Model-based reinforcement learning and navigation in animals and machines

THÈSE N^{O} 8950 (2018)

PRÉSENTÉE LE 19 OCTOBRE 2018
À LA FACULTÉ INFORMATIQUE ET COMMUNICATIONS
LABORATOIRE DE CALCUL NEUROMIMÉTIQUE (IC/SV)
PROGRAMME DOCTORAL EN NEUROSCIENCES

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Dane Sterling CORNEIL

acceptée sur proposition du jury:

Prof. K. Hess Bellwald, présidente du jury
Prof. W. Gerstner, directeur de thèse
Prof. J. Peters, rapporteur
Prof. M. Botvinick, rapporteur
Prof. A. Ijspeert, rapporteur

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Suisse
2018

There’s no need to build a labyrinth when the entire universe is one.
— Jorge Luis Borges, *The Aleph* (1949)

Acknowledgements

First, I want to thank my supervisor, Wulfram Gerstner, for his support and guidance throughout the many research projects that caught my interest during my time at the LCN. To the other members of the LCN during my time here – Olivia, Alex, Laureline, Willem, Moritz, Aditya, Skander, Christian, Carlos, Felipe, Friedemann, Bernd, Mohammadjavad, Tilo, Chiara, Florian, Hesam, Samuel, Lorric, Nicolas, David, Tim, Danilo; thank you for all of the talks, feedback, and happy hour reminders. I especially want to thank Johanni Brea for the many great conversations and the fruitful collaboration.

Outside of the LCN, I want to thank all the members of the Catalyst; I've loved every minute of being on stage with you. To Danny and Saee: you're the best friends and past/future co–workers I could possibly ask for. To Yannick: thank you for being a constant bedrock of support throughout the past year.

Finally, I want to thank my family – Mom, Dad, Drew, Jenny, Owen and Briar – for their love and encouragement during my PhD and my time in Switzerland.

Lausanne, 6 August 2018 D. C.

Abstract

For decades, neuroscientists and psychologists have observed that animal performance on spatial navigation tasks suggests an internal learned map of the environment. More recently, map–based (or model–based) reinforcement learning has become a highly active research area in machine learning. With a learned model of their environment, both animals and artificial agents can generalize between tasks and learn rapidly. In this thesis, I present approaches for developing efficient model–based behaviour in machines and explaining model–based behaviour in animals.

From a neuroscience perspective, I focus on the hippocampus, believed to be a major substrate of model–based behaviour in the brain. I consider how hippocampal connectivity enable path–finding between different locations in an environment. The model describes how environments with boundaries and barriers can be represented in recurrent neural networks (i.e. attractor networks), and how the transient activity in these networks, after being stimulated with a goal location, could be used for determining a path to the goal. I also propose how the connectivity of these map–like networks can be learned from the spatial firing patterns observed in the input pathway to the hippocampus (i.e. grid cells and border cells).

From a machine learning perspective, I describe a reinforcement learning model that integrates model–based methods and "episodic control", an approach to reinforcement learning based on episodic memory. According to episodic control, the agent learns how to act in the environment by storing snapshot–like memories of its observations, then comparing its current observations to similar snapshot memories where it took an action that resulted in high reward. In our approach, the agent augments these real–world memories with episodes simulated offline using a learned model of the environment. These "simulated memories" allow the agent to adapt faster when the reward locations change.

Next, I describe Variational State Tabulation (VaST), a model–based method for learning quickly with continuous and high–dimensional observations (like those found in 3D navigation tasks). The VaST agent learns to map its observations to a limited number of discrete abstract states, and build a transition model over those abstract states. The long–term values of different actions in each state are updated continuously and efficiently in the background as the agent explores the environment. I show how the VaST agent can learn faster than other state–of–the–art algorithms, even changing its policy after a single new experience, and how it can respond quickly to changing rewards in complex 3D environments.

The models I present allow the agent to rapidly adapt to changing goals and rewards, a key component of intelligence. They use a combination of features attributed to model–based and episodic controllers, suggesting that the division between the two fields is not strict. I therefore also consider the consequences of these findings on theories of model–based learning, episodic control and hippocampal function.

Key words: Reinforcement learning, model-based, goal–directed, episodic, navigation, hippocampus, place cells, recurrent neural networks, deep learning, machine learning, sample efficiency, task transfer.

Résumé

Durant des décennies, les neuroscientifiques et psychologues ont observé dans les animuax, lors de tâches de navigation spatiale, des comportements semblait indiquer l'acquisition d'une carte interne de l'environnement. Plus récemment, l'apprentissage par renforcement basé sur carte (ou *model-based*) est devenu un champ de recherche extrêmement actif dans le domain de l'apprentissage automatique (*machine learning*). À l'aide d'un modèle appris de leur environnement, les animaux tout comme les agents artificiels peuvent généraliser d'une tâche à l'autre et apprendre rapidement. Dans cette thèse, je présente des approches permettant de développer un comportement automatique *model-based* efficace et d'expliquer le comportement *model-based* chez les animaux.

Du point de vue des neurosciences, je me concentre sur l'hippocampe, considéré comme étant un sous-état major du comportement *model-based* dans le cerveau. J'examine comment les connexions de l'hippocampe pourraient agir comme un substrat permettant une recherche de chemin entre différents emplacements dans un environnement. Le modèle décrit comment les environnements ayant des limites et des barrières peuvent être représentés dans des réseaux neuronaux récurrents (i.e. réseaux attracteurs), et comment l'activité transitoire dans ces réseaux, après avoir été stimulée par un but d'emplacement, pourrait être utilisée pour la recherche de chemin. Je propose aussi de voir comment la connectivité de ces réseaux cartographiques peut être apprise à partir des modèles d'activation spatiale observés dans la saisie de chemins jusqu'à l'hippocampe (i.e. cellule de grille et cellules de bord).

Du point de vue de l'apprentissage automatique, je décris un modèle d'apprentissage par renforcement qui intègre des méthodes *model-based* et un "contrôle épisodique", une approche à l'apprentissage par renforcement basée sur la mémoire épisodique. Selon le contrôle épisodique, l'agent apprend à se comporter dans un certain environnement en emmagasinant des souvenirs photographiques de ses observations, puis en comparant ses observations en cours avec des clichés mémoriels similaires à l'endroit où a été entreprise une action donnant lieu à une récompense élevée. Dans notre approche, l'agent augmente ces souvenirs du monde réel à l'aide de souvenirs épisodiques simulés hors-ligne en utilisant un modèle appris de l'environnement. Ces "souvenirs simulées" permettent à l'agent de s'adapter plus rapidement lorsque les emplacements à récompense changent.

Suite à cela, je décris la Tabulation d'État Variationnel (*Variational State Tabulation*, VaST), une méthode *model-based* pour apprendre rapidement avec des observations en continu et en grande dimension (comme celles que l'on retrouve dans les tâches de navigation 3D). L'agent VaST apprend à cartographier ses observations jusqu'à un nombre limité d'états abstraits discrets, puis à construire un modèle de transition par-dessus ces états abstraits. Les valeurs à long terme des différentes actions dans chaque état sont mises à jour en arrière-plan, de façon continue et efficace, en même temps que l'agent explore son environnement. Je montre comment l'agent VaST peut apprendre plus rapidement que d'autres algorithmes de l'état de l'art, et même comment il peut changer de stratégie suite à une seule nouvelle expérience, et comment il peut répondre rapidement aux récompenses changeantes dans des environnements 3D complexes.

Les modèles que je présente permettent à l'agent de s'adapter à des buts changeants et à des récompenses à la volée, un composant clé de l'intelligence. Ils utilisent une combinaison de caractéristiques attribuées au *model-based* et aux contrôleurs épisodiques, suggérant que la division entre deux champs n'est pas absolue. Je prends donc aussi en considération les conséquences de ces découvertes sur les théories de l'apprentissage *model-based*, du contrôle épisodique et de la fonction de hippocampe.

Mots clefs : Apprentissage par renforcement, *model-based*, but dirigé, épisodique, hippocampe, cellules de lieu, réseaux neuronaux récurrents, apprentissage profond, apprentissage automatique, *sample efficiency*, transfère de tâche.

Contents

Acknowledgements **i**

Abstract (English) **iii**

Résumé (Français) **v**

List of Figures **xi**

List of Abbreviations **xiii**

1 Introduction **1**
- 1.1 Navigation and cognitive maps 2
- 1.2 Reinforcement learning 5
- 1.3 Structure of the thesis and previously published work 6

2 Background: Reinforcement Learning Approaches **9**
- 2.1 The model–based approach 10
  - 2.1.1 Optimal control: value iteration 10
  - 2.1.2 Synthetic experience 11
  - 2.1.3 Decision–time planning 12
- 2.2 The model–free approach 12
  - 2.2.1 Q–learning 12
  - 2.2.2 Deep Q–learning 13
  - 2.2.3 Policy search 14
- 2.3 Episodic control 14
- 2.4 Hybrid model–free/model–based approaches 16
  - 2.4.1 Dyna–Q 16
  - 2.4.2 Prioritized sweeping 17
  - 2.4.3 Successor representations 17

3 Preplay and Path Planning with Attractor Networks **21**
- 3.1 Introduction 21
- 3.2 Background 22
  - 3.2.1 Attractor network models of hippocampus 22
  - 3.2.2 Spatial representations in attractor networks 25

3.3 Geodesic attractor networks 28
3.4 Trajectories from attractor dynamics 30
3.4.1 Spiking networks 33
3.4.2 Multichart attractor networks 35
3.5 Successor Representation–based geodesic attractors 37
3.5.1 Representing space using the successor representation 37
3.5.2 Path–finding using the successor coordinate mapping 40
3.5.3 The network model 40
3.5.4 Results 41
3.6 From trajectories to headings 42
3.7 Discussion 44

4 Learning Place Cell Maps for Navigation 47
4.1 Introduction 47
4.2 Grid and border cell input to the place cell network 48
4.3 Learning large geodesic place fields from entorhinal input 50
4.4 Learning the attractor map in recurrent weights 53
4.5 Discussion 55

5 Deep Reinforcement Learning with Offline Episodic Control 59
5.1 Introduction 59
5.2 Background 60
5.3 The model and training procedure 61
5.3.1 Learning environment structure with a VAE 62
5.3.2 Learning value functions 63
5.4 Results 66
5.5 Related work 68
5.5.1 Model–based simulated experience 68
5.5.2 Dimensionality reduction for Reinforcement Learning (RL) 69
5.5.3 Non–parametric approaches to RL 70
5.6 Discussion 70

6 Efficient Model–Based RL with Variational State Tabulation 73
6.1 Introduction 73
6.2 Learning the model 76
6.2.1 The variational cost function 77
6.2.2 The reparameterization trick and the Con–crete distribution 79
6.2.3 Learning a tabular transition model 80
6.2.4 Using the model for reinforcement learning 80
6.2.5 Implementation details 82
6.3 Results 82
6.3.1 Dimensionality of the latent representation 86
6.3.2 Visualizing the abstraction 86

6.3.3 Sample efficiency 87
6.3.4 Transfer learning: non–stationary rewards 89
6.3.5 Transfer learning: multiple tasks with a shared network 90
6.3.6 Training on Atari: Pong 91
6.4 Related work 93
6.5 Discussion 96

7 Discussion & Conclusions 97
7.1 Model–based control, episodic control and the hippocampus 97
7.2 Conclusions 98

8 Future Work 101
8.1 Offline Episodic Control 101
8.2 VaST 101

A Hyperparameters for Chapter 6 105
A.1 3D Navigation 105
A.2 Atari: Pong 106

Contributions 107

Bibliography 109

Curriculum Vitae 125

List of Figures

1.1 Model–free vs. model–based learning 2
1.2 Spatial representations in the brain 3

2.1 The successor representation and rapid learning 19

3.1 Schematic of neural activity in a continuous “bump” attractor network 23
3.2 A toroidal bump attractor network 27
3.3 Non–Geodesic vs. Geodesic place fields 28
3.4 A geodesic attractor network in an environment with barriers 29
3.5 Sequential activity generated by low–dimensional representations 31
3.6 Trajectories in an environment with barriers 34
3.7 Sequential activity generated in a spiking attractor network 35
3.8 Multichart attractors 36
3.9 Representing an environment in successor coordinates 39
3.10 Successor coordinate–based attractor networks 42
3.11 Successor coordinate–based network trajectories in different environments . . 43

4.1 Local place fields from combined grid and border cell input 49
4.2 Large place fields from progressive clustering along the dorsoventral axis 51
4.3 Learned large place fields capture global environment properties 52
4.4 Learning attractor dynamics in recurrent networks 54
4.5 Learned large–scale bump attractor dynamics 56

5.1 Offline episodic control network 62
5.2 Limiting the impact of model error 65
5.3 Learning from offline episodes in a GridWorld environment 66
5.4 Combining offline and online experiences in GridWorld 68
5.5 Failures to learn across conditions 69

6.1 Using state tabulation for efficient planning 74
6.2 The Variational State Tabulation (VaST) network model 79
6.3 VaST learns quickly in complex mazes. 85
6.4 Effect of latent dimensionality. 87
6.5 Binary latent state assignment along different dimensions. 88

6.6 VaST allows for rapid policy changes in response to single experiences. 89
6.7 VaST can adapt to changing rewards. 90
6.8 VaST enables transfer learning across tasks in an environment. 91
6.9 Learning to play Pong with variable frame history. 92

List of Abbreviations

RL Reinforcement Learning

TD Temporal Difference

VaST Variational State Tabulation

MFEC Model–Free Episodic Control

kNN k–Nearest–Neighbour

NEC Neural Episodic Control

MDP Markov Decision Process

STDP Spike–Timing Dependent Plasticity

POMDP *partially–observable* Markov Decision Process

CNN Convolutional Neural Network

DCNN deConvolutional Neural Network

VAE Variational Autoencoder

SWR Sharp Wave and Ripple

DQN Deep Q–learning

NEF Neural Engineering Framework

LSH Locality Sensitive Hashing

LSTM Long Short–Term Memory

EM Expectation Maximization

1 Introduction

The ability to adapt to new situations is a cornerstone in any theory of intelligence. Humans show a remarkable capacity to rapidly thrive in new environments and tasks, often by recognizing features and patterns that they have seen in the past. While natural selection has allowed varied forms of life to adapt to extreme and inhospitable environments across the planet, it relies on millions of years of trial, error and noise. However, one solution generated by natural selection – the human brain – can adapt to novel and complex situations in years, days, or even minutes.

This thesis considers how the ability to adapt to new environments and new tasks might be achieved by building a model of the environment. Rather than finding solutions by simply determining what works and what does not, building a model requires learning the underlying dynamics of an environment or state space at some level of abstraction. Armed with a model of the dynamics of an environment, an agent (human, animal or artificial) is often much faster at solving a task in that environment and adapting when the task changes.

As a motivating example, we can consider the problem faced by a rat trying to reach a food reward in a maze (Figure 1.1). We assume that, during the first stage of learning, the rat is placed in the same location in the maze every day with the same location of the reward. The rat can learn whether to turn left, turn right or continue forward at each choice point to find the food. Alternatively, it can learn to build a model (or map) of where it will be following each action at each choice point, and use that model to plan a path to the food. While the second approach is generally more complex, it will allow the rat to adapt faster if the reward's location changes. In the neuroscience and psychology literature, these two types of learning are often referred to as *habitual* vs. *goal–directed*, or *stimulus–response* vs. *stimulus–stimulus* respectively [Holland, 2008, Balleine and O'Doherty, 2010]. In the field of RL, they are referred to as *model–free* vs. *model–based* [Daw et al., 2005, Sutton and Barto, 2018].

In this thesis, I will consider aspects of model–based learning from the perspective of both machine learning and computational neuroscience. In neuroscience, I will focus on the problem of spatial navigation, where model–based learning is closely related to the study of

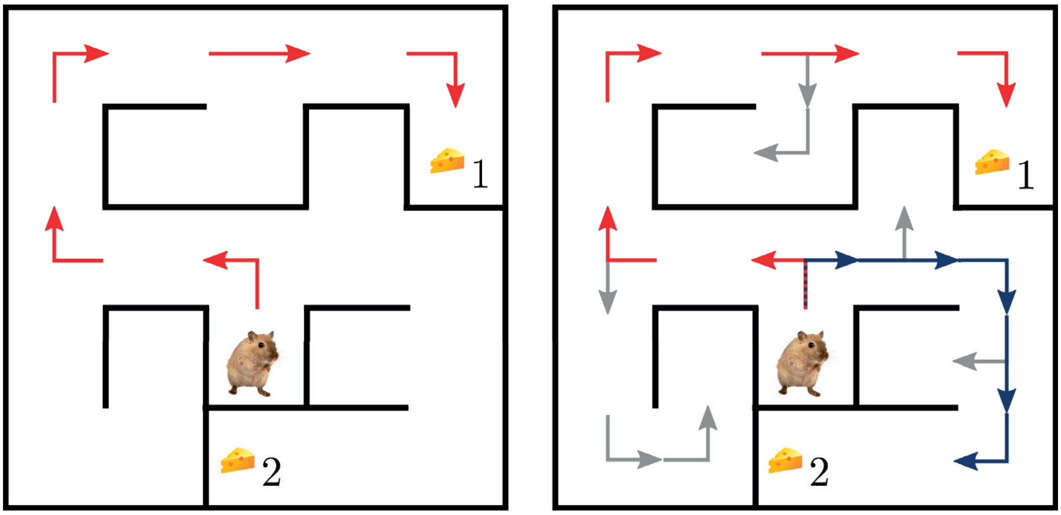

Figure 1.1 – **Model–free vs. model–based learning. Left**: A rat learns to navigate to a food reward in a maze (at Position 1). Here, the rat learns a set of fixed stimulus–response relationships dictating which direction to turn at each intersection in the maze. These responses cannot be easily adapted to a new reward location (Position 2). **Right**: The rat builds an internal model of where it will be after turning in a given direction at each position in the maze. The model can be used to plan a path to either reward location.

cognitive maps [Tolman, 1948, O'Keefe and Nadel, 1979]. In the field of machine learning, I will focus on how artificial agents could use models to learn and adapt to new tasks faster. In the following, I provide a brief overview of relevant research and findings in both neuroscience and machine learning.

1.1 Navigation and cognitive maps

Modern theories of model–based behaviour are based on early behavioural experiments in *latent learning* [Blodgett, 1929]. In these experiments, rats were trained to find the exit in a maze, either with a food reward at the exit (control group) or without a food reward (experimental group). With sufficient training, rats trained with the food reward eventually finished the maze much faster than rats without the food reward. However, when a food reward was later introduced for the experimental group, the researchers found that their performance rapidly improved to match that of the control group. They concluded that the rats in the experimental group had learned something about the structure of the maze even when not incentivized by food, which allowed them to adapt much faster when the food was introduced. The result contrasted against the prevailing theory of operant conditioning [Thorndike, 1898, Sutton and Barto, 2018] in which learning corresponds to reinforcing behaviours that lead to reward. The process of acquiring environmental knowledge in absence of reward was later described by Edward Tolman as building a cognitive map [Tolman, 1948].

The discovery of hippocampal *place cells* in the rat [O'Keefe and Dostrovsky, 1971], which

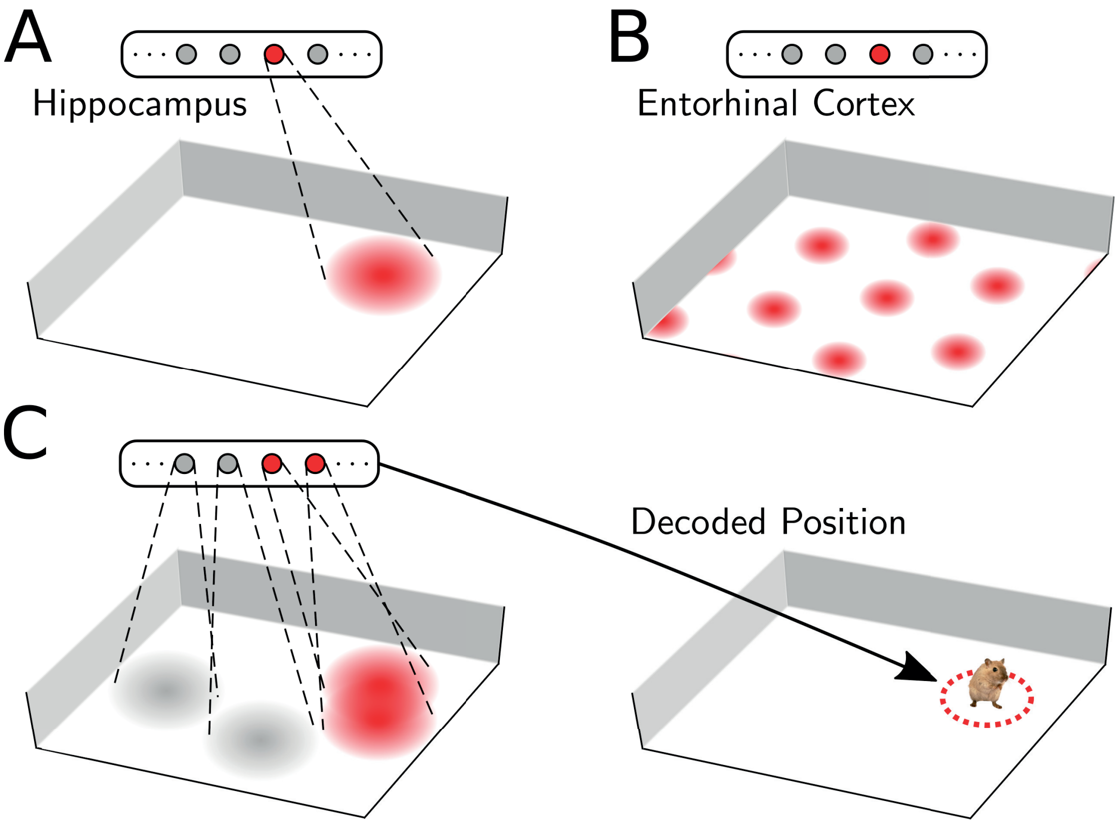

Figure 1.2 – **Spatial representations in the brain. A**: Place cells in the hippocampus respond selectively when the rat is in a particular position within the environment. **B**: Grid cells in the medial entorhinal cortex stereotypically respond according to a hexagonal grid pattern in two–dimensional space. **C**: The population response of multiple place cells in the hippocampus with stable place fields can be decoded downstream to estimate the rat's current location.

tend to fire selectively in a specific region of an environment, provided evidence that the hippocampus may play a role in forming a cognitive map [Moser et al., 2008]. Different place cells within the hippocampus were found to be consistently selective to different locations in an environment (Figure 1.2), such that the population response could be used to determine the animal's current location [O'Keefe, 1976, Wilson and McNaughton, 1993]. The existence of spatially–tuned cells in the hippocampus has since been shown in several other mammalian species, including humans [Nadel, 1991, Rolls et al., 1997, Ekstrom et al., 2003, Ulanovsky and Moss, 2007].

Grid cells were later discovered in the entorhinal cortex [Fyhn et al., 2004, Hafting et al., 2005], the primary input region to the hippocampus. Grid cells have a spatially periodic response, such that a scatter plot of the animal's location when an individual grid cell emits a spike resembles a hexagonal grid (Figure 1.2B) [Moser et al., 2008]. Individual grid cells differ in their grid orientation and offset, with a grid spacing that increases along the dorsoventral axis of the medial entorhinal cortex [Hafting et al., 2005, Moser et al., 2008, Brun et al., 2008], mirroring the increase in place field size in the hippocampus along the dorsoventral axis [Kjelstrup et al., 2008]. Several researchers have suggested that grid cells may act as basis functions for the place cell network in the hippocampus, by combining them across different spatial scales to produce a single, non–periodic place cell response [McNaughton et al., 2006, Solstad et al., 2006, Rolls et al., 2006].

The ability to plan a path within an environment requires more than just a stable representation of the current location; it also requires a model of how different locations are related. Experiments in path integration have shown that the hippocampal representation of position is determined not only by external visual cues, but also by the distance travelled from a starting position, particularly in the absence of familiar landmarks [Gothard et al., 1996, Redish et al., 2000]. In addition, the hippocampal representation of position can persist and continue to be updated after turning off the light in the environment [Quirk et al., 1990, Markus et al., 1994]. The ability to maintain an estimate of the current position while moving in the dark requires the animal to have an internal model of where it will be after moving in a particular direction from a known location, as suggested by the earlier latent learning behavioural experiments [Blodgett, 1929].

Samsonovich and McNaughton [1997], and later Conklin and Eliasmith [2005], proposed that the path integration response of hippocampal place cells could be explained using an *attractor network* model, in which cells with similar and overlapping place fields share recurrent excitatory connections, in combination with an external driving signal which moves the place cell representation in the direction of motion. Attractor network connectivity has since been used to explain several experimental findings on place cell responses [Wills et al., 2005, Colgin et al., 2010, Jezek et al., 2011]. The attractor network model suggests that the spatial relationship between locations is at least partially represented in the hippocampus, in the recurrent connections between place cells.

If the rat were to use a model to plan, one would expect to see, during planning, a neural representation of potential paths through the environment that the rat could follow (i.e. Figure 1.1 right). The observation of sequential place cell activity in the hippocampus during Sharp Wave and Ripple (SWR) events appears to reflect such paths [Lee and Wilson, 2002, Foster and Wilson, 2006, Csicsvari et al., 2007, Diba and Buzsáki, 2007, Johnson and Redish, 2007, Davidson et al., 2009, Karlsson and Frank, 2009, Gupta et al., 2010]. These short events (50 – 300 ms) typically occur when the animal halts during exploration or when the animal is sleeping [Buckner, 2010]. During these events, the location represented by the hippocampus becomes disconnected from the animal's actual location, and moves along a trajectory through the environment. This trajectory can correspond to a path the animal has already taken (*replay*, Lee and Wilson [2002]), the reverse direction of a path the animal has just taken (*reverse replay*, Foster and Wilson [2006]), a path the animal may take in the immediate future (*preplay*, Diba and Buzsáki [2007]), or a path that has no clear relationship to past or future experience [Gupta et al., 2010]. Preplay events have been shown to be predictive of the animal's future path even in open–field environments [Pfeiffer and Foster, 2013], suggesting a link to a path planning mechanism.

When a rat is moving, hippocampal activity is modulated by background theta oscillations (approximately 8 Hz, Buzsáki [2002]). As the rat approaches the place field of a neuron, that neuron typically fires at an earlier and earlier phase in the theta cycle [O'Keefe and Recce, 1993, Skaggs and McNaughton, 1996]. If we consider the activity of several cells within the same theta cycle, this phenomenon can be interpreted as a "sweep" of the hippocampal place representation moving from the animal's location forward in the direction of travel, corresponding to a prediction of future location [Lisman and Redish, 2009]. Like SWR preplay sequences, theta sequences have been found to predict the animal's actual future path through the environment [Huxter et al., 2008, Wikenheiser and Redish, 2015].

Trajectory events like SWR preplay and theta sequences may reflect model–based path planning or some other computation; if they do reflect path planning, it is unclear to what extent they depend on networks external to the hippocampus. However, behavioural evidence also suggests a link between the hippocampus and model–based navigation. Bast et al. [2009] found that the ability to rapidly learn and return to new goal locations in a well–known maze, a hallmark of model–based planning, critically depends on one region of the hippocampus. Incremental learning of constant reward locations (consistent with a model–free approach) was found to not require the hippocampus at all. In other experiments, rapid place learning has been found to depend on NMDA–based plasticity in the recurrently–connected CA3 region of the hippocampus, while incremental learning does not [Nakazawa et al., 2003].

1.2 Reinforcement learning

Modern reinforcement learning theory is founded in the study of *optimal control*: the problem of determining the control signal to supply to a dynamical system in order to maximize a performance criterion over time, subject to a set of constraints [Kirk, 2012, Sutton and Barto, 2018]. In such problems, a perfect model of the system is typically assumed. The *dynamic programming* approach to optimal control suggests breaking the control problem into multiple nested subproblems and solving them recursively. It is anchored in Richard Bellman's Principle of Optimality [Bellman, 1957]:

> *An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.*

For example, in Figure 1.1, we can consider the very simple problem of finding a path to reward location 1 from the top right corner of the maze. This is, in fact, a subproblem of finding an optimal path from any other point in the maze, since the rat will first need to navigate to the top right corner. The rat could build on solutions in the near vicinity of the goal, extending to further distances, until it has determined the optimal path from any point in the maze. This is formalized in the Bellman Equation, discussed in Chapter 2.

The method of Q–learning [Watkins and Dayan, 1992] combined ideas from dynamic programming with Temporal Difference (TD) learning [Sutton and Barto, 2018]. Watkins and Dayan showed that it was possible to learn an optimal policy by incrementally improving an estimate of the long–run future rewards after taking a particular action from a state in the environment. Q–learning relies on the concept of *bootstrapping*: the value of state A depends on the estimated value of state B that follows it, and the estimated value of state B depends on the estimate of its successor state C etc. As in dynamic programming, successfully determining the value of state C would therefore improve the estimate of its predecessor states B and A. Q–learning is model–free: the agent learns only the expected value of taking an action from each state, and takes the action that has the highest expected value at each point in time. Several algorithms later extended Q–learning with model–based components, including Dyna–Q [Sutton, 1990], prioritized sweeping [Moore and Atkeson, 1993, Peng and Williams, 1993, Van Seijen and Sutton, 2013], and successor representations [Dayan, 1993]. Q–learning and its variants and successors are considered in greater detail in Chapter 2.

In 2013, Q–learning was successfully extended to complex, high–dimensional environments like video games by using deep artificial neural networks [Mnih et al., 2013]. Many algorithms for model–free RL in continuous and/or high–dimensional environments have followed, improving the performance or sample efficiency, decreasing the training time, or addressing new types of problems (e.g. Lillicrap et al. [2015], Schaul et al. [2015], Mnih et al. [2016], Pritzel et al. [2017]). However, there have been relatively few successful applications of model–based techniques in combination with deep networks; most have appeared since 2016 [Gu et al., 2016, Racanière et al., 2017, Nagabandi et al., 2017, Oh et al., 2017, Farquhar et al., 2017, Silver et al., 2017a,b, Buesing et al., 2018].

1.3 Structure of the thesis and previously published work

Most chapters in the remainder of the thesis incorporate one or more model–based RL techniques. In Chapter 2, I provide a very brief overview of the field of RL and an introduction to the algorithms that will be considered.

In Chapter 3, I consider the problem of spatial navigation from a neuroscience perspective. I propose a model in which the topology of an environment with boundaries and obstacles is represented in the recurrent connections of the CA3 region of the hippocampus (i.e. a cognitive map), and show how it could allow for planning paths to novel goal locations. Different portions of this chapter were previously published in the proceedings of NIPS 2015 [Corneil and Gerstner, 2015a], and as an extended abstract at COSYNE 2015 [Corneil and Gerstner, 2015b]. In Chapter 4, I extend this work by considering how the environment topology could be learned in attractor networks beginning from simple spatial representations upstream of the hippocampus.

In Chapter 5 and Chapter 6, I describe two different model–based agents primarily from a machine learning perspective (with some possible implications for neuroscience). Chapter 5 considers a deep neural network architecture that combines aspects of model–based RL with the emerging theory of episodic control, allowing the agent to rapidly adapt to changing reward landscapes. Portions of this chapter were previously published as an extended abstract in the proceedings of CCN 2017 [Corneil and Gerstner, 2017].

In Chapter 6, I introduce Variational State Tabulation, an algorithm for enabling fast model–based RL by learning discrete abstractions of environments defined by high–dimensional and/or continuous observations. The model is evaluated primarily in the domain of 3D navigation. Most of this chapter, excluding some new results on multi–task learning and visualizing the abstraction, has been previously published in the proceedings of ICML 2018 [Corneil et al., 2018].

I conclude the thesis with a discussion of the overarching results and promising future research directions.

2 Background: Reinforcement Learning Approaches

The field of Reinforcement Learning is concerned with finding a mapping from observations to actions, in order to maximize a reward signal [Sutton and Barto, 2018]. In the situations considered in this thesis, a reward is often the outcome of an entire sequence of actions over time, like the sequence of turns a rat needs to make to find the food pellet in a maze. RL is one of three major classes of problems considered in the neuroscience of learning and machine learning literature, together with *supervised learning* and *unsupervised learning* [Goodfellow et al., 2016]. Supervised learning is concerned with finding a mapping from observations to labels (e.g. from a picture of a bird in a pond to the label “duck”), while unsupervised learning is concerned with finding some underlying structure in data. In Chapter 6, we will incorporate aspects of unsupervised learning when considering how an agent can learn a model of its environment.

Many problems studied in RL (including the ones in this thesis) belong to the class of Markov Decision Processes (MDPs) [Bellman, 1957]. Consider an agent exploring an environment made up of a set of states $s \in \mathcal{S}$, taking actions $a \in \mathcal{A}$, and receiving real–valued rewards $r \in \mathcal{R}$, with discrete time steps $t = 0, 1, 2, 3$ etc. In an MDP, we have

$$p(s_{t+1}, r_{t+1} | a_{1:t+1}, r_{1:t}, s_{0:t}) = p(s_{t+1}, r_{t+1} | a_{t+1}, s_t), \tag{2.1}$$

i.e. the outcome of the next action a_{t+1} (the distribution over rewards and next states) depends only on the current state s_t; the agent’s history before the current state can be safely ignored. This is referred to as the *Markov assumption* [Rosenblatt, 1974].

In RL, we consider the problem of learning a *policy* $\pi(a|s)$, which describes the action that should be taken in any particular state in order to maximize future reward. Typically, a discount factor $\gamma \in [0, 1]$ is introduced to make rewards in the near future more enticing than rewards in the distant future. Given a particular policy π, the value $V^{\pi}(s)$ of state s is defined as

$$V^{\pi}(s) := \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+1+k} \,\middle|\, s_t = s \right], \tag{2.2}$$

i.e. the expected discounted future reward starting from the state s and continuing under the policy π. The optimal policy π^* is that which maximizes the value function for all states, resulting in the value function $V^*(s)$.

Using the Markov assumption, we find that the state value can be redefined [Sutton and Barto, 2018] as

$$V^*(s) = \max_a \sum_{\substack{s' \in \mathcal{S} \\ r' \in \mathcal{R}}} p(s', r' | a, s) \big[r' + \gamma V^*(s') \big]. \tag{2.3}$$

This is the Bellman optimality equation, an expression of the Principle of Optimality introduced in Chapter 1. It reveals the useful property that following the optimal policy from state s simply means taking the action a that appears in the maximization of $V^*(s)$; all previous states and actions are irrelevant due to the Markov assumption. This is more easily expressed by stating Equation 2.3 in terms of the value of a state–action pair, the so-called *Q–value*, defined as

$$Q^*(s, a) = \sum_{\substack{s' \in \mathcal{S} \\ r' \in \mathcal{R}}} p(s', r' | a, s) \big[r' + \gamma \max_{a'} Q^*(s', a') \big]. \tag{2.4}$$

The optimal policy in the current state s_t is then to take action $a_{t+1} = \arg\max_a Q^*(s_t, a)$.

In some cases, the agent receives observations o_t rather than the underlying states s_t directly. When the observations fully determine the true underlying state, we say the problem is *fully observable* at the current time step [Kirk, 2012]. When there are details about the state s_t that cannot be determined from o_t, we say the problem is *partially observable*. Often, some recent history of observations can then be used to disambiguate the underlying state. In the following, we assume the state is fully observable and consider s_t directly. Some partially observable problems will be considered in Chapter 6.

When the number of states $|\mathcal{S}|$ and actions $|\mathcal{A}|$ are relatively few, we refer to the task as *tabular*. This reflects the fact that the task statistics can be efficiently stored and updated in a fixed–size table or array in memory.

2.1 The model–based approach

2.1.1 Optimal control: value iteration

According to the model–based approach, we can determine the Q–values by solving the right–hand side of Equation 2.4 after learning the task statistics $p(s', r' | a, s)$. The equation can be separated into two terms, giving us

$$\begin{aligned} Q^*(s,a) &= \sum_{\substack{s'\in\mathcal{S}\\ r'\in\mathcal{R}}} p(s',r'|a,s)r' + \gamma \sum_{\substack{s'\in\mathcal{S}\\ r'\in\mathcal{R}}} p(s',r'|a,s)\big[\max_{a'} Q^*(s',a')\big] \\ &= r(s,a) + \gamma \sum_{s'\in\mathcal{S}} p(s'|a,s)\big[\max_{a'} Q^*(s',a')\big] \end{aligned} \qquad 2.5$$

where we have marginalized out the variables irrelevant to the expectation in each term, and $r(s,a) = \mathbb{E}[r|a,s]$ is shorthand for the expected immediate reward after taking action a in state s (i.e. without taking into account future rewards). If the agent learns an accurate model of $r(s,a)$ and $p(s'|a,s)$ by exploration, the optimal Q–values can be determined by iteratively solving for $Q(s,a)$ over all states and actions until a stable solution is reached. This is referred to as *value iteration* [Bellman, 1957]. The update equation is given by

$$Q_{i+1}(s,a) = r(s,a) + \gamma \sum_{s'\in\mathcal{S}} p(s'|a,s)\big[\max_{a'} Q_i(s',a')\big] \quad \forall\ s\in\mathcal{S}, a\in\mathcal{A}, \qquad 2.6$$

where i is the iteration index. Given sufficient iterations for convergence, value iteration arrives at a solution for the value function that induces the optimal policy for the MDP (under the transition model and expected rewards). The algorithm therefore falls into the family of optimal control techniques [Kirk, 2012].

Value iteration is computationally expensive in large state spaces, and can also be inefficient because the values of many states will not change over the course of one iteration. For instance, consider an agent exploring a large state space, with all Q–values initialized to 0. It encounters the first non–zero reward on step $t+1$ after taking action a_{t+1} from state s_t, and updates $r(s_t, a_{t+1})$ accordingly. It then performs value iteration to update its Q–value estimates for all states and actions. In this case, the first update iteration will require $|\mathcal{S}| \times |\mathcal{A}|$ applications of Equation 2.6 despite the fact that only $Q(s_t, a_{t+1})$ will change, a significant inefficiency. This is addressed in part by the method of prioritized sweeping by small backups, discussed in Section 2.4.2 and applied in Chapter 6.

2.1.2 Synthetic experience

An alternative way to leverage a model in order to estimate the optimal value function is to generate *synthetic experiences* by sampling from the transition distribution $p(s'|a,s)$ and evaluating the result. These experiences, extending for one step or multiple steps using a given policy, can be used to update the value function estimate (see Section 2.4.1 for more details). Compared to backups using the full expectation over the next state as in value iteration, sample–based backups require a learning rate parameter (as we will describe in the following sections) and introduce sampling noise; however, they can achieve low error in the value function estimate with less computation [Sutton and Barto, 2018].

2.1.3 Decision–time planning

The preceding approaches use a model to solve or estimate a value function. Alternatively, a model can be used to sample multi–step trajectories from the current state in order to simulate and evaluate many possible future paths, taking the immediate action that leads to the best sampled outcome (given the expected rewards $r(s,a)$). Typically, this process is repeated each time an action is taken and the agent moves to a new state, receiving new observations. This is called *decision–time planning* [Sutton and Barto, 2018]. In linear control theory, the process of optimizing up to a future horizon in order to determine the current control signal, then re–optimizing at the next time step, is referred to as *model predictive control* [Camacho and Alba, 2013].

While not considered in detail in this thesis, decision–time planning is one of the main areas of focus in model–based deep reinforcement learning and the basis of several recent successful algorithms (e.g. Silver et al. [2016, 2017a]). In particular, decision–time planning can be advantageous when the state space is very large such that the agent rarely revisits states, and the transition model is well–known or perfectly known. In this case, the agent may achieve better results by planning at each step rather than estimating or learning a complex value function. Furthermore, decision–time planning avoids wasting computation on determining the policy for states that the agent rarely visits. However, decision–time planning is computationally intensive in general due to the need to re–determine the policy at each step, leading to high latency in selecting an action. In contrast to decision–time planning, model–based algorithms that do not depend on the current state (e.g. value iteration or synthetic experience) have been referred to as *background planning* methods [Sutton and Barto, 2018].

2.2 The model–free approach

2.2.1 Q–learning

Rather than solving for the recursion in Equation 2.4 from a learned model, one can attempt to learn the Q–values $Q^*(s,a)$ directly. This is the approach taken in Q–learning [Watkins and Dayan, 1992]. Consider the agent's observations after taking the next step a_{t+1} from the current state s_t. If the Q–values have already been solved under the optimal policy, we expect from Equation 2.4 that

$$\mathbb{E}_{r',s'}\left[r' + \gamma \max_{a'} Q^*(s',a') - Q^*(s_t,a_{t+1})\right] = \mathbb{E}_{r',s'}[\delta_{td}] = 0, \tag{2.7}$$

where δ_{td} is called the TD–error [Sutton and Barto, 2018]. A non–zero expected TD–error represents an inconsistency in the current estimate of the state–action value. To improve the estimate, one could perform stochastic gradient descent on δ_{td}^2 with respect to the current estimate of $Q(s_t, a_{t+1})$. This yields the update equation

$$Q(s_t, a_{t+1}) \leftarrow Q(s_t, a_{t+1}) + \eta \cdot \left[r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_{t+1})\right], \tag{2.8}$$

where $0 < \eta \leq 1$ is the learning rate. Every time the agent takes a step, it observes the tuple $(s_t, a_{t+1}, r_{t+1}, s_{t+1})$ (original state, action taken, reward received, new state). It then updates $Q(s_t, a_{t+1})$, using its current estimates to determine $\max_{a'} Q(s_{t+1}, a')$. Watkins and Dayan [1992] showed that Equation 2.8 converges to the state–action values $Q^*(s, a)$ under the optimal policy given infinite experience of all possible state–action pairs. Successful Q–learning therefore requires a balance between collecting experience in the state space to achieve a reliable estimate of the Q–values, and maximizing reward by choosing actions with the highest Q–value (the *exploration–exploitation dilemma*). A common approach is to choose a random action with probability ϵ on each step, or $\arg\max_a Q(s, a)$ with probability $1 - \epsilon$, where epsilon is typically annealed over the course of training; this is referred to as an ϵ–greedy strategy [Sutton and Barto, 2018].

The term *model–free* is perhaps a misnomer, since the algorithm clearly maintains a model in the form of the learned Q–values. In contrast to the transition statistics learned by the model–based approach, however, the learned Q–values are dependent on the reward structure. If the reward distribution changes within the same environment, the Q–values will need to be re–learned from scratch; in contrast, a model–based agent will only need to re–learn the expected immediate rewards $r(s, a)$. The parameters learned by a model-free agent are therefore inseparable from the task (i.e. the rewards in an environment) being learned by that agent.

2.2.2 Deep Q–learning

It is reasonable to build a table of Q–values $Q(s, a)$ only when there are a small number of discrete states $s \in \mathcal{S}$. In a continuous and/or high–dimensional state space, the agent may never collect a sufficient amount of experience to ensure that the Q–learning algorithm converges; in some tasks, the agent may never revisit the same state twice. In this case, we need a “non–tabular” approximation. A common practice is to parameterize the Q–values by a set of parameters θ, and minimize δ_{td}^2 with respect to θ for the observed transitions. Generalizing from observed data points to a continuous or high–dimensional function requires *function approximation* [Sutton and Barto, 2018]. The Deep Q–learning (DQN) algorithm uses deep neural networks as Q–value function approximators, and successfully applies Q–learning to complex video game environments [Mnih et al., 2013, 2015].

Several issues can make value learning unstable when using function approximation, particularly with deep networks. First, the network is trained under the assumption that training data examples correspond to independent samples from the training set. However, if the agent learns while exploring the environment according to the current policy, consecutive steps tend to be in the same region of the environment and are therefore highly correlated. As well, the training set can change suddenly in response to changes in the agent's policy. In DQN, this was partially addressed by sampling training data points independently from a *replay memory* of the last 1 million transitions [Lin, 1993, Mnih et al., 2013]. Secondly, we note from Equation 2.8 that both $Q^{\theta}(s_t, a_{t+1})$ and $\max_{a'} Q^{\theta}(s_{t+1}, a')$ are parameterized by θ; optimizing $Q^{\theta}(s_t, a_{t+1})$ can lead to a corresponding change in $\max_{a'} Q^{\theta}(s_{t+1}, a')$, causing parameters to oscillate or diverge. Mnih et al. [2015] therefore parameterized $\max_{a'} Q^{\theta'}(s_{t+1}, a')$ using a separate set of parameters θ', which were copied from θ every 10 000 steps. Finally, rewards and/or output layer error terms were clipped to $[-1, 1]$ to allow the same learning rate to be used across different reward scales in different games.

2.2.3 Policy search

The methods previously described (and those used throughout this thesis) determine a policy $\pi(a|s)$ indirectly by learning a value function over state–action pairs, then choosing the action that maximizes the value function. An alternative model–free approach is to *directly* determine a policy that maximizes the expected discounted returns. In the common case where this is done by optimizing the parameters of the policy using gradient descent, it is referred to as a *policy gradient* method [Peters and Schaal, 2008, Peters, 2010]. Policy gradient methods have convergence guarantees that are lacking with many model–free value function–based approaches [Peters, 2010].

One of the most common policy gradient methods is the REINFORCE algorithm [Williams, 1992, Sutton and Barto, 2018], which can suffer from high variance in the gradient descent update. Attempts to reduce the gradient estimator variance in REINFORCE while maintaining convergence guarantees have resulted in the combination of policy search and value function approaches, through the class of actor–critic algorithms [Konda and Tsitsiklis, 2000].

2.3 Episodic control

When using function approximation for value functions as in DQN, the value function (cumulative discounted rewards) is learned by optimizing a fixed number of network parameters θ. An alternative is to use a non–parametric approach, i.e. to store the returns experienced from a state–action pair directly in memory, and use these raw data points for estimating Q–values in the future. This approach, termed *episodic control*, has been championed from both a neural/behavioural standpoint [Lengyel and Dayan, 2007, Gershman and Daw, 2017] and a machine–learning standpoint [Blundell et al., 2016] as a viable alternative to parametric model–free and model–based methods.

We focus on the formulation of episodic control given by Gershman and Daw [2017]. We consider the case where an agent learns across multiple separate "episodes", and the states s encountered on these episodes exist in some continuous and/or high–dimensional state space (for instance, the color intensity of pixels on a screen while playing a video game). We take $s_{\mu,t}$ to indicate the (potentially vector–valued) state that the agent encountered on step t of episode μ. The index set

$$\mathcal{E}_{sa} = \{(\mu, t) | s_{\mu,t} = s, a_{\mu,t+1} = a\} \tag{2.9}$$

describes the set of episodes and time steps where the agent has encountered a particular state–action pair (s, a) in the past. The Q–value for (s, a) is then given by

$$Q(s,a) = \frac{1}{|\mathcal{E}_{sa}|} \sum_{(\mu,t)\in\mathcal{E}_{sa}} \left[\Big(\sum_{n=1}^{N} \gamma^{n-1} r_{\mu,t+n} \Big) + \gamma^N \max_{a'} Q(s_{\mu,t+N}, a') \right], \tag{2.10}$$

i.e. an average over the discounted returns from all experienced trajectories beginning with (s, a) in the past, truncated to N steps (with the remaining returns estimated from the value of the state on the Nth step). Alternatively, the agent can use the maximum returns experienced thus far from (s, a) [Lengyel and Dayan, 2007, Blundell et al., 2016]. If $N = \infty$, the corresponds to full episodic returns without bootstrapping (i.e. Monte Carlo returns [Sutton and Barto, 2018]).

While parametric N–step model–free methods are common [Sutton and Barto, 2018], an additional key difference lies in how the Q–values of state–action pairs are estimated online:

$$Q(s,a) = \frac{\sum_{s'\in\mathcal{S}_a} K(s,s')Q(s',a)}{\sum_{s'\in\mathcal{S}_a} K(s,s')}, \tag{2.11}$$

where $K(s, s')$ represents some similarity function between states (e.g. a kernel function or a nearest–neighbour measure), and $\mathcal{S}_a$ are the states in memory where the action a was experienced in the past. Equation 2.11 represents an alternative to function approximation for generalizing to state–action pairs that the agent has never seen before. Past observed states may be represented in memory by their associated raw representation (e.g. pixels) or some low–dimensional projection to reduce memory requirements [Blundell et al., 2016].

Returns associated with each state in an episode can be added to the memory at the end of that episode. Individual experiences tend to affect the policy much faster than with 1–step model–free deep RL for two reasons. First, N–step TD methods result in a faster backward diffusion of reward signals than 1–step methods. Secondly, an episode is incorporated into the model immediately rather than being added to a replay memory for eventual incremental update of a function approximator by stochastic gradient descent. Accordingly, episodic controllers have shown much better performance in the early stages of learning [Blundell et al., 2016, Pritzel et al., 2017]. However, they tend to perform worse in later stages, in part because deep networks can eventually learn to generalize and extract the task–relevant features of the state–space more efficiently than is possible with a fixed kernel over raw features. This issue can be partially addressed by learning a state embedding that efficiently compresses the observations [Blundell et al., 2016] or predicts the estimated returns [Pritzel et al., 2017].

2.4 Hybrid model–free/model–based approaches

2.4.1 Dyna–Q

As mentioned earlier, a model can be used to generate synthetic experience by sampling in order to update the value function. One possibility is to use that experience to augment the performance of a model–free Q–learner. This is the approach taken by Dyna–Q [Sutton, 1990]. The agent explores a tabular environment updating the Q–values according to the Q–learning update rule, while also learning $p(s'|a,s)$ and $r(s,a)$ online. Then, for a number of offline backup steps, it updates the Q–values by randomly sampling learned transitions from the estimated model, defined by transition probabilities $p(s'|a,s)$ and estimated rewards $r(s,a)$.

Algorithm 1 Dyna–Q.

```
Initialize $Q(s,a)$ for all s, a
Initialize $N_{sa}, R_{sa}, N_{sa}^{s'} = 0$ for all $s$, $a$, $s'$
Observe current state $s$
loop
    Take action $a$ with $\epsilon$-greedy strategy based on $Q(s,a)$
    Observe $r'$, $s'$
    $Q(s,a) \leftarrow Q(s,a) + \eta \cdot \left[r' + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$    ▷ Q–learning step
    $N_{sa} \leftarrow N_{sa} + 1$; $N_{sa}^{s'} \leftarrow N_{sa}^{s'} + 1$; $R_{sa} \leftarrow R_{sa} + r'$    ▷ Model update
    for $k = 1$ to $N_k$ do
        Sample $s_k$ from states where $\sum_a N_{sa} > 0$
        Sample $a_k$ from actions where $N_{s_k a} > 0$
        Set $r'_k = r(s_k, a_k) = R_{s_k a_k} / N_{s_k a_k}$
        Sample $s'_k \sim p(s'_k | a_k, s_k) = N_{s_k a_k}^{s'_k} / N_{s_k a_k}$
        $Q(s_k,a_k) \leftarrow Q(s_k,a_k) + \eta \cdot \left[r'_k + \gamma \max_{a'} Q(s'_k,a') - Q(s_k,a_k)\right]$    ▷ Offline backup
    end for
    $s \leftarrow s'$
end loop
```

If the number of offline backup steps is $N_k = 0$, then the model is never used and Dyna–Q collapses exactly to Q–learning. Conversely, as $N_k \to \infty$, Dyna–Q updates the value function estimate almost entirely from the learned model. We can therefore see N_k as a parameter that smoothly interpolates between a model–free and a model–based approach, and as a trade–off in terms of background computation requirements.

Notably, the DQN replay memory considered in Section 2.2.2 is equivalent to a form of Dyna–Q in which (a) the transitions kept in memory are limited to those recently encountered by the agent and (b) sampling is biased towards transitions frequently encountered by the agent (since transitions are stored explicitly rather than being used to update tabular statistics, as in Algorithm 1). In this view, DQN is a hybrid approach, where the replay memory is a non–parametric environment model, although the authors consider it to be purely model–free [Mnih et al., 2013].

2.4.2 Prioritized sweeping

Like value iteration, Dyna–Q can be computationally inefficient because many cycles are spent updating Q–values that change very little, or not at all. This is addressed by *prioritized sweeping*, which aims to concentrate update cycles on large changes to the value landscape [Peng and Williams, 1993, Moore and Atkeson, 1993, Van Seijen and Sutton, 2013].

Intuitively, after updating a Q–value according to Step 5 of Algorithm 1, we can consider the priority value $p = |V_i(s) - V_{i-1}(s)| = |\max_a Q_i(s,a) - \max_a Q_{i-1}(s,a)|$, i.e. the change in value of state s after updating $Q(s,a)$. If p is high for a given state s, it is logical to prioritize sampling states s_k in Step 8 that have a high probability of transitioning into state s according to the learned model.

The most efficient form of prioritized sweeping, *prioritized sweeping with small backups* [Van Seijen and Sutton, 2013], is considered in Chapter 6. The version of the algorithm used there updates the Q–values entirely with offline backups, making it a purely model–based approach. In addition, the value of a predecessor state–action pair is updated using the probability of the transition (without sampling or a learning rate); as a result, it can be considered an asynchronous version of value iteration, and likewise converges to the optimal value function for the model [Li et al., 2008, Van Seijen and Sutton, 2013].

2.4.3 Successor representations

As an alternative to background computation, one can consider a factorization of the Q–value that allows the agent to respond quickly to new reward observations. The *successor representation* [Dayan, 1993] does this by building a model of the expected multi–step *future occupancy* of each successor state under the current policy in a tabular environment.

To see how this is useful, we note that the policy and transition probabilities can be combined to give

$$M(s,s') = \sum_{a \in \mathcal{A}} p(s'|a,s)\pi(a|s), \tag{2.12}$$

i.e. the policy–dependent probability that the agent will arrive in state s' after a single step from s.

We form the matrix $\mathbf{M}$ from the entries $M(s,s')$; where $\mathbf{M}$ has the shape $|\mathcal{S}| \times |\mathcal{S}|$. We similarly define $|\mathcal{S}|$–length vectors $\mathbf{v}$ and $\mathbf{r}$, where each entry $v(s)$ in $\mathbf{v}$ represents the value of state s, and each entry $r(s) = \sum_a r(s,a)\pi(a|s)$ in $\mathbf{r}$ represents the expected reward received after exiting state s. In a similar fashion to Equation 2.2, state values $\mathbf{v}$ can be represented as

$$\begin{aligned}\mathbf{v} &= \sum_{k=0}^{\infty} \gamma^k \mathbf{M}^k \mathbf{r} \\ &= \mathbf{r} + \gamma \mathbf{M}\mathbf{r} + \gamma^2 \mathbf{M}^2 \mathbf{r} + \gamma^3 \mathbf{M}^3 \mathbf{r} + \dots \\ &= (\mathbf{I} - \gamma \mathbf{M})^{-1} \mathbf{r} \\ &= \mathbf{L}\mathbf{r},\end{aligned} \tag{2.13}$$

where $\mathbf{L}$ is the successor representation. Each entry $L(s, s')$ gives the future discounted occupancy in state s' starting from state s.

We can similarly define a version of the successor representation that also depends on the initial action a,

$$L(s, a, s') = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k \mathbb{1}[s_{t+k} = s'] \middle| s_t = s, a_t = a \right], \tag{2.14}$$

where $\mathbb{1}[\cdot] = 1$ if $\cdot$ is True and 0 otherwise. This allows the state–action value $Q^{\pi}(s, a)$ under the current policy π to be expressed as

$$Q^{\pi}(s, a) = \sum_{s' \in \mathcal{S}} L(s, a, s') r(s'). \tag{2.15}$$

It is straightforward to adapt a learning rule similar to Equation 2.8 that learns the successor representation $L(s, a, s')$ and $r(s')$ independently.

The successor representation can be beneficial because it isolates the task dynamics, which change slowly as a multi–step function of the policy, from the immediate rewards, which can be updated rapidly. Consider the case where the Q–values and policy have nearly converged to optimality, then the agent experiences a surprising reward (Figure 2.1). In standard Q–learning or Dyna–Q, it can take many experiences and/or offline backups before the Q–values re–converge to accurate estimates. With successor representations, the immediate reward estimates $r(s)$ can change quickly, and the agent immediately gains access to the resulting Q–values *under the existing policy*. In Figure 2.1C, the agent's existing policy would cause it to move left in the left arm of the maze; since the leftmost state now has an expected negative reward, all states that tend to transition into the leftmost state over time have their state values reduced. As a result, the agent will already tend to move right in the left arm of the maze, as it should according to the new optimal greedy policy.

The agent can adjust the policy rapidly because $L(s, a, s')$ implicitly contains a model of the environment dynamics (from the definition of $M(s, s')$ in Equation 2.12), as in a model–based approach. However, $M(s, s')$ also depends on the policy, and as a result $L(s, a, s')$ will need to be re–learned as the policy changes, as in model–free approaches. The successor representation can therefore be seen as a hybrid model–based/model–free approach.

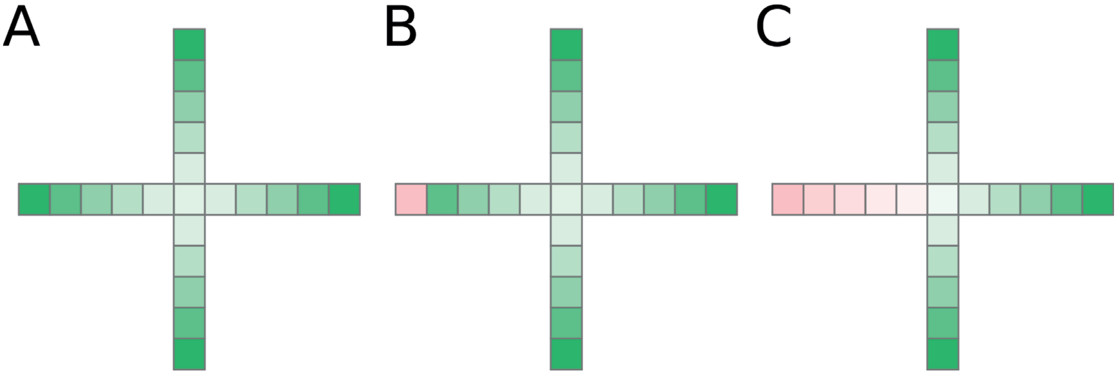

Figure 2.1 – **The successor representation and rapid learning.** **[A]** A simple tabular environment with four possible actions moving in cardinal directions (where the agent stays in the same state if it moves into a wall). At the end of each hallway, the agent receives a reward and the episode terminates; states are shaded according to their learned value after initial training (where darker green corresponds to higher value). **[B]** The left arm stochastically gives a strong negative reward that the agent has not yet experienced. The new state values one step after experiencing the strong negative reward, under standard Q–learning (Equation 2.8). Note that all states except the last one in the left arm still have high values; these could be updated with a background planning method like Dyna–Q or prioritized sweeping. **[C]** The new state values one step after experiencing the strong negative reward, using the successor representation factorization (Equation 2.15). State values in the left arm are immediately reduced.

The future discounted occupancy under a certain policy can also be a useful metric for representing a state space, as considered in Chapter 3.

3 Preplay and Path Planning with Attractor Networks

3.1 Introduction

Animals navigating in a well–known environment can rapidly learn to revisit observed reward locations, often after a single trial [Bast et al., 2009]. The mechanism for rapid path planning remains unknown, though the hippocampus is a natural candidate for investigation given its established role in spatial representation [O'Keefe, 1976]. Experimental results have suggested that the hippocampus is involved in active spatial planning: experiments in "one–shot learning" have revealed the critical role of the CA3 region [Nakazawa et al., 2003, Nakashiba et al., 2008] and the intermediate hippocampus [Bast et al., 2009] in returning to goal locations that the animal has seen only once. In addition, "preplay" activity during Sharp Wave and Ripple (SWR) events in the hippocampus appears to reflect a path planning mechanism [Pfeiffer and Foster, 2013]. This raises the question of whether hippocampal dynamics could support a representation of the current location, a representation of a goal, and the relation between the two.

In this chapter, we propose that a model of CA3 as a "bump attractor" [Samsonovich and McNaughton, 1997, Conklin and Eliasmith, 2005] can be used for path planning. The attractor map represents not only locations within the environment, but also the spatial relationship between locations. In particular, broad activity profiles (like those found in intermediate and ventral hippocampus [Kjelstrup et al., 2008]) can be viewed as a condensed map of a particular environment. In our model, the planned path is observed as sequential activity from the representation of the current position towards the goal location, similar to the preplay observed experimentally in hippocampal activity during navigation tasks [Pfeiffer and Foster, 2013, Wikenheiser and Redish, 2015]. In the model, the sequential activity is initiated by supplying input to the neurons proportional to their activation at the goal site (i.e. reactivating the goal). Unlike other models of rapid goal learning and path planning [Martinet et al., 2011, Ponulak and Hopfield, 2013, Khajeh-Alijani et al., 2015], there is no backward diffusion of a value signal from the goal to the current state during the learning or planning process. Instead, the sequential activity results from the representation of space in the attractor network, even in the presence of obstacles.

After a presentation of the background, the chapter focuses on three different contributions discussed in separate sections. First, we describe how the traditional attractor network model of the hippocampus, based on open–field environments with periodic boundaries, can be extended to realistic environments with boundaries and obstacles. Secondly, we show how an attractor network could give rise to preplay–like trajectories towards goal locations, even if more than one environment is represented in the same attractor network.

Finally, we describe the recurrent structure of the attractor network using a random walk–based successor representation [Dayan, 1993], which represents space according to the number and length of paths connecting different locations. The resulting network can be interpreted as an attractor manifold in a low–dimensional space, where the dimensions correspond to the most relevant eigenvectors of the environment's transition matrix. These low spatial–frequency functions have recently found support in theory as a viable basis for place cell activity [Franzius et al., 2007, Schoenfeld and Wiskott, 2015, Stachenfeld et al., 2014]. We show that, when the attractor network operates in this space and is stimulated with a goal location, the resulting network activity can be interpreted as a viable path to that goal. These trajectories could then be decoded by networks downstream of the attractor network into actions used by the rat to reach a goal. Thus, the bump attractor network can act as a spatial path planning system as well as a spatial memory system.

3.2 Background

3.2.1 Attractor network models of hippocampus

Research in rodent spatial navigation has revealed the existence of place cells in two subregions of the hippocampus: CA3 and CA1 [Moser et al., 2008]. Cells in the CA3 region have significant recurrent interconnectivity and project to the CA1 region, where there are few excitatory recurrent connections [Andersen et al., 2007].

The connectivity profile of CA3 has contributed to multiple models of the CA3 place cell network as a *continuous attractor neural network* [Samsonovich and McNaughton, 1997, Tsodyks, 1999, Doboli et al., 2000, Conklin and Eliasmith, 2005], in loose agreement with experimental findings on the network response of place cells [Wills et al., 2005, Colgin et al., 2010, Jezek et al., 2011]. In a continuous attractor network, the steady–state activity of a population of neurons lies on a low–dimensional manifold [Samsonovich, 2013]. In a spatial bump attractor like that used to model CA3, activity lies on a 2D manifold representing the coordinates $[x_1, x_2]$ of the animal's current location in the environment. A true continuous attractor is not strictly possible with a finite population size, as the steady–state network activity can only collapse to a discrete number of points on the manifold; however, a finite network can be viewed as a (quasi-)continuous attractor if the size of the perturbation required to move the system to a different point on the manifold can be made arbitrarily small by

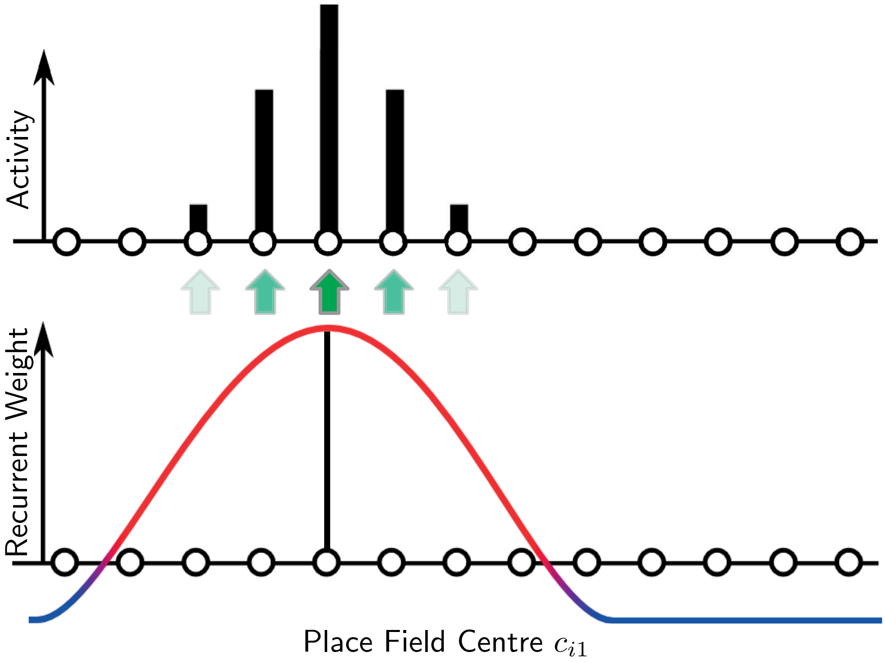

Figure 3.1 – **Schematic of neural activity in a continuous "bump" attractor network**. The schematic shows a cross–section along a single spatial dimension x_1. [**Top**] The white circles represent neurons in a population, arranged along the horizontal axis according to c_{i1}, i.e. the position along the x_1 axis where neuron i responds maximally (the neuron's "place field center"). A feedforward stimulus $\mathbf{x}^{in}$ representing a spatial position in 2D space is applied to the population [green arrows]. The resulting activity in the population is shown. The stimulus induces a Gaussian–like "activity profile" when neurons are aligned according to their place field centers, where the activation of each neuron depends on the distance between its place field center and $\mathbf{x}^{in}$. [**Bottom**] A given neuron (below the black vertical line) recurrently excites neurons with similar place field centers and recurrently inhibits neurons with distant place field centers (as shown by the curve, where red represents excitation and blue represents inhibition). As a result, the activity profile induced by a feedforward stimulus can persist after the stimulus is removed.

increasing the number of neurons [Samsonovich, 2013].

Consider a network of N place cells, where the ith cell responds to the current position $\mathbf{x}^{in}(t) = [x_1(t), x_2(t)]$ in 2D space, with maximal response at that neuron's place field center $\mathbf{c}_i$ (Figure 3.1). The activity a_i of that neuron (analogous to a *firing rate* in a spiking network) is determined in attractor models by

$$\tau \frac{da_i}{dt} = -a_i + g_i \left[\sum_{j=1}^{N} w_{ij}^{rec} a_j + f_i(\mathbf{x}^{in}) \right], \tag{3.1}$$

where g_i is some non–negative monotonically increasing transfer function, w_{ij}^{rec} are the recurrent weights in the network, $\mathbf{x}^{in}$ represents the observed position at time t, and f_i

describes the response of neuron i to an input position (i.e. the place field). Note that g_i can incorporate some multiplicative neuron–specific gain and additive neuron–specific bias (or threshold) before the non–negative nonlinearity.

We describe f_i as a function of the spatial position as

$$f_i(\mathbf{x}^{in}) \propto \exp(-hd^2) \tag{3.2}$$

for some width factor h and distance d (here we take $d = |\mathbf{x}^{in} - \mathbf{c}_i|$). A bump attractor can then be achieved with recurrent weights w_{ij}^{rec} corresponding to excitation between neurons with similar place fields and inhibition between neurons with distant place fields (Figure 3.1 bottom).

Our neuron now has two nonlinearities, g_i and f_i, where f_i is chosen to induce place cell–like activity. Rather than assuming the nonlinearity f_i, we can consider an alternative representation of the input position allowing for a linear response function. For instance, we can achieve a spatial tuning curve as in Equation 3.2 after expressing the input position $\mathbf{x}^{in}$ as an approximately Gaussian–shaped "bump" in space using a set of M orthonormal spatial basis functions x_k^{in} for $k = 1, \dots, M$ [Conklin and Eliasmith, 2005]. In this coordinate system, the nonlinear function of the input position can be expressed as

$$f_i(\mathbf{x}^{in}) = \sum_{k=1}^{M} w_{ik}^{ff} x_k^{in}, \tag{3.3}$$

where the feedforward weights w_{ik}^{ff} depend on the neuron's place field center $\mathbf{c}_i$. We determine the input weights using a normalized vector of coordinates $\mathbf{e}_i = [e_{i1}, e_{i2}, \dots, e_{ik}, \dots, e_{iM}]$, the neuron's preferred direction vector or "encoding vector", optimized to make an approximately Gaussian function in space. The feedforward weights are then $w_{ik}^{ff} = \alpha e_{ik}$ for some scalar input gain factor $\alpha \ll 1$. The dot product $\mathbf{e}_i^T \mathbf{x}^{in}$ defines a similarity measure between the neuron's selectivity and the current input; i.e. the overlap between the input Gaussian and the neuron's Gaussian tuning curve. For the illustrations in this section, the basis corresponds to a set of 2D spatial Fourier functions. Note that the number of functions M required will depend on the width of the Gaussian bumps represented by the input and the network; large bumps can be represented using only a small number of low–frequency Fourier functions.

After this definition, we can re–write Equation 3.1 as

$$\tau \frac{da_i}{dt} = -a_i + g_i \left[\sum_{j=1}^{N} w_{ij}^{rec} a_j + \sum_{k=1}^{M} w_{ik}^{ff} x_k^{in} \right]. \tag{3.4}$$

Here, we assume that the "activity" x_k^{in} of each basis function is available directly as an input to the network; we will consider how this assumption can be relaxed at the end of the next section. Thus, x_k^{in} is a Gaussian centered at the current input location projected onto a basis function k.

3.2.2 Spatial representations in attractor networks

The weight profile shown in Figure 3.1 is the same for all neurons in the network, which is appropriate if we assume the environment is periodic (toroidal) or infinite. Before we generalize the attractor network to represent finite non–toroidal environments, we first consider how to express the representation in the attractor network, and how to determine appropriate recurrent weights. The central idea is that the network activities can be seen as representing a control theoretic vector–valued variable **x** corresponding to a set of coordinates on the basis functions considered in the previous section. Given appropriate recurrent weights, the dynamics of the network representation **x** correspond to a low–pass filter over recent input positions $\mathbf{x}^{in}$ supplied to the network. The analysis here is based primarily on the Neural Engineering Framework (NEF) [Eliasmith and Anderson, 2004] and existing models of bump attractors based on the NEF in toroidal environments [Conklin and Eliasmith, 2005].

In the following, we define an "effective input" $\mathbf{x}^{eff} \in \mathbb{R}^M$ to the network. At first, we will use the effective input as a mathematical construct to derive an appropriate recurrent weight factorization, allowing for an alternative description of the network dynamics in terms of **x**. After determining this factorization, we will show how the effective input corresponds to the combined feedforward and recurrent input to the neurons in the network.

First, we consider how such an effective input vector could be recovered after being multiplied with the network encoding vectors $[\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_j, \ldots, \mathbf{e}_N]$ and passed through the nonlinear transfer function $g_j[\cdot]$. As before, each encoding vector $\mathbf{e}_j \in \mathbb{R}^M$ describes the selectivity of neuron j to an effective input. We assume that the effective input can be approximately linearly decoded using a set of neuron–specific "decoding vectors" $[\mathbf{d}_1, \mathbf{d}_2, \ldots, \mathbf{d}_j, \ldots, \mathbf{d}_N]$. That is, we write

$$\mathbf{x}^{eff} \approx \sum_{j=1}^{N} \mathbf{d}_j g_j \left[\mathbf{e}_j^T \mathbf{x}^{eff}\right], \quad \text{where} \quad \mathbf{d}_j \in \mathbb{R}^M. \tag{3.5}$$

Next, we use the same decoding vectors $\mathbf{d}_j$ and apply them to the population activities as defined in Equation 3.4. Here, we define the result to be **x**, i.e. the population representation of the spatial function (e.g. Gaussian bump) corresponding to the animal's position. That is,

$$\mathbf{x} = \sum_{j=1}^{N} \mathbf{d}_j a_j. \tag{3.6}$$

Finally, we fix the recurrent weights in the population to

$$w_{ij} = (1-\alpha)\mathbf{e}_i^T \mathbf{d}_j. \tag{3.7}$$

Re–evaluating Equation 3.4 using Equation 3.6 and Equation 3.7, we find that

$$\begin{aligned}\tau\frac{da_i}{dt} &= -a_i + g_i\left[\sum_{j=1}^{N} w_{ij}^{rec} a_j + \sum_{k=1}^{M} w_{ik}^{ff} x_k^{in}\right] && 3.8\\ &= -a_i + g_i\left[\sum_{j=1}^{N}\left((1-\alpha)\mathbf{e}_i^T\mathbf{d}_j a_j\right) + \alpha\mathbf{e}_i^T\mathbf{x}^{in}\right] && \text{from Equation 3.7}\\ &= -a_i + g_i\left[(1-\alpha)\mathbf{e}_i^T\mathbf{x} + \alpha\mathbf{e}_i^T\mathbf{x}^{in}\right] && \text{from Equation 3.6}\\ &= -a_i + g_i\left[\mathbf{e}_i^T\left((1-\alpha)\mathbf{x} + \alpha\mathbf{x}^{in}\right)\right].\end{aligned}$$

Applying our decoding vectors to both sides of the equation, summing over the population activities and taking $\mathbf{x}^{eff} = (1-\alpha)\mathbf{x} + \alpha\mathbf{x}^{in}$, we find that

$$\begin{aligned}\tau\sum_i^N \mathbf{d}_i\frac{da_i}{dt} &= -\sum_i^N \mathbf{d}_i a_i + \sum_i^N \mathbf{d}_i g_i\left[\mathbf{e}_i^T\left((1-\alpha)\mathbf{x} + \alpha\mathbf{x}^{in}\right)\right] && 3.9\\ \tau\frac{d\mathbf{x}}{dt} &\approx -\mathbf{x} + \left((1-\alpha)\mathbf{x} + \alpha\mathbf{x}^{in}\right) && \text{from Equation 3.5 and Equation 3.6}\\ \tau'\frac{d\mathbf{x}}{dt} &\approx -\mathbf{x} + \mathbf{x}^{in}, \quad \text{where} \quad \tau' = \tau/\alpha.\end{aligned}$$

The network can thus be seen as an integrator or low–pass filter over the input positions $\mathbf{x}^{in}$ (Figure 3.2). The effective input introduced earlier corresponds to a convex combination of the network representation $\mathbf{x}$ and the input $\mathbf{x}^{in}$ under this weight factorization. Note that as the recurrent weight magnitude increases (i.e. α decreases) the effective time constant of the filter also increases, reflecting the hysteresis induced by the recurrent weights. Strong recurrent weights can also filter out high–frequency noise in the input and allow the network to maintain a memory of the input for some time after it is removed.

To derive the decoding vectors. we take a set of P vectors $[\mathbf{x}_1^{eff}, \mathbf{x}_2^{eff}, \ldots, \mathbf{x}_P^{eff}]$ corresponding to Gaussian profiles of constant width evenly spaced throughout the environment. We then consider the squared error after approximating these effective inputs from their resulting nonlinear representations in Equation 3.5, i.e.

$$E = \frac{1}{2}\sum_{n=1}^{P}\left[\mathbf{x}_n^{eff} - \sum_{j=1}^{N}\mathbf{d}_j g_j\left[\mathbf{e}_j^T\mathbf{x}_n^{eff}\right]\right]^2. \qquad 3.10$$

Finding the optimal decoding vectors for these P effective inputs corresponds to a linear regression problem; they can thus be determined using the Moore–Penrose pseudoinverse of the matrix of representations in response to the inputs. It is straightforward to adjust the solution based on the expected variance of independent, Gaussian noise in each neuron's response profile, corresponding to an L2 regularization [Eliasmith and Anderson, 2004]. Note that each neuron's optimal decoding vector depends on the activity of the other neurons in the population; in general, the error in decoding will decrease as more neurons are added to the population.

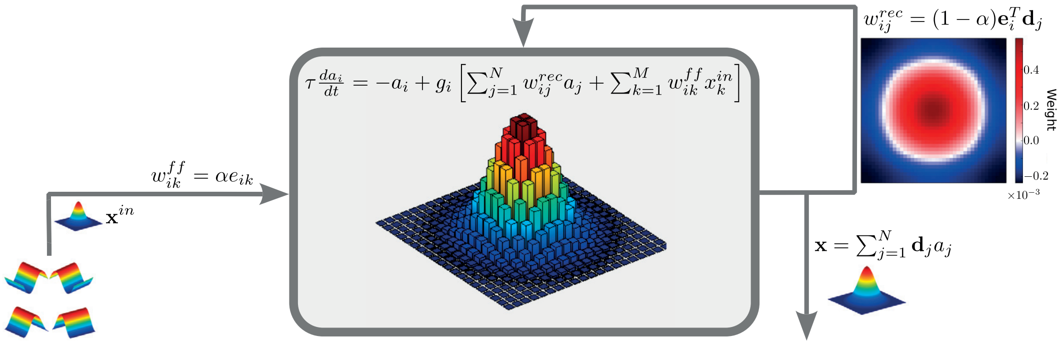
$w_{ij}^{rec} = (1-\alpha)\mathbf{e}_i^T\mathbf{d}_j$
0.4
0.2
0.0
−0.2
$\times 10^{-3}$
Weight
$\tau\frac{da_i}{dt} = -a_i + g_i\left[\sum_{j=1}^{N} w_{ij}^{rec}a_j + \sum_{k=1}^{M} w_{ik}^{ff}x_k^{in}\right]$
$w_{ik}^{ff} = \alpha e_{ik}$
$\mathbf{x}^{in}$
$\mathbf{x} = \sum_{j=1}^{N}\mathbf{d}_j a_j$

Figure 3.2 – **A toroidal bump attractor network**. [**Left**] An input vector $\mathbf{x}^{in}$ is supplied to the network representing the animal's position in space using a basis consisting of low–frequency spatial basis functions. This vector corresponds to a 2D Gaussian bump in a spatial reference frame. The width of the bump depends on the number of basis functions used; additional high–frequency functions result in narrower bumps. [**Middle**] Encoding vectors $\mathbf{e}_i$ also correspond to localized Gaussian bumps in 2D space. As a result, neurons with encoding vectors most similar to the current input have the strongest response. If the neurons are arranged in a 2D grid with equally spaced place field centers, the resulting activity profile likewise resembles a Gaussian function. [**Right**] The activity can be decoded to estimate the network representation $\mathbf{x}$. With recurrent weights that feed $\mathbf{x}$ back into the network, the decoded network activity approximates a low–pass filter of the input. The recurrent outgoing weights from neuron i are shown, plotted in a 2D grid according to the 2D spatial relationship between neuron i's place field center and the place field center of each postsynaptic neuron (where neuron i's place field center corresponds to the middle of the plot). As in Figure 3.1, the neuron excites other neurons with similar place field centers and inhibits those with distant place field centers.

In a toroidal attractor network, the resulting recurrent weights from this procedure are shown in Figure 3.2. Note that they resemble the Gaussian or Mexican–hat–like recurrent connectivity usually imposed heuristically to obtain an attractor network, as in Figure 3.1.

Until this point, we have described the network as a low–pass filter over the input. It is an "attractor network" because the error minimization in Equation 3.10 is performed with respect to a manifold of coefficients that correspond to localized Gaussian bumps in space. As a result, the low–pass filter dynamics are only stable when the network operates in the vicinity of this manifold, and the network activity tends to collapse back to this manifold (i.e. it forms an attractor) [Eliasmith and Anderson, 2004]. For instance, if the initial effective input $\mathbf{x}^{eff} = (1-\alpha)\mathbf{x} + \alpha\mathbf{x}^{in}$ corresponds to a mixture of two narrow Gaussian bumps $\mathbf{x}$ and $\mathbf{x}^{in}$ at different locations in space, the recurrent connectivity will cause the network representation $\mathbf{x}$ to rapidly collapse to a single bump rather than a mixture of the two (Eliasmith and Anderson [2004]).

As an alternative to an orthonormal basis as an input to the network, we note that grid cells in the entorhinal cortex (the primary input region to CA3 [Moser et al., 2008]) correspond to a population of spatially global, periodic functions that can be linearly weighted to form place fields in an open environment [Solstad et al., 2006]. Thus, for a hippocampal bump attractor

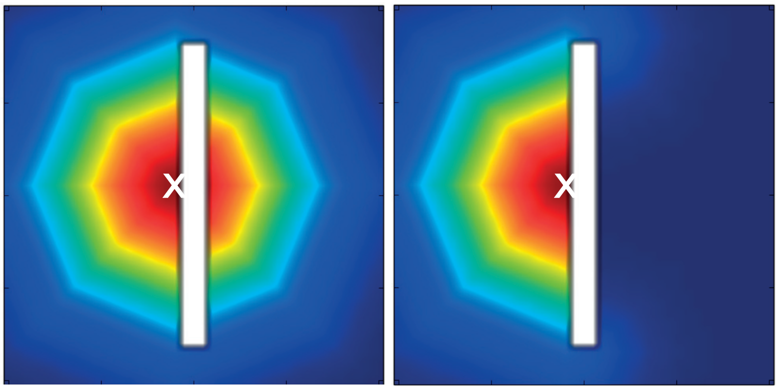

Figure 3.3 – **Non–Geodesic vs. Geodesic place fields**. The colour indicates a cell's simulated activity according to the animal's 2D position in an environment divided by a wall (in white). Here, darker red corresponds to higher activity. The two cells have a maximal response at the same position in the environment, marked with a white cross, which we refer to as the cell's place field center. [**Left**] Place cell response under a non–geodesic metric. Here, place cell activity varies with the shortest path distance from a position on the discrete grid to the place cell center, moving horizontally or vertically between points on the discrete grid, whether or not those points are occupied by a barrier. [**Right**] Actual place cell responses are better represented by fields based on geodesic distance. According to the geodesic metric, the cell's response depends on the shortest path along the discrete grid between the animal's location and the cell's place field center, excluding grid positions occupied by obstacles.

network, we may instead take x_l^{in} as the response of grid cell l to the current location, with $w_{il}^{ff} = \alpha \mathbf{e}_i^T \mathbf{d}_l$ as the connection from grid cell l to place cell i, where $\mathbf{d}_l$ decodes the grid cell representation into an M–dimensional basis representation. In the following we assume that the "activities" of the M basis functions are available directly as input. Later we will consider an explicit grid cell population as input (Chapter 4).

3.3 Geodesic attractor networks

In general, we are interested in representing environments with boundaries and obstacles. In this case, place cell responses respect the topology of the environment; for instance, a cell which responds strongly on one side of a thin wall has little or no response on the opposite side [Gustafson and Daw, 2011].

Rather than using neurons that respond according to the distance between the animal's location and the place field center ignoring obstacles (here referred to as a *non–geodesic* metric), we consider cell responses that scale with *geodesic* distance – i.e. the shortest path the animal could take through the environment between its current position and the place field center. An example profile under a geodesic metric is shown in Figure 3.3 (right), compared to a profile based on distances ignoring the obstacle in the environment (left). We use the term "geodesic" following Gustafson and Daw [2011], where it refers to the fact that obstacles

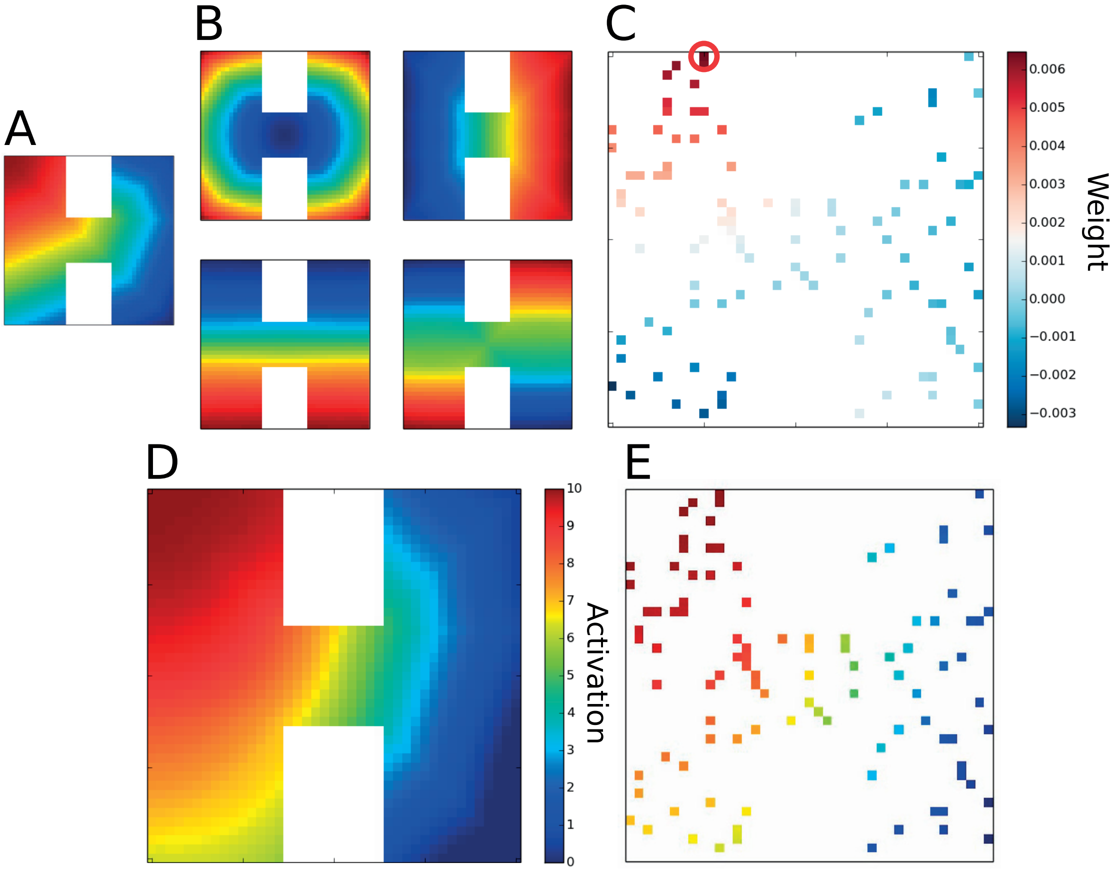

Figure 3.4 – **A geodesic attractor network in an environment with barriers**. [**A**] Example of one of the P large geodesic profiles used for constructing the network weights. This example has a place field centered in the top left corner of the maze. [**B**] The four basis functions associated with the highest magnitude singular values after performing SVD on a set of geodesic spatial profiles evenly spaced throughout the environment. These four low–frequency functions, which varied smoothly across the environment, could explain $> 95\%$ of the variance in the P large geodesic profiles. [**C**] 100 positions were randomly selected in the environment, and the coordinates associated with those positions were used as encoding vectors. Coloured pixels correspond to the 100 sampled positions; as a result, each pixel also defines one neuron's place field center. We plot the 100 outgoing recurrent weights from the ith neuron (which has a place field center circled in red); each pixel's colour gives the value of the recurrent weight from the circled ith neuron to the neuron with a place field center at that pixel's position. Note the similarity to a traditional toroidal attractor weight profile (Figure 3.2), with recurrent excitation between neuron's with similar place field centers and inhibition between neurons with distant place field centers. [**D**] The place field for the ith neuron circled in C, i.e. the steady–state activity of the ith neuron when a constant input stimulus is applied corresponding to each position in the environment. [**E**] The network activity profile for an input corresponding to the top–left corner. The colour bar is shared between D and E.

effectively curve or warp the geometry of the environment.

We consider the problem of building an attractor network that can stably represent such geodesic profiles in a given environment. First, we calculate geodesic spatial functions for $[1,\ldots,n,\ldots,P]$ 2D place field centers evenly spaced throughout an environment with obstacles, where the nth function decreases exponentially with the shortest traversable distance d between the nth place field center and a location in the environment (e.g. Figure 3.4A). The full set of geodesic profiles can be represented with a $P \times Z$ matrix $\mathbf{R}$, where Z is the number of possible positions in the environment (determined by the resolution of the representation).

Taking the Singular Value Decomposition of $\mathbf{R}$ gives $\mathbf{R} = \mathbf{USV}^T$, where $\mathbf{S}$ is a diagonal matrix of singular values in order of decreasing magnitude. Here, $\mathbf{V}$ corresponds to an environment–specific orthonormal basis (e.g. Figure 3.4B), like the Fourier basis in periodic environments. Similar to a Fourier basis, the vectors in $\mathbf{V}$ correspond to topologically–smooth functions, with larger singular values associated with functions of lower spatial frequency.

The nth row of $\mathbf{US}$ gives the coordinates on the basis $\mathbf{V}$ that reconstruct a geodesic profile centered at position n. We took the vectors in $\mathbf{US}$ and truncated them to length M, after arranging them in order of decreasing singular value magnitude. We picked N locations randomly and expressed the locations in the new basis to construct the encoding vectors $[\mathbf{e}_1,\ldots,\mathbf{e}_N]$ for the N neurons in the attractor network. In addition, we used all P rows in $\mathbf{US}$ to generate the decoding vectors minimizing the error in Equation 3.10. Taking $w^{rec}_{ij} = (1-\alpha)\mathbf{e}_i^T\mathbf{d}_j$ and $w^{ff}_{ik} = \alpha e_{ik}$, this process fully defines the recurrent and feedforward weights for the environment–specific attractor network. We used a rectified linear nonlinearity for g_i.

The resulting recurrent weights for an H–maze environment are shown in Figure 3.4C, an example place field is shown in Figure 3.4D, and an activity profile (for a set of 100 neurons with randomly chosen place field centers) is shown in Figure 3.4E.

3.4 Trajectories from attractor dynamics

From Equation 3.9, we can see the network as maintaining a memory of the animal's current location $\mathbf{x}(t)$ that follows a change in feedforward input on the timescale of τ'. We next consider how the attractor network structure could contribute to the sequential activity observed in place cell networks during SWRs and theta cycles [Pfeiffer and Foster, 2013, Wikenheiser and Redish, 2015].

We consider the case where the current state represented by the network, $\mathbf{x}(t)$, and the new input, $\mathbf{x}^{in}(t)$, are significantly different. In this case, we can see the network as generating a low–pass filtered version of the transition between the initial and new stimulation coordinates. In a goal–directed navigation task, $\mathbf{x}(t)$ and $\mathbf{x}^{in}(t)$ may correspond to the animal's current location and a goal location, respectively.

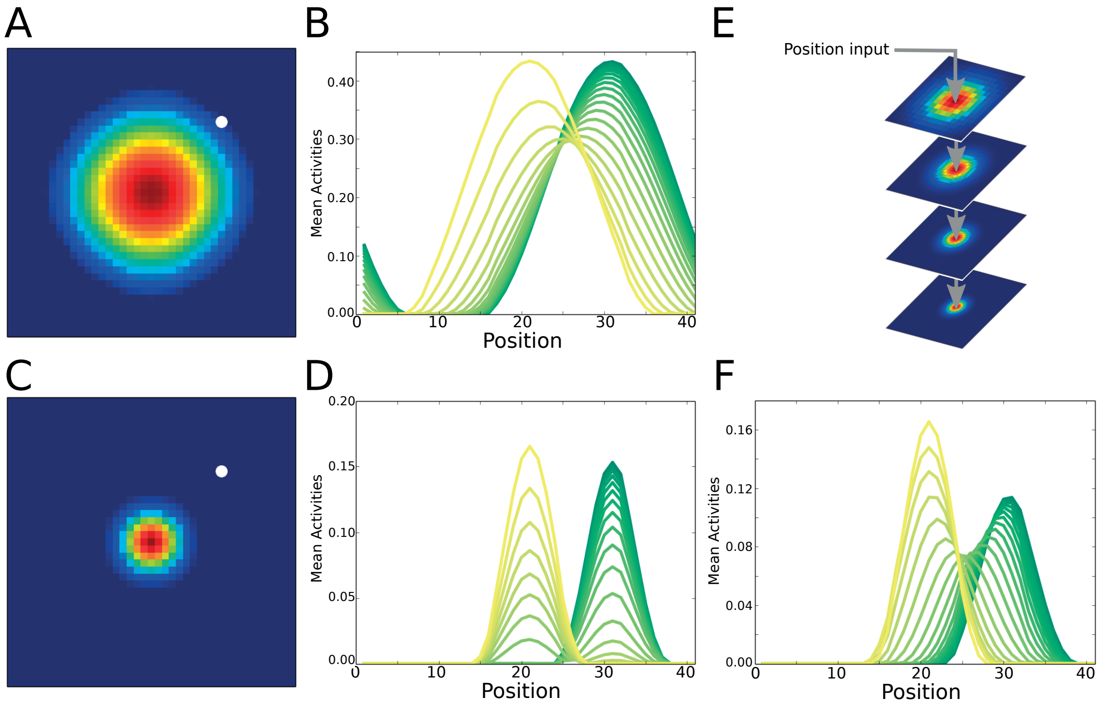

Figure 3.5 – **Sequential activity generated by low–dimensional representations**, with units possessing place fields centers equally spaced along a toroidal grid. [**A**] Neurons with large place fields (generated using 9 Fourier basis functions) have place field centers that evenly tile the entire toroidal environment. They produce a large activity profile [dark red: high activity, dark blue: no activity]. Activity was initialized at the center of the image and stimulation was delivered, centered at the white dot. [**B**] Activity profiles collapsed on the x_1 spatial axis for the first 200 time steps τ of stimulation; profiles are plotted at every 10 steps, moving from yellow to green. Stimulation produced a smooth movement of the bump with an intervening decrease in activity rates. [**C**] The same stimulation process was used on a network of neurons with narrow place fields and narrow resulting activity profiles. [**D**] With narrow fields, the bump decreased at the original position and reappeared at the stimulated location. [**E**] A hierarchical structure was used with four populations of neurons, where the input stimulus was applied to the population with the largest place fields and each population projected to the next population with smaller place fields (Equation 3.11). [**F**] With the hierarchical structure, the bump moves smoothly to the new location at the lowest level.

In mean–field models of periodic continuous attractor networks with broad bump profiles and strong recurrent excitatory feedback, the network activity has been shown to reflect a smooth transition between the initial position and the stimulated position with relatively little change in the profile, a phenomenon referred to as “virtual rotation” [Hansel and Sompolinsky, 1998]. Virtual rotation results from the overlap between the initial activity profile and the input stimulus in combination with the recurrent excitatory dynamics, which pull the network activity towards a bump–like profile.

We reproduce the same phenomenon in a continuous attractor generated according to the

NEF in the top row of Figure 3.5. In this case, we evenly tiled the [41**x**41] grid with neurons, such that each neuron's encoding vector made it maximally receptive to a Gaussian bump at a particular 2D location. When the input and recurrent weights were determined using broad Gaussian bumps (generated from a low–dimensional Fourier basis), we observed virtual rotation (Figure 3.5B). When narrow Gaussian bumps were used (generated from a high–dimensional Fourier basis), the network representation jumped directly to the stimulated location (Figure 3.5D).

The low–pass filter in Equation 3.9 requires the network to represent some convex combination of the initial and final positions. In the low–dimensional, low–frequency basis used to generate the broad activity profiles, the possible positions $[\mathbf{x}_1 \ldots \mathbf{x}_P]$ form a manifold that is nearly linear close to the origin (i.e. closed under addition and multiplication). As a result, the network representation of $\mathbf{x}$ in Equation 3.9 will, at any point in time, be close to the manifold of possible positions. As well, since these points are used to determine the network weights by minimizing the error in Equation 3.10, this manifold is also where the network error is lowest. In contrast, attractor networks generated using narrow bumps correspond to high–dimensional, highly nonlinear manifolds. In other words, convex combinations of narrow Gaussian bumps do not resemble single Gaussian bumps, and they are not well–represented by the network. As a result, the network profile tends to fade from the current position and rise at the stimulated position, without a peak ever appearing between the two (Figure 3.5D).

Drawing on the observation of small place fields in the dorsal hippocampus, but larger place fields in the intermediate and ventral hippocampus [Jung et al., 1994, Kjelstrup et al., 2008], we propose that sequential activity could arise from recurrent dynamics in these more ventral regions. However, preplay trajectories have been observed along the entire dorsoventral axis in theta phase precession [Kjelstrup et al., 2008], and primarily recorded in the dorsal hippocampus during SWR events [Pfeiffer and Foster, 2013]. The model can account for these observations under the hypothesis that sequential activity in dorsal regions (where place fields are narrow) is inherited from intermediate and ventral regions.

Here, we introduce a hierarchical model, where populations with large place fields project to and influence populations with smaller place fields during sequential activity (e.g. Figure 3.5E). At the top level (largest place fields), the dynamics are determined according to Equation 3.4 as usual. For each lower level, the neuron dynamics are amended to

$$\tau \frac{da_i^l}{dt} = -a_i^l + g\left[\sum_{j=1}^{n_l} w_{ij} a_j^l + \sum_{k=1}^{n_{l+1}} w_{ik}^{td} a_k^{l+1}\right] \qquad 3.11$$

where the superscript l denotes the level, the activity of units a_k^{l+1} represent the estimate of the position at a higher level (a broader, lower–frequency representation), and the top–down weights w_{ik}^{td} are determined using the least–squares decoders of these low–frequency components. Notably, the weights at all levels were determined with respect to the same basis functions. As a result, a neuron with a large place field at a given position maximally excites neurons with smaller place fields down the hierarchy near the same position. The top–down signal provides a low–dimensional estimate of the path for the lower level to follow, while the recurrent dynamics at the lower level impose the high–frequency dynamics that keep the activity close to the attractor manifold. By feeding the estimate iteratively through several levels, a smooth path can be generated between distant locations in the environment, even at the level of small place fields.

In order to do this, we generated a four–level network using a 2D Fourier basis, with gradually smaller place fields (higher–dimensional representations) at each level. As shown in Figure 3.5F, a path generated by virtual rotation in a network with large places could also produce a smooth path in the network with small place fields, by filtering the activity through each level successively. This was not possible when a goal signal was supplied directly to the population with small place fields; in that case, the bump jumped directly from the start location to the goal location (Figure 3.5D).

By generating the weights according to the NEF rather than a Gaussian heuristic, the same phenomenon of virtual rotation can be extended to non–toroidal environments with boundaries (Figure 3.6B). Here, the arrows indicate the initial shift in position of the most active neuron after tiling the space such that exactly one neuron had a place field centered at every position, and supplying feedforward input to the network corresponding to a given position (the red dot). As in the toroidal environment, the shift was generally smooth relative to the local topology with large place–field units, but was erratic when place fields were small relative to the size of the environment (Figure 3.6C). However, in a hierarchical 2–layer network, top–down stimulation from the high–level population of 100 neurons with large place fields could induce smooth long–distance movement in the network with small place fields (Figure 3.6D).

3.4.1 Spiking networks

The same behaviour is qualitatively demonstrated in a hierarchical leaky integrate–and–fire (LIF) spiking network with 3 layers (Figure 3.7). We segregated the neurons into excitatory and inhibitory subgroups following the generally sharp division of neurons by chemical effect at outgoing synapses (i.e. Dale's Principle, Eccles et al. [1954]). In this simple toroidal environment, we used a Gaussian weight profile within and between the excitatory subpopulations and a uniform weight profile between excitatory and inhibitory populations, and within inhibitory populations. Fewer neurons were used at each increasing level in the hierarchy, reflecting the decrease in precision (increase in width) of the place representation; as well, we applied the position input at all levels of the hierarchy. Without top–down input, the narrow activity profiles did not move until they eventually jumped directly to the stimulated position. With top–down input from the broader profiles, they moved smoothly to the stimulated position.

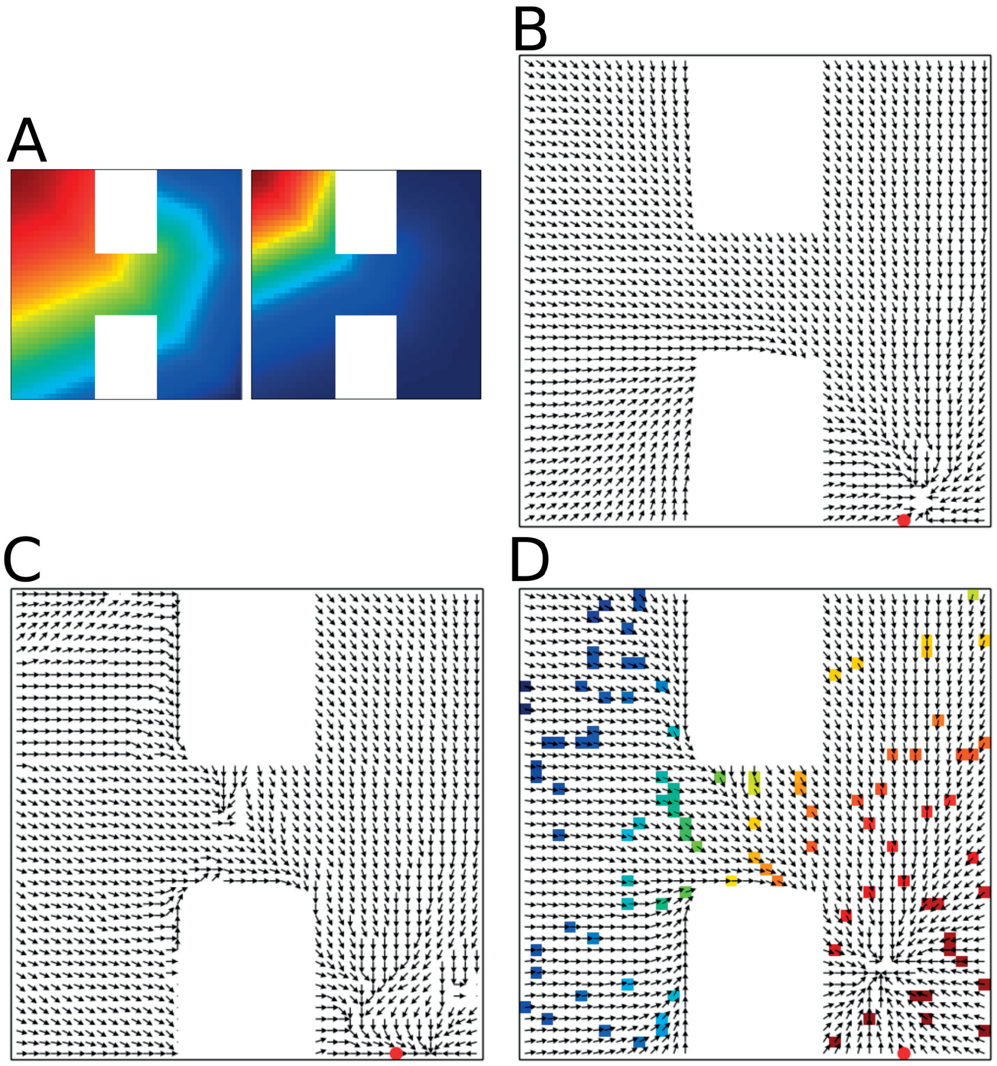

Figure 3.6 – **Trajectories in an environment with barriers**. [**A**] Comparison of a large geodesic place field (left) and a small geodesic field (right), both centered in the top left corner of the maze. [**B**] An attractor network was composed with a large place field neuron centered at each of the 1351 locations in the environment not occupied by a barrier. Activity was initialized at each point in the environment, and weak bump–shaped input applied, centered at the bottom right [red dot]. The direction of shift in the most active unit after the first 10 time steps τ of stimulation is shown by the vector field. [**C**] Paths were generated again using small place fields (higher dimensional representations). With smaller fields, the decoded trajectories were generally accurate near the stimulated location, but activity profiles far from the stimulated location moved erratically or jumped directly to the stimulus (e.g. bottom left of maze). [**D**] A network of 100 large place field neurons with randomly situated centers was generated [coloured overlay represents the steady–state activity in this network for a bump centered at the red dot]. This network was stimulated with an input corresponding to the position of the red dot, and provided top–down stimulation to the lower level, causing activity in the small place field network to move towards a fixed point near the stimulation center from across the environment [vector field].

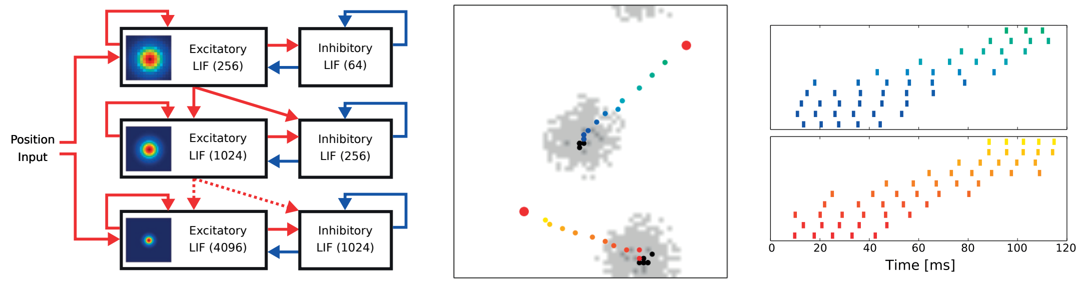

Figure 3.7 – **Sequential activity generated in a spiking, hierarchical, toroidal attractor network**. [**Left**] Network diagram. Each level contains an excitatory and a (global) inhibitory population. Excitatory weights were generated with a Gaussian profile. Profile width was increased and neuron count decreased at each higher level (population sizes are shown in brackets). Red lines: excitatory connections, blue lines: inhibitory connections, dotted lines: manipulated top–down connections. [**Middle**] Overlaid results from two separate trials in the lowest level (smallest profile widths). Neurons are arranged in the grid according to their place field centers. Two different initial activity profiles [grey, spikes in the first 10ms] and stimulation positions [large red dots] are shown. Without the dotted top–down connections, the profiles did not move during the initial stimulation input period [small black dots, activity profile means plotted every 10 ms for 100 ms]. With top–down connections, the profile means moved towards the stimulated position [colored dots, plotted every 10 ms, blue to green for Trial 1 and orange to yellow for Trial 2]. [**Right**] Raster plots for corresponding neurons in middle diagram, showing sequential firing due to hierarchical attractor dynamics.

3.4.2 Multichart attractor networks

Place cells have been experimentally observed to “remap” between different environments, i.e. cells which have correlated firing fields in one environment tend to show no correlation in another environment [McNaughton et al., 1996]. Theorists have shown that a recurrent network, such as CA3, can potentially store many uncorrelated maps simultaneously (a “multichart” representation, Samsonovich and McNaughton [1997]), where each map corresponds to a different environment.

Multichart attractors can be achieved with recurrent connectivity corresponding to the superposition of multiple recurrent weight profiles for multiple environments. Provided that the place cell activities are uncorrelated across environments, and the number of environments is small relative to the network size, the recurrent attractor connections can be superimposed such that the bump is localized only in the environment represented by the feedforward input [Samsonovich and McNaughton, 1997]. The cooperative activity of the neurons maintains the representation in that environment, as the representation has no spatial coherence in any other environment represented in the recurrent weights.

To investigate whether sequential activity could occur in multichart attractors, we generated a geodesic multichart attractor with 100 units for two environments, where each unit was randomly assigned a large place field center in both environments (Figure 3.8). In both environ-

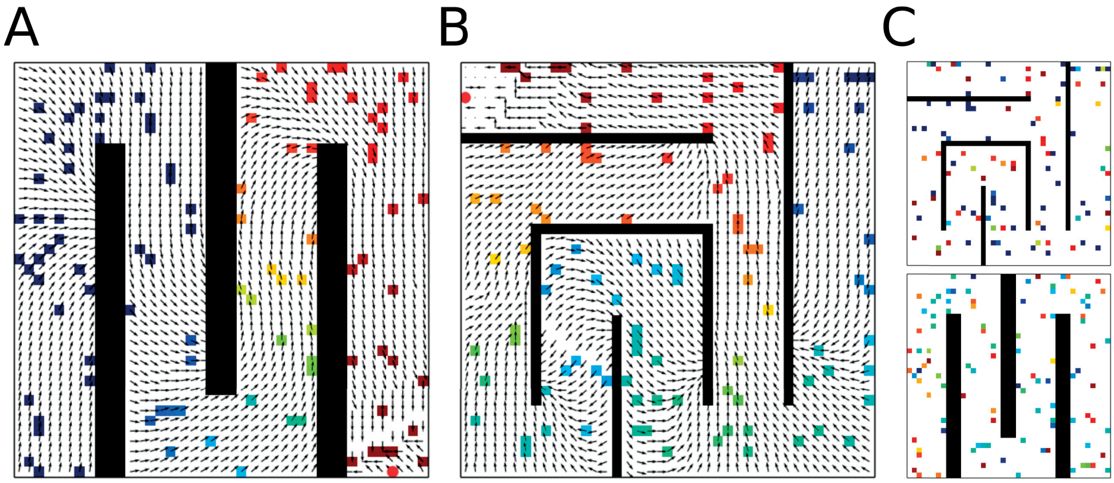

Figure 3.8 – **A small network of large place–field units can generate geodesic trajectories in multiple environments**. Two orthogonal attractors representing different environments were stored in the same network of 100 units. [**A**] Initial activity trajectory from each point in a twisting hallway environment, with stimulation provided at the red dot, as decoded from the network activity. Place field centers and activities, for a bump centered at the red dot, are shown in overlay. [**B**] The same network representing a maze–like environment, with trajectories and activities for stimulation centered at the red dot. Note the discontinuity in the lower left, where two different paths are equidistant from the goal. [**C**] The same activities as in the previous two plots, plotted according to the place field centers in the other environment.

ments, these place fields were well–represented using 6 basis functions. To generate multiple charts in the same network, we determined the recurrent weights solved by minimizing the representation error across both environments; i.e.

$$E = \frac{1}{2}\sum_{n_1=1}^{P_1}\left[\mathbf{x}'_{n_1} - \sum_{j=1}^{N}\mathbf{d}_j a_j(\mathbf{x}'_{n_1})\right]^2 + \frac{1}{2}\sum_{n_2=1}^{P_2}\left[\mathbf{x}'_{n_2} - \sum_{j=1}^{N}\mathbf{d}_j a_j(\mathbf{x}'_{n_2})\right]^2, \qquad 3.12$$

for P_1 positions in environment 1 and P_2 positions in environment 2, where the augmented input vectors correspond to

$$\mathbf{x}'^{T}_{n_1} = [x_{1,n_1}, x_{2,n_1}, x_{3,n_1}, x_{4,n_1}, x_{5,n_1}, x_{6,n_1}, 0,0,0,0,0,0] \quad \text{and} \qquad 3.13$$

$$\mathbf{x}'^{T}_{n_2} = [0,0,0,0,0,0, x_{1,n_2}, x_{2,n_2}, x_{3,n_2}, x_{4,n_2}, x_{5,n_2}, x_{6,n_2}], \qquad 3.14$$

i.e. for each 12–dimensional input vector used to determine the weights, 6 dimensions represented a position in one environment and the other 6 dimensions were set to 0. As a result, all encoding and decoding vectors were 12–dimensional as well. This process results in the generation of two orthogonal attractor manifolds.

We then tested the possibility of sequential activity from virtual rotation in the multichart network. For each environment, the initial movement of the activity profile from each point in that environment was recorded, after stimulating the network with a single goal location. In this case, the movement of the profile was not judged by the shift in the most active unit, but rather by the shift in the maximum position of the spatial function represented by the network activities after decoding them with the optimal decoders from Equation 3.10.

The networks were able to generate trajectories across both environments and displayed diverging activity at equidistant points (e.g. Figure 3.8B, bottom middle of maze), although high–frequency elements like corners were sometimes filtered out by the network (e.g. Figure 3.8A, top left of maze).

3.5 Successor Representation–based geodesic attractors

In the spatial attractor network models presented thus far, the shortest–path distances between all grid positions in the environment are used to determine appropriate recurrent weights. In the following, we consider how large–scale features of the environment could be represented in the network weights using only local distance information about the environment. To do this, we derive the network weights using a local random walk combined with the successor representation framework. In addition, we provide an interpretation of the evolution of the network's activity in light of the successor representation. We focus on the network activity in large place field populations, without the hierarchical structures considered in previous sections.

3.5.1 Representing space using the successor representation

We represent the relationship between locations in an environment including obstacles using the geodesic similarity metric considered in Section 3.3. Given two states $s = [x_1, x_2]$ and $s' = [x'_1, x'_2]$ in the 2D plane, their (symmetric) similarity f is given by

$$f(s,s') = f(s',s) = \exp\left(-hd^2\right) \tag{3.15}$$

where h is a width term as in Equation 3.2 and d is the distance between s and s' along a discrete grid, respecting walls and obstacles (as in the geodesic fields previously considered). The metric is localized such that $f(s,\cdot)$ resembles, as a function of the second argument, a small bump in space truncated by walls with a maximum located at s. Unlike the geodesic functions considered in the previous section, we take a much larger, fixed value for h; the resulting bump is therefore much smaller (Figure 3.9 center bottom, cf. Figure 3.4A). Normalizing the similarity metric gives

$$p(s,s') = \frac{f(s,s')}{\sum_{s'} f(s,s')}. \tag{3.16}$$

The normalized metric can be interpreted as a transition probability from s to s' under a random walk. This random walk results in small steps to positions in the near vicinity of

s; in the 41×41 discrete grid environments considered in the following, $p(s, s') < 0.1\%$ for positions s and s' more than 4 grid positions apart. Taking $p(s, s') = \sum_a T(s'|s, a)\pi(a|s)$, the transition probability can be seen as implicitly incorporating both the action–conditional transition probabilities $T(s'|s, a)$ and a policy $\pi(a|s)$ [Stachenfeld et al., 2014]; the random walk can therefore be interpreted as arising from a deterministic transition function and a policy describing a small random step from the current position. The statistics of the random walk could be determined from one–step observations during random exploration of the environment.

In order to capture large–scale structure using the local transition function $p(s, s')$, we consider the Markov chain described by the transition matrix $\mathbf{P}$, formed from the elements $p(s, s')$. In addition, we assume that there is a single "goal" state in the environment at any point in time, described by the one–hot vector $\mathbf{r}$ with $r(s') = \delta_{s'g}$ (δ denotes the Kronecker delta function and the index g denotes the goal state). Using the successor representation (Section 2.4.3), the expected discounted returns $\mathbf{v}$ from each state while following the random walk is given by

$$\begin{aligned}\mathbf{v} &= \mathbf{r} + \gamma\mathbf{P}\mathbf{r} + \gamma^2\mathbf{P}^2\mathbf{r} + \gamma^3\mathbf{P}^3\mathbf{r} + \ldots \\ &= (\mathbf{I} - \gamma\mathbf{P})^{-1}\mathbf{r} \\ &= \mathbf{L}\mathbf{r}.\end{aligned} \qquad 3.17$$

where γ denotes a discount factor. When we consider only a single goal, we can see the elements of $\mathbf{L}$ as $L(s, s') = v(s|s' = g)$, i.e. the value of state s given that s' is the current goal. We will use this property to generate a spatial mapping that allows for rapidly planning a path between any two points in the environment.

Given that $\mathbf{P}$ is formed from a random walk, a spectral analysis of $\mathbf{L}$ [Coifman and Lafon, 2006, Stachenfeld et al., 2014] gives

$$v(s|s' = g) = z(s') \sum_{l=0}^{n} (1 - \gamma\lambda_l)^{-1} \psi_l(s)\psi_l(s') \qquad 3.18$$

where $z(s')$ is the steady–state occupancy of s' given the transition matrix $\mathbf{P}$, ψ_l are the right eigenvectors of $\mathbf{P}$, and $1 = |\lambda_0| \geq |\lambda_1| \geq |\lambda_2| \cdots \geq |\lambda_n|$ are the $n + 1$ eigenvalues [Coifman and Lafon, 2006]. Large–scale features of the environment are represented in the eigenvectors associated with the largest eigenvalues ([Fiedler, 1989], Figure 3.9 top left). Note that the successor representation describes the space by repeated application of a *local* transition function $p(s, s')$, rather than a single application of a large–scale, global similarity function as in the previous geodesic networks.

We now express the position in the 2D space using a set of "successor coordinates", such that

$$\begin{aligned}s(x_1, x_2) \mapsto \breve{\mathbf{s}} &= \left(\sqrt{\left(1 - \gamma\lambda_0\right)^{-1}}\psi_0(s), \sqrt{\left(1 - \gamma\lambda_1\right)^{-1}}\psi_1(s), \ldots, \sqrt{\left(1 - \gamma\lambda_q\right)^{-1}}\psi_q(s)\right) \\ &= \left(\xi_0(s), \xi_1(s), \ldots, \xi_q(s)\right)\end{aligned} \qquad 3.19$$

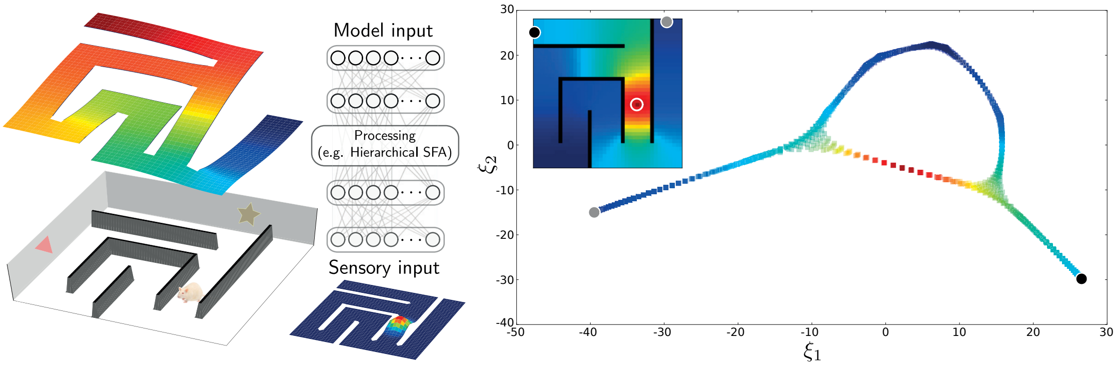

Figure 3.9 – **Representing an environment in successor coordinates**. [**Left**] A rat explores a maze–like environment and passively learns its topology. We assume a process such as hierarchical slow feature analysis, that preliminarily extracts slowly changing functions in the environment (here, the vectors $\xi_1 \ldots \xi_q$). The vector ξ_1 for the maze is shown in the top left. In practice, we extracted the vectors directly from a localized Gaussian transition function (bottom center, for an arbitrary location). **[Right]** This basis can be used to generate a value map approximation over the environment for a given reward (goal) position and discount factor γ (inset). Due to the walls, the function is highly discontinuous in the xy spatial dimensions, but varies smoothly along dimensions where movement is possible. The goal position is circled in white. In the scatter plot, the same array of states and value function are shown on the manifold given by the first two non–trivial successor coordinate dimensions. In this space, the value function is proportional to the scalar product between the states and the goal location. The grey and black dots show corresponding states between the inset and the scatter plot.

where $\xi_l = \sqrt{(1-\gamma\lambda_l)^{-1}}\psi_l$ (see Figure 3.9). This is similar to the "diffusion map" framework by Coifman and Lafon [2006]; with the useful property that, if $q = n$, the value of a given state when considering a given goal is proportional to the scalar product of their respective mappings: $v(s|s' = g) = z(s')\langle \breve{\mathbf{s}}, \breve{\mathbf{s}}' \rangle$. This property allows a network operating in the successor coordinate space to rapidly generate prospective trajectories between arbitrary locations.

The mapping can also be defined using the eigenvectors ϕ_l of a related measure of the space, the normalized graph Laplacian [Mahadevan, 2009]. The eigenvectors ϕ_l serve as the objective functions for slow feature analysis [Sprekeler, 2011], and approximations have been extracted through hierarchical slow feature analysis on visual data [Franzius et al., 2007, Schoenfeld and Wiskott, 2015], where they have been used as an input for generating place cell–like behaviour.

Note that, since the successor coordinates are based on a random walk and not a directed policy resulting from a specific reward landscape, they primarily represent a model–based approach (rather than a hybrid model–free/model–based approach as associated with successor representations in Chapter 2).

3.5.2 Path–finding using the successor coordinate mapping

Successor coordinates provide a means of mapping a set of locations in a 2D environment to a new space based on the topology of the environment. In the new representation, the value landscape is particularly simple. To move from a location $\check{\mathbf{s}}$ towards a goal position $\check{\mathbf{s}}'$, we can consider a constrained gradient ascent procedure on the value landscape, expressed by Equation 3.20:

$$\begin{aligned}\check{\mathbf{s}}(t+\Delta t) &= \underset{\check{\mathbf{s}}\in\check{S}}{\arg\min}\left[(\check{\mathbf{s}}-(\check{\mathbf{s}}(t)+\alpha\nabla v(\check{\mathbf{s}}(t)))^2\right] \\ &= \underset{\check{\mathbf{s}}\in\check{S}}{\arg\min}\left[\left(\check{\mathbf{s}}-(\check{\mathbf{s}}(t)+\tilde{\alpha}\check{\mathbf{s}}')\right)^2\right]\end{aligned} \qquad 3.20$$

where $z(s')$ (see Equation 3.18) has been absorbed into the parameter $\tilde{\alpha}$. At each time step, the state closest to an incremental ascent of the value gradient is selected amongst all states in the environment $\check{S}$. Since the value function is derived under a random walk policy, this corresponds to choosing a state that is more likely to reach the goal in the near future under a random walk. In the following, we will consider how the step $\check{\mathbf{s}}(t)+\tilde{\alpha}\check{\mathbf{s}}'$ can be approximated by a neural attractor network acting in successor coordinate space.

Due to the properties of the transition matrix, ψ_0 is constant across the state space and does not contribute to the value gradient in 3.20. As such, we substituted a free parameter for the coefficient $\sqrt{(1-\gamma\lambda_0)^{-1}}$, which controlled the overall level of activity in the network simulations.

3.5.3 The network model

We use the same network structure described with the NEF in Section 3.3, but here using successor coordinates. Each neuron has an encoding vector given by $\mathbf{e_i} = \frac{\check{\mathbf{s}}_i}{||\check{\mathbf{s}}_i||}$, the normalized successor coordinates of a particular point in space, which corresponds to its place field center. The input to neuron i in the network is then given by

$$\begin{aligned} w_{ik} &= [\mathbf{e}_i]_k, \\ \sum_{k=1}^{m} w_{ik}\check{s}_k^{in} &= \mathbf{e}_i\cdot\check{\mathbf{s}}^{in}. \end{aligned} \qquad 3.21$$

where we assume the input $\check{\mathbf{s}}^{in}$ is given using the basis ξ. As before, we find a set of decoding weights $\mathbf{d}_j$ to recover the least–squares approximation to a set of example effective inputs $\check{\mathbf{s}}^{eff}$ corresponding to locations in the environment, and use them to determine the recurrent weights w_{ij}. With a gain factor α in the feedforward weights and $(1-\epsilon)$ in the recurrent weights, the update equation of the network (following from Equation 3.9) is then given by

$$\tau\frac{d\check{\mathbf{s}}}{dt} \approx -\epsilon\check{\mathbf{s}}+\alpha\check{\mathbf{s}}^{in}. \qquad 3.22$$

Given a location $\check{\mathbf{s}}^{in}$ as an initial input, the network representation $\check{\mathbf{s}}$ approximates the input and reinforces it, allowing a persistent bump of activity to form. When $\check{\mathbf{s}}^{in}$ then changes to a new (goal) location, the input and recovered coordinates conflict. By Equation 3.22, the recovered location moves in the direction of the new input, giving us an approximation of the initial gradient ascent step in Equation 3.20 with the addition of a decay controlled by ϵ. However, the attractor dynamics prevent $\check{\mathbf{s}}$ from moving far from the manifold of actual locations in the environment (determined by the points where the error in Equation 3.10 was minimized). As before, the network activity is decoded after a short stimulation period; here, the state on the manifold of actual states (Figure 3.9 right) closest to the new network representation $\check{\mathbf{s}}$ can be interpreted as a state close to the starting position that ascends the value gradient.

As in earlier non–hierarchical experiments, we used large place fields dominated by low–frequency spatial functions (corresponding to $\gamma = 1$). In addition, we truncated the successor coordinate representation to the first q most significant dimensions, where $q < 6$ in the experiments presented here. Finally, we achieved the best results by balancing the decay and input strength in the network ($\epsilon = \alpha$).

3.5.4 Results

We generated successor coordinate–based attractor networks according to the layout of multiple environments containing walls and obstacles, and stimulated them successively with arbitrary starting points and goals. Here, we use $n = 500$ neurons to represent each environment, with place field centers selected randomly throughout the environment. The network activity resembles a bump across a portion of the environment, as in the previous geodesic attractors (Figure 3.10).

For several networks representing different environments, we initialized the activity at points evenly spaced throughout the environment and provided weak feedforward stimulation corresponding to a fixed goal location (Figure 3.11). After a short delay (5τ), we decoded the successor coordinates from the network activity to determine the closest state (Equation 3.20). The shifts in the network representation are shown by the arrows in Figure 3.11. For two networks, we show the effect of different feedforward stimuli representing different goal locations. The movement of the activity profile was similar to the shortest path towards the goal (Figure 3.11, bottom left), including reversals at equidistant points (center bottom of the maze). Irregularities were still present, however, particularly near the edges of the environment and in the immediate vicinity of the goal (where high–frequency components play a larger role in determining the value gradient).

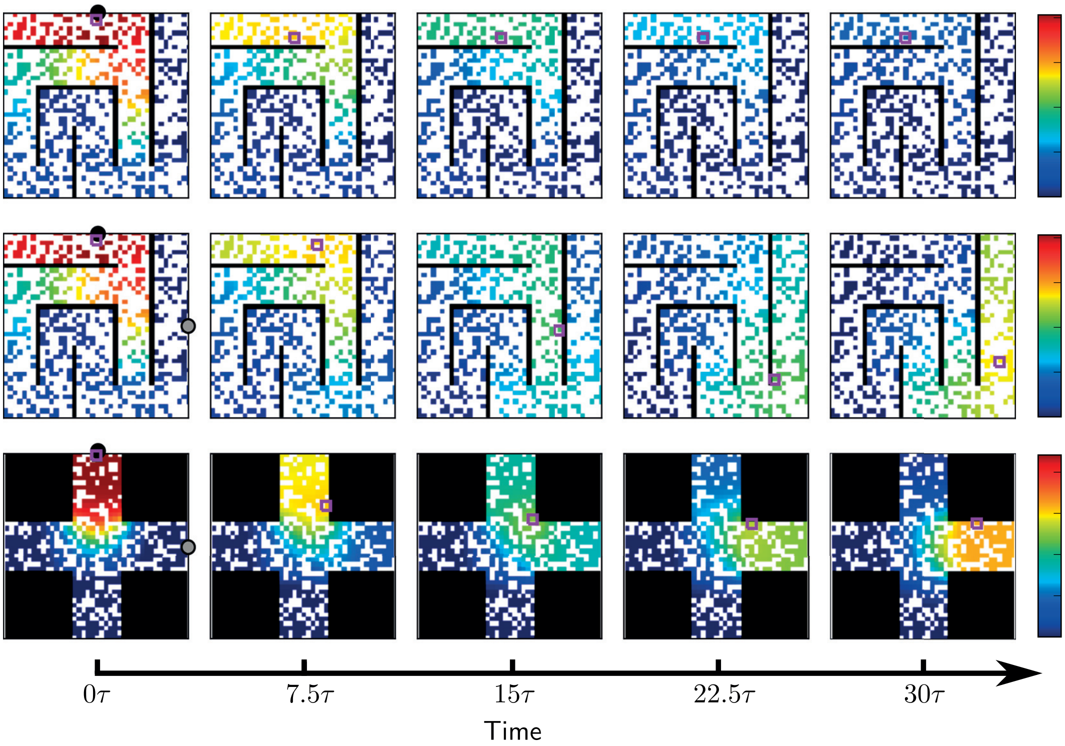

Figure 3.10 – **Successor coordinate–based attractor networks**. Network activities are illustrated over time for different inputs and networks, in multiples of the membrane time constant τ. Purple boxes indicate the most active unit at each point in time. **[Top row]** Activities are shown for a network representing a maze–like environment in a low–dimensional space ($q = 5$). The network was initially stimulated with a bump of activation representing the successor coordinates of the state at the black circle; recurrent connections maintain a similar yet fading profile over time. **[Middle row]** For the same network and initial conditions, a weak constant stimulus was provided representing the successor coordinates at the grey circle; the activities transiently decrease and the center of the profile shifts over time through the environment. **[Bottom row]** Two positions (black and grey circles) were sequentially activated in a network representing a second environment in a low–dimensional space ($q = 4$).

3.6 From trajectories to headings

In the geodesic attractor networks described thus far, we have decoded trajectories according to either the place field center of the most active unit at each point in time (Section 3.4) or by projecting the network representation on to the state manifold at each point in time (Section 3.5). It is worthwhile to consider instead how a downstream network could translate the network activity into a heading for the animal to follow (and ultimately, an action).

Rather than determining a nearby state that ascends the value gradient as in Equation 3.20, a downstream network could decode the gradient in xy space (i.e. the arrows in Figure 3.11) in order to determine an appropriate heading. Several methods for neural differentiation have been proposed [Tripp and Eliasmith, 2010]; for instance, one approach utilizes feedforward

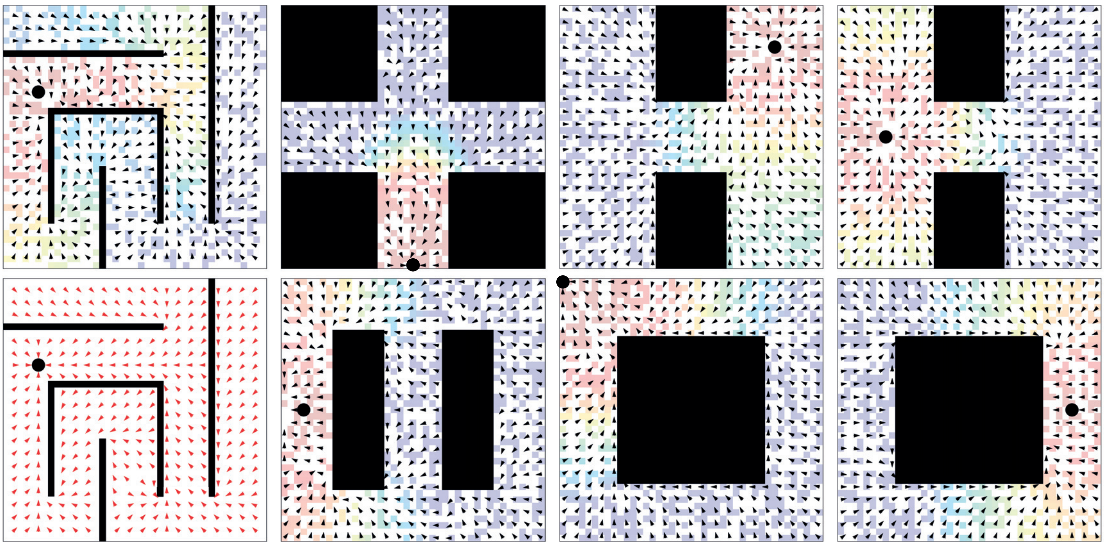

Figure 3.11 – **Successor coordinate–based network trajectories in different environments**. Arrows show the initial change in the location of the activity profile by determining the state closest to the decoded network activity (at $t = 5\tau$) after weakly stimulating with the successor coordinates at the black dot ($\alpha = \epsilon = 0.05$). Pixels show the place field centers of the 500 neurons representing each environment, coloured according to their activity at the stimulated goal site. **[Top left]** Change in location of the represented successor coordinates in a maze–like environment with low–dimensional activity compared to **[Bottom left]** the true shortest path towards the goal at each point in the environment. **[Additional plots]** Various environments and stimulated goal sites using low–dimensional successor coordinate representations.

excitation in combination with delayed, disynaptic inhibition. Suppose that the attractor network projects indirectly to a downstream network, where the activity of each neuron can be described by

$$\tau \frac{da_i^{ds}(t)}{dt} = -a_i^{ds}(t) + g\left[\sum_{j=1}^{N} w_{ij} a_j^{att}(t) + \sum_{j=1}^{N} w'_{ij} a_j^{att}(t-\Delta t)\right], \qquad 3.23$$

where $a_j^{att}(t)$ describes the activity of a neuron j in the attractor network at time t, and the delay $(t-\Delta t)$ is accomplished with a disynaptic delay line. If the disynaptic pathway reverses the sign of the input, and $w_{ij} = \mathbf{e}_i^T \mathbf{d}_j$ where $\mathbf{d}_j$ decodes the successor coordinates, this gives us

$$\tau \frac{da_i^{ds}(t)}{dt} = -a_i^{ds}(t) + g\left[\mathbf{e}_i^T \check{\mathbf{s}}(t) - \mathbf{e}'^T_i \check{\mathbf{s}}(t-\Delta t)\right], \qquad 3.24$$

where $\mathbf{e}_i$ and $-\mathbf{e}'_i$ represent the neuron's response to successor coordinates at time t and time $(t-\Delta t)$, respectively.

Suppose that a neuron in the downstream region is maximally receptive to a particular combination of successor coordinates $\check{\mathbf{s}}_{p_1}(t)$ and $-\check{\mathbf{s}}_{p_2}(t-\Delta t)$ from the attractor network, and that $\check{\mathbf{s}}_{p_1}$ and $\check{\mathbf{s}}_{p_2}$ correspond to neighbouring positions in the environment, offset in some allocentric direction (e.g. $\check{\mathbf{s}}_{p_1}$ is slightly north of $\check{\mathbf{s}}_{p_2}$). If $a_i^{ds}(t)$ is strongly active near the beginning of a trajectory, it suggests that the attractor network representation is also moving north, and that the animal should travel north in order to move closer to the goal position. An entire population receptive to different positions and offsets would therefore give rise to a population code for planned allocentric heading, which could be compared to the animal's current heading by a further downstream network in order to determine an egocentric action (e.g. turn left, turn right or go forward).

This model would require a network of "place × heading" cells downstream of the attractor network in order to decode the heading; in fact, neurons with a place × heading response have been found in the subiculum, the output pathway of the hippocampus [Cacucci et al., 2004]. It is unclear how the cells respond during SWRs or whether they predict the animal's future heading (as opposed to merely reflecting the current heading). However, the cells are strongly predisposed to fire at a particular phase of the local theta rhythm; assuming that this aligns with the end of a theta cycle in CA3/CA1, the subicular cells may summarize the place cell trajectories that occur during theta cycles, which predict the animal's future heading and reflect current goals [Huxter et al., 2008, Wikenheiser and Redish, 2015].

3.7 Discussion

We have presented a spatial bump attractor model generalized to represent environments with arbitrary obstacles, and shown how, with large activity profiles relative to the size of the environment, the network dynamics can be used for long–distance path–finding. This provides a possible explanation for goal–directed activity observed in the hippocampus [Pfeiffer and Foster, 2013, Wikenheiser and Redish, 2015] and an hypothesis for the role that the hippocampus and the CA3 region play in rapid goal–directed navigation [Nakazawa et al., 2003, Nakashiba et al., 2008, Bast et al., 2009], as a complement to an additional (e.g. model–free) system enabling incremental goal learning in unfamiliar environments [Nakazawa et al., 2003].

Recent theoretical work has linked the bump–like firing behaviour of place cells to an encoding of the environment based on its natural topology, including obstacles [Gustafson and Daw, 2011], and specifically to the successor representation [Stachenfeld et al., 2014]. As well, several models have proposed that place cell behaviour can be learned by processing visual data using hierarchical slow feature analysis [Franzius et al., 2007, Schoenfeld and Wiskott, 2015], a process which can extract the lowest frequency eigenvectors of the graph Laplacian generated by the environment [Sprekeler, 2011] and therefore provide an appropriate basis for successor representation–based activity. We provide the first link between these theoretical analyses and attractor–based models of CA3.

Slow feature analysis has been proposed as a natural outcome of a plasticity rule based on Spike–Timing Dependent Plasticity (STDP) [Sprekeler et al., 2007], albeit on the timescale of a standard postsynaptic potential rather than the behavioural timescale we consider here.

However, STDP can be extended to behavioural timescales when combined with sustained firing and slowly decaying potentials [Drew and Abbott, 2006] of the type observed on the single–neuron level in the input pathway to CA3 [Larimer and Strowbridge, 2010], or as a result of network effects. Within the attractor network, learning could potentially be addressed by a rule that trains recurrent synapses to reproduce feedforward inputs (representing positions) during exploration (e.g. [Urbanczik and Senn, 2014]). Both the development of geodesic place fields and learning of recurrent weights are considered in greater detail in Chapter 4.

Our model assigns a key role to neurons with large place fields in generating long–distance goal–directed trajectories. We further propose that such trajectories in dorsal hippocampus (where place fields are much smaller [Kjelstrup et al., 2008]) may be inherited from dynamics in ventral or intermediate hippocampus. The model predicts that ablating the intermediate/ventral hippocampus [Bast et al., 2009] will result in a significant reduction in goal-directed preplay activity in the remaining dorsal region. In an intact hippocampus, the model predicts that long–distance goal–directed preplay in the dorsal hippocampus is preceded by preplay tracing a similar path in intermediate hippocampus.

Recent evidence, since the development of this model, suggests that trajectory events in the dorsal hippocampus move in a “step–like” discontinuous fashion across discrete subpopulations of neurons [Pfeiffer and Foster, 2015]. While this evidence contradicts a model in which trajectories arise from continuous attractor dynamics in dorsal hippocampus, they are still potentially consistent with one in which discrete trajectories in dorsal hippocampus are inherited from continuous trajectories in intermediate/ventral hippocampus. Similar experiments in the intermediate/ventral hippocampus could directly test the hypothesis.

In the model, if an assembly of neurons projecting to the attractor network is active while the animal searches the environment, reward–modulated Hebbian plasticity provides a mechanism for reactivating a goal location. In particular, the presence of a reward–induced neuromodulator would allow for potentiation between the assembly and the attractor network neurons active when the animal receives a reward at a particular location. Activating the assembly would then provide stimulation to the goal location in the network; the same mechanism could allow an arbitrary number of assemblies to become selective for different goal locations in the same environment. Unlike traditional model–free methods of learning which generate a static value map, this would give a highly configurable means of navigating the environment (e.g. visiting different goal locations based on thirst vs. hunger needs), providing a link between spatial navigation and higher cognitive functioning.

4 Learning Place Cell Maps for Navigation

4.1 Introduction

In Chapter 3, we considered how an attractor–based representation of an environment in the hippocampus could contribute to navigation and planning through trajectory events. However, both the feedforward weights and recurrent weights were assumed to be pre–learned, potentially through slow feature analysis. Here, we examine how geodesic place fields of various sizes (i.e. place fields that respect the topology of the environment) could arise from the simple representations of the environment upstream of the hippocampus, and how a local learning rule could result in the recurrent weights implementing an attractor network.

We consider two types of spatial neural responses as input to the place cell network: grid cells (Fyhn et al. [2004], Hafting et al. [2005], briefly introduced in Chapter 1) and border cells [Solstad et al., 2008]. Grid cells, found in the entorhinal cortex (an input pathway to the place cell network of the hippocampus) exhibit spatially periodic firing across an environment, typically resembling a hexagonal grid, at multiple spatial scales along the dorsoventral axis. Border cells, also discovered in the entorhinal cortex, tend to fire at a rate inversely related to the animal's distance to an environmental boundary at a particular orientation. For instance, a cell might respond more strongly as the animal approaches any south–facing wall in its enclosure.

While several theorists have shown that local place cell responses can arise from selectivity to grid cells at multiple spatial scales [McNaughton et al., 2006, Solstad et al., 2006, Rolls et al., 2006], we show how additional selectivity to border cells can result in small–scale place fields that respect the local geometry of the environment (e.g. firing on one side of a wall but not the other, as observed in experiments [Gustafson and Daw, 2011]). We then show how these small-scale place cell responses can, in turn, act as a basis for large–scale place fields like those observed in the ventral hippocampus [Jung et al., 1994, Kjelstrup et al., 2008]. Finally, we show how a local recurrent learning rule could induce attractor weights adapted to the environmental topology.

4.2 Grid and border cell input to the place cell network

Following Solstad et al. [2006], we define a set of grid cell responses according to

$$a_j^{gc}(\mathbf{x}) = g^{gc}\frac{2}{3}\left[\frac{1}{3}\sum_{l=1}^{3}\cos(\mathbf{s}_{lj}^T(\mathbf{x}-\mathbf{r}_{0j}))+\frac{1}{2}\right], \qquad 4.1$$

where $[\mathbf{s}_{1j}, \mathbf{s}_{2j}, \mathbf{s}_{3j}]$ correspond to a set of 2D sinusoidal gratings offset by 0, 60 and 120 degrees from a particular phase angle ϕ, $\mathbf{x}$ corresponds to the animal's 2D position in the environment, and the constants enforce that the grid cell response varies between 0 and g^{gc} across the environment. The grid cell scale is determined by the (equal) wavelength of the three gratings. Each cell is therefore defined by uniformly sampling a grating wavelength from a given range, as well as a a phase ϕ and a 2D offset $\mathbf{r}_{0j}$. The interference pattern of the sinusoidal gratings produces a hexagonal pattern (Figure 4.1A).

Solstad et al. [2006] determine appropriate grid–to–place cell weights algorithmically. Here, we instead follow Sheynikhovich et al. [2009] and "recruit" a place cell i when the animal is at a location $\mathbf{x}$ by setting the feedforward weights according to the normalized presynaptic grid cell activities, i.e.

$$w_{ij} = \frac{a_j^{gc}(\mathbf{x})}{\sqrt{\sum_{m=1}^{\mathcal{N}^{gc}} a_m^{gc}(\mathbf{x})^2}}, \qquad 4.2$$

reflecting the stable solution of a fast self–normalizing competitive Hebbian learning rule, such as Oja's rule [Oja, 1982]. The response of place cell i is then given by

$$a_i^{pc}(\mathbf{x}) = \left[\sum_{j=1}^{\mathcal{N}^{gc}} w_{ij}^{gc} a_j^{gc}(\mathbf{x}) - \theta^{pc}\right]_+, \qquad 4.3$$

where θ^{pc} corresponds to a threshold (or bias), and $[\cdot]_+$ indicates a rectified linear response function. Equation 4.3 can be seen as a discrete time version of the cell model considered in Chapter 3, with a small time constant (and the addition of an explicit threshold θ^{pc}).

Given a sufficiently rich grid cell population, the weight learning rule in Equation 4.2 combined with the interference pattern of the grid cells active at the location that the cell is recruited results in local firing fields centered at the recruitment position (Figure 4.1B left, similar to the results of Sheynikhovich et al. [2009] for a different grid cell model). Throughout this chapter, when we refer to a cell being "recruited", we mean that its feedforward input weights are fixed according to the current normalized presynaptic activity from an upstream population, as in Equation 4.2.

This approach, a combination of the models by Solstad et al. [2006] and Sheynikhovich

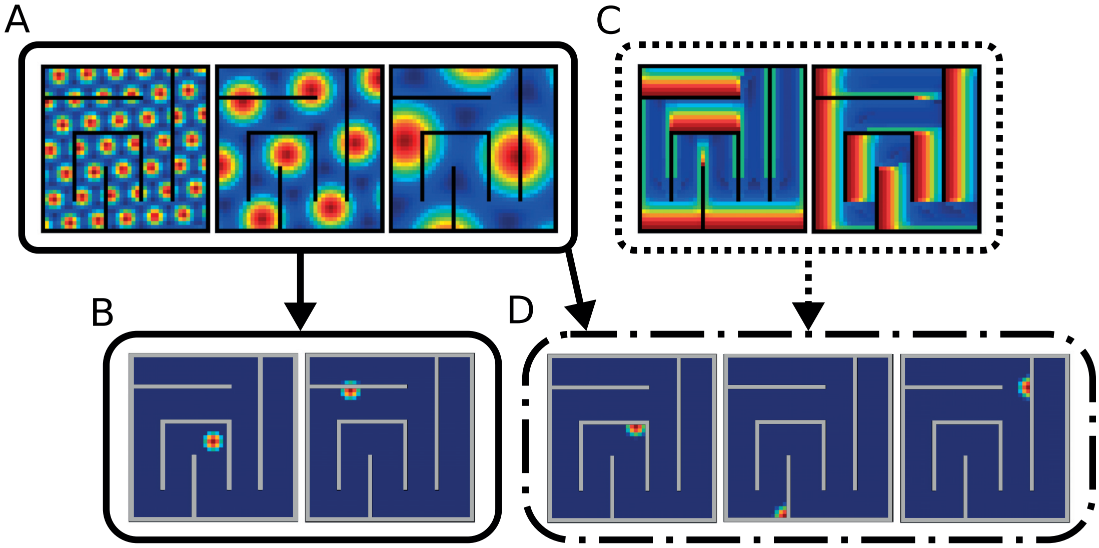

Figure 4.1 – **Local place fields from combined grid and border cell input**. [**A**] Example grid cell responses of differing scale and orientation generated according to Equation 4.1. [**B**] Cells recruited with weights according to Equation 4.2 produce spatially localized place cell–like responses in open field regions of the environment, but can produce non–local responses near boundaries. [**C**] Example border cell responses, generated according to Equation 4.4. [**D**] Combined grid and border input generates spatially localized responses.

et al. [2009], is effective in open–field environments. However, if the grid cell basis does not respect boundaries or obstacles, neither will the resulting place cells (Figure 4.1B right). Evidence suggests that grid cells will form a global pattern unaffected by barriers across an environment [Carpenter et al., 2015], like the pattern resulting from Equation 4.1. In contrast, place cell responses in non–ambiguous environments are generally local and respect boundaries [Skaggs and McNaughton, 1998, Gustafson and Daw, 2011].

We therefore investigate whether entorhinal border cells could allow place cell responses to disambiguate between regions separated by an obstacle. Following Barry et al. [2006], we model border cells according to

$$a_k^{bc}(\mathbf{x}) = g^{bc} \int_{-\pi}^{\pi} \frac{\exp\left[-l_\theta(\mathbf{x})^2/2\sigma_{rad}^2\right]}{\sqrt{2\pi\sigma_{rad}^2}} \frac{\exp\left[-(\theta-\phi_k)^2/2\sigma_{ang}^2\right]}{\sqrt{2\pi\sigma_{ang}^2}} d\theta \tag{4.4}$$

where $l_\theta(\mathbf{x})$ corresponds to the shortest distance to a boundary in direction θ from the current position $\mathbf{x}$ in the environment, ϕ_k determines the border cell's preferred direction vector, and the widths σ_{rad} and σ_{ang} (which we take to be fixed) determine how quickly the cell's response is attenuated with distance from a boundary and the cell's preferred direction, respectively. Equation 4.4 is a simplified version of the "boundary–vector cell" model developed by Barry et al. [2006]. Unlike that model, where the cell's maximal response could occur at an arbitrary distance from the boundary, the cell described by Equation 4.4 always responds maximally at distance 0 from the boundary, and therefore more closely resembles the border cell response found in entorhinal cortex [Solstad et al., 2008].

Combining the grid and border cell populations, we set the feedforward weights of a grid cell j and border cell k to a place cell i at the position of recruitment $\mathbf{x}$ to

$$w_{ij}^{gc} = \frac{a_j^{gc}(\mathbf{x})}{\sqrt{\sum_{m=1}^{\mathcal{N}^{gc}} a_m^{gc}(\mathbf{x})^2 + \sum_{n=1}^{\mathcal{N}^{bc}} a_n^{bc}(\mathbf{x})^2}} \quad \text{and} \quad w_{ik}^{bc} = \frac{a_k^{bc}(\mathbf{x})}{\sqrt{\sum_{m=1}^{\mathcal{N}^{gc}} a_m^{gc}(\mathbf{x})^2 + \sum_{n=1}^{\mathcal{N}^{bc}} a_n^{bc}(\mathbf{x})^2}}, \tag{4.5}$$

with a total of $\mathcal{N}^{gc}$ grid cells and $\mathcal{N}^{bc}$. The place cell's response is then given by

$$a_i^{pc}(\mathbf{x}) = \left[\sum_{j=1}^{\mathcal{N}^{gc}} w_{ij}^{gc} a_j^{gc}(\mathbf{x}) + \sum_{k=1}^{\mathcal{N}^{bc}} w_{ik}^{bc} a_k^{bc}(\mathbf{x}) - \theta^{pc} \right]_+ . \tag{4.6}$$

By scaling the relative strengths g^{gc} and g^{bc} of the grid cells and border cells and the threshold θ^{pc}, the resulting place cells respond on only one side of a thin barrier (Figure 4.1D), despite the place field half–width extending beyond the width of the barrier. This occurs because the border cell population response is effectively anticorrelated across the barrier, resulting in a sharp division between place cell responses across the barrier despite similar grid cell firing patterns.

4.3 Learning large geodesic place fields from entorhinal input

In open–field environments, large place fields can be learned simply by restricting place cell input to low–frequency grid cells [Solstad et al., 2006]. This solution is intuitively appealing, since both grid fields [Hafting et al., 2005] and place fields [Jung et al., 1994, Kjelstrup et al., 2008] increase in size moving along the dorsoventral axis; it is then reasonable that (large–scale) ventral place cells receive input primarily from (large–scale, low spatial frequency) ventral grid cells [Solstad et al., 2006]. We therefore consider whether local fields could develop by combining large–scale grid input with border cell input.

As shown in Figure 4.2A, the combined input often results in multiple firing fields instead of a single place cell response. Cells typically develop multiple fields defined by a “local border cell” response; i.e. responding like border cells constrained to one section of the environment.

Alternatively, we considered how large place fields could develop by successive local clustering along the dorsoventral axis of hippocampus (Figure 4.2B). At the first level (reflecting small–scale, dorsal place fields) cells had their weights fixed at random locations in the environment according to Equation 4.5. We used 1500 grid cells, 1500 border cells and 3000 place cells. We then trained a hierarchical succession of place cell populations, each on the output from the

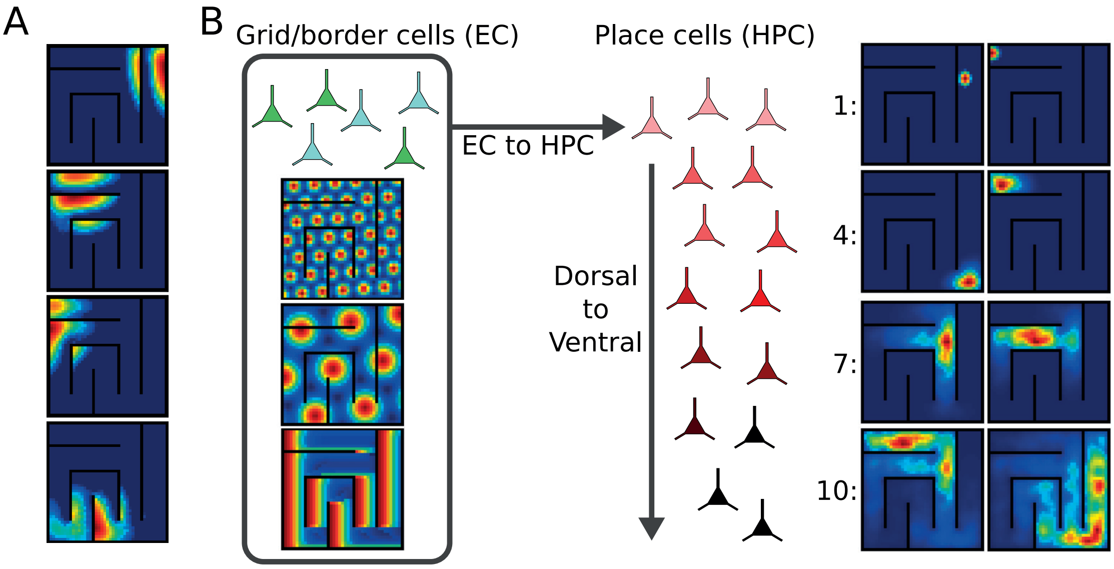

Figure 4.2 – **Large place fields from progressive clustering along the dorsoventral axis**. [**A**] Direct grid and border cell input using only low–frequency grid cells typically results in multiple firing fields. [**B**] Direct grid and border cell input including high–frequency grid cells can generate local firing fields. By iteratively applying the same weight learning rule across multiple populations, each trained on the output from the previous, large geodesic place fields arise. Example place fields are illustrated for population 1 (trained directly on grid and border cell output) and populations 4, 7 and 10.

previous population, i.e.

$$w_{ik}^{ff,n} = \frac{a_k^{pc,n-1}(\mathbf{x})}{\sqrt{\sum_{m=1}^{\mathcal{N}^{n-1}} (a_m^{pc,n-1}(\mathbf{x}))^2}}, \qquad 4.7$$

where $w_{ik}^{ff,n}$ denotes the feedforward weight from neuron k in population $n-1$ to neuron i in population n, and $a_j^{pc,n-1}(\mathbf{x})$ is the response of neuron k to the position $\mathbf{x}$. In total, we trained 10 place cell populations, where population $n = 1$ was trained directly on grid and border cell input and each population $n > 1$ was trained on the output of population $n-1$. This pattern of connectivity is consistent with the observation of dense associational fibers that extend between CA3 cells along the dorsoventral axis of the hippocampus [Amaral and Witter, 1989]. We linearly decreased the cell population size by 200 cells at each level.

We found that the resulting place fields gradually increased in size, reflecting clustering of smaller, local place fields into larger place fields at each level (Figure 4.2B right). While the fields generally maintained locality, there was a gradual spread in the direction of navigable corridors, consistent with intermediate and ventral place fields observed in the hippocampus [Kjelstrup et al., 2008]. Like the place cell responses observed in intermediate and ventral

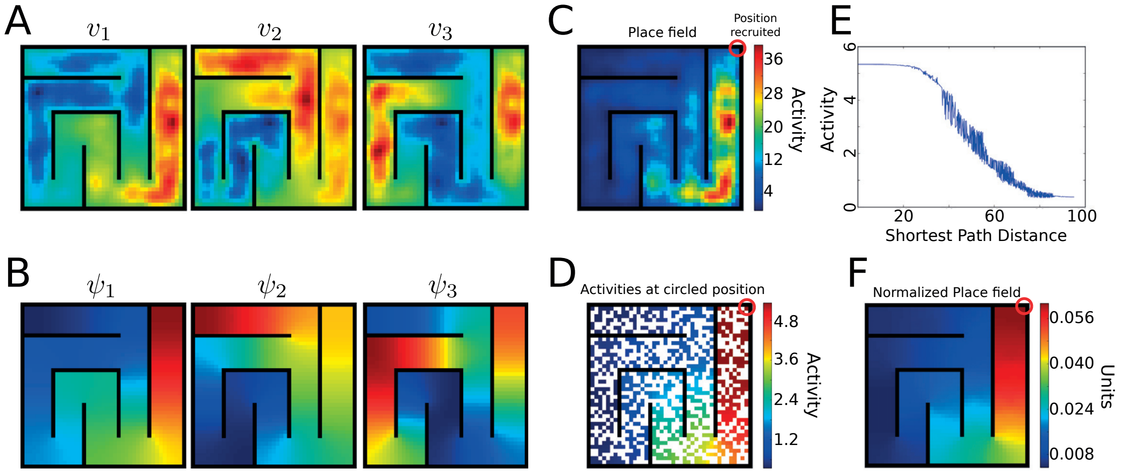

Figure 4.3 – **Learned large place fields capture global environment properties**. [**A**] Singular value decomposition of the population response of large place fields across the environment reveals singular vectors resembling [**B**] the eigenvectors of a random–walk transition matrix. [**C**] Place fields often display peaks far from the recruited position, particularly if the cell is recruited in a corner/edge of the environment. [**D**] Conversely, the activity profile of all cells in the population plotted according to each cell's recruited position is generally smooth, with cells recruited close to the current position in the environment having the strongest response. [**E**] Plotting the same activity as in D as a function of the shortest path distance between each cell's recruited position and the circled position reveals a slow and nearly monotonic relationship. [**F**] After normalizing across the population activity at each point in the environment, the large place fields were smooth and peaked at the recruitment position.

hippocampus, these fields also had reduced spatial coherence (sometimes resulting in multiple field peaks, as in population 10 in Figure 4.3B).

We analyzed the population activity of the large place field units using singular value decomposition, and found that the dominant singular vectors bore a close resemblance to the dominant eigenvectors of the random–walk transition matrix (Figure 4.3A and B, using the local transition function considered in Section 3.5), suggesting that the fields capture important global information about the environment. Population activity was unevenly distributed by position, with cells generally responding more strongly far from the edges of the maze. For instance, in Figure 4.3C, a cell that was recruited in the corner of the maze has a field peak near the center of the hallway. However, the population activity profiles at a given position were smooth (Figure 4.3D and E, plotted for the same cell's recruited position). Place fields were smooth and peaked near the cell's recruited position after normalizing the population activity at each position (Figure 4.3F). The shifted fields reflect both the randomly recruited positions of the cells in the previous layers, and asymmetry effects when a new cell is recruited close to an environment boundary; since there are no cells in the previous layer active on the opposite side of the boundary, the new place field tends to shift away from the boundary towards the interior of the environment. The dependence of the activity rates on the position input could be mitigated by online normalization of the total network activity at each level, using e.g. a pool of recurrently connected inhibitory neurons.

4.4 Learning the attractor map in recurrent weights

Finally, we examine whether a local learning rule could result in recurrent weights that implement an arbitrary attractor network in the place cell population. After simplifying the system to discrete time dynamics, a learning rule for the recurrent weights arises naturally from segregating feed–forward and recurrent input to the place cell population.

Until now, we have considered activities arising from a single feedforward pass through the populations for a given fixed combination of grid and border cell activities. Here, in order to consider the impact of recurrent weights, we introduce the argument t to denote the discrete time step. We use “feedforward” to refer to long–distance connections from more dorsally–located CA3 cells with smaller place fields (i.e. population $n-1$), and “recurrent” to refer to connections between cells within the same region along the dorsoventral axis (i.e. population n). We consider the case where feedforward weights have already been learned (e.g. via the process considered in the last section).

We denote the feedforward postsynaptic activation as $v_i^{ff,n}(\mathbf{x},t) = \sum_k w_{ik}^{ff,n} a_k^{pc,n-1}(\mathbf{x},t)$ for the input position $\mathbf{x}$, i.e. the weighted place cell activities in population $n-1$ at the current time step. After introducing the recurrent weights, we denote the recurrent activation $v_i^{rec,n}(\mathbf{x},t) = \sum_j w_{ij}^{rec} a_j^{pc,n}(\mathbf{x},t)$, i.e. the weighted place cell activities in population n at the current time step. The total activation is then $v_i^n(\mathbf{x},t) = v_i^{ff,n}(\mathbf{x},t) + v_i^{rec,n}(\mathbf{x},t)$.

The place cell population behaves like an integrator if, for some total activation $v_i^n(\mathbf{x},t-1)$ to each cell i, it is approximately matched by the resulting recurrent activation $v_i^{rec,n}(\mathbf{x},t)$ on the next time step, i.e.

$$\begin{aligned}
a_i^{pc,n}(\mathbf{x},t+1) &= \left[\sum_{k=1}^{\mathcal{N}^{n-1}} w_{ik}^{ff,n} a_k^{pc,n-1}(\mathbf{x},t) + \sum_{j=1}^{\mathcal{N}^{n}} w_{ij}^{rec,n} a_j^{pc,n}(\mathbf{x},t) - \theta^{pc}\right]_+ \\
&= \left[v_i^{ff,n}(\mathbf{x},t) + v_i^{rec,n}(\mathbf{x},t) - \theta^{pc}\right]_+ \\
&\approx \left[v_i^{ff,n}(\mathbf{x},t) + v_i^{n}(\mathbf{x},t-1) - \theta^{pc}\right]_+ .
\end{aligned} \tag{4.8}$$

In this case, the population activity will remain approximately constant after the input is removed. The set of all inputs $\mathbf{x}$ for which $v_i^{rec,n}(\mathbf{x},t) \approx v_i^n(\mathbf{x},t-1)$ defines the attractor manifold. The formulation here differs slightly from that in Chapter 3 in that we do not include the weights $(1-\alpha)$ and α that result in a low–pass filter of the input rather than an integrator,

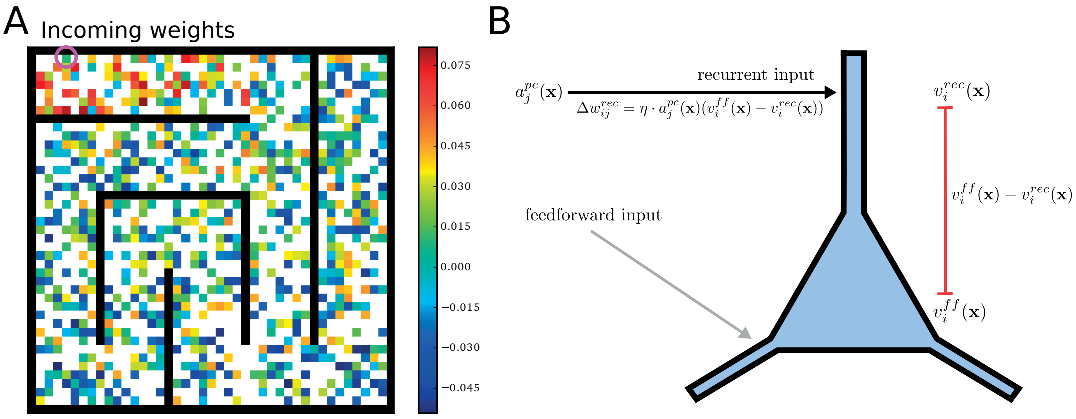

Figure 4.4 – **Learning attractor dynamics in recurrent networks**. [**A**] We used the discrete–time, step–based rule (Equation 4.10) to learn recurrent weights based on position inputs resulting from a random walk in the environment. The learned incoming weights for one unit are shown, plotted according to the recruited position of the presynaptic neuron. If a place cell was never recruited at a given position, it is plotted in white. [**B**] Attractor network weights could be learned using the difference in steady–state postsynaptic activation between compartments dominated by feedforward and recurrent input, with activity driven by the feedforward compartment during the learning stage.

although the resulting attractor network activity is similar.

We assume that the feedforward inputs to the place cell network (that the network receives during exploration of the environment) correspond to samples from the attractor manifold. If we restrict the time course to $t = [0, 1]$ and start the network in a quiescent state ($a_i^{pc,n} = 0 \;\forall\; i \in \mathcal{N}^n$), then $v_i^n(\mathbf{x}, 0) = v_i^{ff,n}(\mathbf{x}, 0)$. In this case, the error function for an integrator is then given by

$$E = \frac{1}{2}\int_t \sum_i \left(v_i^{ff,n}(\mathbf{x}, 0) - v_i^{rec,n}(\mathbf{x}, 1)\right)^2 dt, \tag{4.9}$$

which is a restatement of the error function considered in Chapter 3 (Equation 3.10) in activation space, with the assumption that each feedforward input defines a target for the attractor manifold. Taking the partial derivative with respect to a recurrent weight gives

$$\begin{aligned} \frac{\partial E}{\partial w_{ij}^{rec}} &= -a_j^{pc,n}(\mathbf{x}, 1)(v_i^{ff,n}(\mathbf{x}, 0) - v_i^{rec,n}(\mathbf{x}, 1)) \\ \Delta w_{ij}^{rec,n} &= \eta \cdot a_j^{pc,n}(\mathbf{x}, 1)(v_i^{ff,n}(\mathbf{x}, 0) - v_i^{rec,n}(\mathbf{x}, 1)). \end{aligned} \tag{4.10}$$

We used Equation 4.10 in discrete time to learn recurrent weights for the large place field network, population $n = 10$, based on input from the grid/border cell population filtered through the first 9 place cell populations. We trained the network based on a 50 000 step random walk in the environment with the learning rate annealed over time. For each position, starting from no activity in the population, we first determined (a) the feedforward activations $v_i^{ff,n}(\mathbf{x},0)$, then (b) the resulting activities $a_j^{pc,n}(\mathbf{x},1)$, and finally (c) the new recurrent activations $v_i^{rec,n}(\mathbf{x},1)$. These three terms are sufficient to calculate one iteration of the weight update rule. The resulting incoming weights for one cell are shown in Figure 4.4A.

The learning rule in Equation 4.10 requires the neuron to maintain separate estimates of the feedforward and recurrent activations in order to calculate the weight update. Biologically, one possibility is that the activations correspond to the local potential in two different regions or compartments of the postsynaptic cell: one dominated by local (recurrent) CA3 input and the other by long–distance (feedforward) CA3 input from a more dorsal region. Assuming that the activity of cells in the attractor network is dominated by the feedforward compartment during the learning phase (i.e. the recurrent activation is not integrated into the total activation), then $v_i^n(\mathbf{x},t) = v_i^{ff,n}(\mathbf{x},t)$. In addition, at steady–state, $v_i^{ff,n}(\mathbf{x},t) \approx v_i^{ff,n}(\mathbf{x},t-1)$. The weight change in the recurrent synapses at steady–state would then be proportional to the presynaptic activity and postsynaptic difference in potential between the compartments (Figure 4.4B). Notably, a similar rule has been developed for spiking neurons by Urbanczik and Senn [2014].

After learning the weights, we relaxed the dynamics of the network to continuous time to evaluate the attractor dynamics; i.e. taking

$$\tau \frac{da_i^{pc,n}(\mathbf{x})}{dt} = -a_i^{pc,n}(\mathbf{x}) + \left[\sum_{k=1}^{\mathcal{N}^{n-1}} w_{ik}^{ff,n} a_k^{pc,n-1}(\mathbf{x}) + \sum_{j=1}^{\mathcal{N}^{n}} w_{ij}^{rec,n} a_j^{pc,n}(\mathbf{x}) - \theta^{pc} \right]_+ . \qquad 4.11$$

We observed the evolution of the network activity after supplying brief input corresponding to a location. The network displayed bump attractor dynamics, with bumps of activity remaining stable after the input was removed for much longer than the time constant τ (Figure 4.5A; note that we did not use a decay term α when learning the recurrent weights, so the magnitude of the population activity does not noticeably decrease over time). Similarly, the bump moved along a path between locations when the network was consecutively stimulated with two different locations in the environment (Figure 4.5B). These trajectories were decoded into headings according to the recruited position of the most active neuron before and after stimulation (Figure 4.5C). We found that the decoded heading accuracy was best for long–distance trajectories, and generally worse in the near vicinity of the goal, particularly for goals near the edges of the environment; this reflects both the low precision of the large–scale bumps and the uneven network response near environment boundaries (shown in Figure 4.3).

4.5 Discussion

In this chapter, we considered how the geodesic place fields described in Chapter 3 could be learned from the types of spatial representations found in entorhinal cortex. In addition, we proposed a learning rule for the recurrent weights required to implement attractor network dynamics in arbitrary environments.

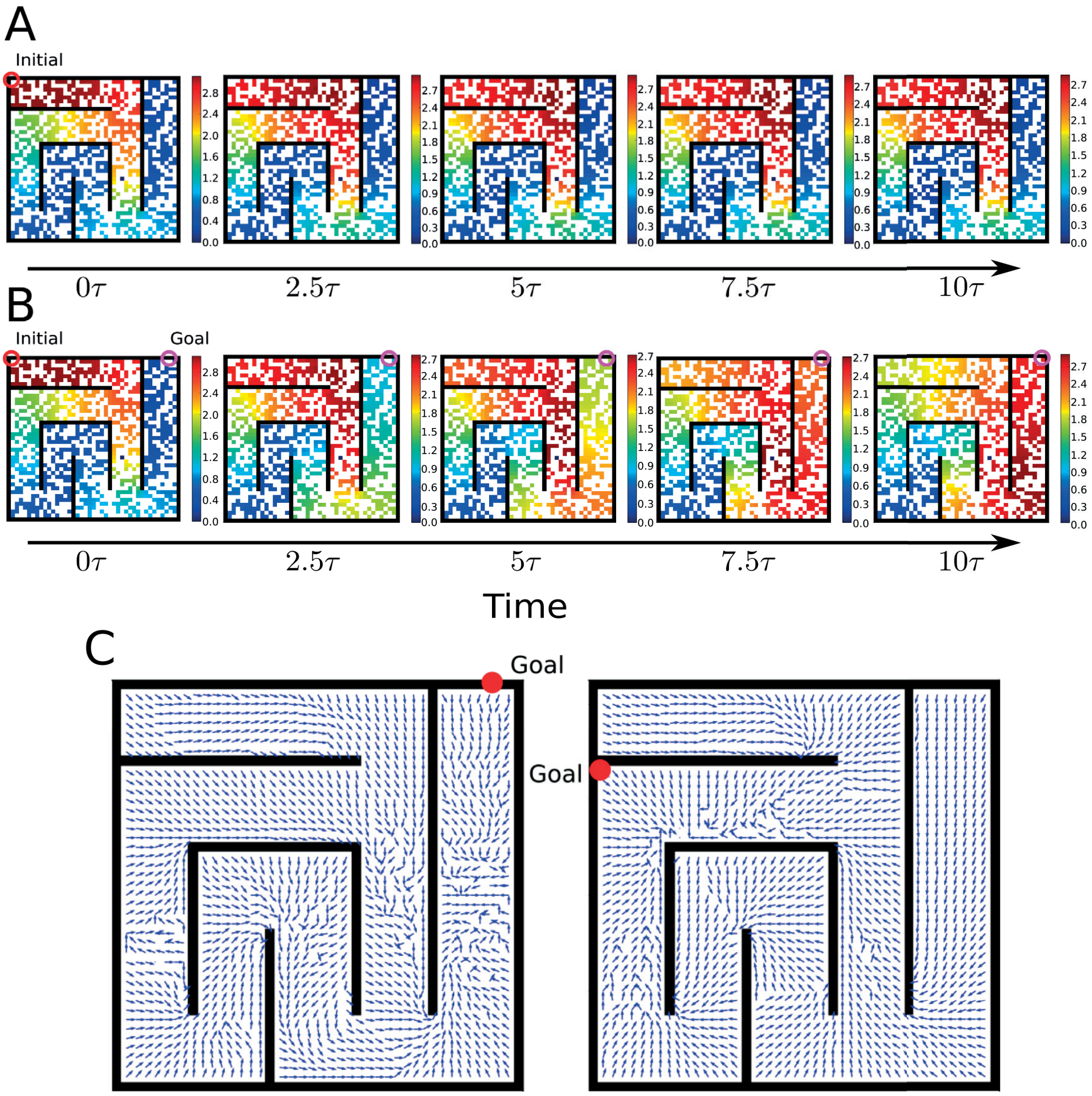

Figure 4.5 – **Learned large–scale bump attractor dynamics**. [**A**] The network was initialized with the steady–state activities $a_j^{pc,n}(\mathbf{x}, 1)$ resulting from a feedforward activation at a particular spatial position (marked “Initial”). The feedforward input was then removed and the network activity evolved according to Equation 4.11. The learned weights resulted in a sustained bump–like activity profile after the stimulus was removed, with some drift. [**B**] After the same initialization, the activity evolved under a feedforward activation corresponding to the position marked “Goal”. [**C**] Headings decoded from initial bump movement after stimulating the network with two locations, according to the change in the recruited position of the most active neuron before and after stimulating the network with the goal position.

Unlike several other models examining the response of spatial cells near barriers [Gustafson and Daw, 2011, Stachenfeld et al., 2017], we assume that grid cells are invariant to boundaries and obstacles. In our grid–to–place cell model, place cells become responsive to boundaries under the influence of border cells. Grid cell barrier invariance is supported by evidence suggesting that grid cells form a globally coherent response as rats discover that different areas on each side of a boundary are in the same environment [Carpenter et al., 2015], although different interpretations of the same data have been proposed [Stachenfeld et al., 2017]. Our model does not necessarily require that grid input is boundary invariant; it only proposes that boundary sensitivity of grid cells is unnecessary when combined with border cell input.

Developmental evidence in rat pups suggests that strong periodicity in grid cell firing arises 1–2 weeks later than stable place cell responses [Langston et al., 2010, Wills et al., 2010], calling into question the necessity of grid cells in the formation of place cell responses. Border cell responses, however, are present from the same time as place cells [Bjerknes et al., 2014]. The combination of entorhinal border cell input with weakly periodic (yet multi–peaked [Langston et al., 2010, Wills et al., 2010]) proto–grid cell input may therefore form a rich enough basis for early place fields, especially given the dense input provided from grid and border cells to hippocampal neurons [Moser et al., 2015].

In our model, large place fields are learned by successive clustering of smaller, local place fields. This dorsal–to–ventral model of place field development contrasts with the ventral–to–dorsal model of preplay activity considered in Section 3.4. While the clustering approach shows how local fields could develop despite non–local grid input, it seems unlikely that ventral place cells are always activated via a multisynaptic cascade along the dorsoventral axis. One possibility is that, after the fields and recurrent dynamics have been shaped, the cells could be directly reactivated by entorhinal or dentate gyrus input, in combination with the soft winner–take–all dynamics of the attractor network.

Finally, we considered how the weights required to implement an attractor network might arise from a two–compartment neuron model. As discussed, the learning rule is qualitatively similar to that described by Urbanczik and Senn [2014], although here it was derived to minimize the error in Equation 3.10. Here, the learning rule was carried out in discrete time, although attractor dynamics were illustrated in continuous time; a continuous time implementation of the learning rule remains for future work. The rule relies on separate estimates of long distance vs. local input to a neuron. CA3–to–CA3 synapses can occur in both the apical and basal dendritic compartments of a CA3 neuron, although a distance–based segregation has not yet been reported [Andersen et al., 2007].

While a learning rule has been proposed before to minimize the least–squares error in Equation 3.10 [MacNeil and Eliasmith, 2011], it uses a (potentially high–dimensional) external error signal to tune the recurrent weights. Conversely, the learning rule here requires only local, scalar information, under the assumption that the feedforward input itself defines a target for the recurrent dynamics.

5 Deep Reinforcement Learning with Offline Episodic Control

5.1 Introduction

In Chapter 1, we considered the problem of learning in an environment without fixed reward locations. To maximize reward efficiently, we discussed how the rat (or more generally, the agent) can leverage a model–based approach by learning a "map" of the environment that can be reused to solve multiple tasks (i.e. reward landscapes). In this chapter we describe, from a machine learning perspective, an approach for learning and utilizing such an environment model to quickly adapt to non–stationary rewards.

When the task is rapidly changing, the agent needs to adjust quickly and build a new policy with very little data; otherwise, it will never learn to exploit the present reward landscape before it changes again. In contrast, given a long–running task with stationary rewards, the learning period will be negligible compared to the total rewards that the agent can expect to receive after reaching asymptotic performance. We can therefore see the model–based approach as one potential way to solve the basic issue of *sample efficiency* faced by the agent: how to best leverage the little experience the agent has in the new reward landscape.

Model–based methods improve sample efficiency by allowing the agent to reuse the experience that does not change between reward landscapes: the structure of the environment. However, it is still critical that a new policy can be learned and exploited quickly. This can be a particular issue if the policy is built on a deep neural network with weights trained by gradient descent; this approach can achieve impressive results but tends to be extremely data inefficient, requiring tens of millions of observations (e.g. Mnih et al. [2015, 2016]). In contrast, Model–Free Episodic Control (MFEC) [Blundell et al., 2016] stores returns to a lookup table and performs no gradient descent on the value function. It achieves significantly higher performance than deep model–free algorithms in the initial stages of learning on complex deterministic tasks like Atari games.

Here, we combine episodic control and model–based approaches to build an agent that can adapt rapidly to changing rewards. We use a Variational Autoencoder (VAE) [Kingma and

Welling, 2013, Rezende et al., 2014] to map observations (corresponding to states) to a normal distribution in latent space. In addition to learning to reconstruct the input observations, the model learns to approximate the transition function between states in the latent space.

Conceptually, our algorithm is inspired by Dyna–Q [Sutton, 1990]. Q–values in the environment are updated based on both real (online) and simulated (offline) experiences, which are generated according to the learned transition model. Simulated offline experiences are generated by sampling initial states from the Normal prior over the latent space, and producing an on–policy rollout through the state space according to the learned transition function (choosing actions according to the approximated Q–values). Unlike existing approaches, the returns from these simulated experiences are stored in a lookup table memory alongside the returns from real episodes, allowing the simulated episodes to rapidly update the policy. The rewards, terminal states, and Q–values are estimated using a k–Nearest–Neighbour (kNN) classifier over real and simulated experiences, such that all three are learned without adjusting the network weights. By storing simulated experiences in a lookup memory (a non–parametric approach) rather than using them to train a deep network, the network can leverage them to update the policy faster. In addition, the kNN memory naturally limits the impact of model error by returning exact matches from real–world experience when possible.

5.2 Background

MFEC [Blundell et al., 2016] can be seen as a variant of the more general class of episodic controllers introduced in Chapter 2. Following an episode lasting T steps and ending in a terminal state s_T, the values of state–action pairs $(s_0, a_1), (s_1, a_2), \dots, (s_{T-1}, a_T)$ visited during the episode are updated according to

$$Q(s_t, a_{t+1}) \leftarrow \begin{cases} R_{t+1} & \text{if } (s_t, a_{t+1}) \notin \mathcal{T} \\ \max\big(Q(s_t, a_{t+1}), R_{t+1}\big) & \text{otherwise,} \end{cases} \tag{5.1}$$

where $R_{t+1} = r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{T-t-1} r_T$ corresponds to the future discounted returns received from step t, and $\mathcal{T}$ is the set of all state–action pairs in the agent's lookup table memory. The use of the max operator for estimating the future returns only works in deterministic environments; otherwise, the Q–value update can be significantly biased by observing a high value but low probability stochastic return. During inference, the agent estimates the Q–values of new (s, a) pairs using a kNN average over the Q–values of existing states in memory where the action a has been taken.

The low–dimensional s_t is determined from the high–dimensional raw observations o_t in one of two ways: by using a fixed random projection to the low–dimensional space, or by learning a low–dimensional embedding with a VAE [Kingma and Welling, 2013, Rezende et al., 2014]. Our model builds on the second approach.

A VAE is an unsupervised deep learning algorithm based on variational Bayes, that maps input observations to a (typically Normal) distribution in N–dimensional latent space. The objective function is given by

$$\begin{aligned}\mathcal{L}(\theta,\phi;o_t) &= \log p(o_t) - \mathcal{D}_{KL}(q_\phi(s_t|o_t)||p_\theta(s_t|o_t)) \\ &= \mathbb{E}_{q_\phi(s_t|o_t)}[\log p_\theta(o_t|s_t)] - \mathcal{D}_{KL}(q_\phi(s_t|o_t)||\mathcal{N}(0,\mathrm{I}))\end{aligned} \qquad 5.2$$

and corresponds to the evidence lower bound (or ELBO) of the observations, using a learned generative distribution (or "decoder") p and an auxiliary posterior distribution (or "encoder") q, assuming a fixed prior $p(s) = \mathcal{N}(0,\mathrm{I})$. The encoder is parameterized by ϕ and the decoder by θ. The gradients are determined using the second line of Equation 5.2; however, we can interpret the training procedure using the first line as finding a model that maximizes the lower bound of the evidence $\log p(o_t)$ (since the KL–divergence $\mathcal{D}_{KL}(q_\phi(s_t|o_t)||p_\theta(s_t|o_t))$ is non–negative).

Training proceeds by using the encoder to map each observation o_t to two N–dimensional vectors s_t^μ and s_t^σ. Using the reparameterization trick [Kingma and Welling, 2013], a new vector $s_t \sim q_\phi(s|o_t)$ is then sampled, where $q_\phi(s|o_t) = \mathcal{N}(s_t^\mu, \mathrm{diag}((s_t^\sigma)^2))$ (and diag indicates a diagonal matrix formed from the entries in $(s_t^\sigma)^2$). The vector s_t is then used by the decoder to perform gradient descent on the reconstruction cost $\log p_\theta(o_t|s_t)$. The second term in Equation 5.2 pressures the posterior distribution $q_\phi(s_t|o_t) = \mathcal{N}(s_t^\mu, \mathrm{diag}((s_t^\sigma)^2))$ generated by the encoder to match the prior distribution $p(s) = \mathcal{N}(0,\mathrm{I})$. Note that the learned encoder output s_t^σ introduces stochasticity into sampling s_t, which causes the VAE to map observations from the environment to some continuous region around the mean s_t^μ (such that all training observations taken together can be approximately mapped to a continuous prior $\mathcal{N}(0,\mathrm{I})$).

Typically, a VAE is used to generate new example observations by sampling from the prior $s \sim \mathcal{N}(0,\mathrm{I})$ after training and evaluating $\mathcal{E}_\theta[o|s]$. In MFEC, the posterior model $q_\phi(s_t|o_t)$ can instead be used to map the observations to a structured latent space. In particular, when the observations correspond to a 2D image, q_ϕ is parameterized using a deep Convolutional Neural Network (CNN) [Krizhevsky et al., 2012], and p_θ is parameterized by a deep deConvolutional Neural Network (DCNN) [Goodfellow et al., 2014], the latent space can incorporate features like translation invariance that are not well–represented by random projections.

In MFEC, the latent embedding stored to memory is taken as the concatenation of s_t^μ and $\log s_t^\sigma$; in our model, we use only the mean vector s_t^μ.

5.3 The model and training procedure

Our model consists of a VAE (composed of an an encoder and a decoder), an auxiliary transition learning network, and a kNN lookup table, illustrated in Figure 5.1. During an initial learning phase, the network is trained to both generate the state observations and to predict the next state under a state–action pair, using the transition network. After the initial learning phase, the agent explores the environment while updating Q–value estimates based on real

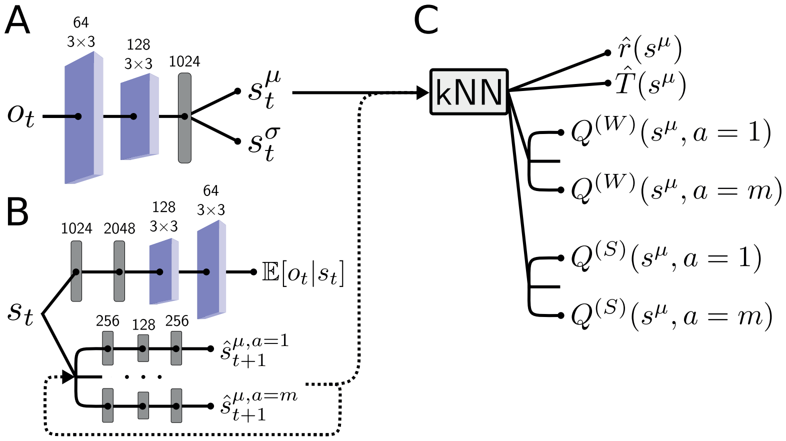

Figure 5.1 – **Offline episodic control network.** The network consists of [**A**] an encoder network, [**B**] a decoder and transition network, and [**C**] a kNN lookup table, which can be accessed in both online mode (from the encoder, storing experiences to $Q^{(W)}$) and offline mode (from the decoder, storing experiences to $Q^{(S)}$). Blue boxes denote convolutional/deconvolutional layers and grey boxes denote fully connected layers. The next state estimate $\hat{s}_{t+1}^{\mu,a}$ is the expectation $\mathbb{E}[s_{t+1}^{\mu,a}|a, s_t]$.

and simulated episodes.

5.3.1 Learning environment structure with a VAE

We use an initial learning phase to allow the agent to develop a model of the environment, ignoring any rewards. In this phase, the agent explores according to a random walk, observing triplets of (o_t, a_{t+1}, o_{t+1}) that are added to a replay memory. Triplets are presented to the network in minibatches. Here, we train the network to optimize the evidence lower bound $\mathcal{L}(\theta,\phi;o_t)$ of a state observation o_t along with an additional loss term $L_t(\phi,\psi;o_t,o_{t+1},a_{t+1})$, according to

$$\begin{aligned} L_t(\theta,\phi,\psi) &= \mathcal{L}(\theta,\phi;o_t) + L_t(\phi,\psi;o_t,o_{t+1},a_{t+1}) \\ &= \mathbb{E}_{q_\phi(s|o_t)}[\log p_\theta(o_t|s)] - \mathcal{D}_{KL}(q_\phi(s|o_t)||p(s)) \\ &\quad + \mathbb{E}_{q_\phi(s|o_t)}[\log p_\psi(s_{t+1}^\mu(o_{t+1})|s,a_{t+1})] \end{aligned} \tag{5.3}$$

where ψ represents the parameters of the transition network.

The first two terms in Eq. 5.3 correspond to the VAE reconstruction and latent/prior losses in Equation 5.2. Using the third loss term, we also train the VAE to generate an estimate of s_{t+1}^μ, the latent space embedding of the next state observation o_{t+1}, conditioned on the action a_{t+1}. Conditioning on the action is achieved by evaluating a specific branch of the transition network, where the number of branches is equal to the number of possible actions (Figure 5.1).

Algorithm 2 Latent Learning with a Random Walk.

```
Initialize replay memory $\mathcal{M}$
Observe $o_0$ and store in $\mathcal{M}$
for $t = 0, \ldots, T$ do
    Select a random action $a_{t+1}$
    Observe $o_{t+1}$
    Store transition $(a_{t+1}, o_{t+1})$ in $\mathcal{M}$.
    if $t$ mod $I_{update} == 0$ then
        Sample a random minibatch of transitions $(o_i, a_{i+1}, o_{i+1})$ from $\mathcal{M}$
        for Sample in minibatch do
            Process $s_i^\mu$, $s_i^\sigma$, $s_{i+1}^\mu$ using the encoder on $o_i$ and $o_{i+1}$
            Sample $s_i \sim \mathcal{N}(s_i^\mu, \mathrm{diag}((s_i^\sigma)^2))$
            Process $\mathbb{E}[o_i|s_i]$ using the decoder on $s_i$
            Process $\mathbb{E}[s_{i+1}^{\mu,a_{i+1}}]$ using the transition network on $s_i$ and $a_{i+1}$
            Perform a gradient descent step on $L_i(\theta, \phi, \psi)$
        end for
    end if
end for
```

5.3.2 Learning value functions

After the initial latent learning phase, the agent explores the environment observing tuples of $(o_t,\ a_{t+1},\ r_{t+1},\ T_{t+1})$, where r_{t+1} denotes the reward received at step $t+1$ and $T_{t+1} = \{0,1\}$ indicates whether the state reached at step $t+1$ was found to be terminating (1) or non–terminating (0). During this phase, the agent updates the estimates $r(s_t^\mu)$ and $T(s_t^\mu)$ in the lookup table according to

$$\begin{aligned} r(s_t^\mu) &\leftarrow r(s_t^\mu) + \eta_r (r_t - r(s_t^\mu)) \\ T(s_t^\mu) &\leftarrow T(s_t^\mu) + \eta_T (T_t - T(s_t^\mu))\ , \end{aligned} \tag{5.4}$$

where η_r and η_T are learning rates. Note that we assume that the status of a state as terminal or non–terminal can change, just like the reward function.

At certain step intervals during exploration, the agent simulates N–step trajectories (truncated episodes) offline using the learned transition network and the lookup table (Figure 5.3B). A simulated episode $\hat{s}_0^\mu \ldots \hat{s}_t^\mu \ldots \hat{s}_N^\mu$ is generated by first sampling $s \sim \mathcal{N}(0, \mathrm{I})$, and evaluating a random branch of the transition network to determine $\hat{s}_0^\mu$. This state is then wrapped back into the transition network (dotted line in Figure 5.1) and an action branch is evaluated according to $a_{t+1} = \arg\max_a \hat{Q}(\hat{s}_t^\mu, a)$ with probability $1-\epsilon$ or a random action with probability ϵ. The same process is repeated for N steps. The simulated returns $\hat{R}_t$ observed after taking an action

a_t from state $\hat{s}^{\mu}_{t-1}$ are given by

$$\begin{aligned} \hat{R}_t &= \hat{r}_t + \gamma \cdot \hat{r}_{t+1}(1-\hat{T}_t) + \dots \\ &\quad + \left[\gamma^{n-t} \cdot \hat{r}_n \prod_{i=t}^{N-1} (1-\hat{T}_i)\right] + \left[\gamma^{N-t+1} \cdot \max_a \hat{Q}(\hat{s}^{\mu}_N, a) \prod_{i=t}^{N} (1-\hat{T}_i)\right], \end{aligned} \tag{5.5}$$

where γ denotes the discount factor [Peng and Williams, 1996]. The values of $\hat{r}_t$ and $\hat{T}_t$ are determined by a kNN lookup on the associated key $\hat{s}^{\mu}_t$; the $(1-\hat{T}_i)$ factors prevent value signals from propagating through a learned terminal state. We found that we achieved better results by discarding the return estimate from the first state $\hat{s}^{\mu}_0$ in the sequence, which also tended to have greater error in observation space (Figure 5.3).

The simulated returns are used to update the Q–values in the lookup table, according to

$$Q^{(S)}(\hat{s}^{\mu}_t, a_t) \leftarrow \hat{Q}^{(S)}(\hat{s}^{\mu}_t, a_t) + \eta_Q(\hat{R}_t - \hat{Q}^{(S)}(\hat{s}^{\mu}_t, a_t)), \tag{5.6}$$

where the S superscript denotes Q–values estimated from simulated trajectories. The estimate $\hat{Q}^{(S)}(\hat{s}^{\mu}_t, a_t)$ on the right side is obtained from a kNN lookup. In addition, we maintained a separate lookup table for Q–values obtained from online (or "real–world") episodes, $Q^{(W)}(s^{\mu}_t, a_t)$. This table is updated using the MFEC algorithm (Equation 5.1) at the end of an episode. Note that the learning rule for simulated Q–values in Equation 5.6 involves averaging over existing table entries, unlike Equation 5.1. We found that this averaging helped to mitigate the impact of model error for simulated rollouts.

During inference, we use a kNN lookup over both real–world and simulated Q–values in the table,

$$\hat{Q}(s^{\mu}_t, a) = \begin{cases} \frac{1}{k}\sum_{i=1}^{k} Q(s^{\mu}_i, a) & \text{if } (s^{\mu}_t, a) \notin (W \cup S) \\ Q(s^{\mu}_t, a) & \text{otherwise,} \end{cases} \tag{5.7}$$

where W are the keys in the real–world lookup table, S are the keys in the simulated rollout lookup table, and $i = 1 \dots k$ indexes the k keys closest to s^{μ}_t already existing in either lookup table. If the key s^{μ}_t is sufficiently close to an existing state in the lookup table, the Q–value for that approximately matching state is returned; if it is close to both a real and simulated key, the real key is returned. In both real and simulated episodes, the agent's policy is determined by an ϵ–greedy strategy. Following Blundell et al. [2016], we used a least–recently used approach to remove keys from the two lookup tables (with a maximum of 1500 entries in each lookup table).

In general, lookup keys obtained from simulated rollouts exist off the manifold of real states due to model error (Figure 5.2). As a result, the estimate Equation 5.7 is dominated by real–world memories in regions of the environment that the agent has recently explored (i.e. where there are many matching keys in W). In addition, the greater the model error in a certain

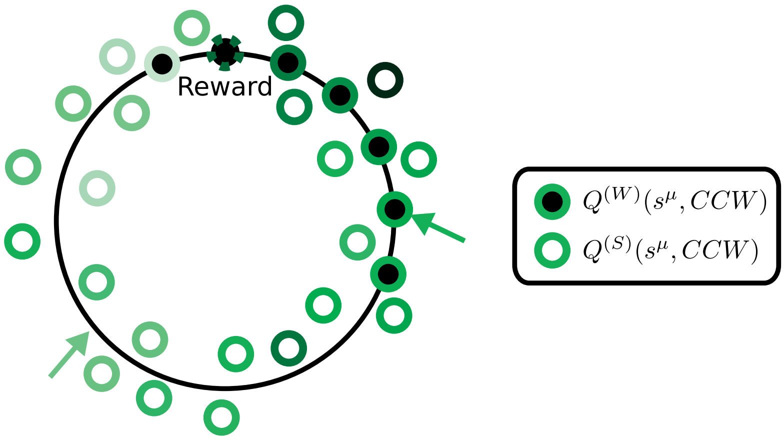

Figure 5.2 – **Limiting the impact of model error.** A schematic example where states sit on a low–dimensional manifold in latent space (a large circle), with a single recently discovered reward state (black dot with a dashed green outline). The reward memories for an action that results in a counter–clockwise transition on the circle are plotted from real–world episodes (black dots) and offline rollouts (white dots), where darker green outlines indicate higher values for the counter–clockwise action. Due to model error, the offline rollouts sit slightly off the manifold. As a result, a nearest–neighbour lookup will be dominated by real–world experience in recently visited states (darker green arrow) and by simulated episodes otherwise (lighter green arrow).

region of the environment, the farther the rollout states in S are from the manifold, and the less they contribute to the estimate. These factors help to mitigate the impact of model error in the value estimate.

Finally, in the experiment involving changing rewards, the Q–values in the lookup table become stale as soon as the rewards shift. Ideally, the agent should be able to empty the Q–values in the lookup table after detecting a reward schedule change. For the environments considered here, where the reward depends deterministically on the state the agent moves into, we used a simple change detection rule: with $\eta_r = 1$, if the agent encounters a state in a real–world trajectory where $r_t = 0$ and $r(s_t^{\mu}) \neq 0$, the Q–value table is emptied (more advanced change detection techniques could be used for other environments; see Discussion). In addition, we dynamically adjusted the value of ϵ for exploration according to

$$\epsilon_t \leftarrow \begin{cases} 1 & \text{if } \sum_i |r(s_i^{\mu})| = 0 \\ \max(\exp(-1/\tau_{\epsilon}) \cdot \epsilon_{t-1}, 0.1) & \text{otherwise.} \end{cases} \tag{5.8}$$

As a result, the agent explores randomly until it encounters a non–zero reward, at which point ϵ begins decreasing; it resets to $\epsilon = 1$ if the agent discovers that a previously rewarded state is no longer rewarding. We used the same value of ϵ for both real–world and simulated episodes.

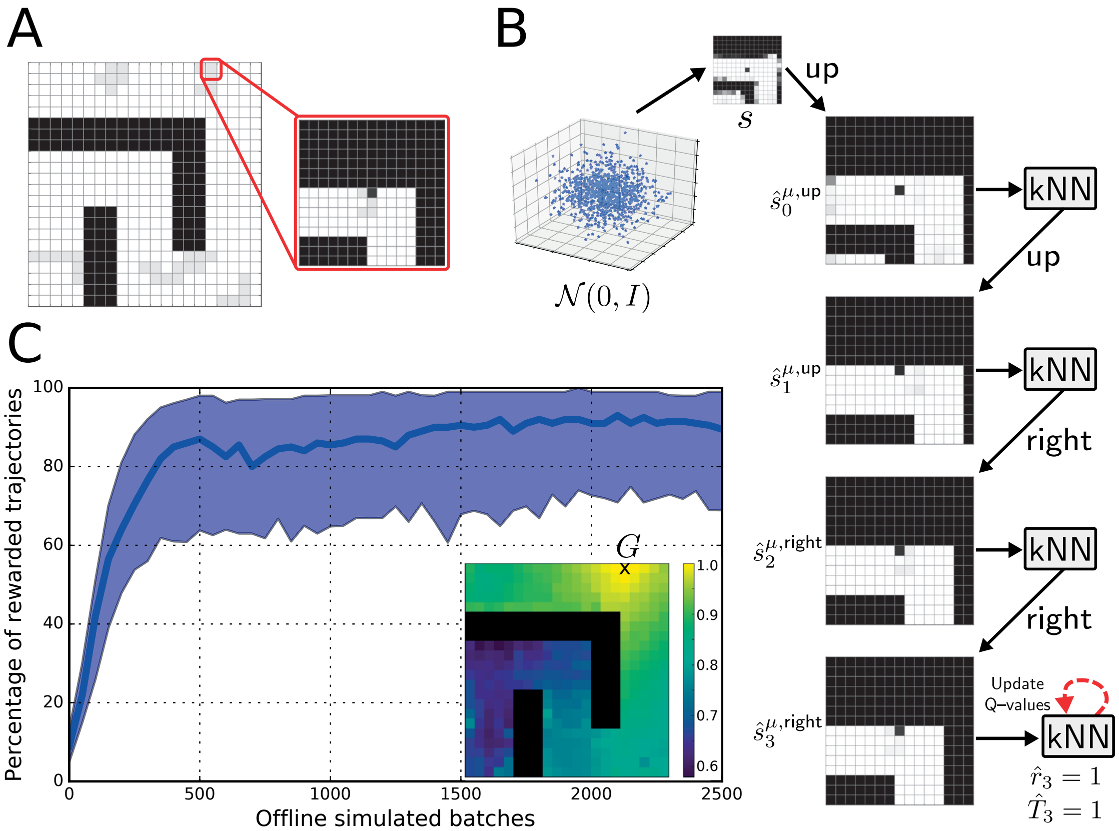

Figure 5.3 – **Learning from offline episodes in a GridWorld environment.** [**A**] The larger plot illustrates the environment excluding exterior walls. The agent's actual observations when it occupies a particular state are highlighted in red, including the black pixels that represent the interior and exterior walls and the agent itself. [**B**] Offline episodes are generated by sampling from the $\mathcal{N}(0, \mathrm{I})$ prior over the initial state, then rolling out an on–policy trajectory according to the Q–values in the lookup table. [**C**] N–step returns were rapidly improved by offline simulation. The percentage of online trials ending in reward is shown as a function of the number of offline rollout batches (beginning with an empty Q–value table, with no online learning). The midline shows the median performance, with the shaded region corresponding to the 10^{th} and 90^{th} percentiles. The inset shows the generated $\max_a \hat{Q}(s, a)$ for an example goal state G.

5.4 Results

We tested the model on a 360–state GridWorld maze (Figure 5.3A), with observations consisting of 15 × 15 pixel boxes around the agent, with four actions (up, down, right, and left). For the purpose of our investigation, we used an environment that was both fully controllable and fully observable at each time step [Kirk, 2012]. By controllable, we mean that there exists a sequence of actions that will transition the agent between any arbitrary pair of states; by observable, we mean that the agent can determine the underlying state from the current observations. The latent learning phase consisted of a 1 000 000 step random walk, sampling a minibatch of 20 transitions every 10 steps from a replay memory of the last 10 000 transitions.

We first tested the agent's ability to learn entirely from offline episodes. The agent experienced each (o, r, T) triplet in the environment, including the triplet associated with a single randomly–chosen rewarding, terminal goal state G. The agent then estimated the value function by generating 2500 minibatches of ten 50–step offline episodes each, with ϵ annealed from 1 to 0.1 over the first 700 minibatches. Online performance was measured every 50 batches, using the proportion of 100 randomly–initialized agents that reached a given G within 100 steps under the offline–generated policy. We used $k = 3$ for the Q–value lookup and $k = 1$ for the $\hat{r}_t$ and $\hat{T}_t$ lookup. The agent's policy significantly improved as a result of the online backups (Figure 5.3). Results are based on 100 random goal states.

Next, we combined online and offline learning with the goal state G changing every 25 000 steps. Each trial consisted of a total of 125 000 steps, i.e. 5 different goal locations. When the agent reached the current goal location, it started a new episode in a random initial state (since the environment was fully controllable, it was possible for the agent to issue a sequence of actions allowing it to transition between any pair of start and goal positions). Every 20 steps in the real world, the agent simulated ten 50–step offline trajectories with initial states sampled from $\mathcal{N}(0, \mathrm{I})$, and updated Q^S accordingly. We used Equation 5.8 with $\tau_\epsilon = 7500$ to determine the policy.

We compared against three baseline conditions across 100 trials. In the first, the agent's policy was based only on Q–values obtained from online experience, corresponding to MFEC with a VAE–based latent representation (except that $\log s_t^\sigma$ was not included in the latent embedding).

In the second baseline, in addition to online learning, we included Dyna–Q–like 1–step offline backups, sampled uniformly from the last 10 000 transition triplets (s_t, a_{t+1}, s_{t+1}) in a replay memory. The immediate rewards $\hat{r}(s_{t+1}^\mu)$ for these transitions were obtained from the lookup table; as a result, the agent could continue to use transitions from the replay memory to update the value function after the rewards had changed. We used a learning rate $\eta = 1$, which is optimal for deterministic environments. The agent made $10 \cdot 50 = 500$ offline backups every 20 steps.

In the third baseline, we simulated 50–step offline trajectories as in the experimental model; however, the initial states were sampled from the replay memory rather than $\mathcal{N}(0, \mathrm{I})$.

The final results are shown in Figure 5.4. The best performance was obtained using offline n–step backups initiated from the latent state prior $\mathcal{N}(0, \mathrm{I})$, followed closely by offline N–step backups initiated from states in the replay memory. The median returns for the two approaches were approximately the same; however, replay memory–initiated trajectories occasionally resulted in very poor performance (Figure 5.4B).

In many subtrials, the agent failed to learn a useful policy during the exploration period after discovering the new reward location. This could result in compounding error over time. For

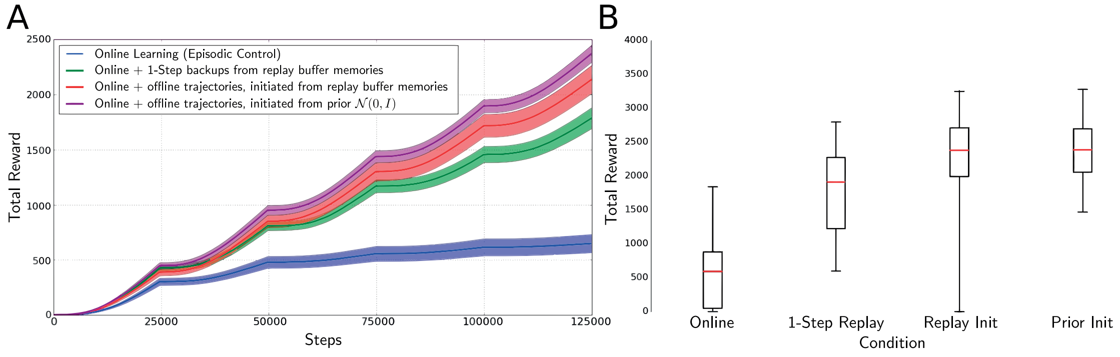

Figure 5.4 – **Combining offline and online experiences in GridWorld.** [**A**] Mean reward ± SEM over all trials for the four conditions. Goal positions were randomly reassigned every 25 000 steps. [**B**] Quartile plot of total reward across the four conditions. Whiskers correspond to 5^{th} and 95^{th} percentiles.

instance, if the agent failed to learn how to consistently reach the reward before ϵ converged to 0.1 during the third subtrial, it would also often fail to discover the reward landscape had changed when the fourth subtrial began; as a result, it would not initiate an exploration period during the fourth and fifth subtrials (typically becoming "stuck" in an uninteresting region of the environment). We analyzed learning failures over the five subtrials in Figure 5.5. The online and 1–step backup agents showed the most accumulating failures over time. The effect was most obvious for the online agent, which had a nearly 100% failure rate by the end of the trial. In contrast, the agents using offline simulated episodes had an approximately constant failure rate over time.

5.5 Related work

5.5.1 Model–based simulated experience

Dyna–Q has traditionally been confined to tabular environments without function approximation or used with linear function approximation [Sutton et al., 2008]. A recent deep network model uses Dyna–Q–like simulated rollouts initiated from memories in a replay buffer to improve learning in a continuous control task [Gu et al., 2016]. Several recent approaches learn and use predictive models of rewards and value functions [Silver et al., 2017b, Oh et al., 2017, Farquhar et al., 2017], but not the observations. Racanière et al. [2017] learn a rollout model and rollout policy trained on the observations and combine it with a model–free pathway to generate an output policy.

The network we present here differs from these existing approaches by storing the results of both online and simulated rollouts in a non–parametric memory, as in an episodic controller (Blundell et al. [2016], Pritzel et al. [2017]). In this regard, our network is most similar to the model from Fraccaro et al. [2018], which also augments a model–based rollout algorithm

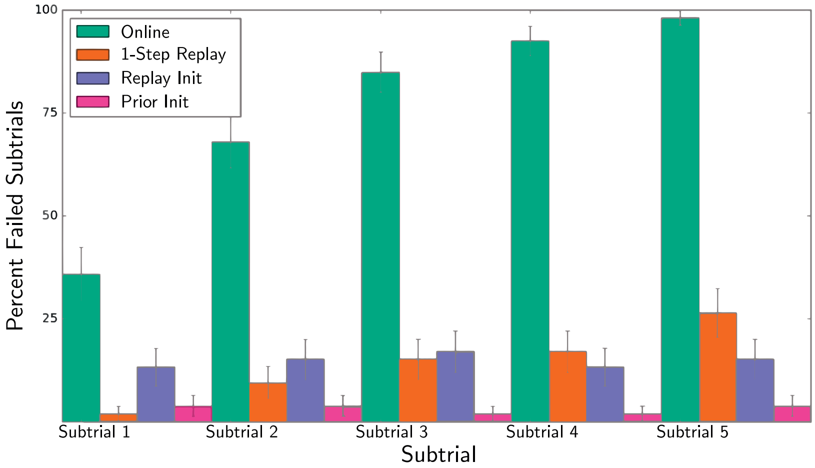

Figure 5.5 – **Failures to learn across conditions.** For each of the five random reward positions (subtrials) presented in each trial, we considered the subtrial a failure if the agent reached the reward position fewer than 10 times over the course of 25 000 steps.

with a lookup table memory; however, their model assumes a spatial task and is not used to store rewards or predict the value function. The structure of our action–conditional transition network is somewhat reminiscent of a conditional variational autoencoder [Sohn et al., 2015]; however the action is not available to the encoder, and the condition is evaluated by training a specific branch of the decoder, where each branch corresponds to a single action (see Figure 5.1).

5.5.2 Dimensionality reduction for RL

There is a rich tradition of study in dimensionality reduction for reinforcement learning, using unsupervised learning techniques like Principal Component Analysis [Curran et al., 2016] as well as task–relevant reward or goal information [Parisi et al., 2017, Rolf and Asada, 2015]. Tangkaratt et al. [2016] consider learning a low–dimensional representation of the high–dimensional input space that captures the transition dynamics of the original state space, in order to apply model–based RL. Luck et al. [2014] integrate dimensionality reduction with policy search through an Expectation Maximization–based approach. Dimensionality reduction for RL has also been proposed as a model for the function of neural circuits [Shah and Alexandre, 2011].

Further, several existing models perform RL using autoencoders and VAEs. Lange and Riedmiller [2010] train a deep autoencoder to create a low–dimensional representation of the input space, and use fitted Q–iteration [Ernst et al., 2005] for generalization. The autoencoder and Q–value function approximator are retrained in batch mode after each episode until a satisfactory approximation is achieved. Barth-Maron [2015] train a deep convolutional autoencoder on GridWorld input, with the latent representation used as a basis for gradient descent SARSA(λ)–based RL [Rummery and Niranjan, 1994]. Finn et al. [2016] also train a deep convolutional autoencoder, using a spatial softmax layer before the bottleneck, as a basis for a robotic continuous control task.

Watter et al. [2015], focusing on continuous state and action spaces, propose *Embed to Control*, in which a VAE is trained with a locally linear dynamics model (unlike the nonlinear transition model we use) and apply stochastic optimal control algorithms in the latent space to determine a policy. Van Hoof et al. [2016] also train a VAE (as well as a denoising autoencoder) to construct a latent state space representation of the input with a learned affine transition model for training the latent representation. They learn a policy using a non–parametric kernel embedding trained to respect the system dynamics [Van Hoof et al., 2015]. The DARLA model uses a VAE with a high weight on the isotropic Gaussian prior to extract independent components of the input space, in order to improve performance on transfer learning with various RL algorithms [Higgins et al., 2017].

Dimensionality reduction is closely related to the study of state space abstractions for reinforcement learning, reviewed by Li et al. [2006] and considered in greater detail in Section 6.4.

5.5.3 Non–parametric approaches to RL

Non–parametric approaches have been used on RL problems beyond the recent episodic control paradigm. They have been applied to model–based controllers in order to mitigate the impact of model error when using continuous differential models for imitation learning on robotics tasks [Atkeson and Schaal, 1997, Atkeson, 1998]. Doshi-Velez et al. [2010] also apply a nonparametric approach for imitation learning, where the data generated by an expert's near–optimal policy is used to reduce uncertainty about the partially–observable world model and the optimal policy. Morimura et al. [2010] describe an approach that uses particle smoothing to estimate the distribution over returns from state–action pairs (as opposed to simply the mean), allowing for risk–sensitive policies. Driessens and Džeroski [2005] combine model–based and instance–based approaches for relational RL, where the model takes the form of a decision tree. Non–parametric approaches have further been applied to inverse RL [Choi and Kim, 2012].

While we use a nearest neighbour–based method here, non–parametric approaches to RL based on kernel methods [Ormoneit and Sen, 2002, Kveton and Theocharous, 2013, Chen et al., 2013, Barreto et al., 2016b] and Gaussian processes [Kuss and Rasmussen, 2004, Engel et al., 2005, Ko et al., 2007, Deisenroth et al., 2009, Deisenroth and Rasmussen, 2011] have also been studied.

5.6 Discussion

In this chapter, we considered how an agent could use a VAE–like architecture to learn the structure of an environment and simulate episodes offline. In contrast to existing approaches, the agent stores the results of offline episodes in a kNN lookup table, which it can use to rapidly generate new policies for changing reward landscapes.

We employed a simple reward change detection method for the agent that would not be suitable for more complex tasks. A more general change detection method could make use of the lookup table to model a distribution over expected rewards in the current state, and employ a surprise measure given the actual experienced reward (e.g. Faraji et al. [2018]) in order to determine when to reset the kNN lookup tables.

The agent was evaluated in a GridWorld environment. Although the underlying environment was tabular, the agent observed the pixel representation of each state, allowing for a measure of similarity between different observations. Compared to several alternative approaches, the agents combining online experience with simulated offline episodes performed the best; the most consistent performance was achieved when offline episodes were initiated with samples from $\mathcal{N}(0, \mathrm{I})$), a probability distribution over the environment observations learned during a random walk. In addition to improving the Q–value estimates, the model–generated offline experience likely provided additional noise in the policy, pushing the agent out of repetitive patterns.

Two of the baselines were based on sampling experiences from a replay memory of recently observed transitions, which is biased towards regions of the environment that the agent has recently explored. This can be both detrimental (if the agent becomes stuck in a cycle over a small number of states, recent experience is less useful in updating the policy) and beneficial (if the agent's performance is improving, recent experience often reflects important regions of the state space, i.e. states that are close to the goal). Further experiments on the optimal length and sampling procedure for the replay memory may be worthwhile, considering that both replay–based agents performed significantly better than vanilla MFEC.

The model is unique in combining model–based and episodic control approaches to RL. In particular, the non–parametric approach of episodic control allows model–based rollouts to be leveraged rapidly by the agent, avoiding the slow process of learning value functions by gradient descent, while limiting model error through the kNN lookup. The model is also potentially relevant to a recent theory of hippocampal function, which suggests that the hippocampus acts primarily as an episodic controller rather than a model–based controller [Gershman and Daw, 2017]. If, as in our model, the hippocampus is an episodic controller that also learns a transition model in the latent space, then both real–world episodes and model–based simulated episodes can be combined to estimate the value function. Our results suggest that this is useful when dealing with non–stationary rewards, a problem area that the hippocampus is particularly associated with (as detailed in Chapter 1).

The model detailed in this chapter relies on forward–focused planning, i.e. simulated trajectories moving forward in time. A sample–efficient alternative is a backwards–focused planning mechanism, such as prioritized sweeping. We consider how to apply backwards–focused planning in complex environments in the next chapter.

6 Efficient Model–Based RL with Variational State Tabulation

6.1 Introduction

Classical RL techniques generally assume a tabular representation of the state space [Sutton and Barto, 2018]. While methods like prioritized sweeping [Moore and Atkeson, 1993, Peng and Williams, 1993, Van Seijen and Sutton, 2013, Sutton and Barto, 2018] have proven to be very sample–efficient in tabular environments, there is no canonical way to carry them over to very large (or continuous) state spaces, where the agent seldom or never encounters the same state more than once. Recent approaches to reinforcement learning have shown tremendous success by using deep neural networks as function approximators in such environments, allowing for generalization between similar states (e.g. Mnih et al. [2015, 2016]) and learning approximate dynamics to perform planning at decision time (e.g. Silver et al. [2017b], Oh et al. [2017], Farquhar et al. [2017], Racanière et al. [2017], Nagabandi et al. [2017]). However, the use of prioritized sweeping for offline updates of Q–values (i.e. background planning [Sutton and Barto, 2018]), has not yet been investigated in conjunction with function approximation by neural networks.

Adjusting the weights in a deep network to learn a value function is a slow procedure relative to learning in tabular environments. As an alternative, episodic controllers (as considered in Chapter 2) improve sample efficiency by using a semi–tabular approach in high–dimensional environments; new episodes can be immediately incorporated into the value function estimate via the lookup table. This chapter describes a model that brings the efficiency of both lookup tables and prioritized sweeping to high–dimensional environments by using a deep network to learn a tabular abstraction of the state space.

As motivation, we consider the T–maze task shown in Figure 6.1A using an MFEC agent (a type of episodic control agent described in Chapter 5). The observations are continuous (x, y) coordinates; the agent can take a fixed–sized step in one of four cardinal directions, with a rebound on hitting a wall, and a terminal reward zone (green). The first episode (red) is spontaneously terminated without reward; the discounted returns along the red trajectory are therefore set to zero. On the second episode (blue), the agent reaches the reward zone, and the

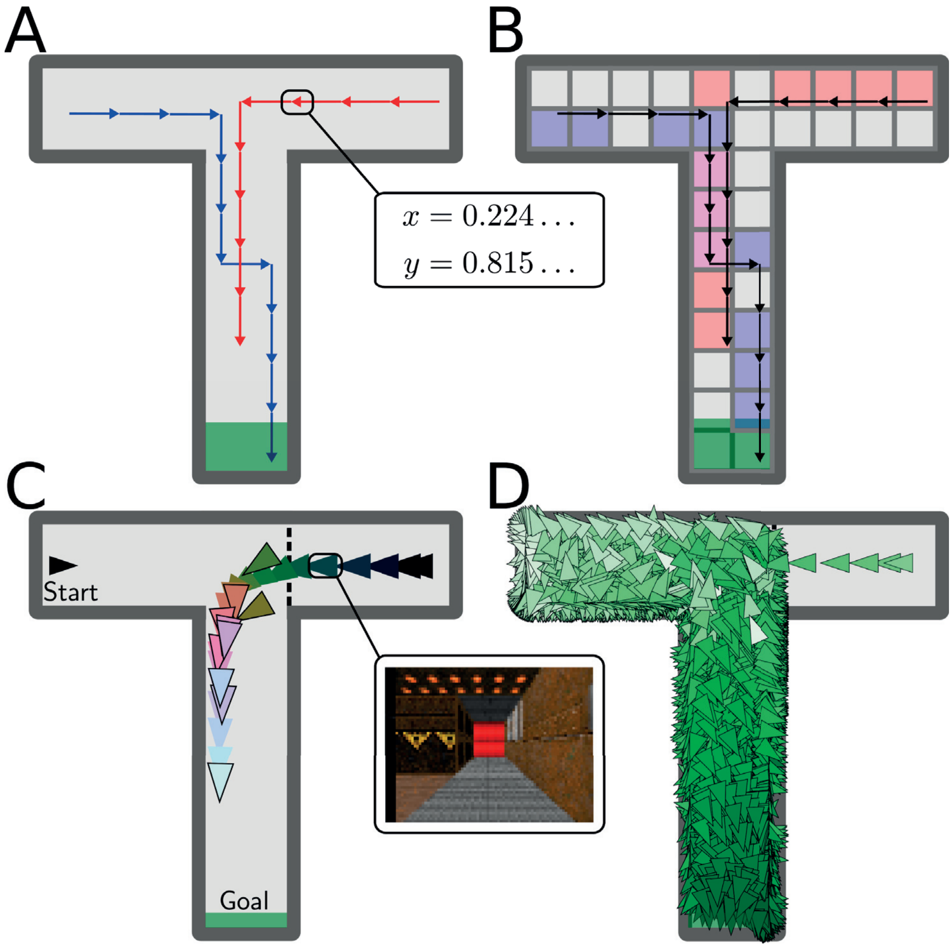

Figure 6.1 – **Using state tabulation for efficient planning**. [**A**] Two episodes in a time discrete MDP with a continuous state space, given by (x, y) coordinates. [**B**] The same episodes after discretizing the state space by rounding. States visited on each trajectory are shaded; magenta states were shared by both trajectories, and can be leveraged by prioritized sweeping. [**C**] 3D navigation with VaST. The agent was trained to run from the start position to the goal, with one arm blocked (dotted line). After training, the agent experienced a trajectory from the blocked arm to the stem of the maze (arrows, no outline). If the observations in this trajectory mapped to existing states in the lookup table, the average coordinates and orientation where those states were previously observed are shown (matching colour arrows, outlined). The observations for one state are illustrated. [**D**] A scatter plot of all state values after training according to the average position and orientation where they were observed; darker green corresponds to higher value.

discounted reward is immediately associated with the states visited along the blue trajectory.

To improve sample efficiency, we consider how an agent could apply the experience of the rewarded trajectory to Q–value estimates in the top–right arm. In particular, a learned a model of the environment could indicate that both trajectories pass through the center of the T–maze, and that discovering a reward at the bottom of the maze should therefore change Q–value estimates in both of the arms at the top. For instance, using prioritized sweeping, a state that changing significantly in value will result in a rapid update of all the state–action pairs that transition into that state [Moore and Atkeson, 1993, Peng and Williams, 1993, Van Seijen and Sutton, 2013]. If both trajectories passed through the same position in the center of the maze, prioritized sweeping could then exploit this fact to rapidly update the value function in both arms.

However, assuming random restarts, the agent in this task never encounters the exact same state more than once. In this case, given a deterministic task, prioritized sweeping as implemented by Van Seijen and Sutton [2013] collapses to the MFEC learning algorithm [Brea, 2017]. We are therefore motivated to consider mapping the observation space to a tabular representation by some form of discretization. For example, with discretization based on rounding the (x, y) coordinates (a simple form of state aggregation [Li et al., 2006, Sutton and Barto, 2018]), the two trajectories in Figure 6.1B now pass through several of the same states. A model–based prioritized sweeping algorithm would allow us to update the Q–values in the top–right arm of the maze to nonzero values after experiencing both episodes, despite the fact that the red trajectory did not result in reward.

Such a discretization can be considered a form of state aggregation, in which multiple observations $o_1 \dots o_n$ map to the same abstract state s. Optimally, the abstraction should be fine enough that all observations in the same state s share similar optimal Q–values [Li et al., 2006], yet coarse enough that the agent can visit the same states multiple times to leverage a densely connected model of the environment.

Even if the underlying task is deterministic, discretization can result in a stochastic state transition model. In the top right of Figure 6.1, the successor state and reward resulting from an action depend on the exact (continuous) position within the discrete state where the action took place, which is hidden at the abstract state level (as shown by the fact that neighbouring states are sometimes “skipped”). In addition, $p(o|s)$ is not a delta function, since multiple observations correspond to the same state. Similarly, $p(s|o)$ may not be a delta function if the abstraction function results in partial membership for observations on the edge of multiple abstract states. We have therefore transformed the system from a continuous *fully–observable* MDP to a discrete *partially–observable* Markov Decision Process (POMDP).

If observations are given by high-dimensional visual inputs instead of (x, y) coordinates, the simple form of state aggregation by rounding (Figure 6.1B) is impractical. Instead, we propose and describe in this chapter the new method of Variational State Tabulation (VaST)[1] for learning discrete, tabular representations from high–dimensional and/or continuous observations. VaST can be seen as an action conditional hybrid ANN–HMM (artificial neural network hidden Markov model, see e.g. [Bengio et al., 1992, Tucker et al., 2017, Ng et al., 2016, Maddison et al., 2016]) with a d-dimensional binary representation of the latent variables, useful for generalization in RL. VaST is trained in an unsupervised fashion by maximizing the

---

[1] The full code for VaST can be found at https://github.com/danecor/VaST/.

evidence lower bound. We exploit a parallelizable implementation of prioritized sweeping by small backups [Van Seijen and Sutton, 2013] to constantly update the value landscape in response to new observations. By creating a tabular abstraction where states are often revisited, the agent can rapidly update state–action values in distant areas of the environment in response to single observations.

In Figure 6.1C&D, we show how VaST can use the generalization of the tabular representation to learn from single experiences. We consider a 3D version of the example T–maze, implemented in the *VizDoom* environment [Kempka et al., 2016]. Starting from the top–left arm, the agent was trained to run to a reward in the bottom of the T–maze stem. During training, the top–right arm of the T–maze was blocked by an invisible wall (dotted line). After training, the agent observed a single, 20–step fixed trajectory (or "forced run") beginning in the top–right arm and ending in the stem, without reaching the reward zone (Figure 6.1C). The agent's early observations in the unexplored right arm were mapped to new states, while the observations after entering the stem were mapped to existing states in the lookup table (corresponding to observations at similar positions and orientations). The agent was able to update the values of the new states by prioritized sweeping from the values of familiar states (Figure 6.1D), without needing to change the neural network parameters, as would be necessary with model-free deep reinforcement learners like DQN [Mnih et al., 2013]. In total, the network generated only 8973 unique states for the 50021 observations during training, indicating significant generalization between similar observations.

6.2 Learning the model

In order to compute a policy using the model–based prioritized sweeping algorithm described by Van Seijen and Sutton [2013], we seek a posterior distribution $p(s_t^\mu | o_{0:t^\mu}^\mu)$ over latent *discrete* states s_t^μ given all past observations $o_{0:t^\mu}$ in the current episode μ. We will later describe how the posterior model is used to build a tabular abstraction.

Classically, the parameters of such a posterior model could be estimated using Expectation Maximization (EM) according to the Baum–Welch algorithm [Baum et al., 1970], with a different transition distribution for each possible action [Chrisman, 1992]. This would require learning an observation model $p_\theta(o_t|s_t)$, a transition model $p_\theta(s_t|a_t, s_{t-1})$ and an initial state distribution $\pi_\theta(s_0)$, all parameterized by θ. At each iteration of the algorithm, the model parameters θ are improved by first calculating the posterior probability of each latent state at each time step under the current parameters, using Bayes' rule.

For the *VizDoom* environment we evaluate on, the observation space is 60×80 pixels and 3 colour channels. An observation model based on a Gaussian Mixture would therefore have over 200 million parameters for each state. With tens of thousands of states necessary to create an accurate model of a complex environment, it rapidly becomes computationally intractable to calculate the posteriors of all states and update the model parameters in a reasonable number of steps. As well, modeling directly on the pixel space fails to capture important features like translation invariance. We overcome these issues by parameterizing the probabilistic models as deep neural networks, and using variational inference to approximate the posterior distribution.

6.2.1 The variational cost function

For a sequence of states $s_{0:T} = (s_0, \dots s_T)$ and observations $o_{0:T} = (o_0, \dots o_T)$, we consider a family of approximate posterior distributions $q_\phi(s_{0:T}|o_{0:T})$ with parameters ϕ, which we assume to factorize given the current observation and a memory of the past k observations, i.e.

$$q_\phi(s_{0:T}|o_{0:T}) = \prod_{t=0}^{T} q_\phi(s_t|o_{t-k:t}), \tag{6.1}$$

where observations before $t = 0$ consist of blank frames. We learn q_ϕ indirectly using the distribution p_θ parameterized by θ. Given a collection of M observation sequences $\mathcal{O} = \{o^\mu_{0:T^\mu}\}^M_{\mu=1}$ and hidden state sequences $\mathcal{S} = \{s^\mu_{0:T^\mu}\}^M_{\mu=1}$, we maximize the log–likelihood $\log \mathcal{L}(\theta; \mathcal{O}) = \sum_{\mu=1}^M \log p_\theta \left(o^\mu_{0:T^\mu}\right)$ of the weight parameters θ, while minimizing $\mathcal{D}_{KL}(q_\phi(\mathcal{S}|\mathcal{O})||p_\theta(\mathcal{S}|\mathcal{O}))$. Together, these terms form the evidence lower bound (ELBO) or negative variational free energy

$$-\mathcal{F}(\theta, \phi; \mathcal{O}) = \log \mathcal{L}(\theta; \mathcal{O}) - \mathcal{D}_{KL}(q_\phi(\mathcal{S}|\mathcal{O})||p_\theta(\mathcal{S}|\mathcal{O})). \tag{6.2}$$

By training on the objective function in Equation 6.2, the parameters θ are optimized to learn a model p_θ that captures the observation distribution. Then, rather than calculating the posterior $p_\theta(\mathcal{S}|\mathcal{O})$ according to that model directly (as in EM), we optimize the variational posterior $q_\phi(\mathcal{S}|\mathcal{O})$ to approximate $p_\theta(\mathcal{S}|\mathcal{O})$, by minimizing the Kullback–Leibler divergence between the two posteriors. Following the VAE approach and using the reparameterization trick (Kingma and Welling [2013], Rezende et al. [2014], introduced in Chapter 5), the objective function can be optimized efficiently by sampling from q_ϕ.

Equation 6.2 can be rewritten as

$$\begin{aligned} -\mathcal{F}(\theta, \phi; \mathcal{O}) &= \log \mathcal{L}(\theta; \mathcal{O}) - \mathcal{D}_{KL}(q_\phi(\mathcal{S}|\mathcal{O})||p_\theta(\mathcal{S}|\mathcal{O})) \\ &= \mathbb{E}_{q_\phi}[\log p_\theta(\mathcal{S}, \mathcal{O})] + \mathcal{H}(q_\phi(\mathcal{S}|\mathcal{O})), \end{aligned} \tag{6.3}$$

where $\mathcal{H}$ denotes the entropy of the distribution. The term inside the expectation evaluates to

$$\begin{aligned} \log p_\theta(\mathcal{S}, \mathcal{O}) = &\sum_{\mu=1}^{M} \log \pi_{\theta_0}(s^\mu_0) + \sum_{\mu=1}^{M} \sum_{t=0}^{T^\mu} \log p_{\theta_\mathcal{R}}(o^\mu_t|s^\mu_t) \\ &+ \sum_{\mu=1}^{M} \sum_{t=1}^{T^\mu} \log p_{\theta_\mathcal{T}}(s^\mu_t|a^\mu_t, s^\mu_{t-1}), \end{aligned} \tag{6.4}$$

where a^μ_t denotes the action taken by the agent on step t of sequence μ, π_{θ_0} is the distribution over initial states, and $\theta_0 \cup \theta_\mathcal{R} \cup \theta_\mathcal{T} = \theta$.

We aim to learn the appropriate posterior distribution q_ϕ by minimizing the variational free energy (maximizing the ELBO). Our cost function from Eq. 6.3 can be written as

$$\mathcal{F}(\theta,\phi;\mathcal{O}) = \sum_{\mu=1}^{M}\sum_{t=0}^{T^\mu}\left[\mathcal{R}_t^\mu + \mathcal{T}_t^\mu - \mathcal{H}_t^\mu\right], \tag{6.5}$$

with reconstruction cost terms

$$\mathcal{R}_t^\mu = -\sum_{s_t^\mu} q_\phi(s_t^\mu|o_{t-k:t}^\mu)\log p_{\theta_\mathcal{R}}(o_t^\mu|s_t^\mu)\,, \tag{6.6}$$

transition cost terms

$$\mathcal{T}_t^\mu = -\sum_{s_t^\mu,s_{t-1}^\mu} q_\phi(s_t^\mu,s_{t-1}^\mu|o_{t-k-1:t}^\mu)\log p_{\theta_\mathcal{T}}(s_t^\mu|a_t^\mu,s_{t-1}^\mu) \tag{6.7}$$

for $t>0$ and $\mathcal{T}_0^\mu = -\sum_{s_0^\mu} q_\phi(s_0^\mu|o_0^\mu)\log\pi_{\theta_0}(s_0^\mu)$, and entropy terms

$$\mathcal{H}_t^\mu = -\sum_{s_t^\mu} q_\phi(s_t^\mu|o_{t-k:t}^\mu)\log q_\phi(s_t^\mu|o_{t-k:t}^\mu)\,. \tag{6.8}$$

We parameterize the posterior distribution q_ϕ (or "encoder") using a deep CNN [Krizhevsky et al., 2012], and the observation model $p_{\theta_\mathcal{R}}$ using a deep DCNN [Goodfellow et al., 2014], as shown in Figure 6.2. We use a multilayer perceptron (3 layers for each possible action) for the transition model $p_{\theta_\mathcal{T}}$, and learned parameters θ_0 for the initial state distribution π_{θ_0}. The architecture is similar to that of a VAE, with the fixed priors replaced by learned transition probabilities conditioned on previous state–action pairs. Unlike the model in Chapter 5, the transition cost terms here arise naturally from evaluating the ELBO under the assumption that the observations come from an MDP.

To allow for a similarity metric between discrete states, we model the state space as all possible combinations of d binary variables, resulting in $N = 2^d$ possible states. Each of the d outputs of the encoder defines the expectation of a Bernoulli random variable, with each variable sampled independently. The sampled states are used as input to the observation and transition networks, and as targets for the transition network.

The reconstruction and transition cost terms can now be used in stochastic gradient descent on $\mathcal{F}$ in θ, by estimating the gradient $\nabla_\theta\mathcal{F}$ with Monte Carlo samples from the variational posterior q_ϕ. To minimize $\mathcal{F}$ also in ϕ, we need to perform backpropagation over discrete, stochastic variables (i.e. over s_t^μ sampled from q_ϕ). There are several methods for doing this (see Discussion). We use the reparameterization trick together with a relaxation of the Bernoulli distribution: the binary Con–crete (or Gumbel–Softmax) distribution (Maddison et al. [2016], Jang et al. [2016]).

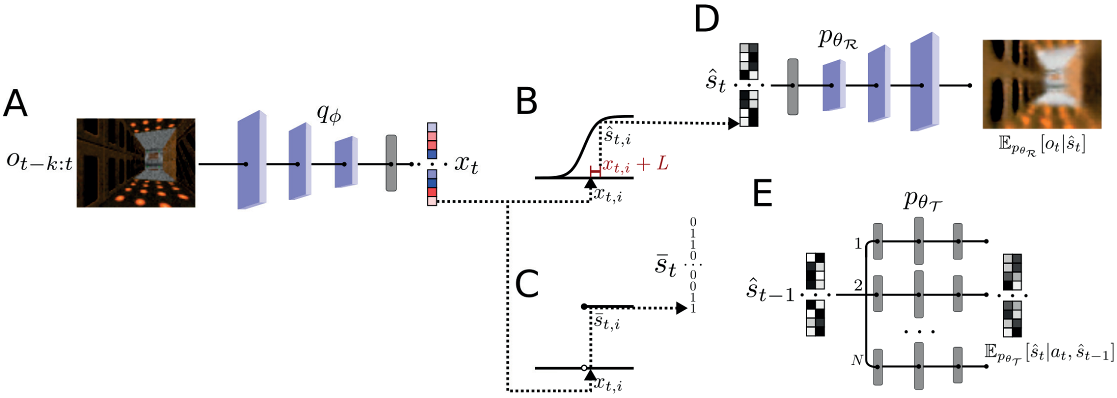

Figure 6.2 – **The network model**. [**A**] CNN encoder q_ϕ. [**B**] Encoder outputs can be used to sample each dimension from a Con–crete distribution for training ($\hat{s}_t$, where the Con–crete distribution corresponds to a logistic activation with added noise L), or [**C**] discretized to the Bernoulli mode $\bar{s}_t$ to update the table. [**D**] DCNN decoder $p_{\theta_\mathcal{R}}$ and [**E**] Transition network $p_{\theta_\mathcal{T}}$, with N possible actions. For illustration, $\mathbb{E}_{p_{\theta_\mathcal{R}}}[o_t|\hat{s}_t]$ and $\mathbb{E}_{p_{\theta_\mathcal{T}}}[\hat{s}_t|a_t, \hat{s}_{t-1}]$ are shown.

6.2.2 The reparameterization trick and the Con–crete distribution

Denoting the i*th* dimension of state s_t as $s_{t,i}$, we consider the i*th* output of the encoder at time t to correspond to $x_{t,i} = \text{logit}(q_\phi(s_{t,i} = 1|o_{t-k:t}))$. Following Maddison et al. [2016], we note that we can achieve a Bernoulli distribution by sampling according to $s_{t,i} = H(x_{t,i} + L)$, where H is the Heaviside step function and L is a logistic random variable. In this form, the stochastic component L is fully independent of ϕ, and we can simply backpropagate through the deterministic nodes [Kingma and Welling, 2013]. However, the derivative of H is 0 almost everywhere. To address this, the Bernoulli distribution can be relaxed into a continuous Con–crete (*continuous* relaxation of *discrete*) distribution [Maddison et al., 2016]. This corresponds to replacing the Heaviside non–linearity with a logistic non–linearity parameterized by the temperature λ:

$$\hat{s}_{t,i} = \frac{1}{1 + \exp(-(x_{t,i} + L)/\lambda)}, \tag{6.9}$$

with $\hat{s}_{t,i} \in [0, 1]$. We use Con–crete samples from the encoder output for the input to both the reconstruction and transition networks and for the targets of the transition network, with temperatures taken from those suggested by Maddison et al. [2016]: $\lambda_1 = 2/3$ for the posterior distribution and $\lambda_2 = 1/2$ for evaluating the transition log–probabilities. The Con–crete relaxation corresponds to replacing the discrete joint Bernoulli samples s_t in the previous loss functions with their corresponding joint Con–crete samples $\hat{s}_t$. We train the network by sampling minibatches of observations and actions $(o^\mu_{t-k-1:t}, a^\mu_t)$ from a replay memory [Riedmiller, 2005, Mnih et al., 2015] of transitions observed by the agent.

6.2.3 Learning a tabular transition model

The model as described learns a joint Con–crete posterior distribution $\hat{q}_\phi(\hat{s}_t|o_{t-k:t})$. We can recover a discrete joint Bernoulli distribution $q_\phi(s_t|o_{t-k:t})$ by replacing the logistic non–linearity with a Heaviside non–linearity (i.e. as $\lambda \to 0$ in Eq. 6.9).

For prioritized sweeping, we need to build a tabular model of the transition probabilities in the environment (i.e. $p(s_t|a_t, s_{t-1})$). We could consider extracting such a model from $p_{\theta_\mathcal{T}}(\hat{s}_t|a_t, \hat{s}_{t-1})$, the transition network used to train the encoder. However, this is problematic for several reasons. The transition network corresponds to Con–crete states, and is of a particularly simple form, where each dimension is sampled independently conditioned on the previous state and action. Moreover, the transition network is trained through stochastic gradient descent and therefore learns slowly; we want the agent to rapidly exploit new transition observations.

We therefore build a state transition table based purely on the encoder distribution $q_\phi(s_t|o_{t-k:t})$, by treating the most probable sequence of states under this distribution as observed data. Since each dimension of s_t is independent conditioned on the observations, the mode $\bar{s}_t$ at time t corresponds to a d–length binary string, where $\bar{s}_{t,i} = H(x_{t,i})$. Likewise, since states within an episode are assumed to be independent conditioned on the causal observation filter, the most probable state sequence for an episode is $\bar{\mathcal{S}}^\mu = \{\bar{s}_0^\mu, \bar{s}_1^\mu \ldots, \bar{s}_{T^\mu}^\mu\}$. We therefore record a transition between $\bar{s}_{t-1}$ and $\bar{s}_t$ under action a_t for every step taken by the agent, and update the expected reward $r(\bar{s}_{t-1}, a_t)$ in the table with the observed reward. Each binary string $\bar{s}$ is represented as a d–bit unsigned integer in memory.

This process corresponds to empirically estimating the transition probabilities and rewards by counting transitions between the most probable states. As the model changes during training, we also revise the counts. As an example, assume the agent encounters states A, B and C successively in the environment. We record transitions $A \to B$ and $B \to C$ in the table, and store the raw observations along with the corresponding state assignments A, B and C in the replay memory. If the observations associated with B are later sampled from the replay memory and instead assigned to state D, we delete $A \to B$ and $B \to C$ from the table and add $A \to D$ and $D \to C$. Both the deletion and addition of transitions through training can change the Q–values.

6.2.4 Using the model for reinforcement learning

The Q–values in the table are updated continuously using the learned transition model $p(\bar{s}_t|a_t, \bar{s}_{t-1})$, expected reward $r(\bar{s}_{t-1}, a_t)$ and prioritized sweeping with small backups (specifically, the "reversed full backups" variant [Van Seijen and Sutton, 2013]). Prioritized sweeping converges to the same solution as value iteration, but can be much more computationally efficient by focusing updates on states where the Q–values change most significantly (particularly on environments with sparse transition matrices). Since learning could result in transition deletions as well as additions, our implementation of prioritized sweeping (Algorithm 4) includes a separate subroutine for deletions.

Given an observation history $o_{t-k:t}$, the agent follows an ϵ–greedy policy using the Q–values $Q(\bar{s}_t, a)$ in the lookup table for all possible actions a. For any pair $(\bar{s}_t, a)$ that has not yet been observed, $Q(\bar{s}_t, a)$ will simply be equal to its initialized value. In order to allow for fast generalization to state–action pairs that have not yet been experience, we therefore estimate the Q–value for new state–action pairs using its nearest neighbours in Hamming distance.

In the following, we simplify the discretized states $\bar{s}$ to s for clarity. We denote $\mathcal{S}$ as the set of all states corresponding to d–length binary strings, $\tilde{Q}(s, a)$ as the Q–value estimate used for action selection, and $Q(s, a)$ as the Q–value for a state–action pair in the lookup table as determined by prioritized sweeping. In order to calculate $\tilde{Q}(s_t, a)$ for a particular state–action pair, we first determine the Hamming distance m to the nearest neighbour(s) $s \in \mathcal{S}$ for which the action a has already been observed, i.e.

$$m = \min_{s\in\mathcal{S}}\{D(s_t, s)|N_{sa} > 0\}, \tag{6.10}$$

where $D(s_t, s)$ is the Hamming distance between s_t and s and N_{sa} denotes the number of times that action a has been taken from state s. We then define the set $\mathcal{S}_{tm}$ of all m–nearest neighbours to state s_t,

$$\mathcal{S}_{tm} = \{s \in \mathcal{S}|D(s_t, s) = m\}, \tag{6.11}$$

and the Q–value estimate used for action selection is then given by

$$\tilde{Q}(s_t, a) := \frac{\sum_{s\in\mathcal{S}_{tm}} N_{sa} Q(s, a)}{\sum_{s\in\mathcal{S}_{tm}} N_{sa}}. \tag{6.12}$$

If (s_t, a) has already been observed, then $m = 0$, $\mathcal{S}_{tm} = \{s_t\}$ and $\tilde{Q}(s_t, a) = Q(s_t, a)$. If $m = 1$, $\tilde{Q}(s_t, a)$ corresponds to an experience–weighted average over all states s with a Hamming distance of 1 from s_t, $m = 2$ to the average over neighbours with a Hamming distance of 2 etc. This Hamming neighbour estimate is parameter–less, and generally much faster than searching for nearest neighbours in continuous space. Note that Equation 6.12 is very similar in form to the instance–based estimate of an episodic controller (Equation 2.11, Gershman and Daw [2017]) where the estimate combines a similarity function based on Hamming distance with an experience weighting, providing a further link to episodic control approaches.

In addition, $\tilde{Q}(s_t, a)$ can be seen as the Q–value of an abstract aggregate state s_{tm} consisting of the m–nearest neighbours to s_t. To show this, we introduce the index set of past experiences $\mathcal{E}_{sa} = \{(\tau, \mu)|s^{\mu}_{\tau} = s, a^{\mu}_{\tau} = a\}$ that contains all the time indices τ for all episodes μ where action a was chosen in state s (taking into account all reassignments as described in Section 6.2.3 and in Algorithm 3). With the above definition of N_{sa} we see that $N_{sa} = |\mathcal{E}_{sa}|$, i.e. there are N_{sa} elements in the set $\mathcal{E}_{sa}$. With this and the update mechanism of prioritized sweeping

(Algorithm 4) we can write

$$Q(s,a) = \frac{1}{N_{sa}} \sum_{\tau,\mu \in \mathcal{E}_{sa}} r^{\mu}_{\tau} + \gamma \frac{1}{N_{sa}} \sum_{\tau,\mu \in \mathcal{E}_{sa}} V(s^{\mu}_{\tau+1}), \tag{6.13}$$

where $V(s) = \max_b \{Q(s,b) | N_{sb} > 0\}$. Substituting this into Equation 6.12, we obtain

$$\tilde{Q}(s_t,a) = \frac{\sum_{s \in \mathcal{S}_{tm}} \left[\sum_{\tau,\mu \in \mathcal{E}_{sa}} r^{\mu}_{\tau} + \gamma \sum_{\tau,\mu \in \mathcal{E}_{sa}} V(s^{\mu}_{\tau+1}) \right]}{\sum_{s \in \mathcal{S}_{tm}} N_{sa}}. \tag{6.14}$$

We now consider an aggregate state s_{tm} by treating all states $s \in \mathcal{S}_{tm}$ as equivalent, i.e. $\mathcal{E}_{s_{tm}a} = \{(\tau,\mu) | s^{\mu}_{\tau} \in \mathcal{S}_{tm}, a^{\mu}_{\tau} = a\}$. With this definition we get $\sum_{s \in \mathcal{S}_{tm}} \sum_{\tau,\mu \in \mathcal{E}_{sa}} = \sum_{\tau,\mu \in \mathcal{E}_{s_{tm}a}}$ and we obtain

$$\begin{aligned} \tilde{Q}(s_t,a) &= \frac{\left[\sum_{\tau,\mu \in \mathcal{E}_{s_{tm}a}} r^{\mu}_{\tau} + \gamma \sum_{\tau,\mu \in \mathcal{E}_{s_{tm}a}} V(s^{\mu}_{\tau+1}) \right]}{N_{s_{tm}a}} \\ &= Q(s_{tm},a), \end{aligned} \tag{6.15}$$

where we used Equation 6.13 to obtain the second equality.

6.2.5 Implementation details

The prioritized backups described by Van Seijen and Sutton [2013] are performed serially with environment exploration. To decrease training time and improve performance, we performed backups independently, and in parallel, to environment exploration and training the deep network.

We implemented state tabulation and prioritized sweeping as two separate processes (running on different CPU cores). The tabulation process (Algorithm 3) acts in the environment and trains the neural networks by sampling the replay memory. The sweeping process (Algorithm 4) maintains the transition table and continuously updates the Q-values using prioritized sweeping.

To perform greedy actions, the tabulation process requests Q–values from the sweeping process. To update the transition table, the tabulation process sends transition updates (additions and deletions) to the sweeping process. Our implementation of the sweeping process performed ~6000 backups/second, allowing the agent to rapidly propagate Q–value changes with little effect on the simulation time.

6.3 Results

We evaluated the VaST agent on a series of navigation tasks implemented in the *VizDoom* environment (see Figure 6.3A, Kempka et al. [2016]). Each input frame consists of a 3–channel [60 × 80] pixel image of the 3D environment, collected by the agent at a position (x, y) and

Algorithm 3 Variational State Tabulation.

```
Initialize replay memory $\mathcal{M}$ with capacity $\mathcal{N}$
Initialize sweeping table process $\mathcal{B}$ with transition add queue $\mathcal{Q}^+$ and delete queue $\mathcal{Q}^-$
for each episode do
    Set $t \leftarrow 0$
    Get initial observations $o_0$
    Process initial state $\bar{s}_0 \leftarrow \arg\max_s q_\phi(s|o_0)$
    Store memory $(o_0, \bar{s}_0)$ in $\mathcal{M}$
    while not terminal do
        Set $t \leftarrow t+1$
        Take action $a_t$ with $\epsilon$-greedy strategy based on $\tilde{Q}(s_{t-1}, a)$ from $\mathcal{B}$
        Receive $r_t$, $o_t$
        Process new state $\bar{s}_t \leftarrow \arg\max_s q_\phi(s|o_{t-k:t})$
        Store memory $(o_t, \bar{s}_t, a_t, r_t)$ in $\mathcal{M}$
        Put transition $(\bar{s}_{t-1}, a_t, r_t, \bar{s}_t)$ on $\mathcal{Q}^+$
        if training step then
            Set gradient list $\mathcal{G} \leftarrow \{\}$
            for sample in minibatch do
                Get $(o_{j-k-1:j}, a_j)$ from random episode and step $j$ in $\mathcal{M}$
                Process $q_\phi(s_{j-1}|o_{j-k-1:j-1})$, $q_\phi(s_j|o_{j-k:j})$ with encoder
                Sample $\hat{s}_{j-1}$, $\hat{s}_j \sim \hat{q}_\phi$ with temperature $\lambda$
                Process $p_\theta(o_j|\hat{s}_j)$, $p_\theta(\hat{s}_j|a_j, \hat{s}_{j-1})$ with decoder and transition network
                Append $\nabla_{\theta,\phi}\mathcal{F}(\theta, \phi; o_{j-k-1:j})$ to $\mathcal{G}$
                for $i$ in $\{j-1, j\}$ do
                    Process $\bar{s}_i^{new} \leftarrow \arg\max_s q_\phi(s|o_{i-k:i})$
                    Get $(\bar{s}_{i-1}, a_i, r_i, \bar{s}_i, a_{i+1}, r_{i+1}, \bar{s}_{i+1})$ from $\mathcal{M}$
                    if $\bar{s}_i \neq \bar{s}_i^{new}$ then
                        Put $(\bar{s}_{i-1}, a_i, r_i, \bar{s}_i)$, $(\bar{s}_i, a_{i+1}, r_{i+1}, \bar{s}_{i+1})$ on $\mathcal{Q}^-$
                        Put $(\bar{s}_{i-1}, a_i, r_i, \bar{s}_i^{new})$, $(\bar{s}_i^{new}, a_{i+1}, r_{i+1}, \bar{s}_{i+1})$ on $\mathcal{Q}^+$
                        Update $\bar{s}_i \leftarrow \bar{s}_i^{new}$ in $\mathcal{M}$
                    end if
                end for
            end for
            Perform a gradient descent step according to $\mathcal{G}$ with given optimizer
        end if
    end while
end for
```

Algorithm 4 Prioritized Sweeping Process.

```
Initialize $V(s) = U(s) = 0$ for all s          ▷ Discretized states $\bar{s}$ are simplified to $s$ for clarity
Initialize $Q(s,a) = 0$ for all s, a            ▷ Initial value is arbitrary; never used
Initialize $N_{sa}, N_{sa}^{s'} = 0$ for all $s$, $a$, $s'$
Initialize priority queue $\mathcal{P}$ with minimum priority cutoff $p_{min}$
Initialize add queue $\mathcal{Q}^+$ and delete queue $\mathcal{Q}^-$
while True do
    while $\mathcal{Q}^+$, $\mathcal{Q}^-$ empty do
        Remove top state $s'$ from $\mathcal{P}$
        $\Delta U \leftarrow V(s') - U(s')$
        $U(s') \leftarrow V(s')$
        for all $(s,a)$ pairs with $N_{sa}^{s'} > 0$ do
            $Q(s,a) \leftarrow Q(s,a) + \gamma N_{sa}^{s'} / N_{sa} \cdot \Delta U$
            $V(s) \leftarrow \max_b \{Q(s,b) | N_{sb} > 0\}$
            add/update $s$ in $\mathcal{P}$ with priority $|U(s) - V(s)|$ if $|U(s) - V(s)| > p_{min}$
        end for
    end while
    for $(s,a,r,s')$ in $\mathcal{Q}^+$ do
        $N_{sa} \leftarrow N_{sa} + 1$; $N_{sa}^{s'} \leftarrow N_{sa}^{s'} + 1$
        $Q(s,a) \leftarrow [Q(s,a)(N_{sa} - 1) + r + \gamma U(s')] / N_{sa}$
        $V(s) \leftarrow \max_b \{Q(s,b) | N_{sb} > 0\}$
        add/update $s$ in $\mathcal{P}$ with priority $|U(s) - V(s)|$ if $|U(s) - V(s)| > p_{min}$
    end for
    for $(s,a,r,s')$ in $\mathcal{Q}^-$ do
        $N_{sa} \leftarrow N_{sa} - 1$; $N_{sa}^{s'} \leftarrow N_{sa}^{s'} - 1$
        if $N_{sa} > 0$ then
            $Q(s,a) \leftarrow [Q(s,a)(N_{sa} + 1) - (r + \gamma U(s'))] / N_{sa}$
        else
            $Q(s,a) \leftarrow 0$
        end if
        if $\sum_b N_{sb} > 0$ then
            $V(s) \leftarrow \max_b \{Q(s,b) | N_{sb} > 0\}$
        else
            $V(s) \leftarrow 0$
        end if
        add/update $s$ in $\mathcal{P}$ with priority $|U(s) - V(s)|$ if $|U(s) - V(s)| > p_{min}$
    end for
end while
```

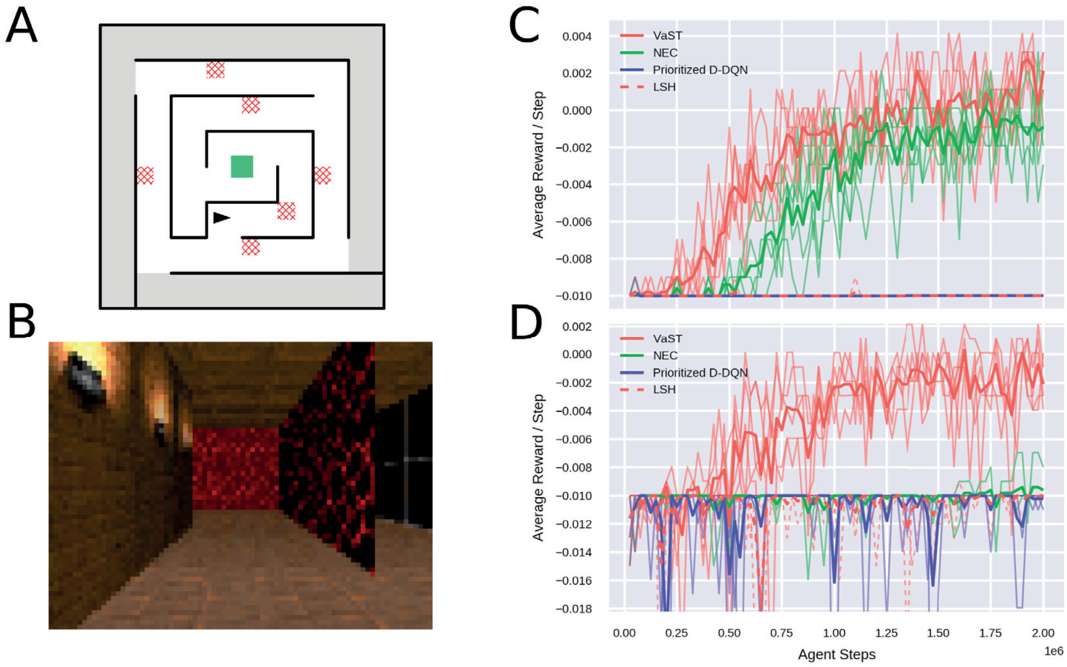

Figure 6.3 – **VaST learns quickly in complex mazes.** [**A**] The agent started at a random position and orientation in the outer rim of the 3D maze (highlighted in grey), and received a reward of +1 on reaching the center of the maze (highlighted in green), with a step penalty of -0.01. Red hatched areas correspond to the hazard regions in the second version of the task, where the agent received a penalty of -1 with a probability of 25%. We used a different texture for each wall in the maze, ending at a corner. The black arrow depicts an example position and orientation. [**B**] The observations for an agent positioned at the black arrow. [**C**] Performance comparison between models for 5 individual runs with different random seeds (mean in bold). Rewards are very sparse (≈ every 20 000 steps with a random policy); with longer training we expect DQN to improve. [**D**] Results for the second version of the task (including hazards).

orientation θ. The agent rarely observes the exact same frame from a previous episode (0.05% – 0.3% of the time in the mazes used here), making it ill–suited for a traditional tabular approach; yet the discovery of new transitions (particularly shortcuts) can have a significant effect on the global policy if leveraged by a model–based agent. We considered the relatively low–data regime (up to 2 million steps). Three actions were available to the agent: move forward, turn left and turn right; due to momentum in the game engine, these give rise to visually smooth trajectories. We also trained the agent on the Atari game Pong (Figure 6.9). For 3D navigation, we used only the current frame as input to the network; each wall in the maze was given a different visual texture, ending at a corner, in order to improve the observability of the task. We tested both 1– and 4–frame inputs for Pong in order to test whether VaST could use frame history to disambiguate states.

We compared the performance of VaST against two recently published sample–efficient model–free approaches: Neural Episodic Control (NEC) [Pritzel et al., 2017] and Prioritized Double–

DQN [Schaul et al., 2015]. We used the structure of the DQN network in [Mnih et al., 2015] for both NEC and Prioritized D–DQN as well as the encoder of VaST (excluding the output layers). Full hyperparameters are given in Appendix A. NEC is similar to both VaST and MFEC in that it is semi–tabular, but unlike MFEC, it can be used in non–deterministic environments. It is an n–step method, unlike VaST and Prioritized DQN, which are both entirely off–policy.

We also compared against prioritized sweeping using Locality Sensitive Hashing (LSH) with random projections [Charikar, 2002], where each bit $\bar{s}_{t,i} = H(v_i \cdot o_t)$, and each fixed projection vector v_i had elements sampled from $\mathcal{N}(0,1)$ at the beginning of training. The environment model and Q–values were determined as with VaST.

In the first task (Figure 6.3), the agents were trained to reach a reward of +1 in the center of a complex maze, starting from a random position and orientation in the outer region. In a second version of the task, we added six “hazard” regions which gave a penalty of -1 with a probability of 25% for each step. The agents were evaluated over a 1000–step test epoch, with $\epsilon = 0.05$, every 25 000 steps. VaST slightly outperformed NEC on the first version of the task and significantly outperformed all of the other models on the more difficult version (Figure 6.3C).

6.3.1 Dimensionality of the latent representation

We used $d = 32$ latent dimensions for the VaST agent in the navigation tasks, corresponding to a 32–bit representation of the environment. We examine the effect of d in Figure 6.4, for both a large maze and a small maze. High–dimensional representations ($d = 64$) tended to plateau at lower performance than representations with $d = 32$, but also resulted in faster initial learning in the larger maze. The agent frequently revisited state–action pairs even using the high dimensional representation (Figure 6.4B). In general, we found that we could achieve similar performance with a wide range of dimensionalities, though the agent was clearly more limited by the dimensionality in the larger, more complex maze.

6.3.2 Visualizing the abstraction

After training an agent in the maze task without hazards, we ran the agent for an additional 100 000 steps in the environment and recorded the position and orientation at each step, as well as the latent discrete state that the observations were assigned to. We plot the binary state assignments along six different dimensions in Figure 6.5.

As the abstract state is conditioned on the observations, we expect the state assignments to be dominated by the visual input; for instance, visual features change slowly as the agent moves in a particular direction along one side of a hallway. The binary states generally reflect this. In some cases, the binary assignment changes almost exclusively in a particular region of the environment (e.g. Figure 6.5 bottom left).

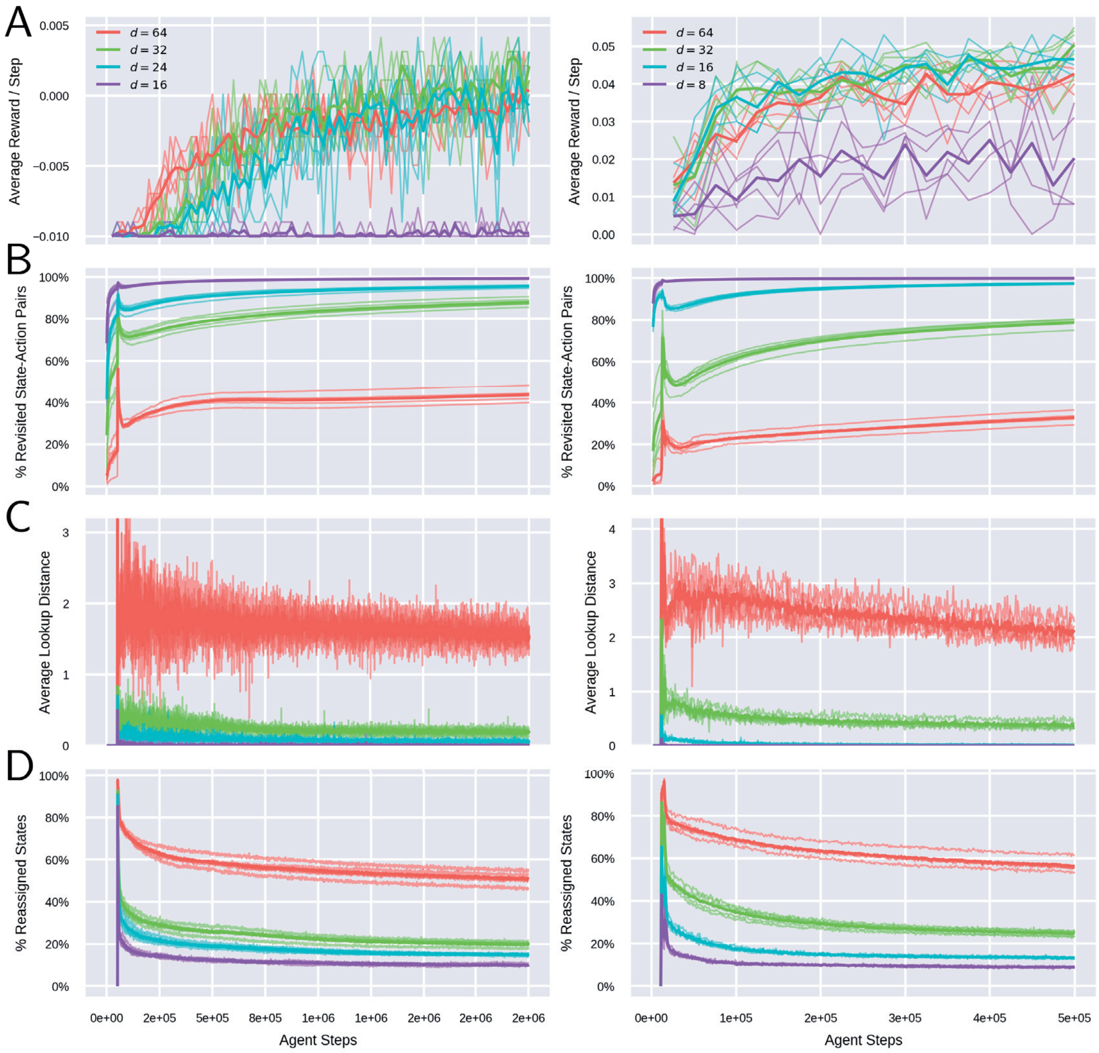

Figure 6.4 – **Effect of latent dimensionality.** The left column corresponds to a large maze (Figure 6.3C) and the right column to a small maze (Figure 6.7 with stationary rewards). [**A**] Average reward. [**B**] Cumulative percentage of revisited state–action pairs over the course of training. The sharp transition at 50 000 steps (10 000 steps in the small maze) corresponds to the beginning of network training. [**C**] The average lookup distance m as a function of time. [**D**] The average percentage of observations from a minibatch that were reassigned to a different state during training.

6.3.3 Sample efficiency

We hypothesized that the VaST agent would be particularly adept at rapidly modifying its policy in response to one new experience. To test this, we designed an experiment in a 3D H–maze (Figure 6.6) that requires the agent to leverage a single experience of a new shortcut. The agent learned to run towards a terminal reward zone (+1) while avoiding a dead end and a terminal penalty zone (-1), with a step penalty of -0.01. After 400 000 steps of training

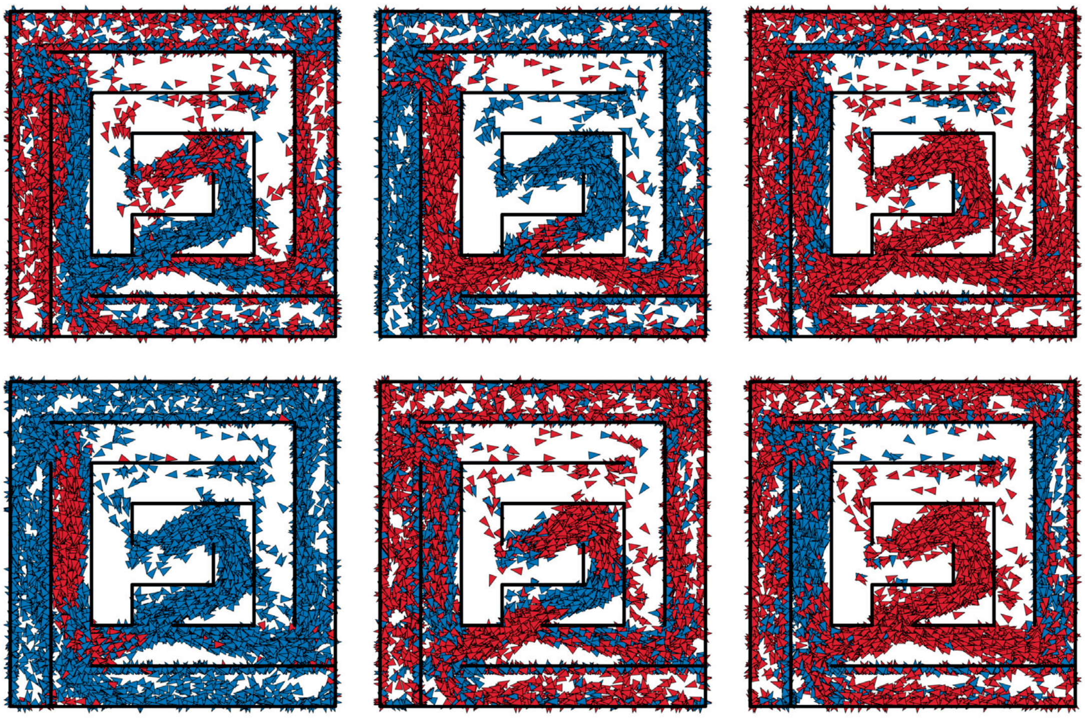

Figure 6.5 – **Binary latent state assignment along different dimensions.** The agent explored the environment for 100 000 time steps after the end of training, with $\epsilon = 0.1$. In each plot, we choose 10 000 times steps t out of those 100 000 steps at random, and plot the position and orientation of each step t as an arrow. Each plot corresponds to a different latent dimension i, where the arrows are shaded according to $\bar{s}_{t,i}$ ($\bar{s}_{t,i} = 0$ is shown in blue and $\bar{s}_{t,i} = 1$ in red).

(when the policy had nearly converged) we introduced a small change to the environment: running into the dead end would cause the agent to teleport to a position close to the reward zone, allowing it to reach the reward much faster. We informed the agent of the teleporter using a single forced run episode, in which the agent collected observations while running from the start box, through the teleporter, to either the reward zone or penalty zone under a fixed, predetermined policy. For the VaST agent, this corresponds to a single experience indicating a new shortcut: the transition between the states before and after the teleporter. After observing either the rewarded or penalized episode, performance rapidly improved as the agent adapted its policy to using the teleporter; in contrast, the agent discovered the teleporter on only 2/5 random seeds without the forced run (Figure 6.6B). The agents switched to using the teleporter regularly approximately 20 000 steps after the forced run, on average (about 160 episodes). The VaST agent adapted to the teleporter more effectively than any of the other models (Figure 6.6C and D).

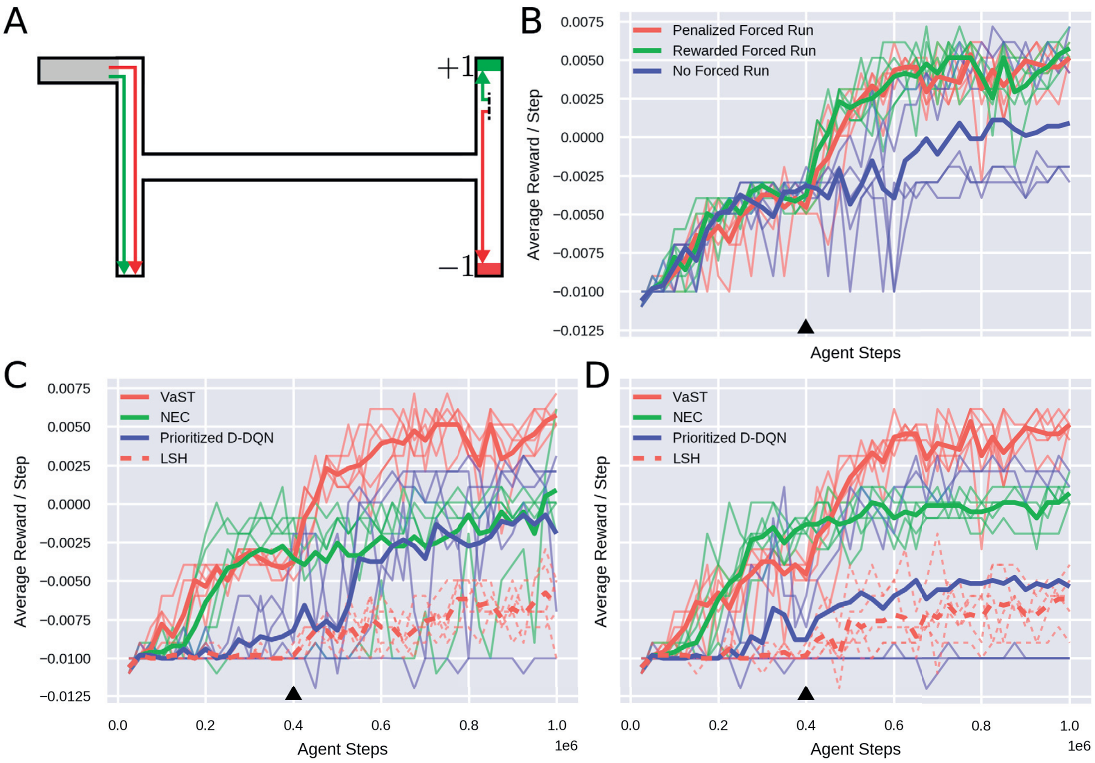

Figure 6.6 – **VaST allows for rapid policy changes in response to single experiences.** [**A**] The agent learned to run from the starting area (grey) to a reward zone (green). After training, a new shortcut (teleporter) was introduced at the bottom of the left arm. The agent either observed no forced run, or a single forced run through the teleporter ending either in the rewarding (green) or the penalizing (red) terminal zone. The forced runs were 58 and 72 steps in length, respectively. [**B**] The teleporter was introduced after 400 000 steps (black triangle). The VaST agent's performance is shown for the three conditions: no forced run, rewarded forced run and penalized forced run. [**C**] Model performance comparison for rewarded forced runs. [**D**] Model performance comparison for penalized forced runs.

6.3.4 Transfer learning: non–stationary rewards

VaST maintains a model of transition probabilities in the environment, separate from immediate rewards or Q–values. If the rewards were suddenly modified, we hypothesized that the existing transition model could allow the agent to rapidly adjust its policy (after collecting enough data to determine that the expected immediate rewards had changed).

We tested this in the 3D maze shown in Figure 6.7A (inset, with example observations show in Figure 6.2). Starting at a random position, the episode terminated at the end of any arm of the maze; the agent received a reward of +1 at the end of horizontal arms, and a penalty of -1 at the end of vertical arms. The reward positions were reversed after 200 000 steps. We used two replay memory sizes ($\mathcal{N}$ = 100 000 and $\mathcal{N}$ = 500 000). Compared to NEC and Prioritized D–DQN, VaST both learned quickly in the initial phase and recovered quickly when

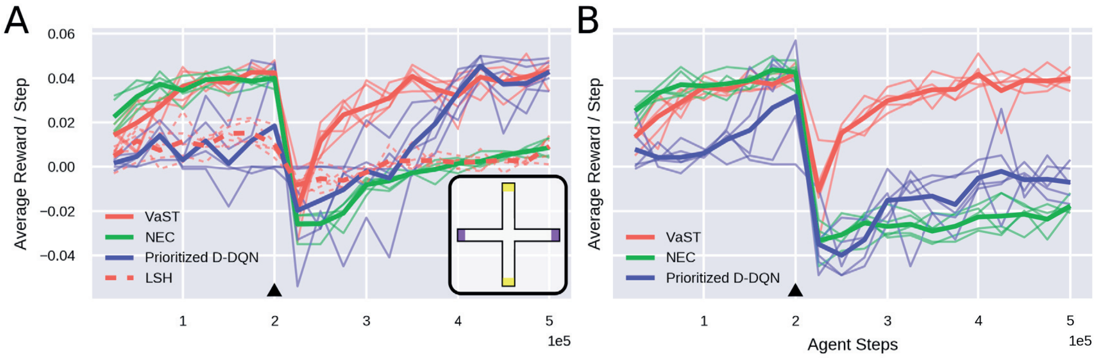

Figure 6.7 – **VaST can adapt to changing rewards.** [**A, Inset**] The maze environment. Horizontal arms (purple) initially yielded a reward of +1 while vertical arms (yellow) yielded a penalty of -1. [**A**] After training for 200 000 steps (black triangle), the rewards and penalties in the maze were reversed. All agents used a replay memory size of $\mathcal{N}$ = 100 000 transitions. [**B**] The same task with a replay memory size of $\mathcal{N}$ = 500 000.

the rewards were reversed. While both NEC and Prioritized D–DQN adapted faster with a smaller replay memory, VaST performed similarly in both conditions.

6.3.5 Transfer learning: multiple tasks with a shared network

The results in Figure 6.7 suggest that VaST could leverage shared structure across multiple parallel tasks in the same environment, or similar environments, in order to learn them using the same network model. In addition, we hypothesized that training the network on multiple tasks would allow the agent to learn each task faster. To test this, we reused the same maze but with four different reward landscapes (Figure 6.8A) corresponding to four separate tasks. The specific task was selected at random at the beginning of each episode, and signaled to the agent from the environment with an index variable (i.e. “Task 1”, “Task 2”, “Task 3” or “Task 4”).

In order to learn the tasks in parallel, we created a version of the agent that could launch multiple prioritized sweeping processes (one for each task). At the beginning of each episode, the agent selects the process corresponding to the signaled task index. It then sends observed transitions to that process, and likewise requests Q–values from that process. In addition, the task index of each observed transition is stored in the replay memory, and used to update the correct process when the state assignments changed. All prioritized sweeping processes continually update Q–values in parallel; as a result, the impact to overall simulation time is minimal.

The agent explored the environment and trained a common network model across all 4 tasks. We trained 5 multi–task agents with different random seeds. As a baseline, we compared the performance of the multi–task agents to 20 single–task agents (5 agents with different random seeds trained on one of the 4 tasks exclusively). The results are shown in Figure 6.8, plotted

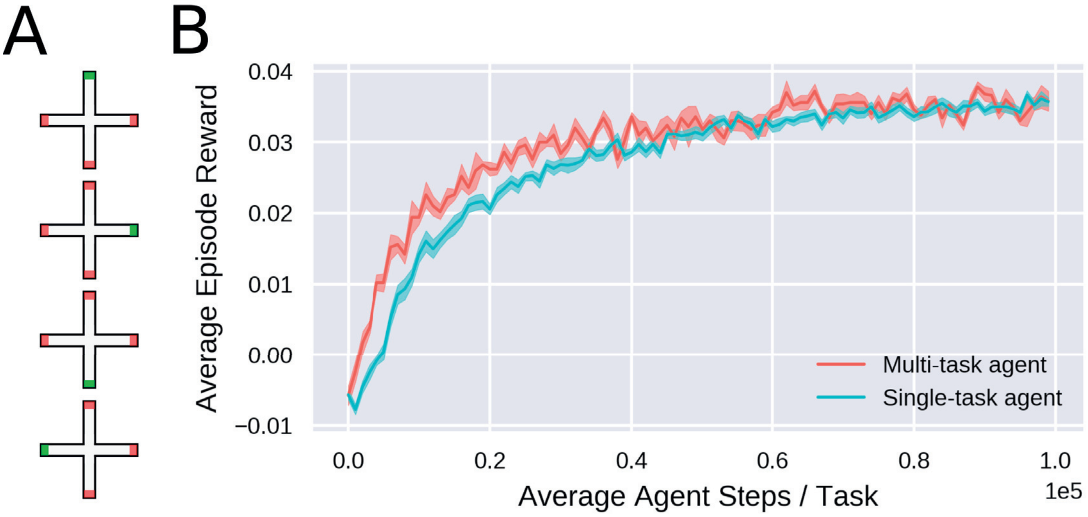

Figure 6.8 – **VaST enables transfer learning across tasks in an environment.** [**A**] The multi–task agent was trained to learn four different tasks corresponding to differing rewarded arms, all in the same environment (reaching the end of an arm always terminated the episode). The task was selected randomly at the beginning of each episode and signaled to the agent. [**B**] The multi–task agent's performance (red) compared to an agent trained on only one of the four tasks (blue), ± SEM. The horizontal axis denotes the average number of steps on each task; e.g. at 1000 steps on the axis, a multi–task agent has taken 4000 steps total across the four task conditions.

as a function of the average steps taken on each task. To ensure a fair comparison, we fixed the exploration rate to $\epsilon = 0.1$ without annealing, and plot performance from the beginning of network training (after initializing each agent's replay memory with 1000 exploratory steps). We plot the training performance every 1000 steps.

Despite sharing a single network for all four tasks, the multi–task agent learned faster than the single–task agent. Since the agent did not share any tabular transition statistics across tasks in the prioritized sweeping processes, the transfer learning can be attributed to training the deep network on multiple tasks with the same environment structure.

6.3.6 Training on Atari: Pong

In addition to 3D navigation, we trained the VaST agent to play the Atari game Pong using the Arcade Learning Environment [Bellemare et al., 2013], with preprocessing steps taken from Mnih et al. [2013]. In Pong, a table tennis–like game played against the computer, the direction of the ball's movement is typically unclear given only the current frame as input; as a result, the state is only partially observable given the current frame. We therefore tried conditioning the posterior distribution q_ϕ on either the current frame ($k = 0$) or the current frame along with the last 3 frames of input ($k = 3$, following Mnih et al. [2013]), in order to test

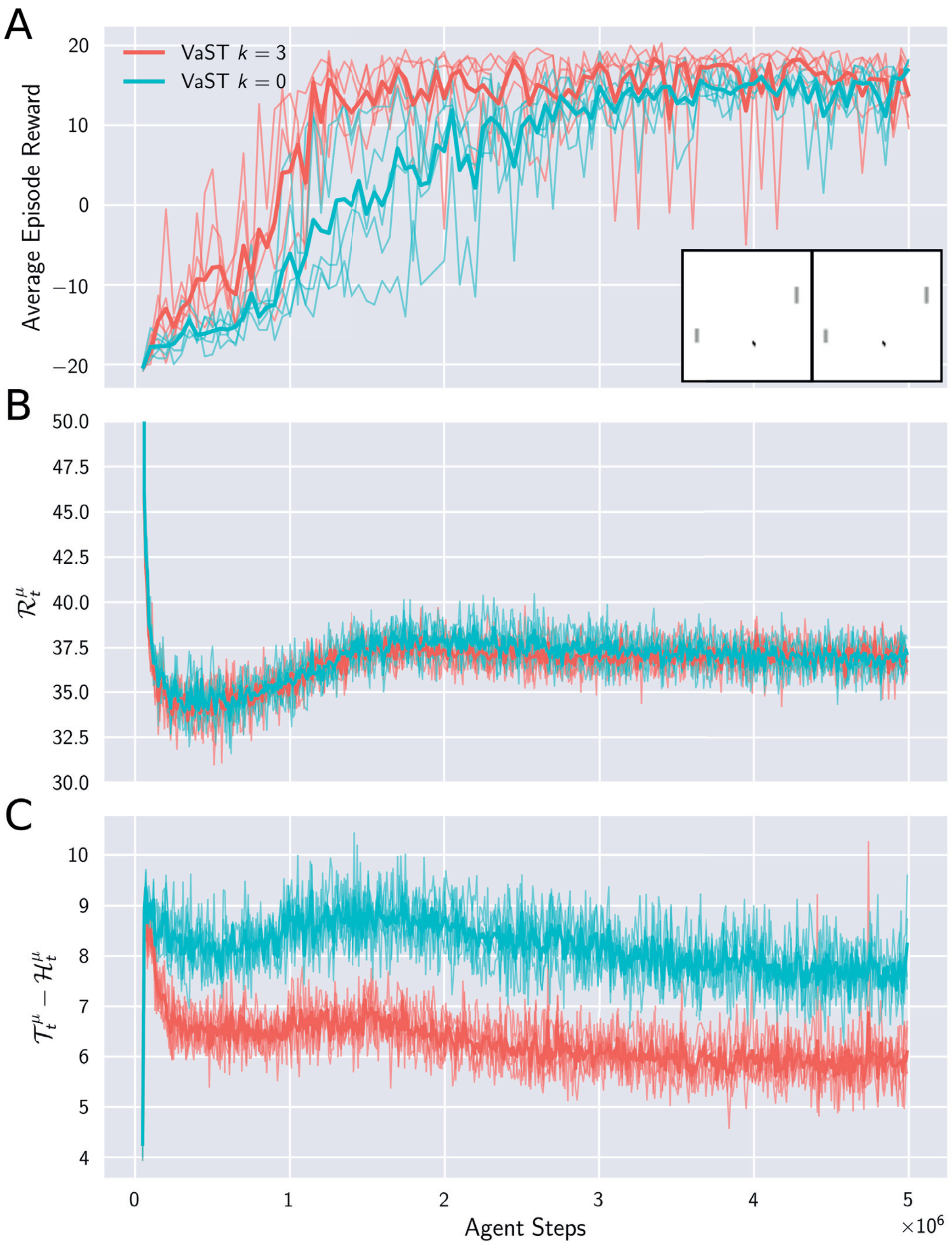

Figure 6.9 – **Learning to play Pong with variable frame history.** [**A**] Test epoch episode rewards for VaST trained over 5 million steps (episode rewards in Pong range from −21 to +21). We tested performance with no frame history ($k = 0$) and with 3 frames of history ($k = 3$) as input to the encoder q_ϕ. [**A, Inset**] Actual observations o_t (left) and reconstructed observations $\mathbb{E}_p[o_t|\hat{s}_t]$ (right) for a trained agent. [**B**] The reconstruction cost component of the free energy cost function, for the two values of k. [**C**] The additional terms of the free energy cost function (transition and entropy).

whether VaST could use the additional frame history to mitigate the partially observability of the task. Using $k = 3$, the performance converged significantly faster on average (Figure 6.9). While the reconstruction cost was the same for $k = 0$ and $k = 3$, the transition and entropy cost terms decreased with additional frame history, showing that the additional frames allowed VaST to disambiguate the transition structure of the task.

6.4 Related work

Model-based reinforcement learning Prioritized sweeping with small backups [Van Seijen and Sutton, 2013] is usually more efficient than but similar to Dyna-Q [Sutton and Barto, 2018], where a model is learned and leveraged to update Q-values. Prioritized sweeping and Dyna-Q are background planning methods [Sutton and Barto, 2018], in that the action selection policy depends on Q-values that are updated in the background independent of the current state. In contrast, methods that rely on planning at decision time (like Monte Carlo Tree Search) estimate Q-values by expanding the decision tree from the current state up to a certain depth and using the values of the leaf nodes. Both background and decision–time planning methods for model-based reinforcement learning are well studied in tabular environments [Sutton and Barto, 2018]. Together with function approximation, usually used to deal with high–dimensional raw (pixel) input, many recent works have focused on planning at decision time. Oh et al. [2017] and Farquhar et al. [2017], extending the predictron [Silver et al., 2017b], train both an encoder neural network and an action–dependent transition network on the abstract states used to run rollouts up to a certain depth. Racanière et al. [2017] and Nagabandi et al. [2017] train a transition network on the observations directly. Racanière et al. [2017] additionally train a rollout policy, the rollout encoding and an output policy that aggregates different rollouts and a model-free policy. Planning at decision time is advantageous in situations like playing board games [Silver et al., 2017a], where the transition model is perfectly known, many states are visited only once and a full tabulation puts high demands on memory. Conversely, background planning has the advantage of little computational cost at decision time, almost no planning cost in well–explored stationary environments and efficient policy updates after minor environment changes.

Exploration The most similar network model to VaST was proposed by Tang et al. [2017], for the purpose of count–based exploration. The model used an autoencoder with a noisy sigmoidal hidden layer, where sigmoidal activations were rounded to produce a binary code; the binary code was then used to estimate the number of times an agent had visited a state before. The visitation count determined an exploration bonus that was added to the reward; however, a different RL algorithm was used to determine the actual policy. In addition, the network model differed from ours in the structure of the loss function, the lack of a learned transition model (either in network weights or in a table), and the lack of a variational Bayesian motivation as in VaST.

Successor representations for transfer learning The hybrid model–based/model–free approach of successor representations has recently been transferred from the tabular domain to deep function approximation [Dayan, 1993, Kulkarni et al., 2016]. Learning separate models of the immediate rewards and the discounted multi–step future occupancy allows for transfer between tasks. However, the future occupancy is learned under a given policy induced by the reward landscape, and the optimal policy will generally change with new rewards. The ability to generalize between tasks in an environment (as VaST does in Figure 6.7 and Figure 6.8) then depends on the similarity between the existing reward landscape (or multiple existing reward landscapes [Barreto et al., 2016a]) and the new reward landscape. Recent work has proposed updating successor representations offline in a Dyna–like fashion using a transition model [Russek et al., 2017, Peng and Williams, 1993]; we expect that prioritized sweeping with small backups could also be adapted to efficiently update tabular successor representations.

Navigation tasks Many methods have been introduced focusing on general spatial navigation tasks by incorporating specialized domain knowledge. These methods are often collectively referred to under the umbrella term Simultaneously Localization and Mapping (SLAM) [Thrun, 2007, Cadena et al., 2016], comprising techniques including extended Kalman filtering and particle filtering. They generally exploit the assumption that the agent is navigating in a 2D spatial environment, and therefore that the true latent space is continuous and three–dimensional (2D spatial dimensions x and y combined with the agent's orientation θ). Under this assumption, the agent begins with a strong model of the relationship between an action and the corresponding displacement in the latent space. SLAM algorithms typically exploit "loop closure" (i.e. recognition of landmarks previously visited on the current trajectory) to build maps and find shortcuts [Cadena et al., 2016].

Mirowski et al. [2016] develop an RL agent for 3D navigation trained using auxiliary tasks based on loop closure prediction and depth prediction to improve the function approximator. Jaderberg et al. [2016] similarly train an RL agent on spatial navigation tasks using unsupervised auxiliary losses not specific to navigation, including reward prediction. Finally, several recent models [Bhatti et al., 2016, Parisotto and Salakhutdinov, 2017, Gupta et al., 2017] augment reinforcement learning algorithms and planners with internal 2D maps to improve performance on navigation problems.

Even though we demonstrate and evaluate our method mostly on navigation tasks, VaST does not leverage any domain knowledge for spatial navigation problems. The inclusion of auxiliary tasks like depth prediction and loop closure prediction, as well as inductive biases regarding the structure of the latent space and the transition model, are likely to further improve performance on spatial navigation tasks (particularly when combined with features like recurrent memory cells to overcome partial observability). The addition of further unsupervised losses not specific to navigation [Jaderberg et al., 2016] is also promising.

State abstraction in reinforcement learning State abstraction has a long history in reinforcement learning [Sutton and Barto, 2018]. Li et al. [2006] and Hutter [2014] consider the conditions under which the abstraction can be used to learn a successful policy. Li et al. [2006] describe a hierarchy of abstractions, and show that several types within the hierarchy result in an abstract MDP which allows Q–learning to converge to an optimal policy that is also optimal in the ground MDP. Without specific constraints on the abstraction, Q–learning in the abstract MDP can result in suboptimal policies [Singh et al., 1994].

The most exact abstraction allowing for an optimal policy is based on bisimulation: it requires that all ground states within an abstract state share the same action–conditional abstract transition probabilities and expected action–conditional immediate rewards [Li et al., 2006]. Notably, VaST is not trained to reconstruct immediate reward statistics from the abstract states; adding this additional loss may therefore improve performance by mitigating model error, at the cost of imposing reward landscape dependence on the latent representation. Both Li et al. [2006] and Hutter [2014] further show that much more efficient abstractions than bisimulation can result in optimal policies, as long as Q–value estimates are consistent within the abstract/aggregate states. Several abstraction algorithms based on aggregating states with equivalent Q–values [Chapman and Kaelbling, 1991, McCallum, 1995] have been proposed; in particular, Chapman and Kaelbling [1991] consider statistical tests on factored binary state representations. A variant of VaST based on reconstructing Q–values is therefore an interesting future direction (see Discussion and Chapter 8). Coarser state aggregations based on equivalent best actions [Jong and Stone, 2005] and similar Bellman residuals [Bertsekas and Castanon, 1989] have also been proposed. State aggregation has further been considered in the domain of Monte Carlo Tree Search [Hostetler et al., 2014].

To our knowledge, VaST is the first approach that uses modern deep learning methods to learn useful and non-linear state discretization for determining a value function. In earlier versions of our model we tried discretizing with standard VAEs (i.e. with a fixed prior distribution and no transition network), with mixed success. The state aggregator $q_\phi(s_t|o_{t-k:t})$ of VaST can be seen as a byproduct of training a hybrid ANN–HMM. Different methods to train ANN-HMMs have been studied [Bengio et al., 1992, Ng et al., 2016, Maddison et al., 2016, Tucker et al., 2017]. While none of these works study the binary representation of the latent states used by VaST for the generalization of Q–values, we believe it is worthwhile to explore other training procedures and potentially draw inspiration from the ANN-HMM literature.

Dimensionality reduction for RL State abstraction based on dimensionality reduction (including the use of autoencoders and variational autoencoders) has a rich history; see Section 5.5.2 for more details.

6.5 Discussion

We found that the VaST agent could rapidly transform its policy based on limited new information and generalize between tasks in the same environment. In stationary problems, VaST performed better than competing models in complex 3D tasks where shortcut discovery plays a significant role. Notably, VaST performs latent learning: it builds a model of the structure of the environment even when not experiencing rewards [Tolman and Honzik, 1930].

We also trained VaST to play the Atari game Pong. In general, we had less initial success training the agent on other Atari games. We suspect that many Atari games resemble deterministic tree Markov Decision Processes, where each state has exactly one predecessor state. In these tasks, prioritized sweeping conveys no benefit beyond MFEC [Brea, 2017]. In contrast, tasks like 3D navigation can be well–characterized by a non–treelike tabular representation (e.g. by using a discretization of (x, y, θ)).

VaST differs from many deep reinforcement learning models in that the neural network is entirely reward–agnostic, where training corresponds to an unsupervised learning task. The reward–agnostic network combined with a parallelizable prioritized sweeping algorithm allows VaST to naturally extend to learning multiple tasks in the same environment; we similarly expect that the model can leverage learning across tasks in differing environments with similar observations and transition statistics.

Many other possible architectures exist for the unsupervised tabulator; for instance, a Score Function Estimator such as NVIL [Mnih and Gregor, 2014, Mnih and Rezende, 2016, Tucker et al., 2017] could be used in place of the Con–crete relaxation for discrete stochastic sampling. In addition, while we chose here to show the strengths of a purely model–based approach, one could also consider alternative models that use value information for tabulation, resulting in hybrid model–based/model–free architectures.

VaST naturally maintains statistics on state–action visitation counts, which could be employed for a tabular count–based exploration approach like MBIE–EB [Strehl and Littman, 2008]. Given that a similar model to VaST has recently been proposed for the purpose of count–based exploration [Tang et al., 2017], this remains a promising avenue for research.

The past decade has seen considerable efforts towards using deep networks to adapt tabular RL techniques to high–dimensional and continuous environments. In this chapter, we showed how the opposite approach – using deep networks to instead transform the environment into a tabular one – can enable the use of powerful model–based techniques.

7 Discussion & Conclusions

7.1 Model–based control, episodic control and the hippocampus

While my thesis primarily investigated model–based approaches, I also found techniques associated with episodic control to be of particular interest. The use of lookup tables in Chapter 5 and Chapter 6 was inspired by recent results showing impressive sample efficiency with episodic controllers [Lengyel and Dayan, 2007, Blundell et al., 2016]. In addition, VaST was motivated by the observation that MFEC corresponds to a simplified version of prioritized sweeping [Brea, 2017], appropriate for deterministic tree MDPs. With VaST, we sought to bring the additional sample efficiency of prioritized sweeping to continuous and high–dimensional environments where states are never revisited.

On a broader scope, the analysis by Brea [2017] blurs the line between episodic control and model–based methods, and brings into question when and whether pure episodic control is a better approach. Notably, episodic control was originally motivated using tree MDPs and compared with a particularly weak model–based controller, which relied on dynamic programming without caching to determine Q–values at each time step [Lengyel and Dayan, 2007]. In contrast, Brea [2017] shows that model–based prioritized sweeping has equivalent sample efficiency to MFEC on deterministic tree MDPs and better sample efficiency on general MDPs. MFEC's primary advantage is then in space complexity: it does not require the memory to maintain a transition model longer than the length of an episode [1].

Gershman and Daw [2017] consider the key features of episodic controllers to be a non–parametric memory and a kernel– or instance–based generalization method (such as kNN) for approximating the value of new states. While prioritized sweeping corresponds to a parametric approach in tabular environments, it does not explicitly require the state space to have a fixed size, and can therefore be applied in a non–parametric fashion. As well, any generalization method associated with episodic control can be directly applied alongside prioritized sweeping in non–tabular environments.

---

[1] Although: Brea [2017] also proposes a version of prioritized sweeping where the transition model is reset after every episode, reducing memory requirements while leveraging revisited states within an episode.

The offline episodic controller in Chapter 5 combines a non–parametric memory and a nearest–neighbour lookup with a model–based simulator. VaST also arguably leverages both of these features attributed to episodic controllers, despite the fact that we consider it to be a purely model–based controller. While the VaST lookup table has a fixed number of possible states (2^d) as in a parametric approach, we grew the state table online under the assumption that the agent will visit only a small number of possible states, as in a non–parametric approach. As well, we used a nearest–neighbour algorithm based on Hamming distance to estimate the value of new states from previously observed states.

Any difference between the controllers is particularly relevant to ongoing questions about the role of the hippocampus in RL. Gershman and Daw [2017] note the dichotomy between characteristics of model–based control in the hippocampus (e.g. map–like place cell coding and preplay and replay phenomena similar to Dyna–like simulation), and the established role of the hippocampus in episodic memory. The authors propose that the hippocampus is primarily an episodic controller, observing that many signatures of episodic control can be confused for those of model–based control (for instance, as I discussed above, episodic memories and statistical world models are indistinguishable when states are never revisited).

In their proposal, a key function of the hippocampus is learning useful kernel functions for encoding episodic memories. As an example, they point out that geodesic place cell coding allows for storing snapshot–like information about the animal's location in a manner that is useful for navigation, consistent with the attractor network model in Chapter 3 (and broadly consistent with the encoders in Chapter 5 and Chapter 6). If the kernel function is trained using a transition model, however, then the animal may also have the ability to simulate episodes in latent space, as in the offline episodic controller. Notably, the observation of place cell sequences that trace out novel, never–before–experienced paths through the environment is difficult to explain in an episodic controller without an environment model [Gupta et al., 2010, Pfeiffer and Foster, 2013, Brea, 2017].

In Section 2.4.1, I noted how Dyna methods smoothly interpolate between model–free and model–based approaches as a function of the number of background sweeps performed on each step, with a tradeoff between sample efficiency and time complexity [Sutton and Barto, 2018]. Similarly, prioritized sweeping can be seen as approximately interpolating between model–based control and episodic control as a function of the amount of transition information stored beyond the current episode. In this case, the tradeoff is between sample efficiency and space complexity. It remains an interesting avenue of research to consider an agent where both background sweeps and transition storage are parameterized, moving between model–free, model–based and episodic control modes.

7.2 Conclusions

Throughout this thesis, I have considered how environment models allow an agent to learn quickly (i.e. with high sample efficiency) and reuse learned structure between different tasks (i.e. task transfer). These models include the attractor networks of Chapter 3 and Chapter 4, which encode a low–dimensional representation of an environment for rapid path–planning, as well as the generative networks of Chapter 5 and Chapter 6, which similarly learn a low–dimensional representation of the environment in order to quickly update a value function in the background.

Task transfer obviously becomes critical when the task dynamics are non–stationary. As discussed in Chapter 5, this is also when sample efficiency is most important: an agent seeking to maximize total cumulative reward should favour fast learning over asymptotic performance if the task dynamics are unlikely to remain fixed for very long. Non–stationary tasks, then, are an important area of study for model–based methods.

In Chapter 5 and Chapter 6, we addressed sample efficiency in part by using a a lookup table. Lookup tables can be updated quickly compared to training a network to approximate a value function with gradient descent; the table entries can also be rewritten rapidly if the task changes, bypassing the problem of training a network to approximate a non–stationary optimal value function.

The lookup tables were paired with deep networks trained using SGD to learn the environment statistics, which were consistent across tasks. The trained environment model allows the lookup table to be exploited more efficiently, by compressing the agent's observations in an environment–specific manner before storing them in the table. The results suggests that pairing network–based "slow" learning with tabular "fast" learning can leverage the best of both approaches to address task transfer, sample efficiency and ultimately task non–stationarity.

The attractor network models presented operate on a similar principle: the slowly–learned network generates a representation of the animal's position that doubles as a representation of the path to that position. This preprocessing enables fast "one–shot" learning of a path to a goal location by reactivating the representation of the goal location. Without the preprocessing step, storing a snapshot of the goal location (using e.g. the visual features at the goal location) is much less useful for planning.

Overall, I found two key methods – prioritized sweeping, and the use of lookup tables (or non–parametric memory) with model–based planning – to be particularly effective and under–explored in both computational neuroscience and deep reinforcement learning. I believe that the two together can enable RL algorithms that learn and adapt to the environment quickly, efficiently and intelligently.

8 Future Work

Here, I outline several worthwhile lines in inquiry based on the models and results described in Chapter 5 and Chapter 6.

8.1 Offline Episodic Control

Surprise–based change detection In complex tasks where the reward is stochastic or the agent rarely revisits a state exactly more than once, the reward change detection rule we used will not scale. As an alternative, a non–parametric version of Bayesian surprise [Itti and Baldi, 2009] or confidence–corrected surprise [Faraji et al., 2018] could be employed with a threshold determining when to reset the lookup table. Note that surprise–based change detection and reset is primarily useful in settings where tasks change but never reoccur, whereas multi–task learning (as employed with VaST) applies when the environment rotates between a small number of tasks.

Alternative objective function We trained the model to map the input observations to a Normal distribution in latent space, while learning a transition model within that latent space. While the Normal prior is useful for sampling initial states for offline trajectories, we found that sampling initial states from a replay memory was almost as effective (Figure 5.4). Removing the Normal distribution prior, the transition model can be naturally incorporated into a variational objective function based on an MDP task structure (as in the VaST model). I suspect this would improve the model's performance.

8.2 VaST

Value–based tabulation Complex MDPs like Atari games often include many features in the observation space that are irrelevant to the policy. For instance, many Atari games include the player's current score at the top of the screen. The score is generally only weakly associated with the state's value and irrelevant in determining the optimal action. Since VaST does not use any reward information in learning the state representation, it will usually learn to represent each possible score as a different state, preventing any generalization across states that are equivalent from a policy perspective. This is less of a problem in 3D navigation, where the observations can be fully determined by three nearly independent variables (the 2D coordinates plus orientation, all of which are relevant to the agent's policy).

Improving VaST's performance on Atari–like environments therefore requires training it to generalize across policy–irrelevant features. We are currently exploring several methods for doing this. In particular, one approach is to predict bootstrapped Q–value estimates in a DQN–like manner from the sampled encoder output s_t. Although these bootstrapped Q–values are learned slowly, they can ideally extract the features of the observation space relevant for predicting the state value early in training. Given a low–dimensional latent embedding, this forces the encoder to prioritize task–relevant features over task–irrelevant features in the binary state representation, generalizing over regions of the state space that are not important for predicting the state value. Early experiments on this approach have been promising. Alternatively, it may be possible to train the network on the tabular Q–value estimates directly, similar to NEC [Pritzel et al., 2017].

Multi–task learning In Section 6.3.5, we showed how VaST could use a common network to transfer learning between tasks in the same environment. This is a particularly conservative form of task transfer; another possibility is to share tabular transition statistics between the tasks as well, differing only in tabular state reward statistics.

Going further, we can ask whether the agent can learn to tell the difference between tasks in an environment when the task is changing without notice. For instance, Doya et al. [2002] propose an EM–like mixture–of–experts approach for learning multiple task models in a tabular environment, where the agent accumulates evidence over time to determine a probability distribution over the current task. This approach could be applied to VaST, allowing the agent to rapidly adapt to unpredictably changing reward landscapes, and potentially to changing environments with similar visual statistics.

Notably, multi–task learning illustrates the downside to value–based tabulation: the network is most generalizable when it makes the fewest assumptions about the reward structure.

Partially observable environments The 3D environments we used to train VaST were almost fully observable, due to the use of a different texture on each wall. However, the results on Pong show that the agent can learn to accumulate information over multiple frames in order to decipher hidden state information.

More generally, we note that the model can be easily conditioned on the full episode observation history by using a recurrent neural network like a Long Short–Term Memory (LSTM) layer [Hochreiter and Schmidhuber, 1997] in the encoder. Given a partially observable environment, the transition network pressures the model to learn a Markovian latent state structure, where observation history in the LSTM layer memory cells would allow the network to differentiate between aliased states. The most significant challenge here is in training the network on extended sequences of observations sampled from the replay memory. Existing approaches either sample full episodes, or sample subsequences and reset the LSTM hidden state to an arbitrary value at the beginning of every subsequence [Hausknecht and Stone, 2015]. A potential alternative approach is to store the LSTM's hidden state in the replay memory, and use it to bootstrap the recurrent network state at the beginning of a sampled subsequence.

Exploration The most similar network model to VaST that we discovered was used to implement tabular count–based exploration [Tang et al., 2017]. It therefore remains a promising direction to see how VaST could combine prioritized sweeping and tabular count–based algorithms like MBIE–EB [Strehl and Littman, 2008] for high–dimensional and continuous environments. Alternatively, measures of familiarity or surprise could be extracted directly from the p or q distributions and used to determine exploration bonuses (see e.g. Bellemare et al. [2016], Houthooft et al. [2016]).

A Hyperparameters for Chapter 6

A.1 3D Navigation

For the three network–based models, hyperparameters were chosen based on a coarse parameter search in two mazes (Figure 6.3 excluding the hazards and Figure 6.6 excluding the teleporter), using the previously published hyperparameters as a starting point for the baselines [Pritzel et al., 2017, Schaul et al., 2015, Mnih et al., 2015]. In all mazes except the smaller Plus–Maze, the agents explored randomly for 50 000 steps to initialize the replay memory before training; ϵ was then annealed from 1 to 0.1 over 200 000 steps. In the Plus–Maze, the agents explored randomly for 10 000 steps and ϵ was annealed over 40 000 steps. We used $\epsilon = 0.05$ for evaluation during test epochs, which lasted for 1000 steps. In all tasks we used a discount factor of 0.99.

The encoder of VaST and the networks for NEC and Prioritized D–DQN all shared the same architecture, as published in [Mnih et al., 2015], with ReLU activations. For all three networks, we used the Adam optimizer [Kingma and Ba, 2014] with $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 1\mathrm{e}{-8}$, and trained on every 4th step. Unless otherwise stated, we used a replay memory size of $\mathcal{N} = 500\,000$ transitions.

VaST We used a latent dimensionality of $d = 32$ unless otherwise stated. For training, we used a minibatch size of 128 and a learning rate of $2 \times 1\mathrm{e}{-4}$. For sweeping, we used $p_{min} = 5 \times 1\mathrm{e}{-5}$. For the Con–crete relaxation, we used the temperatures suggested by Maddison et al. [2016]: $\lambda_1 = 2/3$ for sampling from the posterior and evaluating the posterior log–probability and $\lambda_2 = 0.5$ for evaluating the transition and initial state log–probabilities.

For the decoder architecture, we used a fully–connected layer with 256 units, followed by 4 deconvolutional layers with 4 × 4 filters and stride 2, and intermediate channel depths of 64, 64 and 32 respectively. We used a multi–layer perceptron with 3 hidden layers (with 512, 256 and 512 units respectively) for each action in the transition network.

NEC We used a latent embedding of size 64, $n_s = 50$ for the n–step Q-value backups, and $\alpha = 0.1$ for the tabular learning rate. We performed a 50 approximate nearest–neighbour lookup using the ANNoy library (pypi.python.org/pypi/annoy) on Differentiable Neural Dictionaries of size 500 000 for each action. For training, we used a minibatch size of 32 and a learning rate of $5 \times 1e{-5}$.

Prioritized D–DQN We used the rank–based version of Prioritized DQN with $\alpha = 0.7$ and $\beta = 0.5$ (annealed to 1 over the course of training). We used a minibatch size of 32 and a learning rate of $1e{-4}$ and updated the target network every 2000 steps.

LSH The LSH–based algorithm does not use a neural network or replay memory, since the embedding is based on fixed random projections. We achieved the best results with $d = 64$ for the latent dimensionality. For prioritized sweeping, we used $p_{min} = 5 \times 1e{-5}$.

A.2 Atari: Pong

We used a latent dimensionality of $d = 64$, a replay memory size of $\mathcal{N} = 1\,000\,000$ transitions, and annealed ϵ over 1 000 000 steps. All other hyperparameters were the same as for navigation.

Contributions

Chapter 3 I developed the model for attractor networks in arbitrary environments based on existing toroidal attractor network models [Conklin and Eliasmith, 2005] and the NEF [Eliasmith and Anderson, 2004]; I also introduced the approach for generating multichart attractor networks described in the chapter. I devised the sequential activity model and the hierarchical network structure based in part on the analysis of virtual rotation in [Hansel and Sompolinsky, 1998]. Finally, I developed the "successor coordinates" model and the application to attractor networks drawing in part from the "diffusion map" framework by Coifman and Lafon [2006]. I implemented all of the experiments. I collaborated with Wulfram Gerstner on writing the text for the paper [Corneil and Gerstner, 2015a] which was the foundation for Section 3.5.

Chapter 4 I developed the model of learning hierarchical representations from combined grid and border cell input based in part of the grid cell to place cell models of Solstad et al. [2006] and Sheynikhovich et al. [2009] as well as the boundary vector cell model of Barry et al. [2006]. I implemented all of the experiments. The recurrent learning rule for the attractor network I introduced in the chapter is inspired in part by the weight learning rule in [MacNeil and Eliasmith, 2011], although without the need for an external error signal.

Chapter 5 I developed the network model and devised and implemented the experiments, drawing primarily from existing research on VAEs [Kingma and Welling, 2013, Rezende et al., 2014] and MFEC [Blundell et al., 2016], as well as the Dyna learning framework [Peng and Williams, 1993].

Chapter 6 I developed the model of learning a tabular abstraction of the environment using a VAE–like architecture and the Con–crete distribution, and combining it with parallelized prioritized sweeping, based on Johanni Brea's analysis of prioritized sweeping in tabular environments [Brea, 2017], my existing work in Chapter 5, and the "prioritized sweeping with small backups" algorithm itself [Van Seijen and Sutton, 2013]. I worked with Johanni Brea on determining the implementation of the transition cost in the objective function, as well as the design of the teleporter task. I implemented the model, devised the other experiments and performed all of the experiments. I collaborated with both Johanni Brea and Wulfram

Gerstner on writing the text for the paper [Corneil et al., 2018] which was the foundation for the chapter.

Bibliography

DG Amaral and MP Witter. The three-dimensional organization of the hippocampal formation: a review of anatomical data. *Neuroscience*, 31(3):571–591, 1989.

Per Andersen, Richard Morris, David Amaral, John O'Keefe, and Tim Bliss. *The hippocampus book*. Oxford University Press, 2007.

Christopher G Atkeson. Nonparametric model-based reinforcement learning. In *Advances in neural information processing systems*, pages 1008–1014, 1998.

Christopher G Atkeson and Stefan Schaal. Robot learning from demonstration. In *ICML*, volume 97, pages 12–20. Citeseer, 1997.

Bernard W Balleine and John P O'Doherty. Human and rodent homologies in action control: corticostriatal determinants of goal-directed and habitual action. *Neuropsychopharmacology*, 35(1):48, 2010.

André Barreto, Rémi Munos, Tom Schaul, and David Silver. Successor features for transfer in reinforcement learning. *arXiv preprint arXiv:1606.05312*, 2016a.

André Barreto, Doina Precup, and Joelle Pineau. Practical kernel-based reinforcement learning. *The Journal of Machine Learning Research*, 17(1):2372–2441, 2016b.

Caswell Barry, Colin Lever, Robin Hayman, Tom Hartley, Stephen Burton, John O'Keefe, Kate Jeffery, and N Burgess. The boundary vector cell model of place cell firing and spatial memory. *Reviews in the Neurosciences*, 17(1-2):71–98, 2006.

Gabriel Barth-Maron. *Learning deep state representations with convolutional autoencoders*. PhD thesis, Master's thesis, Brown University, 2015.

Tobias Bast, Iain A Wilson, Menno P Witter, and Richard GM Morris. From rapid place learning to behavioral performance: a key role for the intermediate hippocampus. *PLoS biology*, 7 (4):e1000089, 2009.

Leonard E Baum, Ted Petrie, George Soules, and Norman Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. *The annals of mathematical statistics*, 41(1):164–171, 1970.

M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The Arcade Learning Environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47: 253–279, Jun 2013.

Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information Processing Systems*, pages 1471–1479, 2016.

Richard Bellman. *Dynamic programming.* Princeton University Press, Princeton, NJ, 1957.

Y. Bengio, R. De Mori, G. Flammia, and R. Kompe. Global optimization of a neural network-hidden markov model hybrid. *IEEE Transactions on Neural Networks*, 3(2):252–259, Mar 1992. ISSN 1045-9227. doi: 10.1109/72.125866.

Dimitri P Bertsekas and David Alfred Castanon. Adaptive aggregation methods for infinite horizon dynamic programming. *IEEE transactions on Automatic Control*, 34(6):589–598, 1989.

Shehroze Bhatti, Alban Desmaison, Ondrej Miksik, Nantas Nardelli, N Siddharth, and Philip HS Torr. Playing doom with slam-augmented deep reinforcement learning. *arXiv preprint arXiv:1612.00380*, 2016.

Tale L Bjerknes, Edvard I Moser, and May-Britt Moser. Representation of geometric borders in the developing rat. *Neuron*, 82(1):71–78, 2014.

Hugh Carlton Blodgett. The effect of the introduction of reward upon the maze performance of rats. *University of California publications in psychology*, 1929.

Charles Blundell, Benigno Uria, Alexander Pritzel, Yazhe Li, Avraham Ruderman, Joel Z Leibo, Jack Rae, Daan Wierstra, and Demis Hassabis. Model-free episodic control. *arXiv preprint arXiv:1606.04460*, 2016.

J. Brea. Is prioritized sweeping the better episodic control? *ArXiv e-prints arXiv:1711.06677*, 2017.

Vegard Heimly Brun, Trygve Solstad, Kirsten Brun Kjelstrup, Marianne Fyhn, Menno P Witter, Edvard I Moser, and May-Britt Moser. Progressive increase in grid scale from dorsal to ventral medial entorhinal cortex. *Hippocampus*, 18(12):1200–1212, 2008.

Randy L Buckner. The role of the hippocampus in prediction and imagination. *Annual review of psychology*, 61:27–48, 2010.

Lars Buesing, Theophane Weber, Sebastien Racaniere, SM Eslami, Danilo Rezende, David P Reichert, Fabio Viola, Frederic Besse, Karol Gregor, Demis Hassabis, et al. Learning and querying fast generative models for reinforcement learning. *arXiv preprint arXiv:1802.03006*, 2018.

György Buzsáki. Theta oscillations in the hippocampus. *Neuron*, 33(3):325–340, 2002.

Francesca Cacucci, Colin Lever, Thomas J Wills, Neil Burgess, and John O'Keefe. Theta–modulated place-by-direction cells in the hippocampal formation in the rat. *Journal of Neuroscience*, 24(38):8265–8277, 2004.

Cesar Cadena, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, José Neira, Ian Reid, and John J Leonard. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on Robotics*, 32(6):1309–1332, 2016.

Eduardo F Camacho and Carlos Bordons Alba. *Model predictive control*. Springer Science & Business Media, 2013.

Francis Carpenter, Daniel Manson, Kate Jeffery, Neil Burgess, and Caswell Barry. Grid cells form a global representation of connected environments. *Current Biology*, 25(9):1176–1182, 2015.

David Chapman and Leslie Pack Kaelbling. Input generalization in delayed reinforcement learning: An algorithm and performance comparisons. In *IJCAI*, volume 91, pages 726–731, 1991.

Moses S Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 380–388. ACM, 2002.

Badong Chen, Songlin Zhao, Pingping Zhu, and Jose C Principe. Quantized kernel recursive least squares algorithm. *IEEE transactions on neural networks and learning systems*, 24(9): 1484–1491, 2013.

Jaedeug Choi and Kee-Eung Kim. Nonparametric bayesian inverse reinforcement learning for multiple reward functions. In *Advances in Neural Information Processing Systems*, pages 305–313, 2012.

Lonnie Chrisman. Reinforcement learning with perceptual aliasing: The perceptual distinctions approach. In *AAAI*, pages 183–188, 1992.

Ronald R Coifman and Stéphane Lafon. Diffusion maps. *Applied and computational harmonic analysis*, 21(1):5–30, 2006.

Laura L Colgin, Stefan Leutgeb, Karel Jezek, Jill K Leutgeb, Edvard I Moser, Bruce L McNaughton, and May-Britt Moser. Attractor-map versus autoassociation based attractor dynamics in the hippocampal network. *Journal of neurophysiology*, 104(1):35–50, 2010.

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.

Dane Corneil and Wulfram Gerstner. Attractor network dynamics enable preplay and rapid path planning in maze–like environments. In *Advances in Neural Information Processing Systems*, pages 1684–1692, 2015a.

Dane Corneil and Wulfram Gerstner. Rapid path planning and preplay in maze-like environments using attractor networks. In *COSYNE 2015*, number EPFL-POSTER-207002, 2015b.

Dane Corneil and Wulfram Gerstner. Offline reinforcement learning with simulated trajectories in latent space. In *CCN 2017*, 2017.

Dane Corneil, Wulfram Gerstner, and Johanni Brea. Efficient model-based deep reinforcement learning with variational state tabulation. In *Proceedings of the 35th International Conference on Machine Learning*, 2018.

Jozsef Csicsvari, Joseph O'neill, Kevin Allen, and Timothy Senior. Place-selective firing contributes to the reverse-order reactivation of ca1 pyramidal cells during sharp waves in open-field exploration. *European Journal of Neuroscience*, 26(3):704–716, 2007.

William Curran, Tim Brys, David Aha, Matthew Taylor, and William D Smart. Dimensionality reduced reinforcement learning for assistive robots. In *Proc. of Artificial Intelligence for Human-Robot Interaction at AAAI Fall Symposium Series*, 2016.

Thomas J Davidson, Fabian Kloosterman, and Matthew A Wilson. Hippocampal replay of extended experience. *Neuron*, 63(4):497–507, 2009.

Nathaniel D Daw, Yael Niv, and Peter Dayan. Uncertainty-based competition between prefrontal and dorsolateral striatal systems for behavioral control. *Nature neuroscience*, 8(12): 1704, 2005.

Peter Dayan. Improving generalization for temporal difference learning. *Neural Computation*, 5(4):613–624, 1993.

Marc Deisenroth and Carl E Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472, 2011.

Marc Peter Deisenroth, Carl Edward Rasmussen, and Jan Peters. Gaussian process dynamic programming. *Neurocomputing*, 72(7-9):1508–1524, 2009.

Kamran Diba and György Buzsáki. Forward and reverse hippocampal place-cell sequences during ripples. *Nature neuroscience*, 10(10):1241, 2007.

Simona Doboli, Ali A Minai, and Phillip J Best. Latent attractors: a model for context-dependent place representations in the hippocampus. *Neural Computation*, 12(5):1009–1043, 2000.

Finale Doshi-Velez, David Wingate, Nicholas Roy, and Joshua B Tenenbaum. Nonparametric bayesian policy priors for reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 532–540, 2010.

Kenji Doya, Kazuyuki Samejima, Ken-ichi Katagiri, and Mitsuo Kawato. Multiple model-based reinforcement learning. *Neural computation*, 14(6):1347–1369, 2002.

Patrick J Drew and LF Abbott. Extending the effects of spike-timing-dependent plasticity to behavioral timescales. *Proceedings of the National Academy of Sciences*, 103(23):8876–8881, 2006.

Kurt Driessens and Sašo Džeroski. Combining model-based and instance-based learning for first order regression. In *Proceedings of the 22nd international conference on Machine learning*, pages 193–200. ACM, 2005.

John C Eccles, P Fatt, and K Koketsu. Cholinergic and inhibitory synapses in a pathway from motor-axon collaterals to motoneurones. *The Journal of physiology*, 126(3):524–562, 1954.

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, 2003.

Chris Eliasmith and Charles H Anderson. *Neural engineering: Computation, representation, and dynamics in neurobiological systems.* MIT press, 2004.

Yaakov Engel, Shie Mannor, and Ron Meir. Reinforcement learning with gaussian processes. In *Proceedings of the 22nd international conference on Machine learning*, pages 201–208. ACM, 2005.

Damien Ernst, Pierre Geurts, and Louis Wehenkel. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6(Apr):503–556, 2005.

Mohammadjavad Faraji, Kerstin Preuschoff, and Wulfram Gerstner. Balancing new against old information: The role of puzzlement surprise in learning. *Neural computation*, 30(1): 34–83, 2018.

G. Farquhar, T. Rocktäschel, M. Igl, and S. Whiteson. TreeQN and ATreeC: Differentiable Tree Planning for Deep Reinforcement Learning. *ArXiv e-prints*, October 2017.

Miroslav Fiedler. Laplacian of graphs and algebraic connectivity. *Banach Center Publications*, 25(1):57–70, 1989.

Chelsea Finn, Xin Yu Tan, Yan Duan, Trevor Darrell, Sergey Levine, and Pieter Abbeel. Deep spatial autoencoders for visuomotor learning. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 512–519. IEEE, 2016.

David J Foster and Matthew A Wilson. Reverse replay of behavioural sequences in hippocampal place cells during the awake state. *Nature*, 440(7084):680, 2006.

Marco Fraccaro, Danilo Jimenez Rezende, Yori Zwols, Alexander Pritzel, SM Eslami, and Fabio Viola. Generative temporal models with spatial memory for partially observed environments. *arXiv preprint arXiv:1804.09401*, 2018.

Mathias Franzius, Henning Sprekeler, and Laurenz Wiskott. Slowness and sparseness lead to place, head-direction, and spatial-view cells. *PLoS Computational Biology*, 3(8):e166, 2007.

Marianne Fyhn, Sturla Molden, Menno P Witter, Edvard I Moser, and May-Britt Moser. Spatial representation in the entorhinal cortex. *Science*, 305(5688):1258–1264, 2004.

Samuel J Gershman and Nathaniel D Daw. Reinforcement learning and episodic memory in humans and animals: An integrative framework. *Annual Review of Psychology*, 68, 2017.

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.

Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

Katalin M Gothard, William E Skaggs, and Bruce L McNaughton. Dynamics of mismatch correction in the hippocampal ensemble code for space: interaction between path integration and environmental cues. *Journal of Neuroscience*, 16(24):8027–8040, 1996.

Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous deep q-learning with model-based acceleration. *arXiv preprint arXiv:1603.00748*, 2016.

Anoopum S Gupta, Matthijs AA van der Meer, David S Touretzky, and A David Redish. Hippocampal replay is not a simple function of experience. *Neuron*, 65(5):695–705, 2010.

Saurabh Gupta, James Davidson, Sergey Levine, Rahul Sukthankar, and Jitendra Malik. Cognitive mapping and planning for visual navigation. *arXiv preprint arXiv:1702.03920*, 3, 2017.

Nicholas J Gustafson and Nathaniel D Daw. Grid cells, place cells, and geodesic generalization for spatial reinforcement learning. *PLoS computational biology*, 7(10):e1002235, 2011.

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, 2005.

David Hansel and Haim Sompolinsky. 13 modeling feature selectivity in local cortical circuits. 1998.

Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. *CoRR, abs/1507.06527*, 7(1), 2015.

Irina Higgins, Arka Pal, Andrei A Rusu, Loic Matthey, Christopher P Burgess, Alexander Pritzel, Matthew Botvinick, Charles Blundell, and Alexander Lerchner. Darla: Improving zero-shot transfer in reinforcement learning. *arXiv preprint arXiv:1707.08475*, 2017.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9 (8):1735–1780, 1997.

Peter C Holland. Cognitive versus stimulus-response theories of learning. *Learning & behavior*, 36(3):227–241, 2008.

Jesse Hostetler, Alan Fern, and Tom Dietterich. State aggregation in monte carlo tree search. In *AAAI*, pages 2446–2452, 2014.

Rein Houthooft, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. Vime: Variational information maximizing exploration. In *Advances in Neural Information Processing Systems*, pages 1109–1117, 2016.

Marcus Hutter. Extreme state aggregation beyond mdps. In *International Conference on Algorithmic Learning Theory*, pages 185–199. Springer, 2014.

John R Huxter, Timothy J Senior, Kevin Allen, and Jozsef Csicsvari. Theta phase–specific codes for two-dimensional position, trajectory and heading in the hippocampus. *Nature neuroscience*, 11(5):587, 2008.

Laurent Itti and Pierre Baldi. Bayesian surprise attracts human attention. *Vision research*, 49 (10):1295–1306, 2009.

Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*, 2016.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

Karel Jezek, Espen J Henriksen, Alessandro Treves, Edvard I Moser, and May-Britt Moser. Theta-paced flickering between place-cell maps in the hippocampus. *Nature*, 478(7368): 246, 2011.

Adam Johnson and A David Redish. Neural ensembles in ca3 transiently encode paths forward of the animal at a decision point. *Journal of Neuroscience*, 27(45):12176–12189, 2007.

Nicholas K Jong and Peter Stone. State abstraction discovery from irrelevant state variables. In *IJCAI*, volume 8, pages 752–757, 2005.

Min W Jung, Sidney I Wiener, and Bruce L McNaughton. Comparison of spatial firing characteristics of units in dorsal and ventral hippocampus of the rat. *Journal of Neuroscience*, 14 (12):7347–7356, 1994.

Mattias P Karlsson and Loren M Frank. Awake replay of remote experiences in the hippocampus. *Nature neuroscience*, 12(7):913, 2009.

Michał Kempka, Marek Wydmuch, Grzegorz Runc, Jakub Toczek, and Wojciech Jaśkowski. ViZDoom: A Doom-based AI research platform for visual reinforcement learning. In *IEEE Conference on Computational Intelligence and Games*, pages 341–348, Santorini, Greece, Sep 2016. IEEE.

Azadeh Khajeh-Alijani, Robert Urbanczik, and Walter Senn. Scale-free navigational planning by neuronal traveling waves. *PloS one*, 10(7):e0127269, 2015.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Donald E Kirk. *Optimal control theory: an introduction*. Courier Corporation, 2012.

Kirsten Brun Kjelstrup, Trygve Solstad, Vegard Heimly Brun, Torkel Hafting, Stefan Leutgeb, Menno P Witter, Edvard I Moser, and May-Britt Moser. Finite scale of spatial representation in the hippocampus. *Science*, 321(5885):140–143, 2008.

Jonathan Ko, Daniel J Klein, Dieter Fox, and Dirk Haehnel. Gaussian processes and reinforcement learning for identification and control of an autonomous blimp. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 742–747. IEEE, 2007.

Vijay R Konda and John N Tsitsiklis. Actor-critic algorithms. In *Advances in neural information processing systems*, pages 1008–1014, 2000.

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.

Tejas D Kulkarni, Ardavan Saeedi, Simanta Gautam, and Samuel J Gershman. Deep successor reinforcement learning. *arXiv preprint arXiv:1606.02396*, 2016.

Malte Kuss and Carl E Rasmussen. Gaussian processes in reinforcement learning. In *Advances in neural information processing systems*, pages 751–758, 2004.

Branislav Kveton and Georgios Theocharous. Structured kernel-based reinforcement learning. In *AAAI*, 2013.

Sascha Lange and Martin Riedmiller. Deep auto-encoder neural networks in reinforcement learning. In *The 2010 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2010.

Rosamund F Langston, James A Ainge, Jonathan J Couey, Cathrin B Canto, Tale L Bjerknes, Menno P Witter, Edvard I Moser, and May-Britt Moser. Development of the spatial representation system in the rat. *Science*, 328(5985):1576–1580, 2010.

Phillip Larimer and Ben W Strowbridge. Representing information in cell assemblies: persistent activity mediated by semilunar granule cells. *Nature neuroscience*, 13(2):213–222, 2010.

Albert K Lee and Matthew A Wilson. Memory of sequential experience in the hippocampus during slow wave sleep. *Neuron*, 36(6):1183–1194, 2002.

Máté Lengyel and Peter Dayan. Hippocampal contributions to control: The third way. In *NIPS*, volume 20, pages 889–896, 2007.

Lihong Li, Thomas J Walsh, and Michael L Littman. Towards a unified theory of state abstraction for MDPs. In *ISAIM*, 2006.

Lihong Li, Michael L Littman, and L Littman. Prioritized sweeping converges to the optimal value function, 2008.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Long-Ji Lin. Reinforcement learning for robots using neural networks. Technical report, Carnegie-Mellon Univ Pittsburgh PA School of Computer Science, 1993.

John Lisman and A David Redish. Prediction, sequences and the hippocampus. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 364(1521):1193–1201, 2009.

Kevin Sebastian Luck, Gerhard Neumann, Erik Berger, Jan Peters, and Heni Ben Amor. Latent space policy search for robotics. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 1434–1440. IEEE, 2014.

David MacNeil and Chris Eliasmith. Fine-tuning and the stability of recurrent neural networks. *PloS one*, 6(9):e22885, 2011.

C. J. Maddison, A. Mnih, and Y. Whye Teh. The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables. *ArXiv e-prints arXiv:1611.00712*, November 2016.

Sridhar Mahadevan. *Learning Representation and Control in Markov Decision Processes*, volume 3. Now Publishers Inc, 2009.

Etan J Markus, Carol A Barnes, Bruce L McNaughton, Victoria L Gladden, and William E Skaggs. Spatial information content and reliability of hippocampal ca1 neurons: effects of visual input. *Hippocampus*, 4(4):410–421, 1994.

Louis-Emmanuel Martinet, Denis Sheynikhovich, Karim Benchenane, and Angelo Arleo. Spatial learning and action planning in a prefrontal cortical network model. *PLoS computational biology*, 7(5):e1002045, 2011.

R Andrew McCallum. Instance-based utile distinctions for reinforcement learning with hidden state. In *Machine Learning Proceedings 1995*, pages 387–395. Elsevier, 1995.

Bruce L McNaughton, Carol A Barnes, Jason L Gerrard, Katalin Gothard, Min W Jung, James J Knierim, H Kudrimoti, Y Qin, WE Skaggs, M Suster, et al. Deciphering the hippocampal polyglot: the hippocampus as a path integration system. *Journal of Experimental Biology*, 199(1):173–185, 1996.

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, 2006.

P. Mirowski, R. Pascanu, F. Viola, H. Soyer, A. J. Ballard, A. Banino, M. Denil, R. Goroshin, L. Sifre, K. Kavukcuoglu, D. Kumaran, and R. Hadsell. Learning to Navigate in Complex Environments. *ArXiv e-prints*, November 2016.

Andriy Mnih and Karol Gregor. Neural variational inference and learning in belief networks. In Eric P. Xing and Tony Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 1791–1799, Bejing, China, 22–24 Jun 2014. PMLR.

Andriy Mnih and Danilo Rezende. Variational inference for monte carlo objectives. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 2188–2196, New York, New York, USA, 20–22 Jun 2016. PMLR.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540), 2015.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937, 2016.

Andrew W Moore and Christopher G Atkeson. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine learning*, 13(1):103–130, 1993.

Tetsuro Morimura, Masashi Sugiyama, Hisashi Kashima, Hirotaka Hachiya, and Toshiyuki Tanaka. Nonparametric return distribution approximation for reinforcement learning. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 799–806, 2010.

Edvard I Moser, Emilio Kropff, and May-Britt Moser. Place cells, grid cells, and the brain's spatial representation system. *Annual review of neuroscience*, 31, 2008.

May-Britt Moser, David C Rowland, and Edvard I Moser. Place cells, grid cells, and memory. *Cold Spring Harbor perspectives in biology*, 7(2):a021808, 2015.

Lynn Nadel. The hippocampus and space revisited. *Hippocampus*, 1(3):221–229, 1991.

A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine. Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning. *ArXiv e-prints*, August 2017.

Toshiaki Nakashiba, Jennie Z Young, Thomas J McHugh, Derek L Buhl, and Susumu Tonegawa. Transgenic inhibition of synaptic transmission reveals role of ca3 output in hippocampal learning. *Science*, 319(5867):1260–1264, 2008.

Kazu Nakazawa, Linus D Sun, Michael C Quirk, Laure Rondi-Reig, Matthew A Wilson, and Susumu Tonegawa. Hippocampal ca3 nmda receptors are crucial for memory acquisition of one-time experience. *Neuron*, 38(2):305–315, 2003.

Yin Cheng Ng, Pawel M Chilinski, and Ricardo Silva. Scaling factorial hidden markov models: Stochastic variational inference without messages. In *Advances in Neural Information Processing Systems 29*, pages 4044–4052. 2016.

Junhyuk Oh, Satinder Singh, and Honglak Lee. Value prediction network. In *Advances in Neural Information Processing Systems*, pages 6120–6130, 2017.

Erkki Oja. Simplified neuron model as a principal component analyzer. *Journal of mathematical biology*, 15(3):267–273, 1982.

John O'Keefe. Place units in the hippocampus of the freely moving rat. *Experimental neurology*, 51(1):78–109, 1976.

John O'Keefe and Jonathan Dostrovsky. The hippocampus as a spatial map: Preliminary evidence from unit activity in the freely-moving rat. *Brain research*, 1971.

John O'Keefe and Lynn Nadel. The hippocampus as a cognitive map. *Behavioral and Brain Sciences*, 2(4):520–533, 1979.

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

Dirk Ormoneit and Śaunak Sen. Kernel-based reinforcement learning. *Machine learning*, 49 (2-3):161–178, 2002.

Simone Parisi, Simon Ramstedt, and Jan Peters. Goal-driven dimensionality reduction for reinforcement learning. In *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, pages 4634–4639. IEEE, 2017.

Emilio Parisotto and Ruslan Salakhutdinov. Neural map: Structured memory for deep reinforcement learning. *arXiv preprint arXiv:1702.08360*, 2017.

Jing Peng and Ronald J Williams. Efficient learning and planning within the dyna framework. *Adaptive Behavior*, 1(4):437–454, 1993.

Jing Peng and Ronald J Williams. Incremental multi-step q-learning. *Machine learning*, 22(1), 1996.

J. Peters. Policy gradient methods. *Scholarpedia*, 5(11):3698, 2010. doi: 10.4249/scholarpedia.3698. revision #137199.

Jan Peters and Stefan Schaal. Reinforcement learning of motor skills with policy gradients. *Neural networks*, 21(4):682–697, 2008.

Brad E Pfeiffer and David J Foster. Hippocampal place-cell sequences depict future paths to remembered goals. *Nature*, 497(7447), 2013.

Brad E Pfeiffer and David J Foster. Autoassociative dynamics in the generation of sequences of hippocampal place cells. *Science*, 349(6244):180–183, 2015.

Filip Ponulak and John J Hopfield. Rapid, parallel path planning by propagating wavefronts of spiking neural activity. *Frontiers in computational neuroscience*, 7, 2013.

Alexander Pritzel, Benigno Uria, Sriram Srinivasan, Adrià Puigdomènech Badia, Oriol Vinyals, Demis Hassabis, Daan Wierstra, and Charles Blundell. Neural episodic control. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70. PMLR, 2017.

Gregory J Quirk, Robert U Muller, and John L Kubie. The firing of hippocampal place cells in the dark depends on the rat's recent experience. *Journal of Neuroscience*, 10(6):2008–2017, 1990.

Sébastien Racanière, Théophane Weber, David Reichert, Lars Buesing, Arthur Guez, Danilo Jimenez Rezende, Adrià Puigdomènech Badia, Oriol Vinyals, Nicolas Heess, Yujia Li, et al. Imagination-augmented agents for deep reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 5694–5705, 2017.

A David Redish, Ephron S Rosenzweig, JD Bohanick, BL McNaughton, and CA Barnes. Dynamics of hippocampal ensemble activity realignment: time versus space. *Journal of Neuroscience*, 20(24):9298–9309, 2000.

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*, 2014.

Martin Riedmiller. Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method. In *European Conference on Machine Learning*, pages 317–328. Springer, 2005.

Matthias Rolf and Minoru Asada. Latent goal analysis for dimension reduction in reinforcement learning. In *Machine Learning for Interactive Systems*, pages 26–30, 2015.

Edmund T Rolls, Robert G Robertson, and Pierre Georges-François. Spatial view cells in the primate hippocampus. *European Journal of Neuroscience*, 9(8):1789–1794, 1997.

Edmund T Rolls, Simon M Stringer, and Thomas Elliot. Entorhinal cortex grid cells can map to hippocampal place cells by competitive learning. *Network: Computation in Neural Systems*, 17(4):447–465, 2006.

M Rosenblatt. Markov chains. In *Random Processes*, pages 36–67. Springer, 1974.

Gavin A Rummery and Mahesan Niranjan. *On-line Q-learning using connectionist systems*, volume 37. University of Cambridge, Department of Engineering Cambridge, England, 1994.

Evan M Russek, Ida Momennejad, Matthew M Botvinick, Samuel J Gershman, and Nathaniel D Daw. Predictive representations can link model-based reinforcement learning to model-free mechanisms. *PLOS Computational Biology*, 13(9):e1005768, 2017.

Alexei Samsonovich. Continuous attractor network. *Scholarpedia*, 2013.

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.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

Fabian Schoenfeld and Laurenz Wiskott. Modeling place field activity with hierarchical slow feature analysis. *Frontiers in Computational Neuroscience*, 9:51, 2015.

Nishal Shah and Frédéric Alexandre. Reinforcement learning and dimensionality reduction: a model in computational neuroscience. In *International Joint Conference on Neural Networks IJCNN 2011*, 2011.

Denis Sheynikhovich, Ricardo Chavarriaga, Thomas Strösslin, Angelo Arleo, and Wulfram Gerstner. Is there a geometric module for spatial orientation? insights from a rodent navigation model. *Psychological review*, 116(3):540, 2009.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587): 484–489, 2016.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017a.

David Silver, Hado van Hasselt, Matteo Hessel, Tom Schaul, Arthur Guez, Tim Harley, Gabriel Dulac-Arnold, David Reichert, Neil Rabinowitz, Andre Barreto, and Thomas Degris. The predictron: End-to-end learning and planning. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, 2017b.

Satinder P Singh, Tommi Jaakkola, and Michael I Jordan. Learning without state-estimation in partially observable markovian decision processes. In *Machine Learning Proceedings 1994*, pages 284–292. Elsevier, 1994.

William E Skaggs and Bruce L McNaughton. Theta phase precession in hippocampal. *Hippocampus*, 6:149–172, 1996.

William E Skaggs and Bruce L McNaughton. Spatial firing properties of hippocampal ca1 populations in an environment containing two visually identical regions. *Journal of Neuroscience*, 18(20):8455–8466, 1998.

Kihyuk Sohn, Honglak Lee, and Xinchen Yan. Learning structured output representation using deep conditional generative models. In *Advances in Neural Information Processing Systems*, pages 3483–3491, 2015.

Trygve Solstad, Edvard I Moser, and Gaute T Einevoll. From grid cells to place cells: a mathematical model. *Hippocampus*, 16(12):1026–1031, 2006.

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.

Henning Sprekeler. On the relation of slow feature analysis and laplacian eigenmaps. *Neural computation*, 23(12):3287–3302, 2011.

Henning Sprekeler, Christian Michaelis, and Laurenz Wiskott. Slowness: an objective for spike-timing-dependent plasticity. *PLoS Comput Biol*, 3(6):e112, 2007.

Kimberly L Stachenfeld, Matthew Botvinick, and Samuel J Gershman. Design principles of the hippocampal cognitive map. In Z. Ghahramani, M. Welling, C. Cortes, N.D. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2528–2536. Curran Associates, Inc., 2014.

Kimberly L Stachenfeld, Matthew M Botvinick, and Samuel J Gershman. The hippocampus as a predictive map. *Nature neuroscience*, 20(11):1643, 2017.

Alexander L Strehl and Michael L Littman. An analysis of model-based interval estimation for markov decision processes. *Journal of Computer and System Sciences*, 74(8):1309–1331, 2008.

Richard S Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of ICML 1990*, 1990.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, Cambridge, MA, (in progress) second edition, 2018. URL http://incompleteideas.net/book/the-book-2nd.html.

Richard S. Sutton, Csaba Szepesvári, Alborz Geramifard, and Michael Bowling. Dyna-style planning with linear function approximation and prioritized sweeping. In *Proceedings of the 24th Conference on Uncertainty in Artificial Intelligence*, 2008.

Haoran Tang, Rein Houthooft, Davis Foote, Adam Stooke, OpenAI Xi Chen, Yan Duan, John Schulman, Filip DeTurck, and Pieter Abbeel. # exploration: A study of count-based exploration for deep reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 2753–2762, 2017.

Voot Tangkaratt, Jun Morimoto, and Masashi Sugiyama. Model-based reinforcement learning with dimension reduction. *Neural Networks*, 84:1–16, 2016.

Edward L Thorndike. Animal intelligence: an experimental study of the associative processes in animals. *The Psychological Review: Monograph Supplements*, 2(4):i, 1898.

Sebastian Thrun. Simultaneous localization and mapping. In *Robotics and cognitive approaches to spatial mapping*, pages 13–41. Springer, 2007.

Edward C Tolman. Cognitive maps in rats and men. *Psychological review*, 55(4):189, 1948.

Edward Chace Tolman and Charles H Honzik. Introduction and removal of reward, and maze performance in rats. *University of California publications in psychology*, 1930.

Bryan P Tripp and Chris Eliasmith. Population models of temporal differentiation. *Neural computation*, 22(3):621–659, 2010.

Misha Tsodyks. Attractor neural network models of spatial maps in hippocampus. *Hippocampus*, 9(4):481–489, 1999.

G. Tucker, A. Mnih, C. J. Maddison, D. Lawson, and J. Sohl-Dickstein. REBAR: Low-variance, unbiased gradient estimates for discrete latent variable models. *ArXiv e-prints*, March 2017.

Nachum Ulanovsky and Cynthia F Moss. Hippocampal cellular and network activity in freely moving echolocating bats. *Nature neuroscience*, 10(2):224, 2007.

Robert Urbanczik and Walter Senn. Learning by the dendritic prediction of somatic spiking. *Neuron*, 81(3):521–528, 2014.

Herke Van Hoof, Jan Peters, and Gerhard Neumann. Learning of non-parametric control policies with high-dimensional state features. In *Artificial Intelligence and Statistics*, pages 995–1003, 2015.

Herke Van Hoof, Nutan Chen, Maximilian Karl, Patrick van der Smagt, and Jan Peters. Stable reinforcement learning with autoencoders for tactile and visual data. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pages 3928–3934. IEEE, 2016.

Harm Van Seijen and Richard S Sutton. Efficient planning in MDPs by small backups. In *Proceedings of the 30th International Conference on Machine Learning*, volume 28, 2013.

Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

Manuel Watter, Jost Springenberg, Joschka Boedecker, and Martin Riedmiller. Embed to control: A locally linear latent dynamics model for control from raw images. In *Advances in neural information processing systems*, pages 2746–2754, 2015.

Andrew M Wikenheiser and A David Redish. Hippocampal theta sequences reflect current goals. *Nature neuroscience*, 18(2):289, 2015.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

Tom J Wills, Colin Lever, Francesca Cacucci, Neil Burgess, and John O'keefe. Attractor dynamics in the hippocampal representation of the local environment. *Science*, 308(5723):873–876, 2005.

Tom J Wills, Francesca Cacucci, Neil Burgess, and John O'Keefe. Development of the hippocampal cognitive map in preweanling rats. *Science*, 328(5985):1573–1576, 2010.

Matthew A Wilson and Bruce L McNaughton. Dynamics of the hippocampal ensemble code for space. *Science*, 261(5124):1055–1058, 1993.

Dane Corneil

PhD Candidate, Computational Neuroscience

Contact

Route Cantonale 55,
St-Sulpice VD, 1025
Switzerland

+41 (0) 76 706-3263

dane@corneil.ca
linkedin:danecor

Programming

Python
TensorFlow
NumPy/SciPy
Django
Lua, Java, Matlab
JavaScript & JQuery
CSS & HTML

Interests

Science–inspired
improv comedy and
theatre
www.thecatalyst.ch

Education

2013–Now **PhD Candidate** Computational Neuroscience — EPFL, Switzerland
Laboratory of Computational Neuroscience, Prof. Wulfram Gerstner
Primary research: Model–based reinforcement learning and navigation in animals and machines.
Teaching assistant: Artificial neural networks, unsupervised & reinforcement learning, linear algebra.

2010–2012 **MSc** Neural Systems and Computation — University of Zürich/ETH Zürich, Switzerland
Thesis: Temporal Learning & Inference in Stochastic Winner-Take-All Networks
Adviser: Prof. Giacomo Indivieri
GPA: 5.6/6.0
Coursework: Neuroscience, models of computation, neuromorphic engineering.

2004–2009 **BASc** Honours Systems Design Engineering — University of Waterloo, Canada
Co-operative Program, Psychology Minor, Cognitive Science Option
Final year average: 86.3 %
“Outstanding” or “Excellent” evaluation in all co-operative work terms.
Coursework: Pattern recognition, artificial intelligence, simulating neurobiological systems.

Publications

Eligibility Traces and Plasticity on Behavioral Time Scales: Experimental Support of neoHebbian Three-Factor Learning Rules
W. Gerstner, M. Lehmann, V. Liakoni, **D. Corneil**, and J. Brea
Frontiers in Neuroscience (2018). 2018

Efficient Model-Based Deep Reinforcement Learning with Variational State Tabulation
D. Corneil, Gerstner W., and J. Brea
International Conference on Machine Learning ICML, 2018

Attractor network dynamics enable preplay and rapid path planning in maze–like environments
D. Corneil and W. Gerstner
Neural Information Processing Systems NIPS, 2015, **(Selected for oral presentation)**

Function approximation with uncertainty propagation in a VLSI spiking neural network
D. Corneil, D. Sonnleithner, E. Neftci, E. Chicca, M. Cook, G. Indiveri, and R. Douglas
International Joint Conference on Neural Networks, IJCNN, 2012

Real-time inference in a VLSI spiking neural network
D. Corneil, D. Sonnleithner, E. Neftci, E. Chicca, M. Cook, G. Indiveri, and R. Douglas
International Symposium on Circuits and Systems, ISCAS, 2012

StandEasy: a device for people with Parkinson’s Disease
A. Schulze, A. Murczek, **D. Corneil**, D. Kraan, D. Smith, K. Cerar, J. Ma, and J. Zelek
Proceedings of the IASTED International Conference on Telehealth/Assistive Technologies, 2008

Extended Abstracts

Offline Reinforcement Learning with Simulated Trajectories in Latent Space
D. Corneil and W. Gerstner
Computational and Cognitive Neuroscience CCN, 2017

Rapid path planning and preplay in maze–like environments using attractor networks
D. Corneil and W. Gerstner
Computational and Systems Neuroscience COSYNE, 2015

Learning, inference, and replay of hidden state sequences in recurrent spiking neural networks
D. Corneil, E. Neftci, G. Indiveri, and M. Pfeiffer
Computational and Systems Neuroscience COSYNE, 2014

Work Experience

2016 **Research Science Intern** London, England
Google Deepmind
Working within the Neuroscience team, devised and tested extensions to models of deep reinforcement learning in complex environments.

2014–2016 **Co-founder/Web Developer** Zürich, Switzerland
Ponder
Working with two other PhD students, designed and built a Django–based search engine and aggregator for academic talks and events. Work ranged from logic and database development to front–end web design.

2009–2010 **Web Developer** Waterloo, Ontario, Canada
Pebble
Redesigned and built the customer support section for a smartwatch start–up company using Django.

2008 **Research Assistant** Toronto, Canada
Defence Research & Development Canada
Using AnyLogic (Java-based), developed models of regional stability in war zones based on troop movements. Supervised a research study to determine the extent to which users can internalize and act on war zone dynamics.

2007 **Java Developer** Ottawa, Ontario, Canada
Communications Security Establishment Canada
Acted as a member of an internal Java development team. Researched and compiled a report on the usability of an internal application.