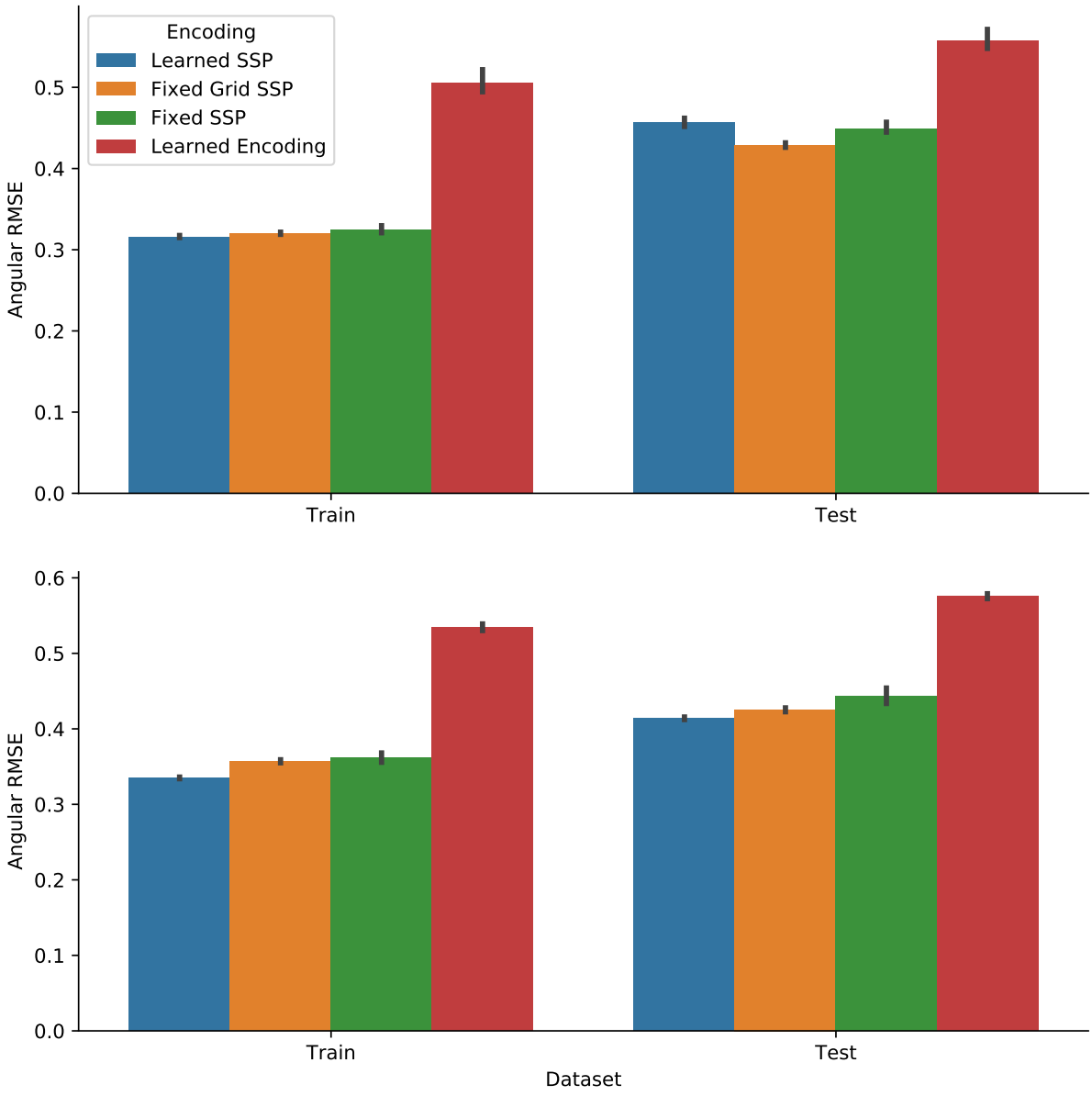


Figure 5.23: **Effect of Regularization**. Performance on the training and testing sets with and without regularization is applied. **Top**: No regularization applied. The learned SSP method performs disproportionately worse on the test set than the other methods, indicating stronger overfitting. **Bottom**: Dropout and weight decay applied during training. Performance on the test set is similar to performance on the training set for all methods. The learned SSP method performs the best in this case.

the ϕ parameters as shown in Figure 5.25. Each of the T pairs of parameters $\phi_{x,k}$ and $\phi_{y,k}$ are plotted on the same image with Gaussian smoothing. This gives an indication of the spatial distribution of the parameters. The magnitude of a the k-th pair of parameters is defined by:

$$m_k = \sqrt{\phi_{x,k}^2 + \phi_{y,k}^2} \tag{5.10}$$

A histogram of this value is plotted in the far right column of Figure 5.25. The random fixed SSP (row A) and the learned SSP without regularization (row C) continue to show many similarities with this visualization. The parameter images appear unstructured and the magnitude histograms have similar shape (corresponding to what would be expected from a random distribution on a 2D space). The fixed grid SSP (row B) and learned SSP with regularization (row E) both exhibit more structure in their parameter images, although different from each other. The fixed grid SSP has an even distribution of magnitudes (by design) which is reflected in the histogram. There is a sharp cutoff in the regularized methods, with the highest density of magnitudes around 0.45 and none below 0.41. One explanation for this phenomenon is that values below this threshold are not very useful for solving the task. A lower value of ϕ corresponds to a larger spatial scale. The environment being trained on is fixed in size, meaning large spatial scales bigger than the environment itself will not be very useful. Even in the version where the L_2 penalty was applied to the ϕ values in addition to the weights (row E) this threshold still shows up. Despite near-zero values incurring a lower cost, no parameters end up in this region, indicating that every parameter is beneficial in some way to solving the task. As expected, there is a clear bias towards smaller magnitudes in general, but there are still many parameters with larger magnitudes, indicating that a mix is beneficial.

5.4 SSP Cleanup Memory

When multiple semantic pointers are bound together, there is always a degradation of representation in order to represent the information in a fixed dimensional space. This degeneration can be seen as a form of noise and accumulates with successive binding. With enough noise built up the system may perform poorly. To prevent this failure mode, it is important to be able to clean up a noisy representation. In the integrated system described later in this chapter (Section 5.7) a cleanup memory will be important when extracting the location of a specific object from a memory containing many objects.

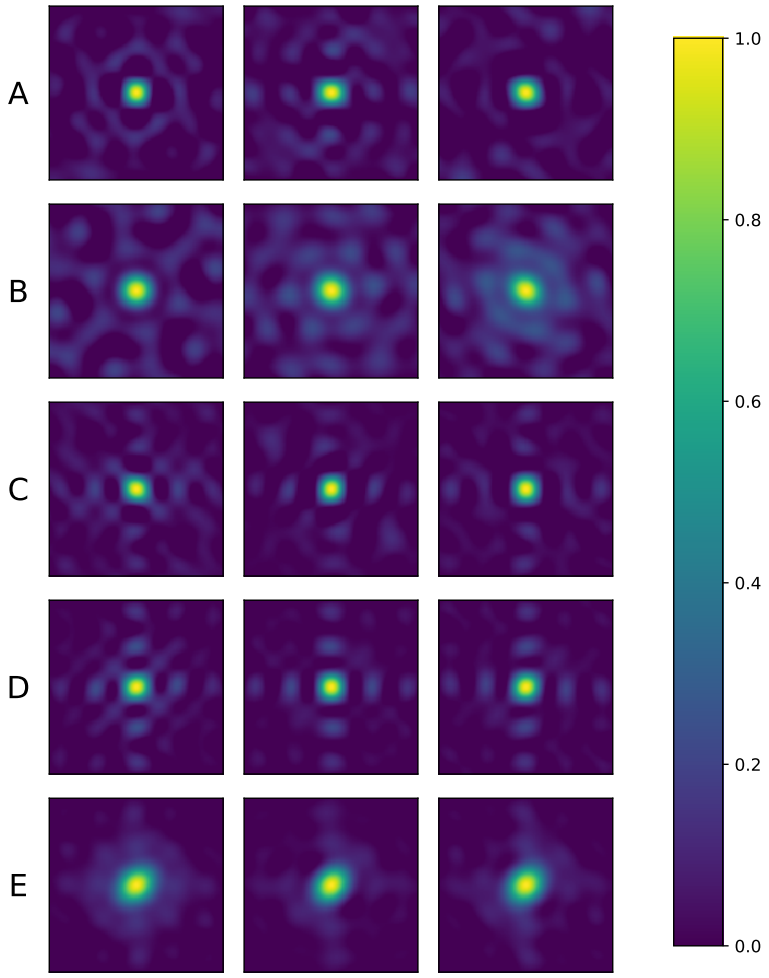


Figure 5.24: **Learning SSP Parameter Heatmaps**. Each row corresponds to a method of generating SSPs, and each column is a different instance of that generation method. **A**: Fixed SSP. Values of ϕ are chosen uniformly at random between $-\pi$ and $/pi$. **B**: Fixed Grid SSP. Values of ϕ chosen by splitting the space into hexagonal toroids with random angles and scales. **C**: Learned SSP. Values of ϕ that are learned by a policy network. **D**: Learned SSP (Regularized). Values of ϕ that are learned by a policy network with regularization to prevent overfitting. **D**: Learned SSP (Regularized ϕ). The L_2 is applied to the values of ϕ as well as the weights.

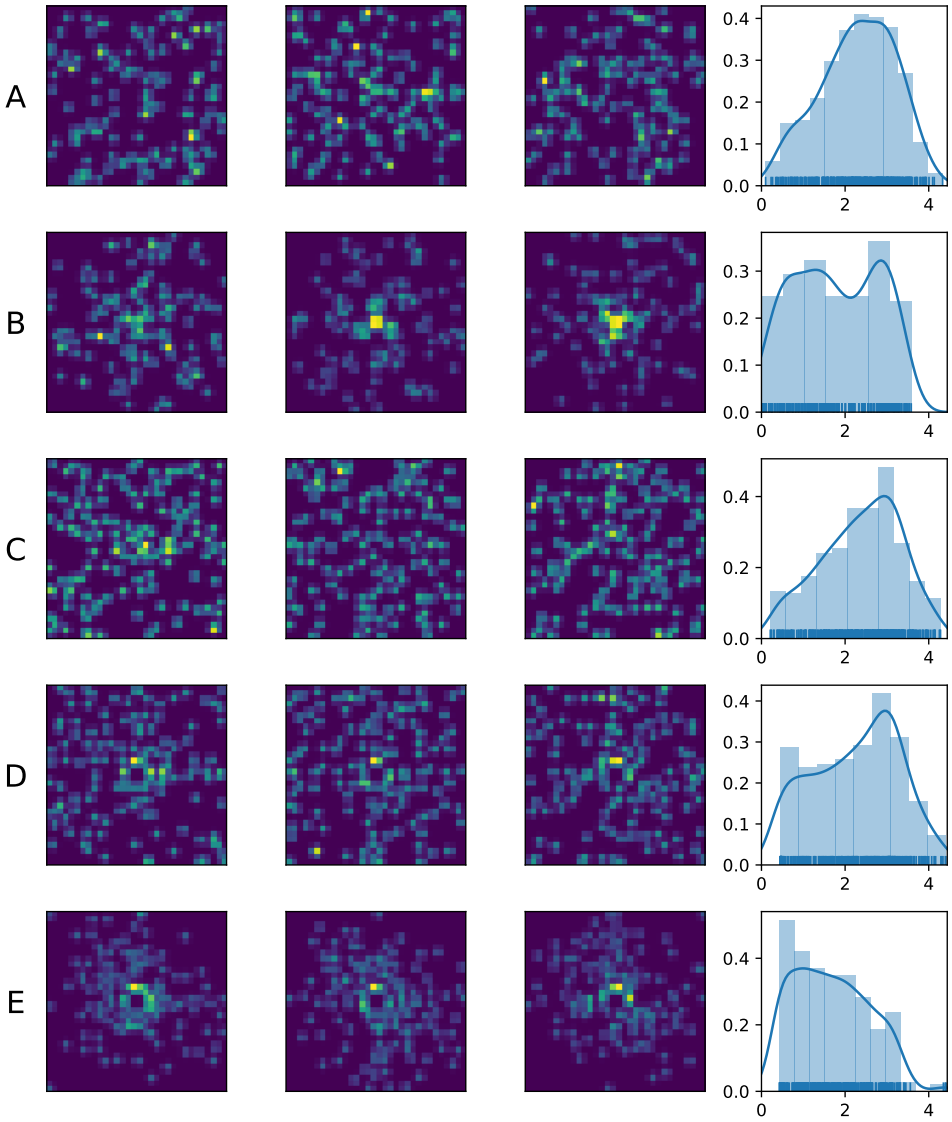


Figure 5.25: **Learning SSP Parameter Histograms**. Each row corresponds to a method of generating SSPs. The first three columns are Gaussian smoothed 2D histograms of the ϕ values of three different instances of each generation method. The far right column is a histogram of the magnitudes for each pair of $\phi_{x,k}$ and $\phi_{y,k}$. **A**: Fixed SSP. **B**: Fixed Grid SSP. **C**: Learned SSP. **D**: Learned SSP (Regularized). **E**: Learned SSP (Regularized ϕ).

One way to achieve this goal is to train a feed-forward neural network to map from the noisy encoding to a clean version. This can be done using supervised learning on pairs of noisy and clean vectors. The resulting function can be thought of as pulling the representation from any value in $\mathbb{R}^d$ to a subspace corresponding to valid SSPs in the domain that was trained over. The cleanup function f provides a mapping from an approximate SSP $\hat{\mathbf{S}}$ to a valid SSP $\mathbf{S}$.

$$f : \hat{\mathbf{S}} \to \mathbf{S} \tag{5.11}$$

Two methods of adding noise are independently used. The first is to simply add normally distributed noise to the vector. The second method involves constructing a memory containing several semantic pointers bound together. Each semantic pointer is a random item vector bound to a random SSP within the region of interest. A query is then performed on the memory by binding to the inverse of one of the random item vectors. The output is a noisy version of the SSP corresponding to the location of that item in memory. For every item in the training set, a new memory is created and a different query is performed. Using this method, the noise profile of the vectors in the training set will more closely match the expected noise profile of the vectors that the cleanup memory will be used with. The memory vectors are created using:

$$M = \sum_{i=1}^{N} S_d(x_i, y_i) \circledast O_i \tag{5.12}$$

where M is the memory vector, N is the number of objects stored in the memory, $S_d(x, y)$ constructs a d-dimensional SSP representing the location (x, y) as defined in Equation 3.4, and O_i is the ith object vector.

$\hat{S}_i$ is a noisy version of the ith SSP. This noise can be generated from querying a memory for the location of the corresponding object:

$$\hat{S}_i = M \circledast O_i^{-1} \tag{5.13}$$

where M is a memory vector constructed by Equation 5.12. Another option is to simply Gaussian noise to the original SSP:

$$\hat{S}_i = S_d(x_i, y_i) + \mathcal{N}_d(\vec{0}, \sigma\mathbf{I}) \tag{5.14}$$

where σ is the standard deviation of the noise. A combination of both noise types can also be used.

The item vectors are generated from a d-dimensional multivariate normal distribution of zero mean, unit variance, and no covariance. The vectors drawn from this distribution are then normalized:

$$\begin{aligned} V_i &\sim \mathcal{N}_d(\vec{0}, \mathbf{I}) \\ O_i &= \frac{V_i}{\|V_i\|} \end{aligned} \tag{5.15}$$

5.4.1 Loss Function

A standard loss function to use for supervised learning is mean squared error (MSE). Another loss function that has useful properties for this task is cosine distance. This loss is defined as the cosine of the angle between the output vector and the target vector. For high dimensional vectors corresponding to a semantic meaning, similarity is often measured as cosine distance, which makes this loss function a natural choice for this task. Since semantic pointers are normalized to be unit length, matching the magnitude of the output is not as important as matching the direction. While mean squared error incurs a penalty for magnitude differences, cosine distance does not. Both loss functions will report zero error when the prediction and the target match, but the path that is taken to get there is different. When plotting both loss functions while training, it becomes evident that there is a local minimum for the MSE that training with a cosine loss gets through. In contrast, when training using MSE, the cosine loss tends to follow monotonically, indicating that there was no local minimum in the cosine loss trajectory that training with MSE gets through. Visualization of this effect when training with stochastic gradient descent is shown in Figure 5.26. Therefore the cosine loss is used in subsequent training.

5.4.2 Spiking Neural Network Implementation

A cleanup memory can also be implemented efficiently in a spiking neural network. This functionality is achieved by setting the connection weights between two populations of neurons appropriately. The first population represents the noisy SSP, and the second will have a clean representation. The encoders of the neural population representing the clean SSP are set to be a mix of band cells, grid cells, and place cells. The decoders from the noisy population are calculated using the least squares optimization in the NEF (Equation 2.6). A set of samples of noisy SSPs and corresponding clean SSPs are used to form the activity matrix and state vector matrix respectively.

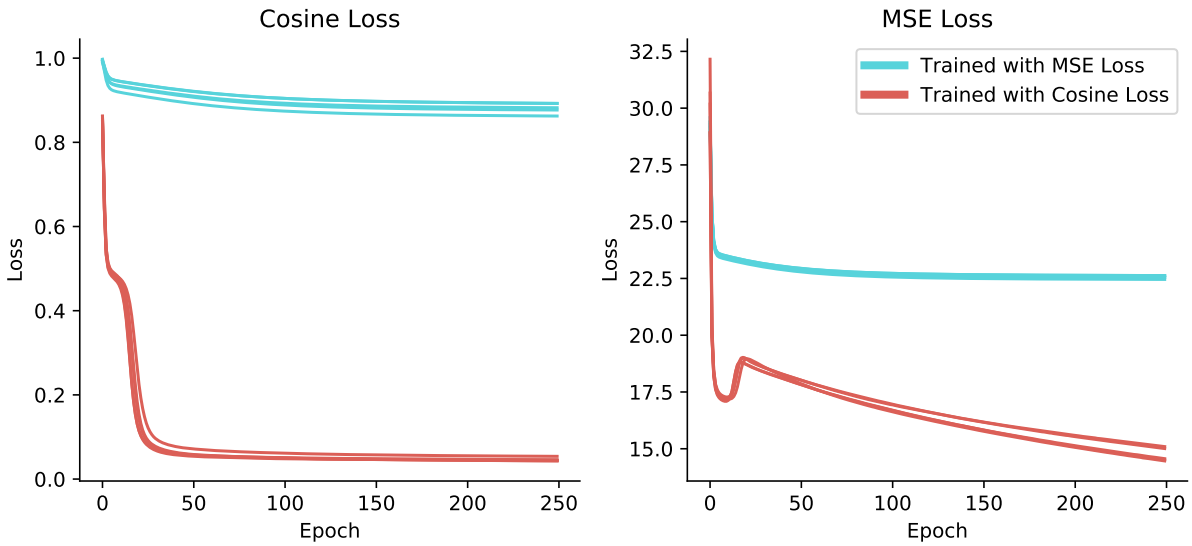


Figure 5.26: **MSE and Cosine Loss During Training**. Using the cosine loss for backpropagation results in superior performance on both loss metrics than using MSE loss for backpropagation. There appears to be a local minimum in the MSE that training with cosine loss is able to get past. Shown are 5 runs of each condition with different random number seeds for both dataset generation and random axis vectors to use. Results are for a 512D encoding, trained with stochastic gradient descent over a space with coordinates ranging from -5 to 5.

5.4.3 Results

Experiments are conducted using networks with 25 neurons per dimension on 256D SSP representations. Noise is generated by extracting the location of an object in a compressed memory containing 7 objects bound to locations. Additional Gaussian noise with $\sigma = 0.005$ is added to each of the 256 dimensions. The artificial neural network is trained on 14,000 pairs of noisy and clean SSPs. The spiking neural network is optimized using the same training set to act as sample points in the least squares optimization. As can be seen in Figure 5.27, the cleanup provides a substantial improvement over the noisy representation. The RMSE of the noisy vectors compared to the ground truth is 1.179. The spiking cleanup reduces the error to 0.734 and the ANN cleanup to 0.572. In terms of cosine distance, the noisy vectors produce an error of 0.648 with respect to the ground truth. The spiking cleanup reduces this error to 0.260 and the AAN cleanup to 0.072.

Another useful measure is how easily a 2D coordinate can be decoded from the representation. One way to approximately decode location is to find the location of the peak in the heatmap. Decoding in this manner, the peak of the result of the cleanup is within 0.5 units of the true peak 96.2% of the time for the spiking cleanup, and 97.6% of the time for the ANN cleanup. Interestingly, the peak in the noisy representation is within this threshold 99.8% of the time. While the noise due to compression degrades the representation in terms of both RMSE and cosine distance, the peak in the heatmap is strongly maintained. This peak corresponds to which SSP the representation is most similar to, regardless of the strength of that similarity. While both cleanup methods move the representation closer to the desired representation, they also move it closer to different SSPs as well, degrading the ability to decode the 2D coordinate in some cases. This indicates that while this cleanup function is extremely helpful, there is still much more room for improvement. These results are summarized in Table 5.2.

It is important to note that the primary use of cleaning up a noisy SSP is to be able to use the representation as a high dimensional vector in other areas of the network, rather than to simply decode to a 2D coordinate. Operations such as circular convolution to shift the representation or bind it to another object rely on the representation being closer to a valid SSP rather than a vector that is just easier to decode.

5.5 Path Integration

Path integration involves the updating of a current position estimation over time given self-motion cues. These cues can come from motor commands as well as sensory input. In

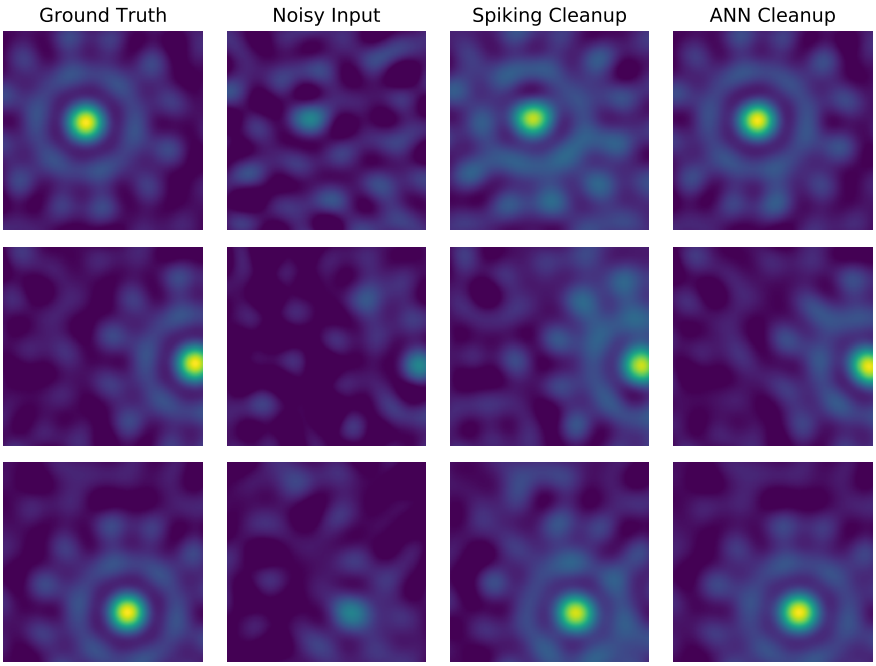


Figure 5.27: **SSP Cleanup Memory Behaviour**. Similarity heatmaps of the ground truth SSP (far left), the noisy input to the network (middle left), the result of the spiking cleanup (middle right), and the result of the artificial neural network cleanup (far right). Each SSP is 256D and is generated by subdividing the space into orthogonal hexagonal tori with random orientations and scales. The spiking network is presented each noisy input in succession for 100ms each. Shown is the average decoded value across a 90ms period starting 10ms after stimulus onset.

Metric	Noisy Input	Spiking Cleanup	ANN Cleanup
RMSE	1.179	0.734	0.572
Cosine Distance	0.648	0.260	0.072
Peak Decode Accuracy	99.8%	96.2%	97.6%

Table 5.2: **Cleanup Performance**. Results are from a 256D SSP. Noise is applied by constructing a normalized semantic pointer containing 7 item-location pairs bound together and circularly convolving with the semantic pointer for one of the items. Each item is represented by a random unit length 256D vector. Results are from a test set generated using items not present in the training set. Locations for the SSPs are chosen uniformly at random from a 10 by 10 space. RMSE and Cosine Distance is computed based on the ground truth SSP. Peak Decode Accuracy is the percentage of the maximum point in each of the heatmaps that is within 0.5 units of the ground truth 2D coordinate.

this experiment the 2D velocity of the agent is given at every time-step. The experiment follows the format of [8] where the velocities are given as an input to an LSTM that is trained to update its hidden representation such that an encoding of the current location can be read out at every time step. This will be important in the final integrated system for updating the internal representation based on velocity commands from a navigational policy.

The model architecture is shown in Figure 5.28. This model is based on the path integration model from [8] but written in Pytorch instead of Tensorflow and supports experimentation with different spatial encoding mechanisms. The LSTM uses the standard implementation in Pytorch [135], with four gates and sigmoid and tanh nonlinearities. The equations that define this LSTM are shown below:

$$
\begin{aligned}
i_t &= \sigma(W_{ii} v_t + b_{ii} + W_{hi} h_{t-1} + b_{hi}) \\
f_t &= \sigma(W_{if} v_t + b_{if} + W_{hf} h_{t-1} + b_{hf}) \\
o_t &= \sigma(W_{io} v_t + b_{io} + W_{ho} h_{t-1} + b_{ho}) \\
g_t &= \tanh\left(W_{ig} v_t + b_{ig} + W_{hg} h_{t-1} + b_{hg}\right) \\
c_t &= f_t \odot c_{t-1} + i_t \odot g_t \\
h_t &= o_t \odot \tanh\left(c_t\right)
\end{aligned}
\tag{5.16}
$$

The velocity input at time t is v_t. The hidden state (h_t) and cell state (c_t) are updated at each time step. There are four gates, input (i_t), forget (f_t), output (o_t), and cell (g_t). σ refers to the sigmoid function, and $\odot$ signifies element-wise multiplication (Hadamard product).

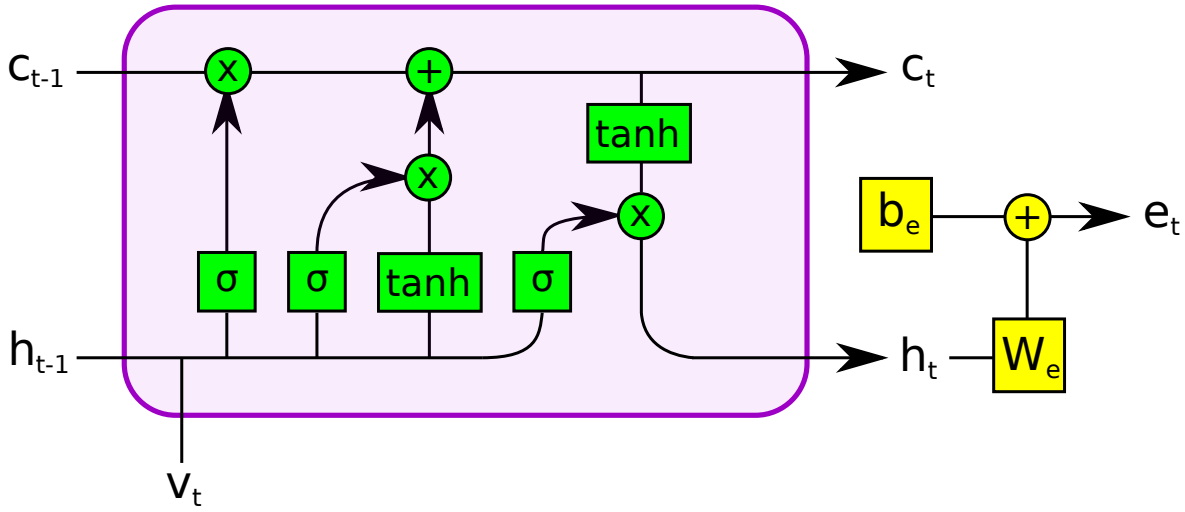


Figure 5.28: **Path Integration Model Architecture**. An LSTM followed by a readout layer transforming the hidden state into an encoded position. The same architecture is used for all encoding methods.

The output of the LSTM is passed through a linear layer to compute the estimate of the representation for the current location.

$$e_t = W_e \cdot h_t + b_e \tag{5.17}$$

Since path integration is relative to the starting point, one additional component is added to the network to compute this starting point for the first step of the trajectory. This is done through a set of weights applied to the encoding of the initial position of the trajectory and serves to initialize the hidden state and cell state of the recurrent network performing the path integration. These weights are learned through backpropagation along with the rest of the network.

$$\begin{aligned} c_0 &= W_c \cdot S(x_0, y_0) \\ h_0 &= W_h \cdot S(x_0, y_0) \end{aligned} \tag{5.18}$$

T	15 s
L_{wall}	2.2 m
dt	0.02 s
μ_{ang}	0 deg/s
σ_{ang}	330 deg/s
α_{lin}	0.13 m/s
d_p	0.03 m
p_{ang}	660 deg/s
p_{lin}	0.25

Table 5.3: **Path Integration Dataset Generation Parameters**

5.5.1 Dataset Generation

Data is generated from a simulated agent in an square enclosure with no obstacles. The state of the agent is given by its (x, y) position and heading. At every time-step the agent will move forward and turn with a particular speed. The linear speed is drawn from a Rayleigh distribution with scale parameter α_{lin} and the angular speed is drawn from a normal distribution with mean μ_{ang} and standard deviation σ_{ang}. These distributions were chosen as they produce trajectories similar to real rats foraging in an environment [144].

Collisions with the boundary are avoided by slowing down the linear speed and increasing the angular speed when the agent approaches a boundary. This causes the agent to turn around before colliding. The agent is considered as approaching the boundary when it is within a distance of d_p to any wall. While the agent is within this bound, its linear speed is slowed down by multiplying by the factor p_{lin} and its angular speed is changed by p_{ang} in the direction corresponding to the smallest rotation needed to face away from the closest wall.

All of the parameter values used to generate the dataset are based on the work from [8] and shown in Table 5.3. T is the length of time for each trajectory, L_{wall} is the length of each wall in the square arena, and dt is the simulation time-step. Unless otherwise specified, a dataset containing 7500 unique trajectories is used for each experiment. The starting position of the agent is chosen uniformly at random for each trajectory.

5.5.2 Training

For training, trajectories of 100 time-steps are sampled from the dataset. Each time-step of the trajectory consists of the (x, y) position of the agent and the velocity command

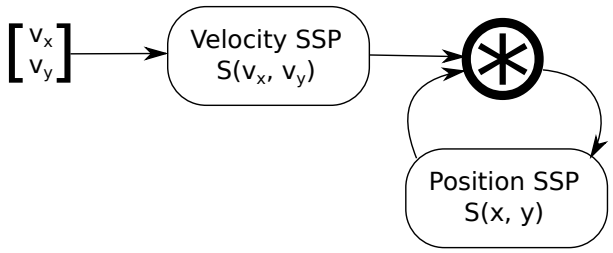


Figure 5.29: **SSP Path Integration Architecture**. Input velocity is encoded as an SSP and then circularly convolved with an existing position SSP to update the position. This update occurs every time-step.

issued to get to the next point in the trajectory. Batches of 10 trajectories are used for each optimization step. A dropout rate of 50% is used on the fully connected linear layer during training. These parameters were chosen to match those used in related work on a similar task [8].

The output of the network is a sequence of vectors corresponding to the encoded position at each step in the trajectory. The loss is computed between these output encodings and the true encodings generated from the (x, y) position from the trajectory. The loss function used is the sum of the mean squared error and the cosine distance between the model estimate and the true target. Weights and biases are updated using backpropagation with the RMSProp optimizer [175].

5.5.3 Circular Convolution Network

The previously described approaches use an artificial neural network as a function approximator to compute path integration. In the case of an SSP representation, this approximation is not required as the true function can be derived. Integrating a velocity signal to update a position representation can be described as a circular convolution of the current location with the velocity encoded as an SSP. This operation shifts the represented location in the direction of the velocity. A recurrent network that computes this path integration is shown in Figure 5.29. This network can also be implemented efficiently with spiking neurons.

5.5.4 Results

The circular convolution network requires no training. Each other network is trained for 1500 epochs on a dataset containing 7500 trajectories with 100 time steps in each. Performance on an independent testing set of 1000 trajectories is shown in Figure 5.30.

Unsurprisingly, the circular convolution network performs the best. This is due to the fact that an SSP can represent any continuous coordinate precisely and circular convolution provides the exact update needed to move this representation based on a displacement (velocity multiplied by the time step). The small amount of error recorded can be attributed to measurement error from decoding the representation into a 2D coordinate for comparison to the ground truth.

Decoding is accomplished by using a dense nearest neighbors lookup by discretizing the environment into 16,384 (128×128) locations. The environment has a length of 2.2 units, meaning each bin covers a 0.017 unit square. The maximum measurement error occurs when the true value of the location is on the corner of one of these squares, which corresponds to an error of $\sqrt{(2 \times (0.017/2)^2)} = 0.012$. Any error above this value can be attributed to the network itself. For the 2D method there is no decoding required, so all error is from the network.

The Hex SSP encoding provides the best performance of the neural network approaches. Surprisingly, the 2D method does not work as well as Hex SSP or RBF despite having a simpler underlying function to learn. This is likely due to the fact that this function needs to be learned precisely, as deviations in any dimension of a 2D output has a larger impact than deviations in any dimension of a 256D output. While the Legendre method can represent positions precisely, the transformation required to move between nearby positions is likely difficult to learn due to the fact that each component in the representation changes with a different frequency and there are many high frequency components. The SSP encodings can represent positions precisely, are resilient to perturbations due to an imperfect network, and have a consistent transformation to nearby representations. These properties all contribute to successful path integration performance.

5.6 Localization

The goal of the localization task is given a set of measurements of the environment from an agent, determine the location of the agent in the environment. The sensor measurements come from a low resolution RGB-D camera (provides colour and depth at each pixel the

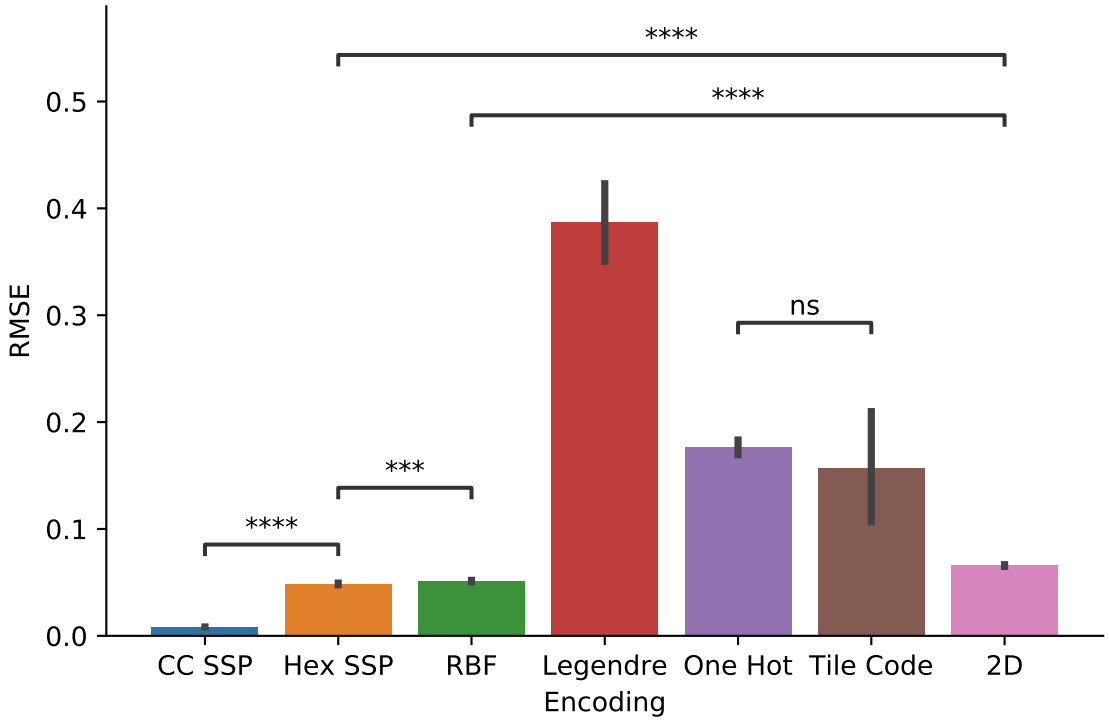


Figure 5.30: **Path Integration Results**. Each network is evaluated on 1000 random trajectories 100 time steps in length. RMSE is calculated based on the difference between the true location and the decoded output representation for every time step in all trajectories. 'CC SSP' refers to the circular convolution network. All other methods refer to a network trained as described in Section 5.5.2. 256D encoding is used for all methods except '2D' which produces 2D coordinates directly.

agent can see in the simulated environment). The output location will be given using the encoded location. This will be used in the final integrated system to allow sensor inputs to construct an internal representation of position that can also be updated through path integration and used by a policy for computing actions.

To solve this task, a feed-forward artificial neural network is trained using supervised learning. Data points are pairs of measurements and corresponding encoded location.

5.6.1 Dataset Generation

Multiple environment layouts are randomly generated, using either the 'blocks' or 'maze' structure. A colour function assigns an RGB value to every obstacle in the environment. For this experiment, this is accomplished through one Gaussian function for each colour channel, centered at 3 different points in the environment, emulating overhead light sources. This colour information is added to help distinguish different parts of the environment. Due to the simplicity of the 2D environment, many geometrical features are replicated in different areas (i.e., all horizontal hallways will give the same reading). Adding colour increases the complexity of the environment and helps to distinguish between different areas within the environment. For each environment, a set of random locations are chosen from the free space regions. The agent is placed in this state and then measurements are taken from each of its sensors. The measurements are the distances to the nearest obstacle and the colour of that obstacle, along each of 36 directions evenly spaced around the agent. Each sample in the dataset consists of the vector of sensor measurements, the ground truth encoding for that x-y location, and an ID vector for the environment being used (context).

5.6.2 Results

Experiments are run across 10 different environments. Models are trained for 50 epochs on a training set of 100,000 samples using the Adam optimizer with a learning rate of 0.001. The network hidden size is 2048 units and the output encoding is 256D (except for the baseline 2D method). A dropout fraction of 0.1 is used for regularization during training. A unique set of 5000 data points is used for validation. Results on a separate 10,000 data point test set are shown in Figure 5.32. All of the continuous encoding methods perform similarly on this task, with the Legendre method performing the best overall.

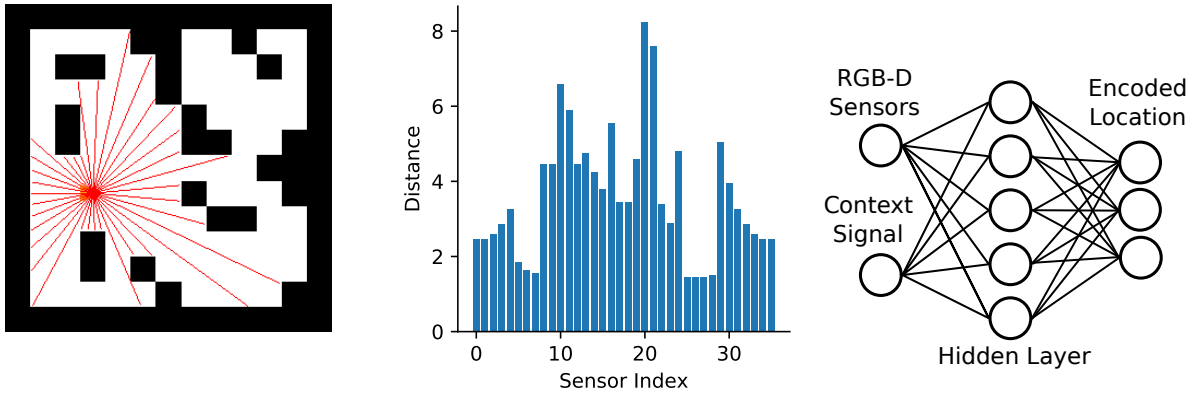


Figure 5.31: **Localization Task**. **Left**: Agent positioned in the environment with distance sensors shown. **Middle**: Distance readings from each sensor. **Right**: Localization network structure.

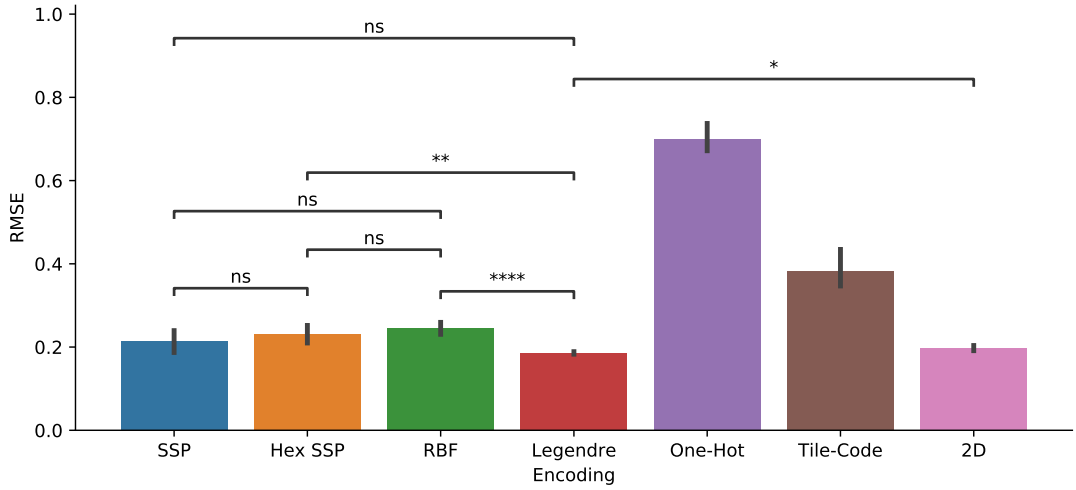


Figure 5.32: **Localization Results**. Training a network to output an encoding of the agent's current location based on 36 sensor readings. Each network was trained on 10 unique environments with a context signal to distinguish them. Each network has 2048 hidden units and a 256D output.

5.6.3 Spiking Neural Network

A spiking neural network can be trained on this task using the same techniques as the spiking policy network. A feedforward network of 4096 LIF neurons is used. This network is trained on 3,000,000 samples of sensor measurements and corresponding SSPs across 10 unique environments. An L_2 penalty is applied to the network weights during training with a factor of 10^{-5} applied to this penalty when combined with the mean square error loss of the network output. Training is conducted for 25 epochs using the Adam optimizer with a learning rate of 0.001.

Testing is accomplished by placing the agent at a location and injecting sensor measurements from that location into the network for 30 time-steps (30ms). The output spikes are filtered with a synaptic time constant of 0.1 and decoded as a 256 SSP. This procedure is repeated for a series of free space locations, with a visualization of the results shown in Figure 5.33. Each data point is coloured based on the ground truth location and is plotted at the decoded location. The RMSE of the decoded coordinates across all environments is 0.1097 for the spiking network and 0.0711 for the equivalent non-spiking network.

5.7 Integrated System

The localization, path integration, policy network and spatial reasoning afforded by SSPs can all be connected together to form an integrated system that allows an embodied agent to solve spatial navigation tasks. A spatial task that uses all of the components described in the previous sections is navigation towards semantically defined goals. Specifically, the agent is given an object to find, such as a guitar or a basketball, and it needs to issue motor commands to move in its environment to reach this object. This requires a representation in memory of each potential object associated to its location. The agent needs a way of localizing itself in the environment, in order to know where it is in its current mental map. It is also beneficial for the agent to be able to update its estimate based on self motion cues. Given an estimate of the location of the object of interest and the location of itself, the agent needs to produce a motor command that will move it towards the desired object. In general this policy can be more complicated than simply moving in the direction of the goal, as there may be obstacles blocking that path. The policy needs to take into account the shape of the environment and the pathways available between different regions. A diagram depicting the integration of components for this task is shown in Figure 5.34.

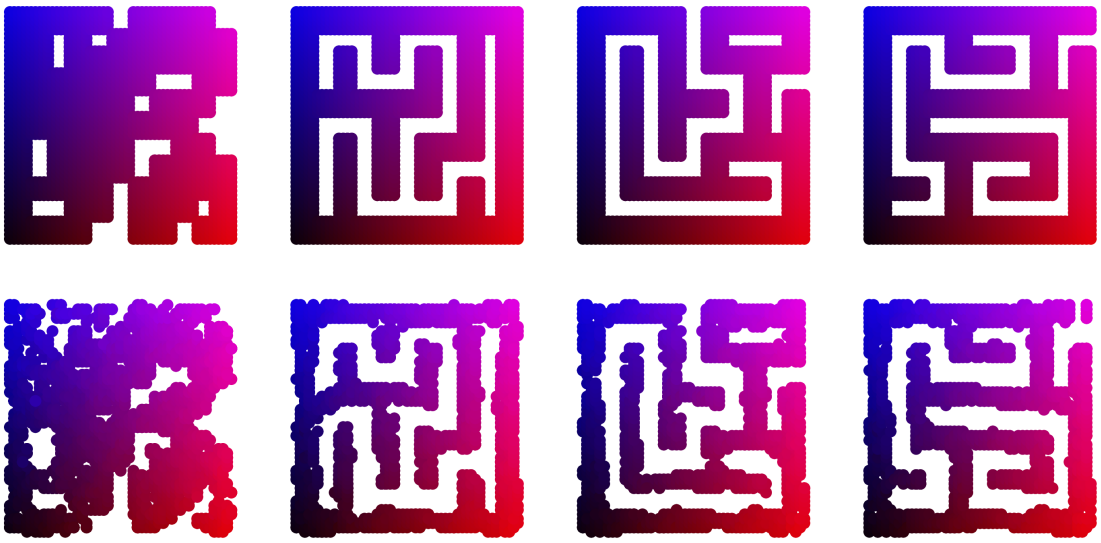


Figure 5.33: **Spiking Localization** Visualizing results on four different environments. **Top**: Ground truth. Each free space in the environment is coloured based on the x and y location. **Bottom**: Network output. The network is given a set of sensor readings from each location in the ground truth and produces as estimated location. A point is plotted at the location estimate, with the colour of the point corresponding to the ground truth.

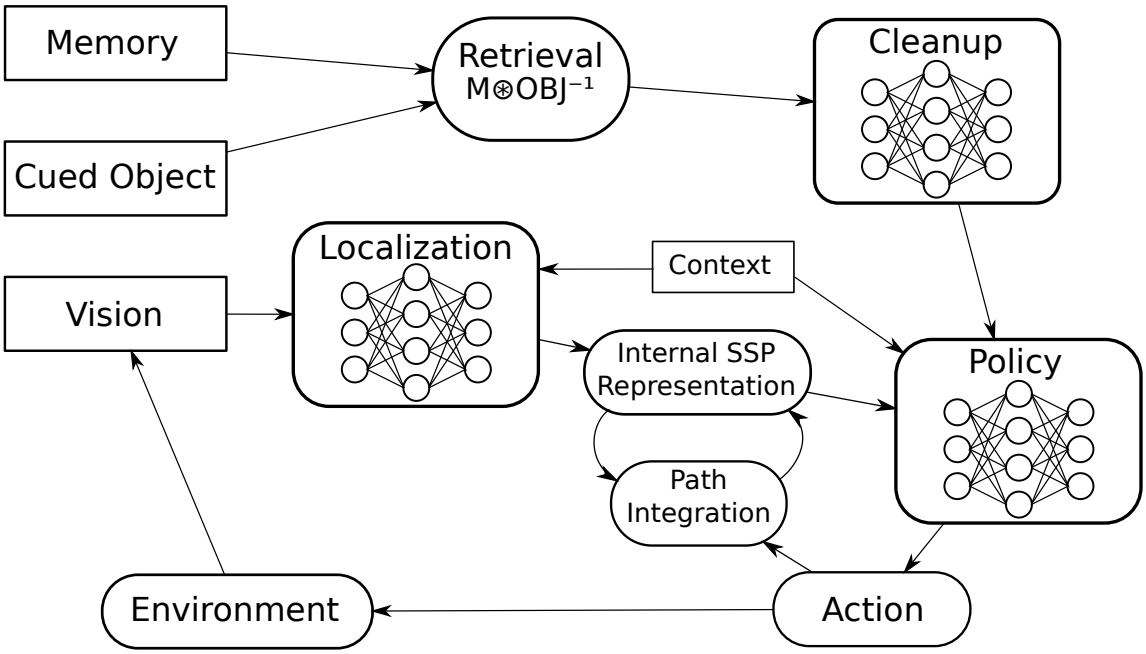


Figure 5.34: **Integrated System**. Every time-step the agent is given a series of vision sensor measurements from its current location, a context signal corresponding to which of several environments it is in, a representation of a cued goal object, and access to a memory containing a compressed semantic pointer representation of objects and their locations. The agent estimates its own location based on sensor data and context cues. This location estimate is combined with estimates from past time-steps updated through path integration. The agent computes its estimated goal location through memory retrieval and cleanup. Both location and goal estimates are fed into a policy network which computes a desired action. The action is taken in the environment and the sensor readings in the next time-step will be from the agent's new location.

5.7.1 Simulation Environment

In this work a simulated world is created for the agent rather than using a real-world robot. The use of simulation allows rapid iteration of design, parallel execution of experiments, accurate measurement of performance, and easily reproducible results.

The environment is written in the Python programming language. Python was chosen as it is the same language that Nengo and popular deep learning libraries such as PyTorch and TensorFlow are written in.

The simulation environment is designed as a stand-alone Python package that is open-source. This allows other researchers to use the simulator for their own projects or to create their own agents to compare to this work. Wrappers are provided for the popular OpenAI Gym [21] framework for reinforcement learning.

A variety of configuration options are available for this 2D environment. For the experiments below, the agent is configured for holonomic (i.e., unconstrained) movement and controlled by a 2D velocity command. Gaussian noise with $\sigma = 0.25$ is applied to the velocity signal at each time step. Sensor input consists of 36 RGB-D measurements from evenly spaced directions around the agent.

5.7.2 Performance

A 512D SSP is used for the internal representation. This SSP is generated by choosing random orientations and scales for the 85 orthogonal hexagonal toroids that make up the space. The policy network contains 2 hidden layers of 2048 neurons each. It is trained with Adam for 100 epochs on a dataset of 500,000 sample points. Regularization is achieved through a dropout fraction of 0.15 and noisy inputs. Each sample has Gaussian noise with $\sigma = 0.2$ added to all coordinates before they are encoded, and additional Gaussian noise with $\sigma = 0.01$ added to the encoded coordinate. The localization network contains 1 hidden layer with 4096 neurons. It is trained with Adam for 100 epochs on a dataset of 500,000 sample points using a dropout fraction of 0.15. The cleanup network contains 1 hidden layer with 512 neurons. It is trained with Adam on 14,000 pairs of noisy and clean vectors. The context signal for each of the environments the networks are trained on is a 256D random unit vector. In total, the network contains 8704 neurons.

Evaluation is performed across 10 different environment layouts. On each trial, the agent is placed at a random location in one environment with 10 different goal objects at random locations. The agent is cued with a semantic pointer for one of those goal objects

and the agent must then move to the location of that object. A trial is considered successful if the agent reaches the goal object within 1000 time-steps. Additional experiments are conducted replacing the localization and cleanup components with ground truth information to give an indication of the error induced by their neural network approximation. Results for 500 trials on each environment are shown in Figure 5.35. An example of the agent's behaviour based on the cued goal is shown in Figure 5.36.

A variant of the navigation system with all components implemented in spiking neurons is tested on the same task. Performance of the spiking network is similar to the standard neural network implementation. Counter-intuitively, replacing the localization component with ground truth position information leads to a slight decrease in performance. One explanation for this phenomena is that the localization network tends to have a bias towards returning estimates towards the center of hallways (see Figure 5.33) and the optimal actions for locations near the center are more directed towards the goal. It is likely that the learned policy is weaker when right up against a wall and could lead to the agent being stuck, while an imprecise localization can provide an input to the policy in a more well-learned regime, causing a better action to be chosen. Using a more accurate cleanup memory leads to an overall increase in performance. This is expected as even a perfect policy will fail if given an incorrect goal. The performance of the overall system is dependent on each component, and these components can be improved further through increased network complexity and training.

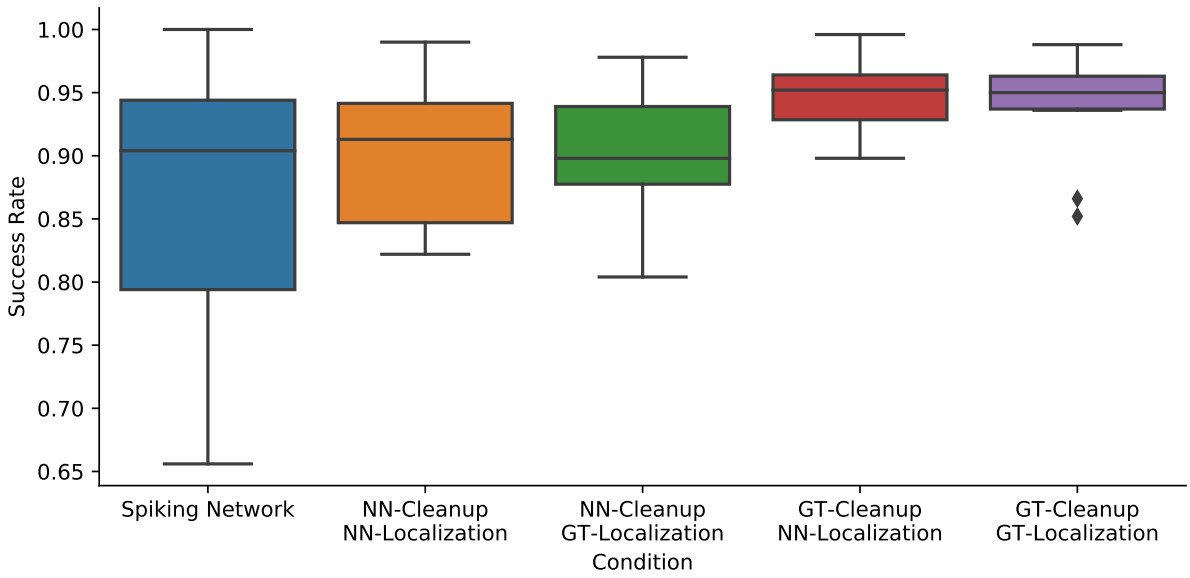


Figure 5.35: **Integrated System Results**. For the first condition, all components are implemented by a spiking neural network. In all other cases the policy is a standard artificial neural network. The cleanup memory and localization function are either a neural network (NN) or provide the ground truth (GT). Each data point is the mean success rate across 500 trials of a particular environment.

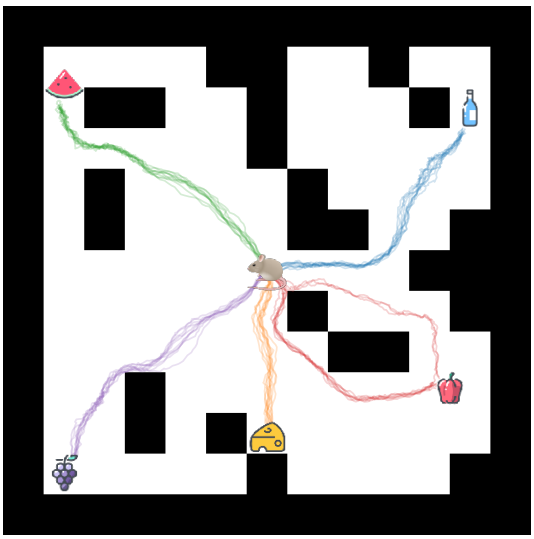


Figure 5.36: **Goal-Finding Behaviour**. This example shows the behaviour of the agent depending on which food item it is told to find. 10 trials for each goal are overlaid on the environment. The colour of the trajectory corresponds to which goal was cued. On each trial in this example, the agents starts at the location indicated by the mouse icon.

Chapter 6

General Encoding

Researchers have recently proposed that the brain areas involved in spatial memory are used for encoding more than just physical space and can represent and compute across more abstract spaces as well [44, 43, 158]. Studies in humans have implicated medial temporal lobe regions involved in navigating musical space [171], social space [170], and associational space [191]. Rats trained to use a joystick to manipulate sound exhibited frequency selective neural responses in the hippocampus and entorhinal cortex analogous to place cells and grid cells [6]. Memory techniques such as the 'method of loci' [20] allow semantic memories that do not normally have a spatial component to be encoded within a spatial context. This has been shown to dramatically improve recall for long lists of items.

6.1 Encoding Arbitrary Features

The SSP encoding method is not limited to spatial information, it can be applied to any metric quantity. This section will explore the effectiveness of using SSPs as a more general encoding mechanism for any continuous feature by using it across a variety of machine learning problems that have continuous inputs.

The Penn Machine Learning Benchmarks (PMLB) contain a growing set of over 285 curated benchmark datasets for evaluating and comparing machine learning algorithms for supervised classification and regression [130]. These datasets come from a wide range of application areas, including biomedical studies, image classification, spam detection, astronomical data, and many more. The number of datasets and variety of domains makes these benchmarks a good choice for testing the generalizability of the SSP encoding method.

Hidden Layer Size	512
Learning Rate	0.001
Max Epochs	600
Solver (>1000 samples)	Adam
Solver (<1000 samples)	L-BFGS

Table 6.1: **Hyperparameters**

The datasets in PMLB contain a mix of categorical and continuous features. Since SSPs are designed for continuous information, analysis is restricted to the subset of datasets that contain only continuous features. This results in 51 datasets for classification and 120 datasets for regression. The average number of samples in the regression datasets are about five times more than in the classification datasets. To save on computation, the regression datasets used are further trimmed down by selecting only those datasets that contain 10 or less features, resulting in 71 datasets. Overall, 122 datasets are used in the experiments.

6.1.1 Experiments

A standard pre-processing method for all domains is to scale the input features to have zero mean and unit variance. This pre-processing method is applied to all datasets. For the 'Normalized' case, this result is then fed into a neural network to be classified. For the 'SSP' case, each feature is encoded into a one dimensional SSP and then fed into the neural network. To ensure a fair comparison, the hyperparameters of the neural network structure and its training are held constant across both cases. These hyperparameters are shown in Table 6.1. The neural network implementation used in these experiments is the MLPClassifier and MLPRegressor from scikit-learn [136].

The size of the datasets in PMLB are not consistent and vary from 42 samples to 1,025,010 samples, with a mean of 22,669 and median of 500. 75% of the data is used for training and 25% is held out for a test set. To ensure consistency, the same training and test split is used across all experiments. For smaller datasets, the L-BFGS solver can often converge faster and perform better than Adam [19]. L-BFGS is used on all datasets with less than 1000 training samples, and Adam is used on all others. When training, 10% of the training data is held out as a validation set and early stopping is used if the performance on this validation set does not improve for 5 epochs. Training ends when either the early stopping criteria has been met, or after 600 epochs.

To form a baseline to situate these results, experiments are also run using One-Hot Encoding, Tile Coding, and Gaussian Basis Functions. Each feature is encoded independently using a 256D version of the encoding, ensuring that the dimensionality of each encoding for a particular dataset is the same. These encodings need to be defined over a fixed size space. The bounds are chosen to be at a distance of three times the variance around the mean. This encompasses 99.73% of normally distributed data. Features outside of these bounds can still be represented, although not as accurately.

Fixed Dimensional Encoding

Another way to encode features is to use a different axis vector for each feature value and then circularly convolve all of these vectors together. This is the method that is used in the previous chapters and results in an encoding with a dimensionality independent of the number of features. This encoding method can also be further modified in an analogous manner to the hexagonal SSP encoding. The transformation that maps the 3-D hexagonal coordinates to a 2-D plane can be extended to an N dimensional subspace embedded in an $N+1$ dimensional space. One way to formulate this transformation is to linearly project the positive axis-aligned unit vectors of the $N+1$ dimensional space to $N+1$ points in N dimensional space. A natural arrangement is to use a simplex, which is the generalization of the notion of a triangle to arbitrary dimensions [33]. This is an N dimensional shape which contains $N+1$ vertices. For a regular simplex, all side lengths are equal (distance between any two vertices is constant) and all vertices are equidistant from the centroid. These properties make the vertices of a regular simplex correspond to the largest arrangement of equal hyperspheres that can be in contact with one another.

Modifying the hexagonal SSP equation (3.25) to arbitrary feature dimensions produces Equation 6.1.

$$\begin{aligned} S_N(\mathbf{f}, d) &= X_1^{g_1} \circledast X_2^{g_2} \circledast ... \circledast X_{N+1}^{g_{N+1}} \\ \begin{bmatrix} g_1 & g_2 & ... & g_{N+1} \end{bmatrix} &= \begin{bmatrix} f_1 & f_2 & ... & f_N \end{bmatrix} A_{N\times(N+1)} \end{aligned} \tag{6.1}$$

Where $\mathbf{f}$ is the N dimensional feature vector to be encoded, $\mathbf{g}$ is the feature vector linearly transformed into $N+1$ dimensional space via the A matrix. The A matrix is formed by stacking the coordinates of the vertices of an N dimensional regular simplex to form an $N \times (N+1)$ transformation matrix. The simplex is designed to have a centroid at the origin and the distance from the origin to any vertex is 1. $X_i \in \mathbf{X}$ is a set of $N+1$ random unitary vectors of dimensionality d.

Encoding	Size
Hex SSP	$256 \times f$
SSP	$256 \times f$
Combined SSP	256
Simplex SSP	256
RBF	$256 \times f$
One Hot	$256 \times f$
Tile Code	$256 \times f$
Normalized	f

Table 6.2: **Encoding Sizes**

In the results section, the method where each axis represents an orthogonal feature is referred to as 'Combined SSP' (C-SSP in the plots) and the method using the vertices of a regular simplex is referred to as 'Simplex SSP' (S-SSP in the plots). An additional variant is using a single axis vector to encode each feature independently, but instead of generating this vector randomly it is generated from first defining a Hex SSP in 2D space and then just using one of the two resulting axis vectors. This variant is referred to as 'Hex SSP'. The encoding size for each method relative to the number of continuous features, f, is shown in Table 6.2.

6.1.2 Results

For the classification datasets, the metric reported is the mean accuracy of the network predictions. For the regression datasets, the metric used is the coefficient of determination:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{6.2}$$

where y_i is the target and $\hat{y}_i$ is network's prediction of the target for sample i and $\bar{y}$ is the mean of all n target values. The maximum score possible is 1.0. A model that would always predict the mean value of the target would achieve a score of 0.0. The score can be negative for models that perform worse than simply predicting the mean. Results are shown in Figure 6.1. As a form of outlier rejection, negative scores on the regression datasets are clipped to zero when generating the plot.

For the classification datasets many encodings perform similarly well; with the SSP, Hex SSP, RBF, and Normalized performing the best on average. For the regression datasets, the

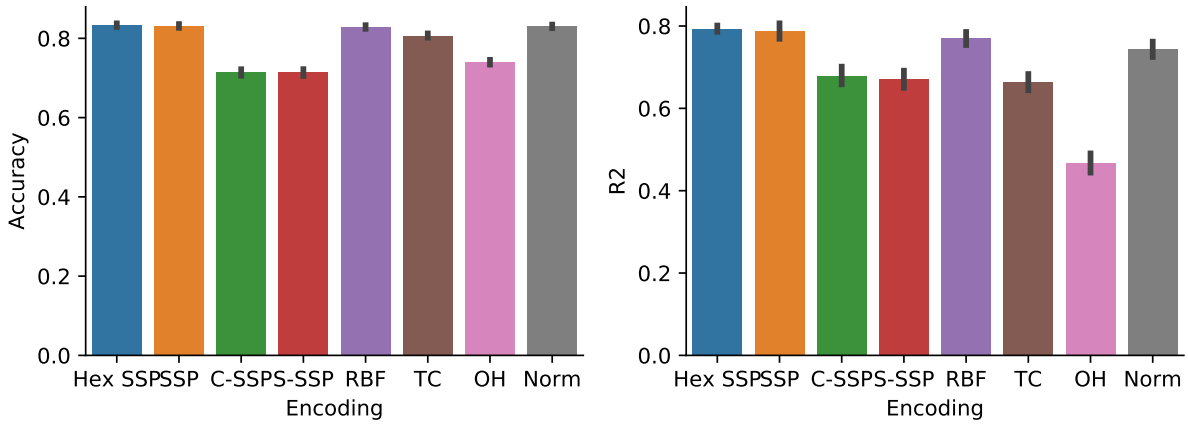


Figure 6.1: **Performance on all Datasets**. **Left**: Classification datasets. **Right**: Regression datasets.

same four encodings perform the best, but with more variance. Since these results combine scores from a variety of different datasets, they can be hard to interpret. For example, on some datasets an improvement of 1% accuracy could be considered a major improvement, but on others that same difference could be considered just noise. The variance in the scores from the regression datasets are higher than classification. One strong contributor for this variance is that there are negative scores on some datasets for each encoding, and for which dataset this occurs is dependent on the encoding itself.

Another method of analysing the results to work around the issues outlined above is to see which encoding performed the best on each dataset. In general when choosing among methods for a particular problem, the method that works best on that problem will be selected, regardless of how well it works on unrelated problems. Results using this metric are shown in Figure 6.2. Pairwise comparisons of the Hex SSP encoding to other top encodings across both classification and regression datasets are shown in Figure 6.3. The SSP encodings where each feature is treated as independent outperformed both methods where features were mixed into a single fixed dimensional vector. One potential explanation for this finding is that when these different features are combined into the same manifold, any function that is learned must be sensitive to all features. Often there are features that are more important than others, and some features have little to no importance at all. A change in any feature will cause the value in every dimension of the representation to change, making it more difficult to ignore unimportant features. Unlike representing physical space, each feature used is of a different kind. For example, one feature might be

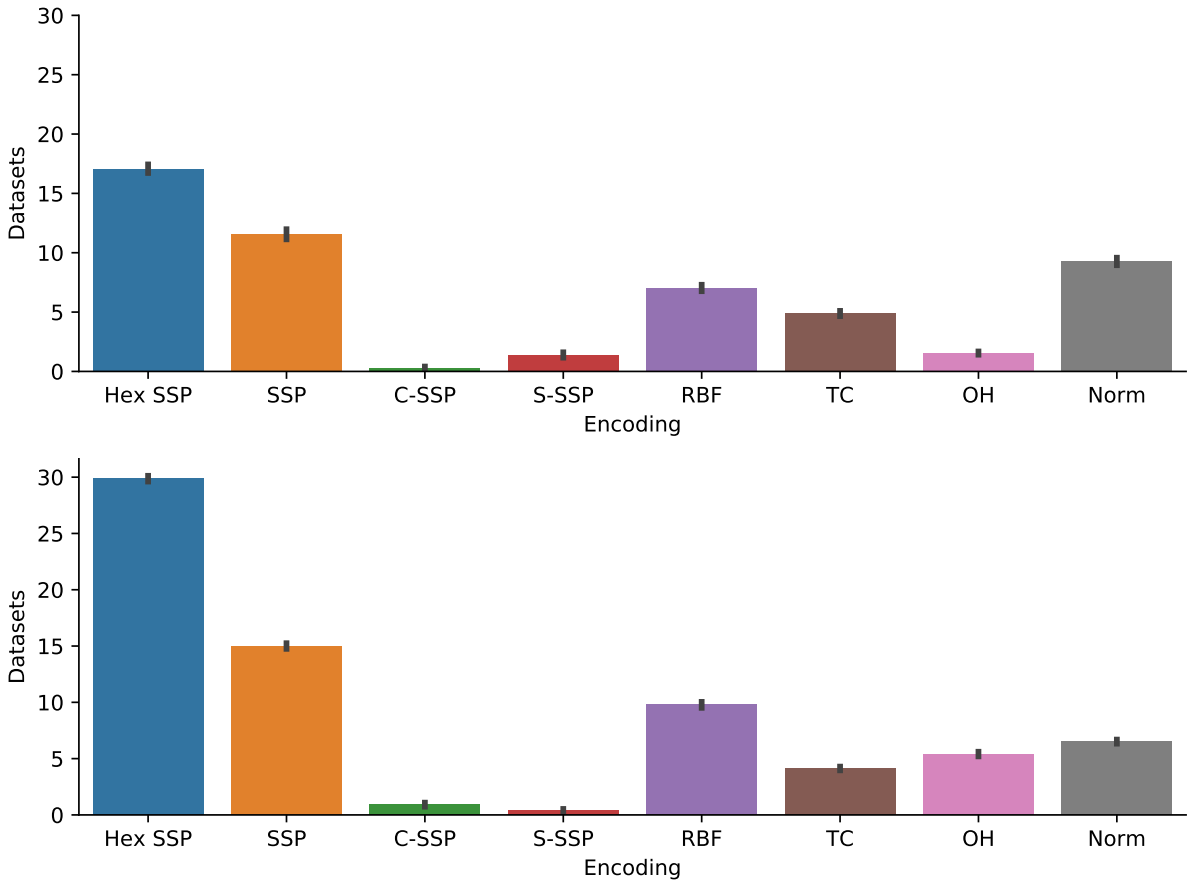


Figure 6.2: **Best Encoding on each Dataset**. Number of datasets for which each encoding outperformed all other encodings. Displayed with bootstrapped confidence intervals. **Top**: Classification datasets. **Bottom**: Regression datasets.

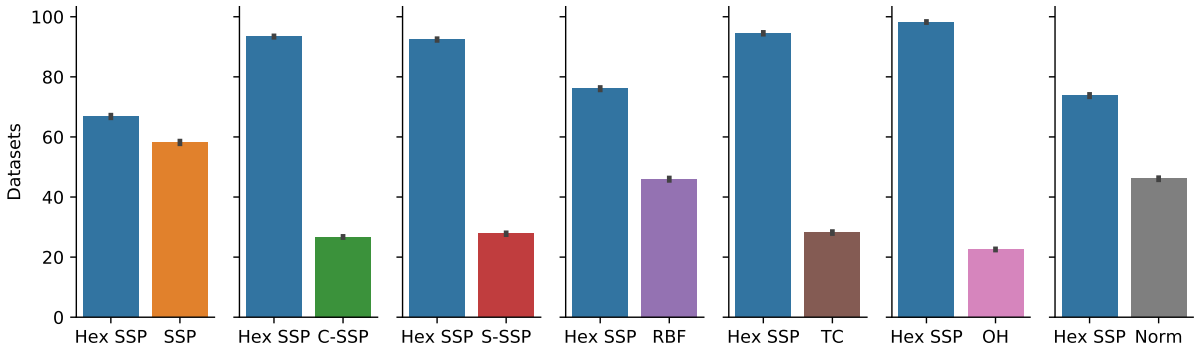


Figure 6.3: **Pairwise Comparison of Encodings**. Pairwise results comparing the Hex SSP encoding to the other methods.

a length, another a colour, and another time. Naively forming a metric space across these disparate features may not be as effective as first scaling and shifting them based on the domain or the class of function that needs to be computed. Another disadvantage of the dependent encodings is that information is lost when compressing the features to a fixed dimensional space. The more features present in the dataset, the greater the effect of this compression.

To give an indication of the variability of these results, bootstrapped confidence intervals [42] are calculated. This procedure produces an estimate of the true mean of a distribution where the number of samples are limited. Since a large number of datasets are used and there are many encodings tested for each dataset, it is not feasible to re-run the experiments using a multitude of unique random number seeds. Data points available are subsampled with replacement to produce many estimates of the mean. In this case there are 9 different runs available. Sets of 3 runs are sampled 100 times to produce the confidence interval about the mean. As can be seen in the figures, the findings are quite robust to random variation.

Overall the Hex SSP encoding is the best choice to use on the majority of these datasets compared to any other encoding tested. This however does not mean that it should always be chosen, as the results are problem dependent. An interesting line of future work would be to discover the properties of a dataset that make it more amenable to one encoding method over another, and predict which encoding will perform the best on a new dataset based on meta features such as the number of classes, types of features, ranges of feature values, etc.

6.2 Encoding Properties

This section explores various properties of SSPs compared to other encoding methods to gain insight into why SSPs are effective on so many problems.

6.2.1 Distance

High dimensional spaces can be very difficult to visualize, but the distances between representations in the encoded space can help give insights into how that space behaves. Two common distance measures to use are Euclidean distance (straight line distance between two vectors) and Cosine distance (angle between two vectors).

Figure 6.4 shows the two distance metrics for each encoding. The plots are generated by computing the distance from the representation of the origin to a series of points along a line that crosses the origin. Ideally the distance will be zero at the origin and increase monotonically the further the point is from the origin. Symmetry in the distance measure in both directions from the origin is also desired.

For SSPs, further away locations are nearly orthogonal, but for the other methods they are exactly orthogonal. Though this may be a disadvantage for some uses of the representation, a threshold can be applied to make detecting further away distances just as easy. All of the continuous encodings behave approximately linearly near the reference point. The discrete encodings do not have the resolution to be linear, but maintain monotonicity.

The SSP methods produce a symmetric distance measure. From a given starting point, representing a location d units away in a particular direction is always the same distance as the representation for a location d units away in the opposite direction. This is true for both the Cosine and Euclidean distance metric. This also holds for any starting location, not just the origin as shown in the figure. In addition, if two points are offset by the same displacement, their distance in the represented space will remain the same. These properties are summarized in Equation 6.3:

$$\begin{aligned} D(S(x), S(x+a)) &= D(S(x), S(x-a)) \\ D(S(x), S(x+a)) &= D(S(x+b), S(x+b+a)) \end{aligned} \tag{6.3}$$

where D is the distance metric (Euclidean or Cosine), S is a function that converts a vector to an SSP, and x, a, and b are vectors of the same dimensionality. These relations hold for the Euclidean distance metric due to the fact that SSPs map points onto a Clifford

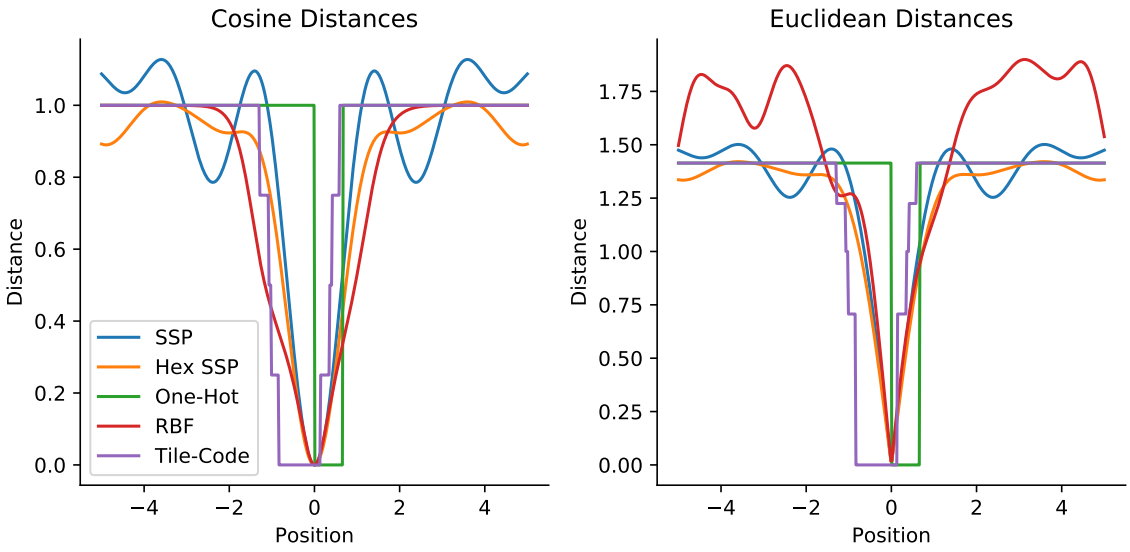


Figure 6.4: **Distance in Encoded Space**. For each encoding, the distance from the representation for (0, 0) to points along the line from (-5, 0) to (5, 0) is computed. The two distance measures reported are Euclidean (straight line distance between two vectors) and Cosine (one minus the cosine of the angle between two vectors). 256D encodings are used. The plotted Euclidean distance for Tile-Code is scaled down by $\sqrt{n_{tiles}}$ to be aligned visually with the other encodings.

torus, preserving spatial relations. In a similar manner, since movement in a particular direction along this manifold corresponds to moving along an arc of constant curvature, the cosine distance will be directly related to the distance moved in the input space. None of the other methods used have these properties.

Consistent distance properties at different locations could help in learning operations that generalize to different parts of the space. An example of such an operation is shifting the represented location via circular convolution. Convolving the same vector with any location causes the same shift. For the other encodings, any shift operation will have to perform a different computation at each location.

6.2.2 Decoding

It is useful to be able to reconstruct a 2D coordinate from the higher dimensional representation. For example, a system may perform computations in the higher dimensional space, but need to control an actuator or visualization system with a lower dimensional signal. To quantify the ability for each representation to be decoded, a neural network is trained on pairs of encoded coordinates and raw 2D coordinates to learn how to decode each method. Results are shown in Figure 6.5.

6.2.3 Smoothness

When observing the output of the decoding network as it trains, it is evident that some encodings already produce a smooth manifold in the output from the beginning, and training is mostly a matter of stretching and flattening the result to match the ground truth. This process can be seen in Figure 6.6.

Starting with an output that is already smooth may be an indication that the neural network will have success in learning the desired transformation. Visualization of the output of each encoding when presented to a randomly initialized neural network is shown in Figure 6.7. In this case there are 512 hidden units and 2 output units. Output is coloured based on the location of the input coordinate, so that similar colours share similar locations. To give a baseline for the encodings, the 2D coordinates are also fed into a random network and visualized alongside the other methods.

As can be seen qualitatively, the 2D, SSP, and RBF produce smooth outputs, while the One-Hot, Tile-Code, and Legendre methods do not.

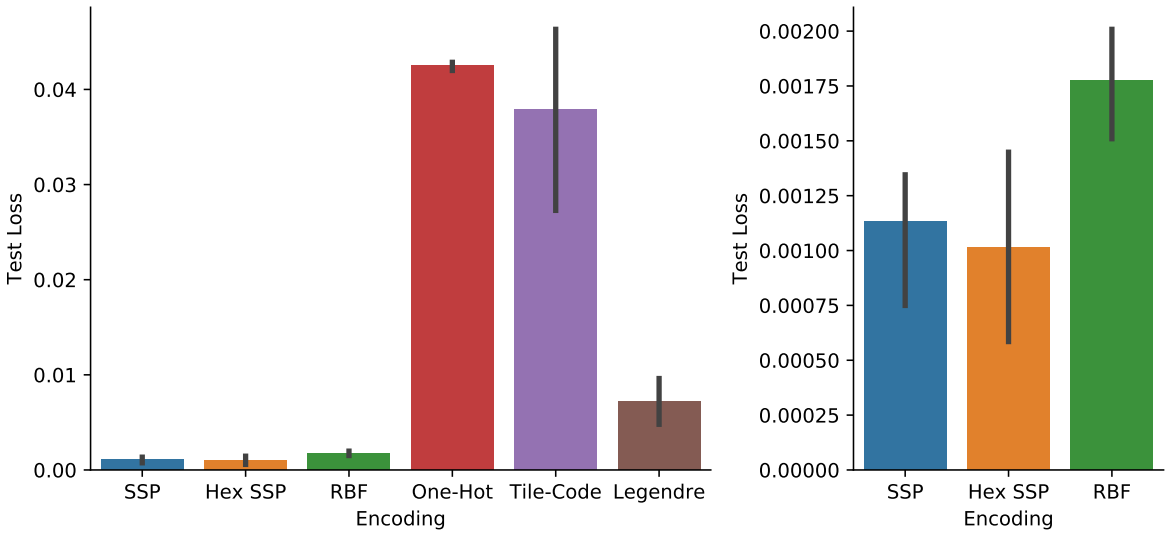


Figure 6.5: **Decoding 2D Position**. The network is trained to output a 2D position from a 256D encoded input. Trained with coordinate values between -5 and 5. **Left**: The test loss after 50 epochs training with Adam. **Right**: Zoomed in results of the top 3 encodings.

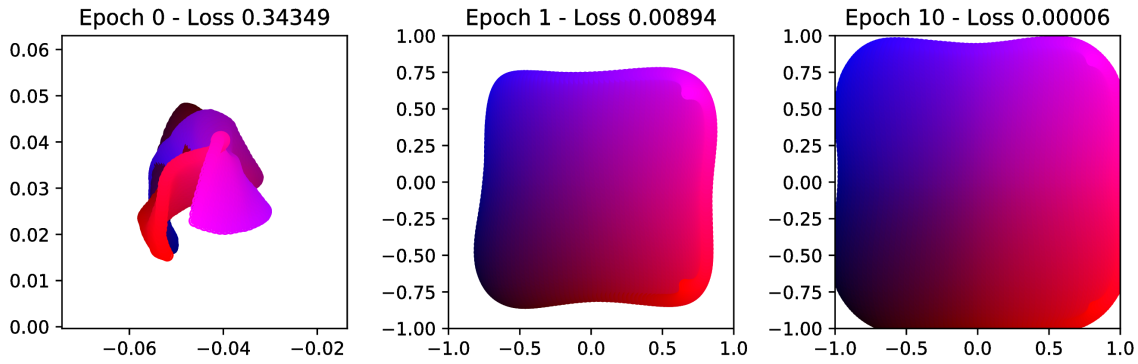


Figure 6.6: **Training Decode Network**. Output of the network at different stages of training to decode an SSP to the corresponding 2D coordinate (values between -1 and 1). Output is coloured based on the location of the input coordinate by modifying the RGB channels. Value of the blue channel corresponds to x position, value of the red channel corresponds to y position.

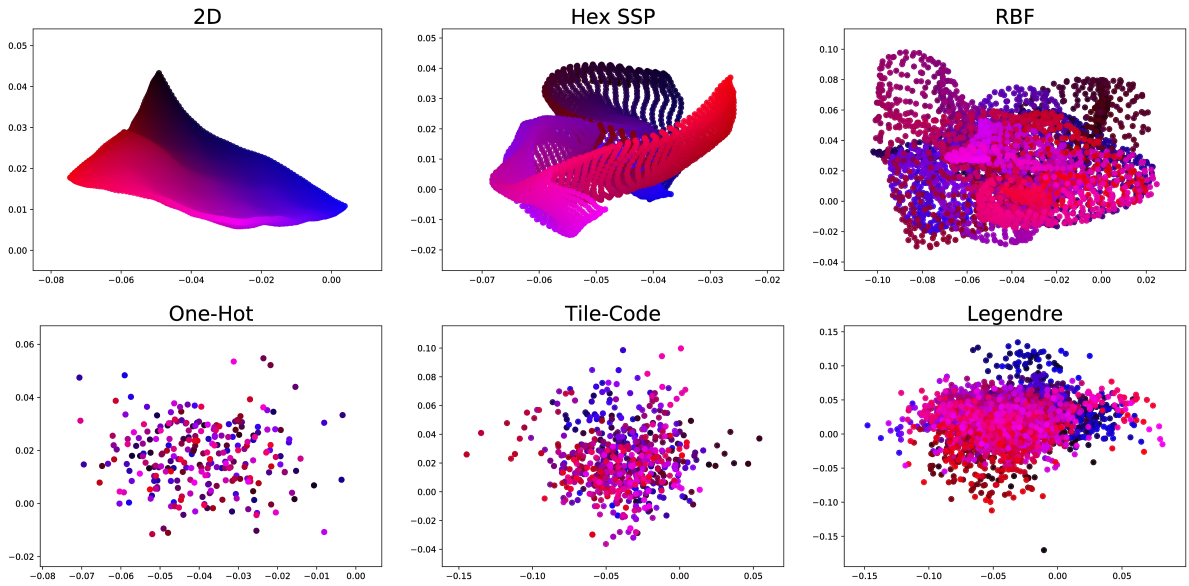


Figure 6.7: **Output of an Untrained Neural Network**. Each encoded point is run through a neural network with 512 hidden units, 2 output units, and ReLU activation functions. Output is coloured based on the location of the input coordinate by modifying the RGB channels. Value of the blue channel corresponds to x position, value of the red channel corresponds to y position.

6.2.4 Function Learning

Another useful property is the ease with which simple functions can be learned. More complicated functions can often be defined in terms of simpler building blocks. If the smaller components are difficult to learn, it is likely that a function defined by a combination would be difficult as well.

Three simple functions of two points are considered for this experiment: the distance between the two points, the relative direction between the points, and the location of the midpoint. Two variations of presenting the input to the network are considered. The first is summing the encoded values together, the second is concatenating them. Results are shown for various 256D encoding methods in Figure 6.8. In all cases a neural network with 512 hidden units and ReLU activation functions is trained for 50 epochs using the Adam optimizer. A dataset of 10,000 training and 10,000 test samples is used. Training and test loss converged for all cases. Each variant of the experiment is run 5 times with different random number seeds for both the dataset generation and the encoding definition.

The networks trained using SSP encodings produce superior results in comparison to the other methods. This gives an indication that a neural network using SSPs will likely outperform the other methods on more complex functions that can be built from these simple functions as well. In addition to the desired function being learned, there is an inherent 'decode' required to produce the output. Based on the results in Figure 6.5 (note the scale of the y-axis) all encodings are able to perform this decode fairly accurately, so the majority of the discrepancy in the results is likely due to learning the desired function itself.

One explanation for why SSPs perform so well on these simple functions is because the consistent relationship between encoded locations encourages more general computations to be learned, as opposed to trying to memorize the dataset. For example, moving continuously from the SSP representation of one location to another corresponds to moving along an arc on a torus. Therefore, the mean of two representations will be on a point interior to the torus, where the closest location on the arc itself to this point is the exact middle of the two representations. In this manner, one way to learn how to compute the mean of two points is to learn an attractor to the hypersurface that all encoded points reside on. This same attractor can be applied to any points in the space. In contrast, the high dimensional surface that the other encodings lie on is irregular or non-continuous, making such a general function difficult to produce. The SSP methods showed the greatest improvement over other methods on the midpoint task, which could be due to this property of the representation space.

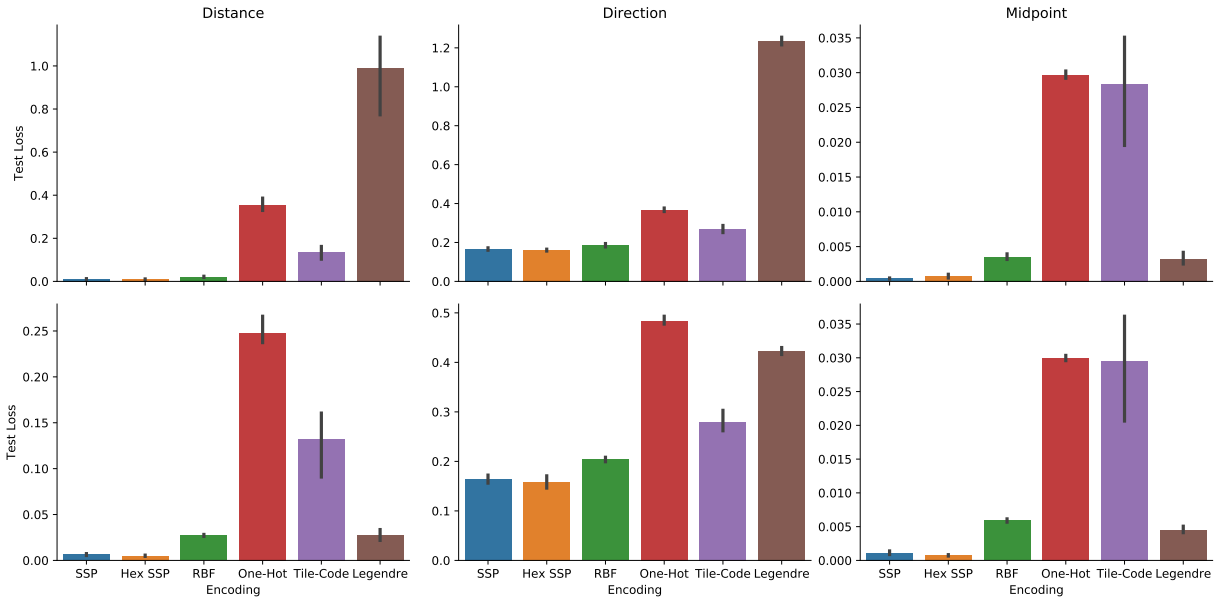


Figure 6.8: **Learning Simple Functions**. A network with 512 hidden units is trained to compute functions of two encoded coordinates. Each encoding converts the 2D coordinate to a 256D vector. **Top**: The encoded coordinates presented to the network are summed together to produce a single 256D vector. **Bottom**: The encoded coordinates are concatenated together to produce a 512D input vector. **Left**: The network must produce the distance between the two 2D coordinates. **Middle**: The network must produce the direction between the two 2D coordinates. Since the coordinates are considered unordered, the direction is a value between 0 and π. **Right**: The network must produce the coordinate corresponding to the midpoint between the two inputs.

Chapter 7

Conclusions

In this thesis I have introduced spatial semantic pointers (SSPs), a method of representing continuous valued information as a semantic pointer. This allows continuous quantities to be used within the Semantic Pointer Architecture (SPA); a theory which describes how models of higher-level cognition based on compressed vectors can be implemented in a biologically plausible neural network. I have demonstrated that SSPs provide many useful properties for a spatial representation, including the ability to represent multiple locations simultaneously, represent regions of space, and shift items in a memory. A spiking neural network can efficiently implement SSPs and their associated operations. A single population of neurons can represent a memory with multiple objects at various locations. The memory can be queried for the location of a particular object, or for what objects are at a particular location or region of space.

While the non-spiking implementation of networks using SSPs tend to have slightly better performance, the ability to construct spiking networks provides many advantages. The first is facilitation of comparisons with neural data recorded from animals. This allows the accuracy of the model to be quantified and for detailed predictions to be made on the neural level. Another benefit of spiking networks is that they are implementable in neuromorphic hardware that can take advantage of the parallel processing and lower power consumption that comes from communicating with spikes.

The space of vectors that can be used for SSPs forms a continuous manifold that preserves Euclidean geometry. The choice of two axis vectors defines a 2D toroidal surface within this manifold that all encoded coordinates will lie upon. This toroidal surface can be designed such that projections into different orthogonal sub-spaces produce hexagonal tori with different orientations and scales. Neurons sensitive to these sub-spaces have

hexagonally symmetric firing fields and resemble the properties of grid cell modules found in the brain (they maintain the same orientation and scale, but can vary in phase). The SSP formulations consistent with the observed neural tuning found in animals performed the best on a navigation task.

In addition to use of SSPs for modelling spatial cognition, this representation has also proven to be useful for machine learning problems. No explicit bounds need to be defined for this encoding, reducing the hyperparameters required to define this representation. Any real value can be encoded and is guaranteed to produce a vector of unit length with all elements between -1 and 1, ensuring input is in an ideal space for a neural network. Consistent Euclidean and cosine distance properties are maintained in the encoded space, helping functions to generalize to different parts of the space. Compared to other common encoding methods, SSPs offer superior performance when training a neural network to produce a navigational policy.

Furthermore, I have demonstrated that improvement in performance of artificial neural networks using SSP encoding is not limited to spatial tasks. Across a set of 122 diverse problems, SSP based encodings performed the best on a significant majority (61.2%) of the datasets.

An overarching theme of this work is that representation is critical for both biological and artificial neural networks. Insights from the kinds of representations plausible for brains to use can be applied to machine learning systems. Despite the many differences in these two paradigms, the common goal of an intelligent system can arise through similar functions and representations.

7.1 Future Work

There are many avenues of future research that may stem from this work. Potential directions are outlined in the sections below.

7.1.1 Exploring the effects of different axis vectors

In the majority of the experiments in this work the vectors used to define the x and y axes for the SSPs were chosen randomly. An interesting line of future work would be to explore the effects of different axis vector choices on the usefulness of the representation for different tasks. It may be possible to design an axis vector to be optimal for a particular

use case. For a given task, there may more effective ways to learn the basis to use rather than have it predetermined. The relation between the x and y axis vectors is important. Exploring this relationship may lead to insights on how to design optimal sets of vectors to cover N-dimensional space.

7.1.2 Reinforcement Learning

One natural extension to this work is to explore in greater detail learning behavioural policies online through reinforcement learning. The current work illustrates the effectiveness of an SSP representation after the learning process has stabilized, as well as preliminary work demonstrating that reinforcement learning can be used to learn an effective policy. How this spatial representation influences learning new environments through experience is still an open question.

7.1.3 Encoding Continuous Signals

The experiments on policy learning for solving mazes clearly demonstrates the effectiveness of SSPs encoding spatial locations, but there may be other broad classes of problems that SSPs excel on. Preliminary experiments on the PMLB datasets indicate that SSPs offer a good representation on some of the datasets but not others. There may be a link between the effectiveness of a continuous value encoding and the class of signal that is being encoded or the type of function that is being learned. One line of future work is to characterize which tasks are better suited for an SSP representation.

7.1.4 Applications on Real-World Robotics

Another line of future work is to apply SSPs to real-time systems in the physical world. Neuromorphic hardware is being developed that can run spiking neural networks in real-time with a fraction of the power usage of conventional computing platforms [105, 16]. Since SSPs can be implemented efficiently using spiking neural networks, this representation can be used with these specialized hardware systems.

7.1.5 Detailed Modelling of Biological Systems

In addition to applications in artificial systems, there is opportunity to extend this work to improve models of biological systems.

One example would be extending the SPAUN [48] model to handle tasks involving spatial cognition. SPAUN is currently the world's largest functional brain model, and allowing it to be embodied in a robotic agent with spatial reasoning abilities would be a major improvement.

References

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016.

[2] Monireh Abdoos, Nasser Mozayani, and Ana LC Bazzan. Hierarchical control of traffic signals using q-learning with tile coding. *Applied intelligence*, 40(2):201–213, 2014.

[3] Forest Agostinelli, Matthew Hoffman, Peter Sadowski, and Pierre Baldi. Learning activation functions to improve deep neural networks. *arXiv preprint arXiv:1412.6830*, 2014.

[4] James S Albus. Data storage in the cerebellar model articulation controller (cmac). *Journal of Dynamic Systems, Measurement, and Control*, 97(3):228–233, 1975.

[5] Jose M Alvarez and Mathieu Salzmann. Learning the number of neurons in deep networks. In *Advances in Neural Information Processing Systems*, pages 2270–2278, 2016.

[6] Dmitriy Aronov, Rhino Nevers, and David W Tank. Mapping of a non-spatial dimension by the hippocampal–entorhinal circuit. *Nature*, 543(7647):719–722, 2017.

[7] Jan Balaguer, Hugo Spiers, Demis Hassabis, and Christopher Summerfield. Neural mechanisms of hierarchical planning in a virtual subway network. *Neuron*, 90(4):893–903, 2016.

[8] Andrea Banino, Caswell Barry, Benigno Uria, Charles Blundell, Timothy Lillicrap, Piotr Mirowski, Alexander Pritzel, Martin J Chadwick, Thomas Degris, Joseph Mo-

dayil, et al. Vector-based navigation using grid-like representations in artificial agents. *Nature*, 557(7705):429, 2018.

[9] Marian Stewart Bartlett, Gwen Littlewort, Claudia Lainscsek, Ian Fasel, and Javier Movellan. Machine learning methods for fully automatic recognition of facial expressions and facial actions. In *2004 IEEE International Conference on Systems, Man and Cybernetics (IEEE Cat. No. 04CH37583)*, volume 1, pages 592–597. IEEE, 2004.

[10] William F Basener. *Topology and Its Applications*. John Wiley & Sons, Inc, 2006.

[11] Roberto Battiti. First-and second-order methods for learning: between steepest descent and newton's method. *Neural computation*, 4(2):141–166, 1992.

[12] Trevor Bekolay, James Bergstra, Eric Hunsberger, Travis DeWolf, Terrence C Stewart, Daniel Rasmussen, Xuan Choo, Aaron Russell Voelker, and Chris Eliasmith. Nengo: a python tool for building large-scale functional brain models. *Frontiers in neuroinformatics*, 7, 2013.

[13] Trevor Bekolay, Carter Kolbeck, and Chris Eliasmith. 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*, volume 35, 2013.

[14] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

[15] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48, 2009.

[16] Peter Blouw, Xuan Choo, Eric Hunsberger, and Chris Eliasmith. Benchmarking keyword spotting efficiency on neuromorphic hardware. In *Proceedings of the 7th Annual Neuro-inspired Computational Elements Workshop*, pages 1–8, 2019.

[17] Peter Blouw and Chris Eliasmith. Inferential role semantics for natural language. In *Proceedings of the 39th Annual Conference of the Cognitive Science Society*, pages 142–147, Philadelphia, Pennsylvania, 2017. Cognitive Science Society.

[18] Charlotte N Boccara, Francesca Sargolini, Veslemøy Hult Thoresen, Trygve Solstad, Menno P Witter, Edvard I Moser, and May-Britt Moser. Grid cells in pre-and parasubiculum. *Nature neuroscience*, 13(8):987, 2010.

[19] Raghu Bollapragada, Dheevatsa Mudigere, Jorge Nocedal, Hao-Jun Michael Shi, and Ping Tak Peter Tang. A progressive batching l-bfgs method for machine learning. *arXiv preprint arXiv:1802.05374*, 2018.

[20] Gordon H Bower. Analysis of a mnemonic device: Modern psychology uncovers the powerful components of an ancient system for improving memory. *American Scientist*, 58(5):496–510, 1970.

[21] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[22] Neil Burgess. Spatial cognition and the brain. *Annals of the New York Academy of Sciences*, 1124(1):77–97, 2008.

[23] Neil Burgess, Caswell Barry, and John O'keefe. An oscillatory interference model of grid cell firing. *Hippocampus*, 17(9):801–812, 2007.

[24] Anthony N Burkitt. A review of the integrate-and-fire neuron model: I. homogeneous synaptic input. *Biological cybernetics*, 95(1):1–19, 2006.

[25] Patrick Byrne, Suzanna Becker, and Neil Burgess. Remembering the past and imagining the future: a neural model of spatial memory and imagery. *Psychological review*, 114(2):340, 2007.

[26] Richard Cattrambone and Stellan Ohlsson, editors. *A Spiking Neuron Model of Serial-Order Recall*, Portland, OR, 08/2010 2010. Cognitive Science Society.

[27] Martin J Chadwick, Amy EJ Jolly, Doran P Amos, Demis Hassabis, and Hugo J Spiers. A goal direction signal in the human entorhinal/subicular region. *Current Biology*, 25(1):87–92, 2015.

[28] Feng-Xuan Choo. *Spaun 2.0: Extending the Worlds Largest Functional Brain Model*. Phd thesis, University of Waterloo, 2018.

[29] Allen Chieng Hoon Choong and Nung Kion Lee. Evaluation of convolutionary neural networks modeling of dna sequences using ordinal versus one-hot encoding method. In *2017 International Conference on Computer and Drone Applications (IConDA)*, pages 60–65. IEEE, 2017.

[30] Laura Lee Colgin, Edvard I Moser, and May-Britt Moser. Understanding memory through hippocampal remapping. *Trends in neurosciences*, 31(9):469–477, 2008.

[31] John Conklin and Chris Eliasmith. A controlled attractor network model of path integration in the rat. *Journal of computational neuroscience*, 18(2):183–203, 2005.

[32] James W Cooley and John W Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of computation*, 19(90):297–301, 1965.

[33] Harold Scott Macdonald Coxeter. *Regular polytopes*. Courier Corporation, 1973.

[34] Eric Crawford, Matthew Gingerich, and Chris Eliasmith. Biologically plausible, human-scale knowledge representation. In *35th Annual Conference of the Cognitive Science Society*, pages 412–417, 2013.

[35] Eric Crawford, Matthew Gingerich, and Chris Eliasmith. Biologically plausible, human-scale knowledge representation. *Cognitive Science*, 2015.

[36] Balázs Csanád Csáji et al. Approximation with artificial neural networks. *Faculty of Sciences, Etvs Lornd University, Hungary*, 24(48):7, 2001.

[37] M Day, R Langston, and Richard GM Morris. Glutamate-receptor-mediated encoding and retrieval of paired-associate learning. *Nature*, 424(6945):205–209, 2003.

[38] William de Cothi and Hugo J Spiers. Spatial cognition: Goal-vector cells in the bat hippocampus. *Current Biology*, 27(6):R239–R241, 2017.

[39] Edsger W Dijkstra et al. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[40] Grayson M DuBois and Joshua L Phillips. Working memory concept encoding using holographic reduced representations. In *MAICS*, pages 137–144, 2017.

[41] Nicole Dumont and Chris Eliasmith. Accurate representation for spatial cognition using grid cells. In *42nd Annual Meeting of the Cognitive Science Society*, Toronto, ON, 2020. Cognitive Science Society.

[42] Bradley Efron. Bootstrap confidence intervals for a class of parametric problems. *Biometrika*, 72(1):45–58, 1985.

[43] Howard Eichenbaum and Neal J Cohen. Can we reconcile the declarative memory and spatial navigation views on hippocampal function? *Neuron*, 83(4):764–770, 2014.

[44] Howard Eichenbaum, Paul Dudchenko, Emma Wood, Matthew Shapiro, and Heikki Tanila. The hippocampus, memory, and place cells: is it spatial memory or a memory space? *Neuron*, 23(2):209–226, 1999.

[45] Arne D Ekstrom, Michael J Kahana, Jeremy B Caplan, Tony A Fields, Eve A Isham, Ehren L Newman, and Itzhak Fried. Cellular networks underlying human spatial navigation. *Nature*, 425(6954):184–188, 2003.

[46] Chris Eliasmith. *How to build a brain: A neural architecture for biological cognition*. Oxford University Press, 2013.

[47] Chris Eliasmith and Charles H Anderson. *Neural engineering: Computation, representation, and dynamics in neurobiological systems*. MIT press, 2003.

[48] Chris Eliasmith, Terrence C Stewart, Xuan Choo, Trevor Bekolay, Travis DeWolf, Yichuan Tang, and Daniel Rasmussen. A large-scale model of the functioning brain. *science*, 338(6111):1202–1205, 2012.

[49] Tamir Eliav, Maya Geva-Sagiv, Michael M Yartsev, Arseny Finkelstein, Alon Rubin, Liora Las, and Nachum Ulanovsky. Nonoscillatory phase coding and synchronization in the bat hippocampal formation. *Cell*, 175(4):1119–1130, 2018.

[50] Meng Joo Er, Shiqian Wu, Juwei Lu, and Hock Lye Toh. Face recognition with radial basis function (rbf) neural networks. *IEEE transactions on neural networks*, 13(3):697–710, 2002.

[51] Scott Fujimoto, Herke Van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*, 2018.

[52] Marianne Fyhn, Torkel Hafting, Alessandro Treves, May-Britt Moser, and Edvard I Moser. Hippocampal remapping and grid realignment in entorhinal cortex. *Nature*, 446(7132):190–194, 2007.

[53] Stephen I Gallant and Phil Culliton. Positional binding with distributed representations. In *2016 International Conference on Image, Vision and Computing (ICIVC)*, pages 108–113. IEEE, 2016.

[54] Xiang Gao and Tao Zhang. Unsupervised learning to detect loops using deep neural networks for visual slam system. *Autonomous robots*, 41(1):1–18, 2017.

[55] Ross W Gayler. Vector symbolic architectures answer Jackendoff's challenges for cognitive neuroscience. *arXiv preprint cs/0412059*, 2004.

[56] Pierre Georges-François, Edmund T Rolls, and Robert G Robertson. Spatial view cells in the primate hippocampus: allocentric view not head direction or eye position or place. *Cerebral Cortex*, 9(3):197–212, 1999.

[57] Mariana Gil, Mihai Ancau, Magdalene I Schlesiger, Angela Neitz, Kevin Allen, Rodrigo J De Marco, and Hannah Monyer. Impaired path integration in mice with disrupted grid cell firing. *Nature neuroscience*, 21(1):81–91, 2018.

[58] Lisa M Giocomo, Eric A Zilli, Erik Fransén, and Michael E Hasselmo. Temporal frequency of subthreshold oscillations scales with entorhinal grid cell field spacing. *Science*, 315(5819):1719–1722, 2007.

[59] Fernand Gobet, Peter CR Lane, Steve Croker, Peter CH Cheng, Gary Jones, Iain Oliver, and Julian M Pine. Chunking mechanisms in human learning. *Trends in cognitive sciences*, 5(6):236–243, 2001.

[60] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[61] Jan Gosmann. Precise multiplications with the NEF. Technical report, University of Waterloo, December 2015.

[62] Jan Gosmann. *An Integrated Model of Context, Short-Term, and Long-Term Memory*. Phd thesis, University of Waterloo, 2018.

[63] Jan Gosmann and Chris Eliasmith. Optimizing semantic pointer representations for symbol-like processing in spiking neural networks. *PloS one*, 11(2), 2016.

[64] Jan Gosmann and Chris Eliasmith. Vector-derived transformation binding: An improved binding operation for deep symbol-like processing in neural networks. *Neural Computation*, 31(5):849–869, 05 2019.

[65] Roddy M Grieves and Kate J Jeffery. The representation of space in the brain. *Behavioural Processes*, 135:113–131, 2017.

[66] Alexis Guanella, Daniel Kiper, and Paul Verschure. A model of grid cells based on a twisted torus topology. *International journal of neural systems*, 17(04):231–240, 2007.

[67] DJ Gunaratnam and JS Gero. Effect of representation on the performance of neural networks in structural engineering applications. *Computer-Aided Civil and Infrastructure Engineering*, 9(2):97–108, 1994.

[68] Torkel Hafting, Marianne Fyhn, Tora Bonnevie, May-Britt Moser, and Edvard I Moser. Hippocampus-independent phase precession in entorhinal grid cells. *Nature*, 453(7199):1248–1252, 2008.

[69] Torkel Hafting, Marianne Fyhn, Sturla Molden, May-Britt Moser, and Edvard I Moser. Microstructure of a spatial map in the entorhinal cortex. *Nature*, 436(7052):801–806, 2005.

[70] T Hartley and N Burgess. Models of spatial cognition. encyclopedia of cognitive science. ed. nadel l, 2003.

[71] Jennifer Hasler and Harry Bo Marr. Finding a roadmap to achieve large neuromorphic hardware systems. *Frontiers in neuroscience*, 7:118, 2013.

[72] Joao F Henriques and Andrea Vedaldi. Mapnet: An allocentric spatial memory for mapping environments. In *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8476–8484, 2018.

[73] D Hilbert. Über die stetige abbildung einer linie aufein flächenstück. *Mathematische Annalen*, 38:459–460, 1891.

[74] Ashley Hill, Antonin Raffin, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, Rene Traore, Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. Stable baselines. https://github.com/hill-a/stable-baselines, 2018.

[75] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[76] Etsuro Hori, Eiichi Tabuchi, Nobuhisa Matsumura, Ryoi Tamura, Satoshi Eifuku, Shunro Endo, Hisao Nishijo, and Taketoshi Ono. Representation of place by monkey hippocampal neurons in real and virtual translocation. *Hippocampus*, 13(2):190–196, 2003.

[77] Eric Hunsberger. *Spiking Deep Neural Networks: Engineered and Biological Approaches to Object Recognition*. Phd thesis, University of Waterloo, 2018.

[78] Maciej M Jankowski and Shane M OMara. Dynamics of place, boundary and object encoding in rat anterior claustrum. *Frontiers in behavioral neuroscience*, 9:250, 2015.

[79] Maciej M Jankowski, Johannes Passecker, Md Nurul Islam, Seralynne Vann, Jonathan T Erichsen, John P Aggleton, and Shane M OMara. Evidence for spatially-responsive neurons in the rostral thalamus. *Frontiers in behavioral neuroscience*, 9:256, 2015.

[80] Kathryn J Jeffery. Place cells, grid cells, attractors, and remapping. *Neural plasticity*, 2011, 2011.

[81] Michael N Jones and Douglas JK Mewhort. Representing word meaning and order information in a composite holographic lexicon. *Psychological review*, 114(1):1, 2007.

[82] Ivana Kajić, Jan Gosmann, Brent Komer, Ryan W. Orr, Terrence C. Stewart, and Chris Eliasmith. A biologically constrained model of semantic memory search. In *Proceedings of the 39th Annual Conference of the Cognitive Science Society*, London, UK, 2017. Cognitive Science Society.

[83] Ivana Kaji, Tobias Schrder, Terrence C. Stewart, and Paul Thagard. The semantic pointer theory of emotion: Integrating physiology, appraisal, and construction. *Cognitive Systems Research*, 2019.

[84] Pentti Kanerva. The spatter code for encoding concepts at many levels. In *ICANN94*, pages 226–229. Springer, 1994.

[85] David R Kelley, Jasper Snoek, and John L Rinn. Basset: learning the regulatory code of the accessible genome with deep convolutional neural networks. *Genome research*, 26(7):990–999, 2016.

[86] D Kingma and J Ba. Adam: A method for stochastic optimization in: Proceedings of the 3rd international conference for learning representations (iclr15). *San Diego*, 2015.

[87] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2014.

[88] Felix Klein. Zur nicht-euklidischen geometrie. *Mathematische Annalen*, 37(4):544–572, 1890.

[89] Alexander Kleiner, Markus Dietl, and Bernhard Nebel. Towards a life-long learning soccer agent. In *Robot Soccer World Cup*, pages 126–134. Springer, 2002.

[90] Kunikazu Kobayashi, Tadashi Kurano, Takashi Kuremoto, and Masanao Obayashi. Cooperative behavior acquisition in multi-agent reinforcement learning system using attention degree. In *International Conference on Neural Information Processing*, pages 537–544. Springer, 2012.

[91] Brent Komer and Chris Eliasmith. Efficient navigation using a scalable, biologically inspired spatial representation. In *42nd Annual Meeting of the Cognitive Science Society*, Toronto, ON, 2020. Cognitive Science Society.

[92] Brent Komer, Terrence C. Stewart, Aaron R. Voelker, and Chris Eliasmith. A neural representation of continuous space using fractional binding. In *41st Annual Meeting of the Cognitive Science Society*, Montreal, QC, 2019. Cognitive Science Society.

[93] Stephen M Kosslyn, Thomas M Ball, and Brian J Reiser. Visual images preserve metric spatial information: evidence from studies of image scanning. *Journal of experimental psychology: Human perception and performance*, 4(1):47, 1978.

[94] Mark A Kramer. Nonlinear principal component analysis using autoassociative neural networks. *AIChE journal*, 37(2):233–243, 1991.

[95] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[96] Emilio Kropff, James E Carmichael, May-Britt Moser, and Edvard I Moser. Speed cells in the medial entorhinal cortex. *Nature*, 523(7561):419–424, 2015.

[97] Julija Krupic, Neil Burgess, and John O'Keefe. Spatially periodic cells are neither formed from grids nor poor isolation. *arXiv preprint arXiv:1512.06248*, 2015.

[98] Julija Krupic, Neil Burgess, and John OKeefe. Neural representations of location composed of spatially periodic bands. *Science*, 337(6096):853–857, 2012.

[99] Ilya Kuzovkin, Raul Vicente, Mathilde Petton, Jean-Philippe Lachaux, Monica Baciu, Philippe Kahane, Sylvain Rheims, Juan R Vidal, and Jaan Aru. Activations of deep convolutional neural networks are aligned with gamma band activity of human visual cortex. *Communications biology*, 1(1):1–12, 2018.

[100] Alban Laflaquière and Michael Garcia Ortiz. Unsupervised emergence of egocentric spatial structure from sensorimotor prediction. In *Advances in Neural Information Processing Systems*, pages 7156–7166, 2019.

[101] Adam Daniel Laud. *Theory and application of reward shaping in reinforcement learning*. Phd thesis, University of Illinois at Urbana-Champaign, 2004.

[102] Máté Lengyel, Zoltán Szatmáry, and Péter Érdi. Dynamically detuned oscillations account for the coupled rate and temporal code of place cell firing. *Hippocampus*, 13(6):700–714, 2003.

[103] Moshe Leshno, Vladimir Ya Lin, Allan Pinkus, and Shimon Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural networks*, 6(6):861–867, 1993.

[104] Colin Lever, Stephen Burton, Ali Jeewajee, John O'Keefe, and Neil Burgess. Boundary vector cells in the subiculum of the hippocampal formation. *Journal of Neuroscience*, 29(31):9771–9777, 2009.

[105] Chit-Kwan Lin, Andreas Wild, Gautham N Chinya, Yongqiang Cao, Mike Davies, Daniel M Lavery, and Hong Wang. Programming spiking neural networks on intels loihi. *Computer*, 51(3):52–61, 2018.

[106] Thomas Lu, Aaron R. Voelker, Brent Komer, and Chris Eliasmith. Representing spatial relations with fractional binding. In *41st Annual Meeting of the Cognitive Science Society*, Montreal, QC, 2019. Cognitive Science Society.

[107] Zhou Lu, Hongming Pu, Feicheng Wang, Zhiqiang Hu, and Liwei Wang. The expressive power of neural networks: A view from the width. In *Advances in neural information processing systems*, pages 6231–6239, 2017.

[108] Nandor Ludvig, Hai M Tang, Baiju C Gohil, and Juan M Botero. Detecting location-specific neuronal firing rate increases in the hippocampus of freely-moving monkeys. *Brain research*, 1014(1):97–109, 2004.

[109] Adam H Marblestone, Greg Wayne, and Konrad P Kording. Toward an integration of deep learning and neuroscience. *Frontiers in computational neuroscience*, 10:94, 2016.

[110] Alexander Mathis, Andreas VM Herz, and Martin Stemmler. Optimal population codes for space: grid cells outperform place cells. *Neural computation*, 24(9):2280–2317, 2012.

[111] Nobuhisa Matsumura, Hisao Nishijo, Ryoi Tamura, Satoshi Eifuku, Shunro Endo, and Taketoshi Ono. Spatial-and task-dependent neuronal responses during real and

virtual translocation in the monkey hippocampal formation. *Journal of Neuroscience*, 19(6):2381–2393, 1999.

[112] BL McNaughton, Carol A Barnes, and J O'keefe. The contributions of position, direction, and velocity to single unit activity in the hippocampus of freely-moving rats. *Experimental Brain Research*, 52(1):41–49, 1983.

[113] Bruce L McNaughton, Francesco P Battaglia, Ole Jensen, Edvard I Moser, and May-Britt Moser. Path integration and the neural basis of the'cognitive map'. *Nature Reviews Neuroscience*, 7(8):663–678, 2006.

[114] Paul A Merolla, John V Arthur, Rodrigo Alvarez-Icaza, Andrew S Cassidy, Jun Sawada, Filipp Akopyan, Bryan L Jackson, Nabil Imam, Chen Guo, Yutaka Nakamura, et al. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345(6197):668–673, 2014.

[115] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[116] Michael J Milford, Gordon F Wyeth, and David Prasser. Ratslam: a hippocampal model for simultaneous localization and mapping. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 1, pages 403–408. IEEE, 2004.

[117] George A Miller. *WordNet: An electronic lexical database*. MIT press, 1998.

[118] Stefan Milz, Georg Arbeiter, Christian Witt, Bassam Abdallah, and Senthil Yogamani. Visual slam for automated driving: Exploring the applications of deep learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 247–257, 2018.

[119] Edvard I Moser, Emilio Kropff, and May-Britt Moser. Place cells, grid cells, and the brain's spatial representation system. *Annu. Rev. Neurosci.*, 31:69–89, 2008.

[120] Robert U Muller, Elizabeth Bostock, Jeffrey S Taube, and John L Kubie. On the directional firing properties of hippocampal place cells. *Journal of Neuroscience*, 14(12):7235–7251, 1994.

[121] Robert U Muller and John L Kubie. The effects of changes in the environment on the spatial firing of hippocampal complex-spike cells. *Journal of Neuroscience*, 7(7):1951–1968, 1987.

[122] Andrew Mundy, James Knight, Terrence C Stewart, and Steve Furber. An efficient spinnaker implementation of the neural engineering framework. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2015.

[123] Alexander Neckar, Sam Fok, Ben V Benjamin, Terrence C Stewart, Nick N Oza, Aaron R Voelker, Chris Eliasmith, Rajit Manohar, and Kwabena Boahen. Braindrop: A mixed-signal neuromorphic architecture with a dynamical systems-based programming model. *Proceedings of the IEEE*, 107(1):144–164, 2018.

[124] Douglas L Nelson, Cathy L McEvoy, and Thomas A Schreiber. The university of south florida free association, rhyme, and word fragment norms. *Behavior Research Methods, Instruments, & Computers*, 36(3):402–407, 2004.

[125] Erkki Oja. Neural networks, principal components, and subspaces. *International journal of neural systems*, 1(01):61–68, 1989.

[126] John O'Keefe and Neil Burgess. Geometric determinants of the place fields of hippocampal neurons. *Nature*, 381(6581):425–428, 1996.

[127] John O'Keefe, Neil Burgess, James G Donnett, Kathryn J Jeffery, and Eleanor A Maguire. Place cells, navigational accuracy, and the human hippocampus. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 353(1373):1333–1340, 1998.

[128] John O'Keefe and Jonathan Dostrovsky. The hippocampus as a spatial map. preliminary evidence from unit activity in the freely-moving rat. *Brain research*, 34(1):171–175, 1971.

[129] John O'Keefe and Michael L Recce. Phase relationship between hippocampal place units and the eeg theta rhythm. *Hippocampus*, 3(3):317–330, 1993.

[130] Randal S. Olson, William La Cava, Patryk Orzechowski, Ryan J. Urbanowicz, and Jason H. Moore. Pmlb: a large benchmark suite for machine learning evaluation and comparison. *BioData Mining*, 10(1):36, Dec 2017.

[131] Jeff Orchard. Oscillator-interference models of path integration do not require theta oscillations. *Neural computation*, 27(3):548–560, 2015.

[132] Jeff Orchard, Hao Yang, and Xiang Ji. Does the entorhinal cortex use the fourier transform? *Frontiers in computational neuroscience*, 7:179, 2013.

[133] Y-H Pao and Yoshiyasu Takefuji. Functional-link net computing: theory, system architecture, and functionalities. *Computer*, 25(5):76–79, 1992.

[134] EunHye Park, Dino Dvorak, and André A Fenton. Ensemble place codes in hippocampus: Ca1, ca3, and dentate gyrus place cells have multiple place fields in large environments. *PloS one*, 6(7):e22349, 2011.

[135] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[136] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

[137] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[138] Adrien Peyrache, Marie M Lacroix, Peter C Petersen, and György Buzsáki. Internally organized mechanisms of the head direction sense. *Nature neuroscience*, 18(4):569–575, 2015.

[139] Tony A Plate. Estimating analogical similarity by dot-products of holographic reduced representations. In *Advances in neural information processing systems*, pages 1109–1116, 1994.

[140] Tony A Plate. Holographic reduced representations. *IEEE Transactions on Neural networks*, 6(3):623–641, 1995.

[141] Joseph D Prusa and Taghi M Khoshgoftaar. Improving deep neural network design with new text data representations. *Journal of Big Data*, 4(1):7, 2017.

[142] James Ranck Jr. Head direction cells in the deep cell layer of dorsal presubiculum in freely moving rats. In *Society for Neuroscience Abstracts*, volume 10, 1984.

[143] Daniel Rasmussen. Nengodl: Combining deep learning and neuromorphic modelling methods. *Neuroinformatics*, 17(4):611–628, 2019.

[144] Florian Raudies and Michael E Hasselmo. Modeling boundary vector cell firing given optic flow as a cue. *PLoS computational biology*, 8(6):e1002553, 2012.

[145] Sean N Riley and Jim Davies. A spiking neural network model of spatial and visual mental imagery. *Cognitive Neurodynamics*, pages 1–13, 2019.

[146] Françcois Rivest, Yoshua Bengio, and John Kalaska. Brain inspired reinforcement learning. In *Advances in neural information processing systems*, pages 1129–1136, 2005.

[147] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[148] Edmund Rolls. The mechanisms for pattern completion and pattern separation in the hippocampus. *Frontiers in systems neuroscience*, 7:74, 2013.

[149] Edmund T Rolls et al. Spatial view cells and the representation of place in the primate hippocampus. *Hippocampus*, 9(4):467–480, 1999.

[150] Benjamin Rosman and Subramanian Ramamoorthy. Learning spatial relationships between objects. *The International Journal of Robotics Research*, 30(11):1328–1342, 2011.

[151] Alon Rubin, Michael M Yartsev, and Nachum Ulanovsky. Encoding of head direction by hippocampal place cells in bats. *Journal of Neuroscience*, 34(3):1067–1080, 2014.

[152] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[153] Deepti Moyi Sahoo and Snehashish Chakraverty. Functional link neural network approach to solve structural system identification problems. *Neural Computing and Applications*, 30(11):3327–3338, 2018.

[154] Alexei Samsonovich and Bruce L McNaughton. Path integration and cognitive mapping in a continuous attractor neural network model. *Journal of Neuroscience*, 17(15):5900–5920, 1997.

[155] Sandeep Silwal (https://math.stackexchange.com/users/138892/sandeep-silwal). Sum of squared inner product of vector with spokes around unit circle is constant. Mathematics Stack Exchange. URL: https://math.stackexchange.com/q/3053523 (version: 2018-12-27).

[156] Ayelet Sarel, Arseny Finkelstein, Liora Las, and Nachum Ulanovsky. Vectorial representation of spatial goals in the hippocampus of bats. *Science*, 355(6321):176–180, 2017.

[157] Francesca Sargolini, Marianne Fyhn, Torkel Hafting, Bruce L McNaughton, Menno P Witter, May-Britt Moser, and Edvard I Moser. Conjunctive representation of position, direction, and velocity in entorhinal cortex. *Science*, 312(5774):758–762, 2006.

[158] Daniela Schiller, Howard Eichenbaum, Elizabeth A Buffalo, Lila Davachi, David J Foster, Stefan Leutgeb, and Charan Ranganath. Memory and space: towards an understanding of the cognitive map. *Journal of Neuroscience*, 35(41):13904–13911, 2015.

[159] William E Skaggs, James J Knierim, Hemant S Kudrimoti, and Bruce L McNaughton. A model of the neural basis of the rat's sense of direction. In *Advances in neural information processing systems*, pages 173–180, 1995.

[160] Trygve Solstad, Charlotte N Boccara, Emilio Kropff, May-Britt Moser, and Edvard I Moser. Representation of geometric borders in the entorhinal cortex. *Science*, 322(5909):1865–1868, 2008.

[161] Sameet Sreenivasan and Ila Fiete. Grid cells generate an analog error-correcting code for singularly precise neural computation. *Nature neuroscience*, 14(10):1330, 2011.

[162] Kenneth O. Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127, 2002.

[163] Hanne Stensola, Tor Stensola, Trygve Solstad, Kristian Frøland, May-Britt Moser, and Edvard I Moser. The entorhinal grid map is discretized. *Nature*, 492(7427):72–78, 2012.

[164] Terrence C. Stewart, Xuan Choo, and Chris Eliasmith. Dynamic behaviour of a spiking model of action selection in the basal ganglia. In *10th International Conference on Cognitive Modeling*, 2010.

[165] Terrence C. Stewart, Yichuan Tang, and Chris Eliasmith. A biologically realistic cleanup memory: Autoassociation in spiking neurons. *Cognitive Systems Research*, 12:84–92, 2011.

[166] Peter Stone and Richard S Sutton. Scaling reinforcement learning toward robocup soccer. In *Icml*, volume 1, pages 537–544. Citeseer, 2001.

[167] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[168] Jeffrey S Taube. The head direction signal: origins and sensory-motor integration. *Annu. Rev. Neurosci.*, 30:181–207, 2007.

[169] Jeffrey S Taube, Robert U Muller, and James B Ranck. Head-direction cells recorded from the postsubiculum in freely moving rats. i. description and quantitative analysis. *Journal of Neuroscience*, 10(2):420–435, 1990.

[170] Rita Morais Tavares, Avi Mendelsohn, Yael Grossman, Christian Hamilton Williams, Matthew Shapiro, Yaacov Trope, and Daniela Schiller. A map for social navigation in the human brain. *Neuron*, 87(1):231–243, 2015.

[171] Sundeep Teki, Sukhbinder Kumar, Katharina von Kriegstein, Lauren Stewart, C Rebecca Lyness, Brian CJ Moore, Brian Capleton, and Timothy D Griffiths. Navigating the auditory scene: an expert role for the hippocampus. *Journal of Neuroscience*, 32(35):12251–12257, 2012.

[172] LT Thompson and PJ Best. Place cells and silent cells in the hippocampus of freely-behaving rats. *Journal of Neuroscience*, 9(7):2382–2390, 1989.

[173] Kurt A Thoroughman and Reza Shadmehr. Learning of action through adaptive combination of motor primitives. *Nature*, 407(6805):742–747, 2000.

[174] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.

[175] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

[176] Anne Treisman. The binding problem. *Current opinion in neurobiology*, 6(2):171–178, 1996.

[177] Oliver Trujillo. A spiking neural model of episodic memory encoding and replay in hippocampus. Masters thesis, University of Waterloo, Waterloo, Ontario, 2014.

[178] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research*, 11(Dec):3371–3408, 2010.

[179] Aaron Voelker, Ivana Kajić, and Chris Eliasmith. Legendre memory units: Continuous-time representation in recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 15544–15553, 2019.

[180] Aaron R. Voelker, Eric Crawford, and Chris Eliasmith. Learning large-scale heteroassociative memories in spiking neurons. In *Unconventional Computation and Natural Computation*, London, Ontario, 2014. Springer International Publishing.

[181] Klaus Volkert. Space forms: a history. *Bulletin of the Manifold Atlas*, 2013.

[182] Hongwei Wang and Hong Gu. Prediction of chaotic time series based on neural network with legendre polynomials. In *International Symposium on Neural Networks*, pages 836–843. Springer, 2009.

[183] Samuel Justo Waskow and Ana Lcia Cetertich Bazzan. Improving space representation in multiagent learning via tile coding. In *Brazilian Symposium on Artificial Intelligence*, pages 153–162. Springer, 2010.

[184] Toshihiko Watanabe, Yuichi Saito, Takeshi Kamai, and Tomoki Ishimaru. Tile coding based camera modeling for 3d sensing. In *2014 IEEE 6th International Conference on Awareness Science and Technology (iCAST)*, pages 1–6. IEEE, 2014.

[185] JR Weeks. *The Shape of Space*. Marcel Dekker, Inc., 1985.

[186] Aldis P Weible, David C Rowland, Caitlin K Monaghan, Nicholas T Wolfgang, and Clifford G Kentros. Neural correlates of long-term object memory in the mouse anterior cingulate cortex. *Journal of Neuroscience*, 32(16):5598–5608, 2012.

[187] Shuhuan Wen, Yanfang Zhao, Xiao Yuan, Zongtao Wang, Dan Zhang, and Luigi Manfredi. Path planning for active slam based on deep reinforcement learning under unknown environments. *Intelligent Service Robotics*, pages 1–10, 2020.

[188] Matthew A Wilson and Bruce L McNaughton. Dynamics of the hippocampal ensemble code for space. *Science*, 261(5124):1055–1058, 1993.

[189] Yoko Yamaguchi, Naoyuki Sato, Hiroaki Wagatsuma, Zhihua Wu, Colin Molter, and Yoshito Aota. A unified view of theta-phase coding in the entorhinal–hippocampal system. *Current opinion in neurobiology*, 17(2):197–204, 2007.

[190] Daniel LK Yamins, Ha Hong, Charles F Cadieu, Ethan A Solomon, Darren Seibert, and James J DiCarlo. Performance-optimized hierarchical models predict neural responses in higher visual cortex. *Proceedings of the National Academy of Sciences*, 111(23):8619–8624, 2014.

[191] Dagmar Zeithamova, April L Dominick, and Alison R Preston. Hippocampal and ventral medial prefrontal activation during retrieval-mediated learning supports novel inference. *Neuron*, 75(1):168–179, 2012.

[192] Guodong Zhang, Chaoqi Wang, Bowen Xu, and Roger Grosse. Three mechanisms of weight decay regularization. *arXiv preprint arXiv:1810.12281*, 2018.

[193] Honghui Zhang and Joshua Jacobs. Traveling theta waves in the human hippocampus. *Journal of Neuroscience*, 35(36):12477–12487, 2015.

[194] Jingwei Zhang, Lei Tai, Joschka Boedecker, Wolfram Burgard, and Ming Liu. Neural slam: Learning to explore with external memory. *arXiv preprint arXiv:1706.09520*, 2017.

[195] Kechen Zhang. Representation of spatial orientation by the intrinsic dynamics of the head-direction cell ensemble: a theory. *Journal of Neuroscience*, 16(6):2112–2126, 1996.

[196] Xiang Zhang and Yann LeCun. Which encoding is the best for text classification in chinese, english, japanese and korean? *arXiv preprint arXiv:1708.02657*, 2017.

[197] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In *Advances in neural information processing systems*, pages 649–657, 2015.

[198] Eric A Zilli and Michael E Hasselmo. Coupled noisy spiking neurons as velocity-controlled oscillators in a model of grid cell spatial firing. *Journal of Neuroscience*, 30(41):13850–13860, 2010.

APPENDICES

Appendix A

Source Code

Source code for this work is organized across multiple repositories based on functionality.

A.1 Packages for General Use

Code designed as an installable Python package for use in other projects.

The source code for generating 2D environments that an agent can move around in is available online at: https://github.com/bjkomer/gridworlds.

The source code for creating and using Spatial Semantic Pointers is available online at: https://github.com/bjkomer/spatial-semantic-pointers.

A.2 Scripts for Experiments

Code for reproducing all of the experiments from various chapters in this thesis.

The source code for running the experiments in the first section of Chapter 4 is available online at: https://github.com/ctn-waterloo/cogsci2019-ssp.

The source code for the policy, localization, cleanup, and integrated system experiments in Chapter 5 is available online at: https://github.com/bjkomer/ssp-navigation.

The source code for running all other experiments and generating figures is available online at: https://github.com/bjkomer/ssp-experiments.