



The Learning Hypothesis on Spatial Receptive Field Remapping

Citation

Kuo, Henry. 2024. The Learning Hypothesis on Spatial Receptive Field Remapping. Bachelors Thesis, Harvard University Engineering and Applied Sciences.

Link

<https://dash.harvard.edu/handle/1/42719184>

Terms of use

This article was downloaded from Harvard University's DASH repository, and is made available under the terms and conditions applicable to Other Posted Material (LAA), as set forth at

<https://harvardwiki.atlassian.net/wiki/external/NGY5NDE4ZjgzNTc5NDQzMGIzZWZhMGFIOWI2M2EwYTg>

Accessibility

<https://accessibility.huit.harvard.edu/digital-accessibility-policy>

Share Your Story

The Harvard community has made this article openly available. Please share how this access benefits you. [Submit a story](#)

The Learning Hypothesis on Spatial Receptive Field Remapping

A THESIS PRESENTED

BY

HENRY KUO

TO

THE DEPARTMENT OF COMPUTER SCIENCE

AND THE FACULTY OF THE COMMITTEE ON DEGREES IN NEUROSCIENCE

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE WITH HONORS

OF BACHELOR OF ARTS

HARVARD UNIVERSITY

CAMBRIDGE, MASSACHUSETTS

MARCH 2024

Neuroscience Concentration
Division of Life Sciences
Harvard University

The Harvard College Honor Code

Members of the Harvard College community commit themselves to producing academic work of integrity – that is, work that adheres to the scholarly and intellectual standards of accurate attribution of sources, appropriate collection and use of data, and transparent acknowledgement of the contribution of others to their ideas, discoveries, interpretations, and conclusions. Cheating on exams or problem sets, plagiarizing or misrepresenting the ideas or language of someone else as one’s own, falsifying data, or any other instance of academic dishonesty violates the standards of our community, as well as the standards of the wider world of learning and affairs.

Signature: _____

©2024 – HENRY KUO
ALL RIGHTS RESERVED.

The Learning Hypothesis on Spatial Receptive Field Remapping

ABSTRACT

The spatial pattern of grid cells changes due to information such as the location of rewards. Past research has discovered that after rats learn the location of rewards in a cheeseboard environment, their grid cells activate more near reward locations, distorting the grid pattern. However, it is unknown whether this distortion has any purpose at a population level. In this work, we use reinforcement learning (RL) theory to explain how changes in spatial activation patterns could be the result of the agent trying to optimize learning progress such as learning speed or convergence learning error. Here, we assume that distortions of the grid patterns correspond to changes in features in temporal difference (TD) learning. First, we show that with fixed features, parameters such as learning rate (γ) and batch size (B) linearly change the TD learning convergence learning error. Specifically, the convergence learning error is proportional to $O(\frac{\gamma}{B})$. Though this result depends on a Gaussian approximation when feature dimension is high, we demonstrate the linear scaling phenomenon on continuous complex environments such as MountainCar-vo. Second, we show that using an evolutionary method to adjust radial basis function features (place cell features) would evolve features that are centered near reward locations. Finally, we use experimental data from rats' medial entorhinal cortex (MEC) to show that the features after learning provide faster TD learning convergence speed and lower convergence error. These results support the theory that animals change their perceptions to optimize learning progress.

Contents

1	INTRODUCTION	1
1.1	Perception of the Environment	2
1.2	Learning with Rewards	4
1.3	Perception Affects Learning	7
1.4	Learning Affects Perception	11
2	METHODS AND EXPERIMENTS	12
2.1	Theory on the Loss Dynamics of Temporal Difference (TD) Learning	12
2.2	Evolving Features to Learn Faster	17
2.3	Working with Experimental Data	19
3	RESULTS	29
3.1	Verification of Theory in MountainCar-vo	29
3.2	Cross-entropy Adaptation Algorithm	30
3.3	TD Learning Curves of Experimental Data	31
4	DISCUSSION	40
4.1	Convergence Speed and Error Depends on Hyperparameters and Features	41
4.2	Features Match the Value Function	41
4.3	Feature Fit Depends on Policy	42
4.4	Discrepancy Between Learning Curves and Spectral Analysis	42
4.5	Modeling Policy Iteration	42
4.6	Representation Learning	43
4.7	Large-Scale Neural Recordings	43
4.8	Verification of Other Experimental Findings	44
4.9	Conclusion	44
	APPENDIX A APPENDIX	45
A.1	Additional Cross-entropy Based Basis Adaptation Results	45
A.2	Additional TD Learning Curves with Experimental Data	46
A.3	Plots of True Value Functions	46

A.4	Plots of Learned Value Functions	46
A.5	Equipment and Tools	46
REFERENCES		54

Listing of Figures

1.1	Example of place and grid cell firing patterns.	2
2.1	The MountainCar-vo environment.	14
2.2	Value functions of MountainCar-vo obtained by Tabular Q-Learning and TD Learning.	16
2.3	Trajectories and rate maps from experimental data.	22
2.4	Trajectories sampled from approximated policies.	26
3.1	Evaluating the TD Learning theory on MountainCar-vo.	34
3.2	Results of the Cross-entropy Based Basis Adaptation Algorithm.	35
3.3	TD Learning curves of the random policy.	37
3.4	TD Learning curves of the post-learning policy.	38
3.5	Spectral perspective of TD learning on the post-learning policy.	39
A.1	Additional results of the Cross-entropy Based Basis Adaptation Algorithm	47
A.2	TD Learning curves of the pre-learning policy.	48
A.3	TD Learning curves of the post-learning policy with peak normalization.	49
A.4	Plots of true value functions	50
A.5	Plots of true value functions (cont.)	51
A.6	Plots of true value functions (cont.)	52
A.7	Plots of learned value functions.	53

Listing of Tables

3.1	Properties of experimental data.	36
-----	--	----

TO MY PARENTS, THE SHOULDERS OF WHOM I HAVE STOOD ON.

Acknowledgments

I would like to thank my thesis advisor Cengiz Pehlevan for his support and advice. I am very grateful to have had the opportunity to work in such an innovative environment.

I would also like to thank Paul Masset for meeting me at a reinforcement learning symposium and subsequently inviting me to join the lab. Thank you Blake Bordelon for your invaluable work on the theory of learning and input on mathematics. Thank you Sasha Rayshubskiy for supporting my first research experience at Harvard.

Finally, thanks to my friends (in last name order) Rudra Barua, Rachel Guo, Trevor Jones, Angie Ling, Sean Yang for supporting me throughout my college years.

List of Contributions

Experimental data was solely collected by Boccara et al., 2019, with permission requested by P. Masset. The theory of TD learning was conceived by B. Bordelon (Section 2.1, 2.3.4). The code for all experiments in this thesis was independently developed by H. Kuo.

1

Introduction

Think about the last time you learned to perform a new task. After learning the task, you might find yourself focusing on different parts of the environment compared to before learning the task. In previous research, rats were found to change their perception of the world after learning a task (Boccaro et al., 2019; Sanguinetti-Scheck & Brecht, 2020; Ginosar et al., 2023). Why do animals change their perceptions of the environment after learning? We hypothesize that they do so to learn faster. In this thesis, we will attempt to examine this hypothesis through a computational perspective.

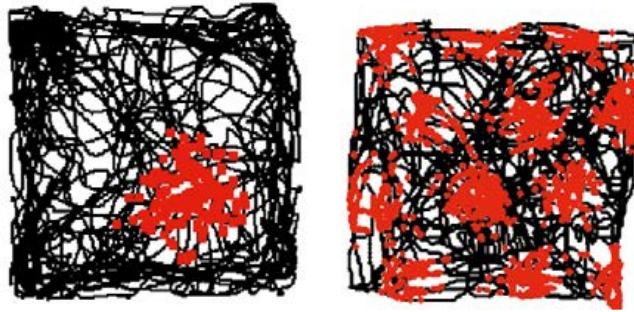


Figure 1.1: Firing patterns for a place and a grid cell. (a) Place cells only fire at specific locations in the environment. Figure from (Jeffery, 2007). (b) Grid cells firing in a periodic hexagonal pattern based on the animal's location. Figure from (Hafting et al., 2005)

1.1 PERCEPTION OF THE ENVIRONMENT

To answer this question, we must first understand how animals perceive the environment. One essential part of animals' perception of the environment involves positioning oneself. It is suggested that place cells and grid cells (Figure 1.1) may be involved in providing animals a signal to determine their position.

O'Keefe and Dostrovsky, 1971, discovered place cells in the hippocampus, which fire when the animal is at specific places in the environment. To understand how these firing patterns were generated, Hafting et al., 2005, recorded from cells one synapse upstream, in the dorsocaudal medial entorhinal cortex (dMEC). They identified neurons that may form the underlying fabric of how rats locate themselves in the environment. These cells, deemed "grid cells", fire in a periodic hexagonal pattern based on the rat's location in the environment.

How can we model animals' perception of the environment? The animal takes in a representation of the environment $s \in \mathcal{S}$, where \mathcal{S} is the set of all possible states that the agent can be in. This representation could come from different sensory inputs from the animal such as vision or odor information. Place and grid cells can then be thought of as a function ψ that maps a state to neural

firing rate:

$$\text{neural firing rates} = \psi(s). \tag{1.1}$$

For example, since place cells have maximal firing rate at a certain location μ , we can model place cells with the Radial Basis Function (RBF) as follows:

$$\psi(\mathbf{s}; \mu) = \exp(-\|\mathbf{s} - \mu\|) \tag{1.2}$$

where \mathbf{s} is the coordinate of the agent and μ is the coordinate of where the place cell fires maximally, and $\|\cdot\|$ is the norm of a vector.

We can extend the model by concurrently recording from k different place cells and grid cells at once. Therefore, the output of the function is not just a scalar, but a vector of neural firing rates. We can write ψ as a function that maps the vector state space \mathcal{S} to a vector space Ψ :

$$\psi: \mathcal{S} \rightarrow \Psi. \tag{1.3}$$

Note that ψ does not have to be a linear map, nor neural firing rates. In fact, it is useful in many cases that ψ provides some kind of non-linear mapping between state space and feature space. For example, we can map states to the Fourier basis with the dimension of the state space. For a state space with two dimensions (e.g., x - y plane), let N_1, N_2 denote the number of possible states for each axis. Let each state \mathbf{s} be represented as a columnar vector of 2 numbers, $\mathbf{s} = (s^1, s^2)$. The i th feature for the Fourier basis can be written as:

$$\psi^j(\mathbf{s}) = \begin{pmatrix} \cos(\mathbf{s}^\top \mathbf{c}^j) \\ \sin(\mathbf{s}^\top \mathbf{c}^j) \end{pmatrix} \quad (1.4)$$

where $\mathbf{c}^i = \left(\frac{2\pi}{N_1} c_1, \frac{2\pi}{N_2} c_2 \right)$, c_j be any number in $\{0, 1, \dots, N_j - 1\}$. Each feature uses a different selection of c_j s. Therefore, there are $N_1 \times N_2$ features in total, making the dimension of $\psi(\mathbf{s})$ be $N_1 \times N_2 \times 2$.

1.2 LEARNING WITH REWARDS

After the agent perceives the environment, they could use this information to perform different behaviors. Studies have surmised that grid cells are used for navigation (e.g., the problem of finding the fastest route to a goal location) (Bush et al., 2015), due to them being able to act as a global metric for space (Hafting et al., 2005).

However, the precise role of these cells is still unclear. Recent research has found that grid cells are subject to perturbations of the environment such as changing the geometry of the experimental arena (Stensola et al., 2015). This suggests that the cells provide a local metric for proximal space rather than a global metric, and the perception information could also be used for tasks such as learning (Ginosar et al., 2023; Dang et al., 2021). We believe that this may be a better characterization of the roles of grid cells.

One way animals learn is through trial and error. As the animal explores the environment, it learns to repeat actions that are more rewarding, and not perform actions that are not rewarding. To achieve this goal, animals must be able to predict how much reward they will eventually get performing an action at a state. With this prediction, they can adjust their actions to eventually maximize reward from whatever state they start from.

This prediction can also be learned through trial and error. As the animal explores the environment, sometimes the prediction may be different from reality. To update the prediction, the animal can use a feedback signal that tells the animal how far their prediction is from reality. This feedback signal is called *reward prediction error* (RPE).

Researchers have found evidence of this signal in the monkey brain (Schultz et al., 1997). Through recordings of dopamine neurons in the ventral tegmental area (VTA), it has been shown that these dopaminergic neurons may be able to represent RPE. The monkeys were first put under long periods of dehydration, then presented with various appetitive stimuli such as apple or fruit juice. When presented with the positive unexpected reward, dopamine neurons had higher activity, which represented high RPE. After the monkey got accustomed to the reward, the dopamine neurons no longer exhibited higher activity during reward presentation. This is because the brain's prediction matched reality. Interestingly, the dopamine neurons exhibited suppressed activity if rewards were removed unexpectedly.

To model the above trial-and-error learning process, we can use the theory of Reinforcement Learning (RL) (Sutton & Barto, 2018). Through employing this model on machines, researchers have successfully created RL algorithms that have been applied to many real world scenarios such as Atari Games, Backgammon, Go, and autonomous vehicles (Mnih et al., 2015; Mnih et al., 2016; Silver et al., 2016; Silver et al., 2017).

Two main components compose of the RL model: the environment and the agent. The agent interacts with the environment, receives feedback, then updates itself to perform better in the future. The environment is typically modeled as a Markov Decision Process (MDP). An MDP consists of three sets, the set of states \mathcal{S} , the set of actions \mathcal{A} , and the set of rewards \mathcal{R} . At each time step t , the agent receives some representation of the environment's state, $s_t \in \mathcal{S}$, and selects an action $a_t \in \mathcal{A}(s_t)$. On the next time step $t + 1$, the agent receives a reward $r_{t+1} \in \mathcal{R}$ and ends up in a new state, s_{t+1} .

In an MDP, the probability of an agent ending up in a new state only depends on the current state and action. This is also called the *Markov Property*. In other words, there exists a deterministic function p such that,

$$p(s', r|s, a) = \Pr[s_t = s', r_t = r | s_{t-1} = s, a_{t-1} = a] \quad (1.5)$$

for all $s', s \in \mathcal{S}, a \in \mathcal{A}$, and $r \in \mathcal{R}$. The function p is also known as the *dynamics* function, as it completely characterizes the dynamics of an MDP environment. This will be useful later in the thesis to characterize the dynamics of the environment and the behavior of rats from experimental data.

The agent's goal is to maximize the expected discounted sum of future rewards, starting from any state s_t . Specifically, the goal is to maximize the *discounted return* G_t starting from time t :

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \quad (1.6)$$

where γ is a parameter such that $0 \leq \gamma \leq 1$, also called the *discount rate*. The discount rate is important to model cases when the agent wanders indefinitely.

Many RL algorithms can be used to achieve this goal. They usually involve estimating *value functions*. Value functions are functions that estimate how good it is to be in a given state (or be in a given state and perform a certain action). This is similar to the prediction that the animal has to make to predict the consequences of their actions. For example, in the monkey example, cues are given to the monkey before rewards. If the monkey perceives no cue, the brain will predict that they will receive no reward in the future. On the other hand, if the monkey senses a cue, the brain will

predict receiving a reward shortly after.

The value function relies on what actions the agent will take in the future. For example, if the monkey decides to perform no action after receiving the cue, it should predict that no reward will be received in the future. We say that the agent is under a *policy* that defines their behavior. A policy π is a function from states to probabilities of selecting each action. If the agent is following policy π at time t , the probability that it takes action a_t upon seeing state s_t is $\pi(a_t|s_t)$.

The value function $v_\pi(s_t)$, represents the expected discounted return from s_t , under policy π :

$$v_\pi(s) = \mathbb{E}[G_t|s_t = s] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s \right] \quad (1.7)$$

where $\mathbb{E}_\pi[\cdot]$ denotes the expected value of the random variable, under the condition that the agent follows π at every time step thereafter.

The agent can only estimate this function through interacting with the environment. To make this differentiation, we use uppercase V_π to denote the estimated value function under policy π , while lowercase v_π to denote the true value function. The agent's goal is to approximate the true value function well in all states. In other words, the agent's aim is to obtain a function V_π such that:

$$V_\pi(s) \approx v_\pi(s), \forall s \in \mathcal{S}. \quad (1.8)$$

1.3 PERCEPTION AFFECTS LEARNING

There are many methods to model and estimate the true value function v_π . In this thesis, we focus on modeling the value function as a linear function of perception. In particular,

$$V_{\pi}(s; \mathbf{w}) = \psi(s) \cdot \mathbf{w} \quad (1.9)$$

where \mathbf{w} is a vector that has the same dimension as Ψ , also called “weights”. An intuitive explanation of this model is that the animal uses their perception of the environment to come up with a prediction of how much reward they expect to get at this state. As the animal explores the environment, it updates the weights \mathbf{w} to obtain a better estimate of the true value function.

There are several benefits to modeling the value function this way. One benefit is that it separates the non-linear part (ψ) from the linear part (\mathbf{w}) for ease of analysis. We can also model different perception models (ψ) and see how well they can represent the value function. Some perception models are more expressive and can estimate the true value function better.

How can we learn \mathbf{w} ? In neuroscience research, it has been found that the Temporal Difference (TD) learning algorithm may likely be employed in animals to update their predictions (Amo et al., 2022). The TD learning algorithm computes a prediction error, also known as TD error, at every time step the animal takes and uses this prediction error to update weights. The dopamine neurons described in Section 1.2 also seem to mirror TD error (Amo et al., 2022). The TD error is defined as follows:

$$\delta_t = r_t + \gamma V_{\pi}(s_{t+1}) - V_{\pi}(s_t). \quad (1.10)$$

Notice that the first part of the equation, $r_t + \gamma V_{\pi}(s_{t+1})$, resembles a more accurate prediction of future return due to the agent receiving r_t from the environment. The difference between this term and the current prediction, $V_{\pi}(s_t)$, is the RPE.

The TD error itself (Equation 1.10) does not tell us how to update the weights to improve our prediction. To find an update rule for \mathbf{w} , we can employ Stochastic Gradient Descent (SGD) methods (Robbins & Monro, 1951). These methods optimize a function by slowly adjusting the parameters to the function by the gradient of the function with respect to the parameter.

In our case, we would like to minimize the error between the true value function v_π and our prediction V_π across all states. This can be quantified by the *mean square value error* $\overline{VE}(\mathbf{w})$, the sum of squared errors through all states:

$$\overline{VE}(\mathbf{w}) \equiv \sum_{s \in \mathcal{S}} [v_\pi(s) - V_\pi(s; \mathbf{w})]^2. \quad (1.11)$$

When our prediction is exactly equal to the true value function across all states, the error becomes zero. To find the \mathbf{w} that minimizes \overline{VE} , SGD methods determine that we should update the \mathbf{w} as follows:

$$\mathbf{w}_{n+1} = \mathbf{w}_n - \frac{1}{2} \eta \nabla_{\mathbf{w}} \overline{VE}(\mathbf{w}) \quad (1.12)$$

where n is the update iteration, $0 \leq \eta \leq 1$ is the learning rate, which determines how much the agent should update their predictions on each update. Combining Equations 1.11-1.12, we obtain:

$$\mathbf{w}_{n+1} = \mathbf{w}_n + \eta \sum_{s \in \mathcal{S}} [v_\pi(s) - V_\pi(s)] \nabla_{\mathbf{w}} V_\pi(s) \quad (1.13)$$

$$= \mathbf{w}_n + \eta \sum_{s \in \mathcal{S}} [v_\pi(s) - V_\pi(s)] \psi(s). \quad (1.14)$$

Notice that the summation in Equation 1.14 iterates through all states. During learning, it is impossible to iterate through all states nor obtain the true value function. In the TD learning algorithm, we approximate by averaging across a batch of steps and using the TD error (Equation 1.10), which gives us the *TD update rule*:

$$\mathbf{w}_{n+1} = \mathbf{w}_n + \eta \frac{1}{T} \sum_{t=0}^{T-1} [r_t + \gamma V_{\pi}(s_{t+1}) - V_{\pi}(s_t)] \psi(s_t) \quad (1.15)$$

where T is the number of time steps in a batch. We can now describe the full TD learning algorithm in Algorithm 1.

Algorithm 1 TD Learning Algorithm

1. Input: the policy π
 2. Parameter: learning rate $0 \leq \eta \leq 1$, discount rate $0 \leq \gamma \leq 1$
 3. Initialize: $\mathbf{w}_0 = 0$, the zero vector
 4. For each update n from 1:
 - (a) Follow policy π for an episode, obtaining $(s_0, a_1, r_1, s_2, a_2, r_2, \dots, s_T, a_T, r_T)$
 - (b) Save new weights $\mathbf{w}_{n+1} = \mathbf{w}_n + \eta \frac{1}{T} \sum_{t=0}^{T-1} [r_t + \gamma V_{\pi}(s_{t+1}) - V_{\pi}(s_t)] \psi(s_t)$
-

Notice how the weight update is dependent on the feature function ψ . This provides a clue that different perceptions of the environment affect learning. For example, for a given reward function, there may be feature functions that would allow our agent to learn faster compared to other feature functions.

1.4 LEARNING AFFECTS PERCEPTION

In the above model, we treated ψ , the perception of the environment, as a function that does not change throughout learning. However, in previous experiments, researchers found that learning actually affects perception. For example, the properties of place cells and grid cells often change after rats learn a new task. In one experiment (Boccaro et al., 2019), rats were first lowered into an open arena to measure the activity fields of the cells. Then, the rat was trained to chase for food-rewards in three places in the arena and come back to the start location. After the training session, the activity fields of the cells were recorded again. The researchers found that in the activity fields after learning moved toward locations with reward (see Figure 2.3 for an example neuron).

Why does perception change after learning a new task? This is a question that has remained unsolved. This thesis hypothesizes that animals adapt their perception to make learning faster. Concretely, learning faster means that the value error \overline{VE} decreases faster on average. Specifically, we will (1) create a theory that predicts learning dynamics of TD learning on the average case and verify it on a complex continuous environment (2) show how perception changes for an agent that intends to learn faster, and (3) analyze if post-learning features derived from experimental data (Boccaro et al., 2019) leads to faster learning.

2

Methods and Experiments

2.1 THEORY ON THE LOSS DYNAMICS OF TEMPORAL DIFFERENCE (TD) LEARNING

In Section 1.3 we introduced TD learning, an algorithm used to estimate the value function for a certain policy. How can we predict the value error at each iteration, given ψ and other hyperparameters? As the learning process is stochastic, we are interested in the value errors of the average case.

In our previous paper, we have created a theory that predicts the value error in TD learning for the

average case (Bordelon et al., 2023).

One key assumption for this theory is the Gaussian Feature Assumption:

Gaussian Feature Assumption. *The learning curves for a TD learner with high dimensional features $\{\psi(s_t)\}_{t=1}^T$ over random trajectories τ can be approximated by the learning curves of a TD learner trained with Gaussian features $\psi_G \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma + \boldsymbol{\mu}\boldsymbol{\mu}^\top)$ where $\boldsymbol{\mu} = \langle \psi(s_t) \rangle_{\tau \sim p(\tau)}$ and $\Sigma(t, t') = \langle \psi(s_t)\psi(s_{t'})^\top \rangle_{\tau \sim p(\tau)}$.*

Here, $\langle \cdot \rangle$ denotes expectations (averages) and $\langle \cdot \rangle_{\tau \sim p(\tau)}$ is the expectation over trajectories. With this assumption, the theory predicts that the average convergence value error scales linearly to $\frac{\eta\gamma^2}{B}$, as represented in the following equation:

$$\langle \overline{VE}(\mathbf{w}^*) \rangle = \langle [v_\pi(s) - V_\pi(s; \mathbf{w}^*)]^2 \rangle \sim O\left(\frac{\eta\gamma^2}{B}\right) \quad (2.1)$$

where \mathbf{w}^* denotes final convergence weights, B is the number of batches used in each TD update. In our thesis, one batch equals one episode. While the theory matches experiments on simple discrete environments with random walk policies, do TD learners still follow this assumption on more complex continuous environments? We will test this theory on the environment MountainCar-vo.

In this section, we will first introduce the MountainCar-vo environment. Second, we explain how this environment can be solved by discretizing the state space, giving us a policy π that can consistently reach the goal. Finally, since it is difficult to find the dynamics function of the discretized state space, finding a closed solution for the true value function (v_π) is difficult. We provide a method based on the theory to evaluate value error when the true value function is difficult to obtain.

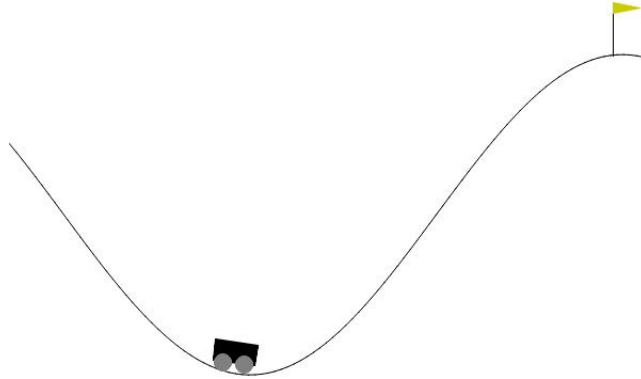


Figure 2.1: A visualization of the initial position of the car in MountainCar-v0 (Moore, 1990). The car starts at the bottom of the hill. The goal of the agent is to control the car to reach the flag at the top of the hill. Visualization by Towers et al., 2023.

2.1.1 MOUNTAINCAR-V0 ENVIRONMENT

MountainCar-v0 (Moore, 1990; Towers et al., 2023) is a continuous state space environment where the goal of the agent is to control a car to drive up a sinusoidal-shaped mountain as fast as possible (Figure 2.1). At the beginning, the car is initialized to a random position close to the bottom of the mountain. At each time step, the agent is given the car’s position along the x -axis and velocity as a real number. The agent has two possible actions at each timestep: (1) accelerate the car to the left, or (2) accelerate the car to the right. The agent receives a reward of -1 at each time step until it reaches the goal. After it reaches the goal, the agent obtains reward 0 until the end of the episode (in this thesis, 350 steps).

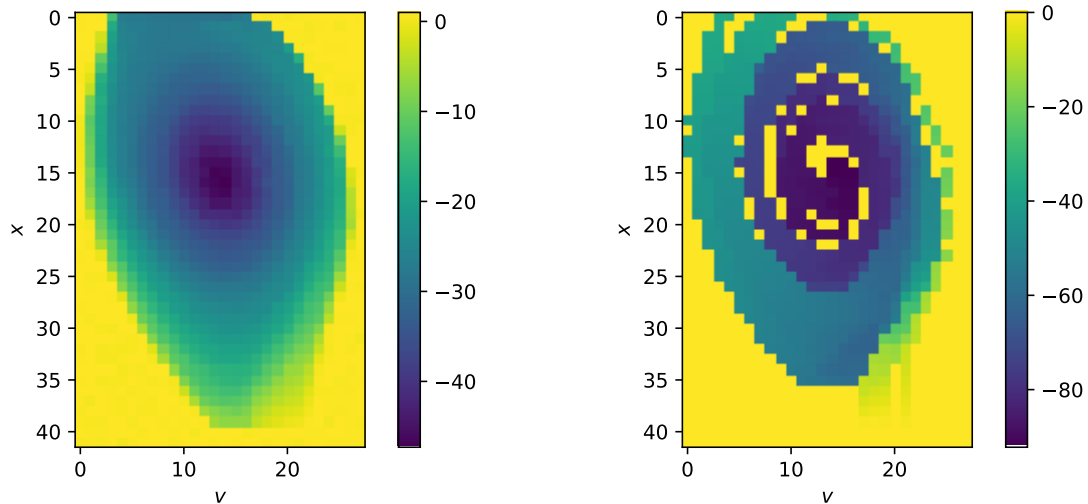
There are a couple of things that make this environment difficult to learn. First, the input state space is continuous, with two real numbers ranging $[-1.2, 0.6]$ and $[-0.07, 0.07]$. The agent must be able to handle continuous state space variables. Second, the agent must learn to first accelerate further away from the goal (to the left) first, and use gravity to swing the car to the right. The agent

cannot reach the goal only by accelerating towards the goal (to the right).

2.1.2 DISCRETIZED TABULAR ϵ -GREEDY Q LEARNING

To ease the analysis of computing the value error at each learning iteration, we discretize the position and velocity into 42 and 28 states, respectively. We can then use the tabular ϵ -greedy Q Learning algorithm (Sutton & Barto, 2018; C. Watkins, 1989; C. J. C. H. Watkins & Dayan, 1992) to learn a policy π that is not optimal but consistently reaches the target within 350 time steps. For the Q Learning algorithm, we use parameters $\epsilon = 0.1, \gamma = 0.99, \eta = 0.01$. ϵ is the probability of the agent executing an action that is not optimal at each time step to explore the environment, γ is the discount factor, and η is the learning rate. The output of the algorithm is the value function $Q(s, a)$, which is the expected discounted return of the agent executing action a at state s . After learning this Q function, the agent can use the policy $\pi(s) = \arg \max_a Q(s, a)$ to optimize future return. Note that the parameter ϵ is only used during the training process, as the agent does not need to explore the environment after training. For brevity, we will not explain the algorithm in detail. What is important is that this algorithm can give us the Q function and a policy that can solve the MountainCar-vo environment consistently.

Discretizing the environment enables us to easily learn a policy, but makes the environment stochastic. This is because the agent may be in any of the continuous states represented by a discretized bin. Depending on where the agent is in the discretized bin, the agent could jump to two or more different other bins in the next state. This makes it difficult to recover the dynamics function (Equation 1.5) of MountainCar-vo in a closed form because the probability of the agent transitioning from one discrete state to another depends on where the agent is located within the discretized bin.



(a) V^* estimated with tabular ϵ -greedy Q-Learning

(b) V^π TD Converged Value Function

Figure 2.2: Value functions of MountainCar-v0 learned by Tabular Q-Learning and TD Learning. (a) Value function learned by Tabular Q-Learning that approximates the value function of an optimal policy. (b) An example value function of a policy (V^π) learned by TD learning. Notice that the value function does not equate to that in (a) due to the policy π not reaching all states in the environment.

2.1.3 COMPUTING THE TRUE VALUE FUNCTION

The above challenges of discretizing an environment motivate us to find an iterative method to compute the true value function. Though it is possible to obtain a value function from the Q function learned using the Q learning algorithm, the value function we obtain is not for the policy we intend to evaluate.

Figure 2.2 shows the differences between the value functions computed from the Q function and the value function computed by TD learning. We can see that the value function computed by TD learning has more unknown values due to the policy seldom reaching those states during learning. Q-Learning, on the other hand, reaches those states due to its exploration of the environment. In addition, the converged value function for TD learning is very different from that of Q-Learning. We can see that the lowest value in the value function of TD learning can be as low as -80, where

the lowest value in Q-Learning is only -40. Due to the different state distributions and different convergence value function of ϵ -greedy Q-Learning and TD Learning, we cannot directly use the optimal value function as the true value function to compute the value error in TD Learning.

To obtain the true value function, we run TD learning with the smallest learning rate and largest batch size, on the most number of iterations. This is motivated by the theory (Equation 2.1), as convergence error scales linearly to $O\left(\frac{\eta}{B}\right)$. As we are only looking at linear relations between the convergence value error and η/B , this remains a valid method to determine the true value function for our purposes.

2.1.4 FOURIER BASIS AS FEATURES

The features we chose for this experiment is the full Fourier basis (Equation 1.4) for its representation power and ease of computation. This means that we should be able to learn any value function using this feature function. In our discretized version of MountainCar-v0, for each state \mathbf{s} , the dimension of $\psi(\mathbf{s})$ is $42 \times 28 \times 2$.

2.2 EVOLVING FEATURES TO LEARN FASTER

The theory does not provide us with insight on what features would make TD learning faster. We would need to empirically find features that would make learning faster. Would the obtained features match observations in neuroscience experiments? In (Menache et al., 2005), the authors proposed an algorithm that could find features for TD learning that would minimize convergence TD error. In the algorithm, they evolved the feature function using the cross-entropy (CE) adaptation algorithm (de Boer et al., 2005). In this section, we explain the cross-entropy basis adaptation algorithm modified to evolve features that would allow a TD learner to learn faster.

Let the feature function be parameterized by θ , i.e., $\psi(\cdot; \theta)$. For example, in the case of RBFs

(Equation 1.2), this θ could be the centers of the place cells. Let θ be sampled from a probability density function (pdf) $f(\cdot; \nu)$, where ν is a meta-parameter that parameterizes the pdf. For example, if f is the Gaussian pdf, ν would consist of the mean and variance of the gaussian. The CE algorithm's goal is to find the best θ that would speed up learning the most. The high-level overview of this algorithm is to sample a few θ s from f at each iteration and measure scores that represent learning speed. With this information, the algorithm adjusts ν such that it is more likely to sample θ s that have better scores in the next iteration.

We provide more details about the algorithm as follows. First, candidate parameter vectors $\theta^1 \dots \theta^N$ are independently sampled from $f(\cdot | \nu_m)$, where ν_m denotes the ν at iteration m . Then, the scores for each parameter $S(\theta^j), j = 1 \dots N$ are computed. In our case, the score function is defined as:

$$S(\theta^j) = \overline{VE}(\theta_{500}^j) \quad (2.2)$$

where \overline{VE} is the mean square value error (Equation 1.11), θ_{500}^j is the weights after running TD RL for 500 iterations (lower is better for the score function). After finding the scores, we order the scores in ascending order $S_{(1)} \leq S_{(2)} \leq \dots \leq S_{(N)}$. We take the lowest k scores. Let $S_{(k)} \leq S \leq S_{(k+1)}$. Then we solve the following to determine the new ν :

$$\nu_{m+1} = \arg \max_{\nu} \sum_{j=1}^N I_{\{S(\theta^j) \leq S\}} \log f(\theta^j; \nu) \quad (2.3)$$

where I is the indicator function. Equation 2.3 essentially maximizes the probability of the top scoring θ s appear in the next iteration.

For Gaussian f , Equation 2.3 has a closed-form solution (Rubinstein & Kroese, 2004). Assume each element θ_q in θ is sampled independently from a Gaussian with mean μ_q and variance σ_q^2 , $\theta_q \sim \mathcal{N}(\mu_q, \sigma_q^2)$. Then for iteration $m + 1$,

$$\begin{aligned}\mu_q^{(m+1)} &= \frac{1}{k} \sum_{j=1}^N I_{\{S(\theta^j) \leq S\}} \theta_q^j, \\ \sigma_q^{2(m+1)} &= \frac{1}{k} \sum_{j=1}^N I_{\{S(\theta^j) \leq S\}} \left(\theta_q^j - \mu_q^{(m+1)} \right)^\top \left(\theta_q^j - \mu_q^{(m+1)} \right).\end{aligned}\tag{2.4}$$

In other words, the new mean is the mean of the top-scoring θ s in the previous iteration, and the new variance is the mean of the variances of the top-scoring θ s in the previous iteration. We can summarize the algorithm in Algorithm 2.

Algorithm 2 Cross-entropy Based Basis Adaptation Algorithm

1. Parameter: k , the number of top scores to use.
 2. Initialize: ν_1 , the initial parameter from which we will sample θ from.
 3. For each update m from 1:
 - (a) Generate $(\theta^1, \dots, \theta^N)$ from pdf $f(\cdot; \nu_m)$.
 - (b) For every $j = 1 \dots N$
 - i. Estimate $S(\theta^j)$ by running TD Learning using $\psi(\cdot; \theta^j)$ as the features and obtain the score using Equation 2.2.
 - (c) Compute the new ν_{m+1} using Equation 2.3, or Equation 2.4 if f is a Gaussian.
-

2.3 WORKING WITH EXPERIMENTAL DATA

To test our hypothesis that animals change their perception to learn faster, we seek to obtain features of rats before and after learning, then run TD learning using the features and observe if TD learning is faster with the features after learning.

The neural data is from Boccara et al., 2019. The authors of the paper provided us with full neural recordings from their paper, which we used for our analysis. We obtained neural recordings from the medial entorhinal cortex (MEC) of rats, before and after the rats learn the locations of three reward locations on a cheeseboard maze. The neural activity data is aligned with the location and velocity of the rat, which provides us with enough information to reconstruct the activity fields of the recorded neurons. For each recording trial, the rat sees a new cheeseboard maze with rewards at different randomly selected locations. Within a trial, the same neurons are recorded before and after the rats learn the locations of the rewards. In total, we have data from 4 rats, 13 recording sessions, and 262 neurons.

In this section, we describe how to process the neural recording data and create the simulation environment. First, spatial firing rate maps are generated from the neural activity data. Second, the rate maps are normalized to generate features for TD learning. Third, we discuss the construction of a gridworld-like environment that simulates the cheeseboard maze and the behavior of the rat. Finally, we explain a quantitative method based on the TD learning theory to determine whether a feature is better for learning.

2.3.1 GENERATING NEURONAL SPATIAL FIRING MAPS FROM NEURAL ACTIVITY DATA

The first step of creating features from the recording data is to generate neuronal spatial firing rate maps from the data. The spatial firing rate maps can be thought as a probability distribution that gives rise to the observed neural activity data. Since the data is noisy, we need to use methods to interpolate and give an estimate of the underlying probability distribution. Kernel density estimation methods, in particular, are non-parametric methods that use kernels as weights to smooth the data. Here, our input data are sampled pairs of location and neuron activity. We choose the Triweight Kernel (Dunn et al., 2017), the same method used in Boccara et al., 2019, to create the spatial firing rate maps. This kernel has the benefit of having compact support and is widely used to create spatial

firing rate maps. We will briefly explain how to create the neural firing rate map of a single neuron using the 2D Triweight Kernel and show example firing rate maps from the data.

Let T be the total number of sampled location and activity pairs we obtain from neural recordings. Let S be the number of sampled pairs only when the neuron is firing. Let $\mathbf{x}_i = (x_i, y_i), i = 1 \dots T$ be all the locations that the rat has visited, and $\xi_j = (\xi_j, \chi_j), j = 1 \dots S$ be the locations of where the neuron has fired. Let a_j be the number of spikes recorded from the neuron at time j . Then the spatial firing rate at location $\mathbf{x} = (x, y)$ is defined as follows:

$$f(\mathbf{x}) \equiv \frac{\sum_{j=1}^S a_j K(\mathbf{x}|\xi_j)}{\sum_{i=1}^T \Delta t K(\mathbf{x}|\mathbf{x}_i)} \quad (2.5)$$

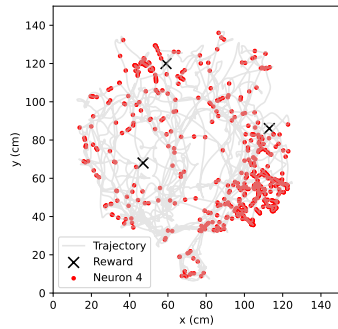
where Δt is the time difference between each sample. The 2D Triweight Kernel K is defined as:

$$K(\mathbf{x}|\xi) \equiv \begin{cases} \frac{4}{9\pi\sigma^2} \left[1 - \frac{\|\mathbf{x}-\xi\|^2}{9\sigma^2} \right]^3, & \|\mathbf{x} - \xi\| < 3\sigma \\ 0, & \text{otherwise,} \end{cases} \quad (2.6)$$

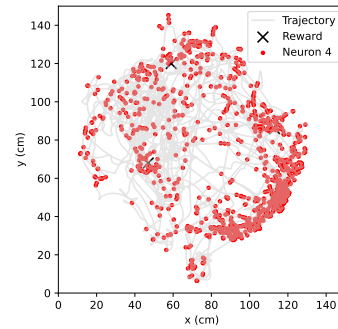
where σ is a parameter that controls the degree of smoothing. The larger the σ , the more points are considered and averaged together to form the activity rate at location \mathbf{x} . The coefficients are chosen to ensure the function K integrates to one within a 2D circle with radius $r = 3\sigma$:

$$\begin{aligned} & \int_{r=0}^{r=3\sigma} \frac{4}{9\pi\sigma^2} \left[1 - \frac{r^2}{9\sigma^2} \right]^3 2\pi r dr \\ &= \frac{8}{9\sigma^2} \cdot \frac{9^4 \sigma^8}{5832\sigma^6} = 1. \end{aligned}$$

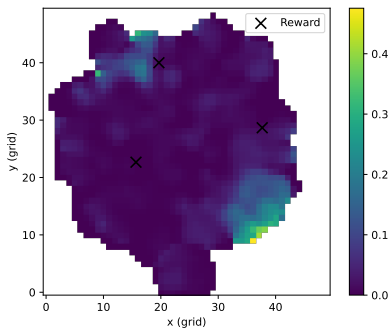
In this thesis, we chose $\sigma = 3$ cm to divide the 150 cm x 150 cm cheeseboard maze into 2,500 3 cm x 3 cm bins, same as the data preprocessing methods in (Boccaro et al., 2019). We also only chose data when the rat moved with speed > 3 . This ensures that the rate map does not use data that



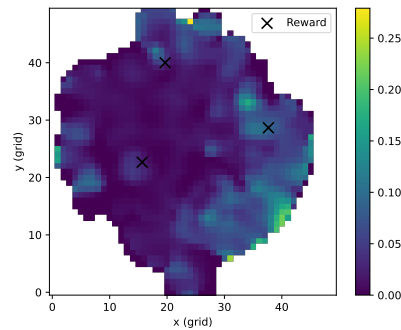
(a) Pre-learning Trajectory



(b) Post-learning Trajectory



(c) Pre-learning Rate Map



(d) Post-learning Rate map

Figure 2.3: Example trajectories and rate maps before and after the rat learns the reward location. Data from *mj c2-3*, neuron 4. (a,b) Original trajectories of the rat and the locations where the neuron fired. (c,d) Firing rate maps generated with the Triweight kernel. Experimental data collected by [Boccaro et al., 2019](#).

contains sensor artifacts.

Figure 2.3 shows the rate maps of an example neuron generated with the Triweight kernel. Note that the rate map pre-learning and post-learning have different shapes since the rat explores different parts of the cheeseboard maze during the two recordings. We also notice that in pre-learning, the rat would wander around in the cheeseboard maze, while in post-learning, the rat would directly head to the reward locations after being released into the maze.

2.3.2 NORMALIZING SPATIAL FIRING RATE MAPS AS FEATURES

After generating the firing rate maps, we can use them to create features in learning. Recall that perception is a function that maps state space to feature space (Equation 1.3). In our simulations, we choose ψ as the function that maps a certain state s to a vector of the neuronal firing rates at state s of k corresponding rate maps. For example, if we recorded data from 4 neurons in a trial, the function ψ maps each state to a vector of size 4.

This introduces the problem of normalizing the rate maps as it may affect learning dynamics. For example, let $c > 1$ be a constant. Comparing the feature maps $c\psi(\cdot)$ and $\psi(\cdot)$, we can absorb c into the learning rate in the TD update rule (Equation 1.15). Therefore, TD learning on the former feature map is essentially TD learning on the latter with a higher learning rate. From the theory (Section 3.1), we would expect the former to have faster error convergence rate with a higher convergence error.

We compare two different methods of normalizing the rate maps. The first method is no normalization, directly using the raw rate map values. The second method is to normalize the highest firing rate within a rate map to 1.

For each trial, we have two different feature maps: (1) The pre-feature map, and (2) the post-feature map, generated from the neural recordings before and after the rat learned the food locations, respectively.

2.3.3 GRIDWELL ENVIRONMENT AND POLICY

Next, we discuss how to model the cheeseboard maze and the behavior of the rat. A 50 x 50 grid world was used to simulate the discretized cheeseboard maze. The environment is set up such that the agent receives a reward of 1 at the three reward locations and 0 otherwise. The three reward locations are the same ones as in the data.

In our experiments, we evaluated the pre- and post-learning features on three policies. The first policy is the random policy (π_{rand}). For this policy, the agent chooses to move left, right, up, and down randomly at each step. The second and third policies are based on the behavior of the rat (π_{pre}, π_{pos}). Evaluating these policies is motivated by the observation that the behaviors of rats could change drastically throughout learning (Figure 2.3). How can we model the policy of rats based on observations of the rat?

One method is by approximating the probability of a rat transitioning from one state to another, $P(s_{t+1}|s_t)$. A naïve method would be to estimate this probability by looking at how many times the rat transitions to another state from a starting state. For example, suppose the rat completed 5 state transitions that start from position (25, 25). If we observe that for 3 out of the 5 transitions, the rat will end up at (24, 25) at the next time step, and for 2 transitions the rat will end up at (26, 26), then the transition probability $(25, 25) \rightarrow (24, 25) = \frac{3}{5} = 0.6$, and $(25, 25) \rightarrow (26, 26) = \frac{2}{5} = 0.4$.

This effort is futile because of the number of transitions required to construct the *transition probability matrix*. The transition probability matrix is the matrix \mathbf{P} where $\mathbf{P}_{ij} = P(s_{t+1} = j|s_t = i)$ is the probability of the rat transitioning from state i to j . This matrix has around $50^4 = 6,250,000$ entries. However, for each of the 2,500 states, there are usually only less than 10 transitions to compute the transition probability in the experimental data, with some of the states having only 1 to 3 transitions. This motivates a need for a method to interpolate the data.

Notice that,

$$P(s_{t+1}|s_t) = \frac{P(s_{t+1}, s_t)}{P(s_t)} \quad (2.7)$$

by Bayes rule. If we can estimate the two distributions s_t, s_{t+1} and s_t from the data, then we can com-

pute $P(s_{t+1}|s_t)$. Fortunately, kernel density methods can be used to estimate these two distributions. The key insight is to view each transition as a 4D data point for the distribution s_t, s_{t+1} . We can adjust the constants for the 4D triweight kernel, which can be obtained by making the kernel integrate to 1 within a 4D sphere:

$$K_{4D}(\mathbf{x}|\xi) \equiv \begin{cases} \frac{30}{\pi} \frac{1}{(3\sigma)^4} \left[1 - \frac{\|\mathbf{x}-\xi\|^2}{9\sigma^2} \right]^3, & \|\mathbf{x} - \xi\| < 3\sigma \\ 0, & \text{otherwise,} \end{cases} \quad (2.8)$$

where we chose $\sigma = 2$ to estimate $P(s_{t+1}, s_t)$. The distribution $P(s_t)$ can be estimated similarly with a 2D triweight kernel.

Note that what we estimate is actually $p(\cdot, \cdot | s, \pi(s))$, where p is the environment dynamics function (Equation 1.5). However, this is sufficient for TD learning (Algorithm 1) as the algorithm only uses the state to state transition probabilities.

When constructing the matrix \mathbf{P} , we did additional processing to remove artifacts and speed up learning. In particular, we only computed the matrix entries where $\|s_{t+1} - s_t\| < 1.5$ (the 8 grids surrounding s_t). In addition, we also let $P(s_{t+1}|s_t) = 0$ for all $s_{t+1} = s_t$. This is so that the agent would keep moving in TD learning. After pre-processing, we normalize to create the probability matrix \mathbf{P} .

It is possible that \mathbf{P} does not have values for all states, or makes the agent move to a state that does not have access to other states. Therefore, we need to check connection probabilities of the graph induced by the matrix \mathbf{P} (Tarjan, 1972). Concretely, we create a directed graph $G(V, E)$ such that edge $(u, v) \in E \Leftrightarrow \mathbf{P}_{uv} > 0$. We can then analyze the number of strongly connected components (SCCs) and weakly connected components (WCCs) using the Python NetworkX package (Hagberg et al., 2008). An SCC is a set of vertices in the graph such that for each pair of vertices (u, v) , there exists a path $u \rightsquigarrow v$ and $v \rightsquigarrow u$. A WCC is a set of vertices in the graph such that for each pair of

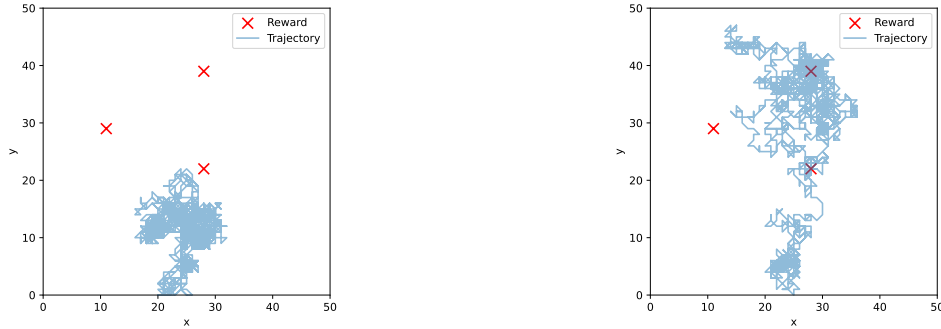


Figure 2.4: Example mjc4-2 trajectories sampled from the approximated policies for 3,000 time steps. (Left) Sampled trajectory from π_{pre} . (Right) Sampled trajectory from π_{pos} . The policy π_{pos} directly heads toward the reward locations, hence the trajectory covers more reward locations. Policies estimated from experimental data collected by Boccara et al., 2019.

vertices (u, v) , there exists a path $u \rightsquigarrow v$ or $v \rightsquigarrow u$. To mitigate the effect of some graphs not being strongly connected, we start the agent at a random vertex in the graph at the start of each episode.

Furthermore, we also check if all the reward locations are reachable in the graph.

Moreover, since the rats may not discover every location in the cheeseboard maze, the rate maps (features) may not contain all states in the 50×50 grid. We compute a set of valid states where we have data to compute the rate map. Using this set, only errors on the valid states count towards the value error. Similarly, the agent is made to stay at the original state if it wants to transition to an invalid state.

We can verify the approximated policy is similar to the rat’s policy by plotting the trajectories of the approximated policy with \mathbf{P} . For the approximated pre-learning policy (π_{pre}), the agent wanders randomly on the field. On the other hand, the approximated post-learning policy (π_{pos}) will rush towards the three reward locations (Figure 2.4).

Finally, modeling the rat’s behavior with \mathbf{P} has the added benefit of allowing us to easily compute the true value function in closed form. Using the Bellman equations (Sutton & Barto, 2018), we derive the true value function:

$$v_\pi = (\mathbf{I} - \gamma\mathbf{P})^{-1} \mathbf{R}, \quad (2.9)$$

where we take the Moore-Penrose pseudo-inverse, since $\mathbf{I} - \gamma\mathbf{P}$ is not necessarily full rank.

2.3.4 SPECTRAL ANALYSIS

Plotting the learning curves with features from experimental cell data is not enough to determine if the features after the rat learns the location of rewards are better for learning. This motivates a method to isolate the effect of features from other parameters such as learning rate. For example, higher learning rates in TD Learning will lead to faster value error loss but higher convergence value error (Section 2.1). If we see a learning curve for feature ψ_1 that has faster initial value error loss but higher convergence value error compared to that of another feature ψ_2 , this can be due to unfair normalizations of the experimental cell data that leads to overall higher values in ψ_1 compared to ψ_2 (e.g., in the case where ψ_1 equals a constant factor times ψ_2).

Given the difficulties of normalizing experimental cell data, we would like to quantitatively analyze if a feature is aligned to the reward function. One method is to use the derived TD Learning theory (Section 2.1, Bordelon et al., 2023) to predict the average weights at each iteration using a spectral perspective.

Let the matrix $\mathbf{A} = \bar{\Sigma} - \gamma\bar{\Sigma}_+$ and decompose it into its eigenvectors and eigenvalues $\mathbf{A}\mathbf{u}_k = \lambda_k\mathbf{u}_k$, where the eigenvalues λ_k may be complex. Let the true weights of the value function be in this basis $\mathbf{w}_{TD} = \sum_k w_k\mathbf{u}_k$. The theory predicts that on average, the weights at the n th iteration will be:

$$\langle \mathbf{w}_n \rangle = \sum_k |1 - \eta\lambda_k|^n \exp(n\theta_k i) w_k \mathbf{u}_k, \quad (2.10)$$

where $|\cdot|$ is the complex modulus and $\theta_k = \text{Arg}(1 - \eta\lambda_k)$. We can then order the modes by $|1 - \eta\lambda_k|$. Given this ordering, tasks that can be learned efficiently are those with most of the norm of w_k in the

modes with small timescales. Therefore, we can compute the function

$$C(k) = \frac{\sum_{l < k} w_l^2}{\sum_l w_l^2} \quad (2.11)$$

to quantify how well aligned a task is to a feature. If this quantity rises rapidly with k , then the task can be learned from a small number of samples (Canatar et al., 2021).

3

Results

3.1 VERIFICATION OF THEORY IN MOUNTAINCAR-VO

We first present the verification of the Gaussian Feature Assumption (Section 2.1) in MountainCar-vo. The results are summarized in Figure 3.1. As expected, we see that the value error plateaus to an error level determined by both the learning rate (Figure 3.1a) and batch size (Figure 3.1c). Plotting the mean convergence value errors across different learning rates and batch sizes also reveals that the

convergence value error is proportional to $O(\frac{\gamma}{B})$.

The results show that the convergence value errors depend on the choice of learning rate and batch size. This is because in each time step of learning, old estimates of the value function are bootstrapped to determine the new estimate of the value function. For larger learning rates, the old estimate of the value function is typically larger, which gives rise to new estimates that deviate from the true value function. This process repeats until the algorithm converges at a value function that differs from the true value function. In other words, in reinforcement learning, increasing the learning rate also induces bias in the resulting value function estimate.

Similarly, larger batch sizes typically give less noisy estimates of the value function at each iteration. This creates better estimates at each learning iteration and results in a smaller bias in the convergence value function estimate in the next iteration.

Notice that there are wider differences in convergence learning rates between seeds for the largest learning rate. This may be because, in higher learning rate regimes, the learned value function may converge to local optima, fail to converge, or widely fluctuate between different runs.

3.2 CROSS-ENTROPY ADAPTATION ALGORITHM

Figure 3.2 shows the results of the Cross-entropy Adaptation Algorithm (Algorithm 2) for a single trial (mjc1-2). The results of other trials can be found in Section A.1. In this experiment, a total of 50 features in the form of RBFs were used and ran on 100 iterations of optimizing the meta parameters in Algorithm 2. The meta parameters were initialized to a random reachable location determined by the transition probability matrix of π_{pos} . In each iteration, 20 different feature functions were sampled, where only the top performing 5 were selected to evolve the meta parameters. Evaluation is done through TD Learning with batch size $B = 1$ and 500 batches. The width of the RBFs are not evolved to better examine the evolution of the centers.

From Figure 3.2(c, e), we can see that the centers of the final meta parameters have evolved to become closer to the reward locations. We can verify that the final meta parameters produce features that allows faster learning in Figure 3.2(d).

Interestingly, we also observe that not all RBF centers moved toward the reward centers. In fact, the RBF centers that are furthest from the reward locations became even further away from the reward centers.

3.3 TD LEARNING CURVES OF EXPERIMENTAL DATA

In this section, we present the results of TD learning with features from experimental data. Table 3.1 shows some properties of the data. We can see that in each trial, we have recordings of 12-45 neurons from rats mjc2-4, while we only have recordings of 3-5 neurons from the rat mjc1. Note that this number of neurons is still significantly less than the high-feature-dimension assumption in the TD learning theory (Section 2.1). For example, the Fourier basis in the MountainCar-vo simulation (Section 3.1) has a feature dimension of $42 \times 28 \times 2 = 2,352$. Therefore, some of the theoretical results from the theory may not directly translate to results based on experimental data.

From the data, we can also see that there are balanced amounts of data points gathered from each trial. Here, each data point is a measurement of the rat's x, y location, speed, and the number of times each neuron fired within a 25.6ms bin. This means that there are plenty of data points to construct the firing rate map and transition probability matrix \mathbf{P} .

Furthermore, both the data from before learning and after learning induce transition probability matrices that are not strongly connected. The trials that have graphs that are not strongly connected does not seem to have a pattern. The number of SCCs is relatively small, with the maximum being 4. All policies are able to reach the 3 target reward locations.

In the sections below, we analyze the results of TD learning on π_{rand} , π_{pre} , and π_{pos} with no fea-

ture normalization. We also have learning curves with peak normalization for evaluating π_{pos} in Section A.2.

3.3.1 RANDOM POLICY AND PRE-LEARNING POLICY

Figure 3.3 shows the results of TD Learning on π_{rand} . We can see that in 11 of the 13 trials, the convergence value error of ψ_{pos} is higher than that of ψ_{pre} . In some of the trials, such as mjc2-3, the value error for ψ_{pos} decreases faster but converges to a higher value error. This is similar to the learning dynamics of an agent with higher learning rate. In general, there does not seem to be a significant improvement in using the features post-learning than in pre-learning, except for mjc1-2 and mjc2-4. This means that the features used post-learning may not be suitable for evaluating the random policy.

Furthermore, Figure A.2 shows the results of TD Learning on π_{pre} . We can see results similar to the ones on π_{rand} , even though π_{pre} is generated from real rat trajectories. This is may be due to the similarities in the true value functions $v_{\pi_{rand}}$ and $v_{\pi_{pre}}$ (Figure A.4-A.6). This suggests that π_{pre} is similar to π_{rand} as the rat was exploring the cheeseboard maze.

3.3.2 POST-LEARNING POLICY

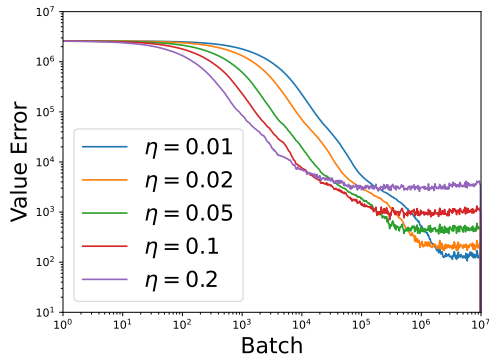
Figure 3.4 and Figure A.3 show the results of TD Learning on π_{pos} . In all trials but mjc3-3, we can see lower convergence value errors and/or higher convergence speeds for ψ_{pos} compared to ψ_{pre} . This supports the hypothesis that the features after learning could help the rat learn faster.

Comparing the learning curves of π_{pos} and π_{rand} , we can see that evaluating TD Learning with π_{pos} provides us learning curves that have lower convergence variances compared to π_{rand} . This suggests that the features are more suitable to learn the value functions derived from policies more similar to the behaviors of the rat.

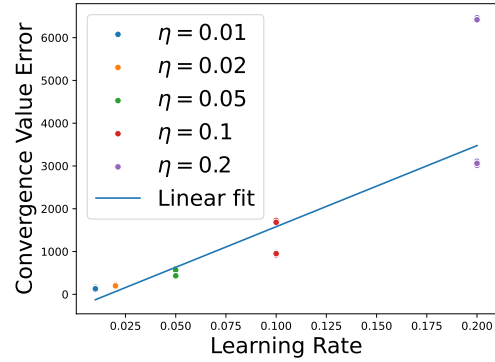
Different regularization techniques also does not seem to qualitatively affect the TD learning curves (Figure A.3). For peak normalization, in most trials, we can see that ψ_{pos} still performs better than ψ_{pre} . We can see that though the variance between seeds is higher for peak normalization, the qualitative differences between the learning curves of features ψ_{pre} and ψ_{pos} remain mostly the same.

3.3.3 SPECTRAL ANALYSIS

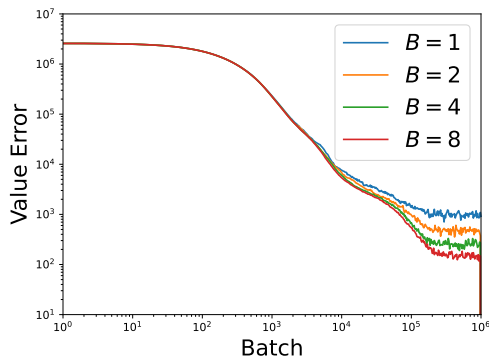
Figure 3.5 show the results of spectral analysis on different trials. We can see that spectral analysis somewhat supports the learning curves in Figure 3.4. For 8 out of 13 trials (mjc1-2, mjc2-1, mjc2-2, mjc2-3, mjc2-4, mjc3-2, mjc4-1, and mjc4-2), the function $C(k)$ rises faster with k . We can verify that these learning curves converge to the final weights faster.



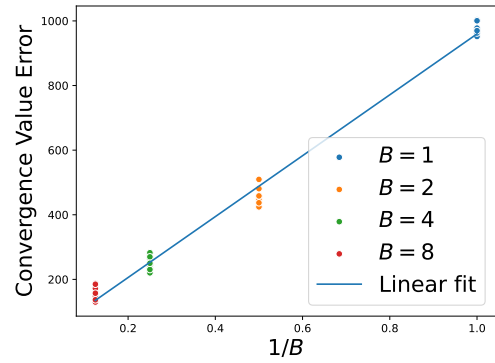
(a) Learning curves of varying Learning Rates



(b) Scaling of value error with Learning Rate

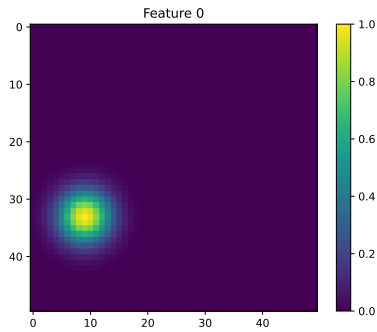


(c) Learning curves of varying Batch Sizes

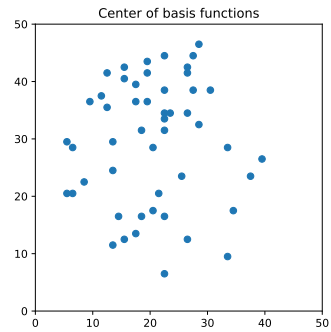


(d) Scaling of value error with Batch Size

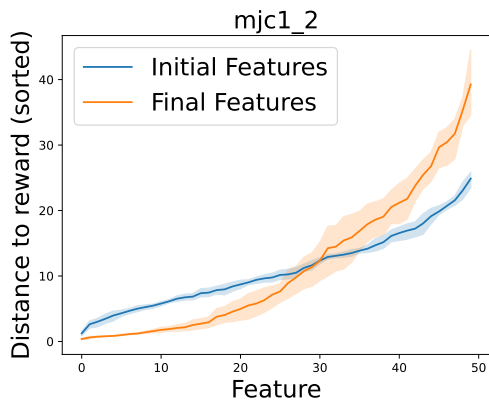
Figure 3.1: Results of testing learning curves in a MountainCar-v0 environment. (a) Learning curves of policy evaluation with varying learning rates when $B = 1$. Convergence value errors were computed by averaging the 100k batches before batch 10M. (b) Linear scaling of convergence value error with the learning rate. (c) Learning curves of policy evaluation with varying batch size B when $\eta = 0.1$. Convergence value errors were computed by averaging the 100k batches before batch 1M. (d) Linear scaling of convergence value error with the inverse of batch size. (a-d) Shaded area denotes 95% confidence interval. The target value function is the same across both experiments. Each dot represents a different seed. A total of 10 seeds were used.



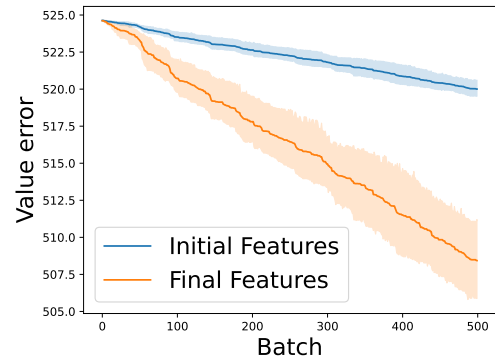
(a) Example radial basis function (RBF) feature



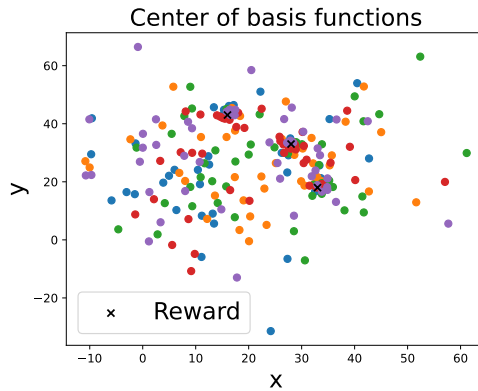
(b) Example initial centers of RBFs



(c) Distance of mean RBF centers to nearest reward location



(d) TD learning curves of initial features and final features



(e) Centers of the RBFs after evolution

Figure 3.2: Results of Cross-entropy based basis adaptation algorithm (Algorithm 2) with π_{pos} from trial mjc1-2. (a) An example of an RBF feature with width 10. The peak of the RBF is set to 1. (b) An example of initial centers of RBFs. (c) The distance of RBF centers to nearest reward locations sorted from smallest to largest. (d) An example TD Learning curve of the top performing initial features and final features. Batch size $B = 1$, $\eta = 0.01$. (c-d) Shaded area denotes 95% confidence interval. A total of 5 seeds were used. (e) The centers of the RBFs after learning for 5 seeds. Each color represents the final centers from a seed.

Policy		Data Points	Filtered Data Points	Neurons	SCC	WCC	Reward Reachable?
pre	mjc1-1	45184	37468	5	3	1	Y
	mjc1-2	50903	40320	3	2	1	Y
	mjc1-3	44378	37106	3	1	1	Y
	mjc2-1	70224	46603	17	3	1	Y
	mjc2-2	63657	33949	42	1	1	Y
	mjc2-3	60708	27554	33	2	1	Y
	mjc2-4	49145	26293	45	1	1	Y
	mjc2-5	46566	28815	33	1	1	Y
	mjc3-1	53128	24934	15	3	1	Y
	mjc3-2	46015	23879	17	3	1	Y
	mjc3-3	63851	32103	20	2	1	Y
	mjc4-1	34788	23065	12	3	1	Y
	mjc4-2	38186	23803	17	2	1	Y
pos	mjc1-1	65722	50588	5	3	1	Y
	mjc1-2	59402	45725	3	1	1	Y
	mjc1-3	46648	35711	3	3	1	Y
	mjc2-1	46564	26475	17	1	1	Y
	mjc2-2	36267	23862	42	4	1	Y
	mjc2-3	72877	39361	33	1	1	Y
	mjc2-4	47851	28398	45	2	1	Y
	mjc2-5	50843	31339	33	1	1	Y
	mjc3-1	54429	28668	15	3	1	Y
	mjc3-2	93303	46802	17	2	1	Y
	mjc3-3	68497	26551	20	1	1	Y
	mjc4-1	46090	26135	12	2	1	Y
	mjc4-2	115889	50654	17	1	1	Y

Table 3.1: Properties of experimental data. “pre” represents data collected before the rat learns the location of the rewards. “pos” represents data collected after the rat learns the location of the rewards. “mjc i - j ” represents the i th rat’s j th trial. “Data Points” is the total number of usable data points, used to generate the transition probability matrix of the policy. “Filtered Data Points” is the number of data points used to generate the spatial firing rate map, which is filtered by $speed > 3$. “SCC” represents the number of strongly connected components (SCC) (Tarjan, 1972) in the transition probability matrix. An agent can move from any state to any other state within an SCC. “WCC” represents the number of weakly connected components (WCC) (Graham et al., 1972). In an WCC, there exists a path between any two states. “Reward Reachable?” represents if the reward location is present in the transition probability matrix. Experimental data collected by Boccara et al., 2019.

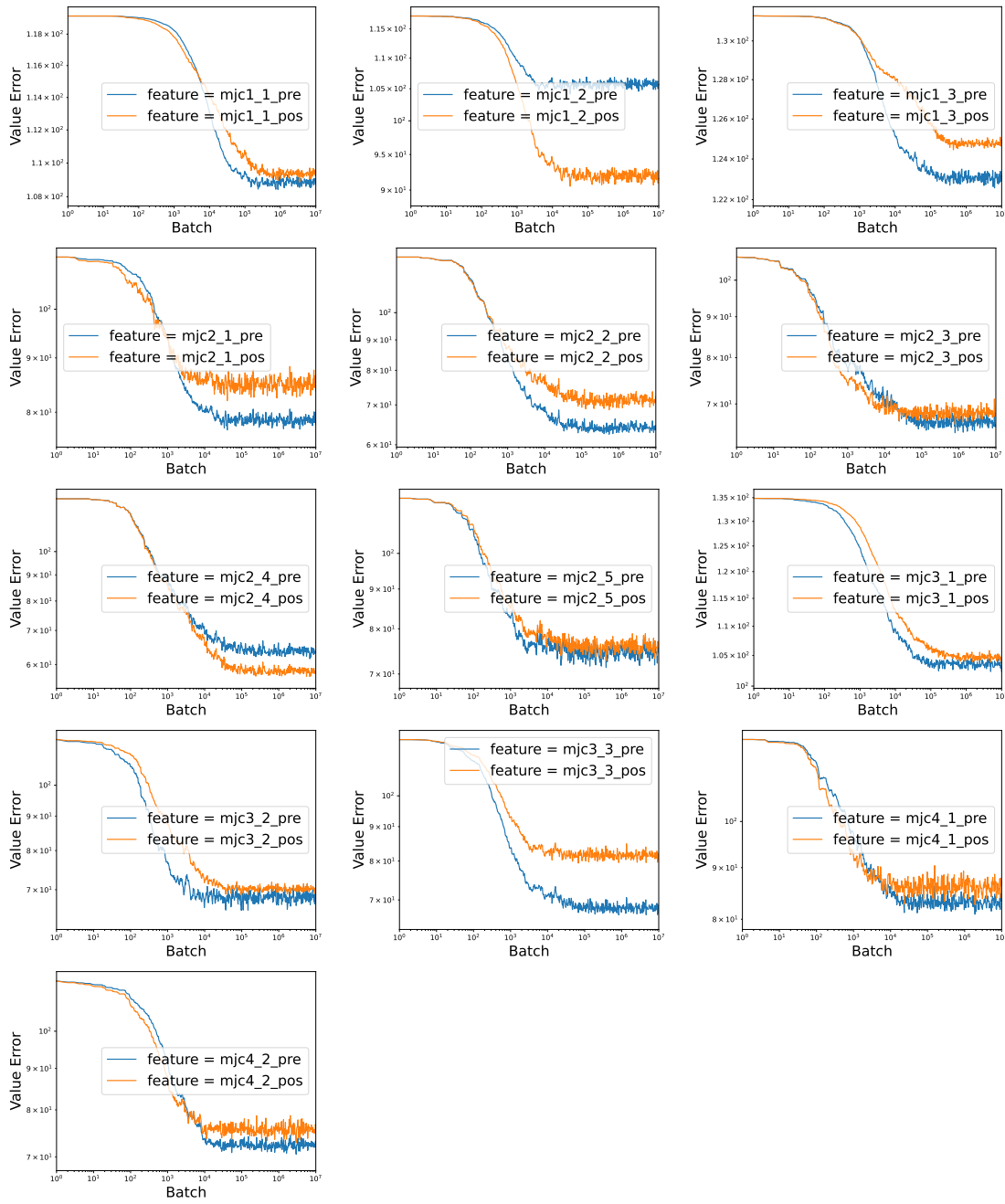


Figure 3.3: Learning curves of TD Learning evaluated on the random policy (π_{rand}) with no normalization. The learning rate is set to 1.0. Shaded area denotes 95% confidence interval. A total of 5 seeds were used. Experimental data collected by Boccarda et al., 2019.

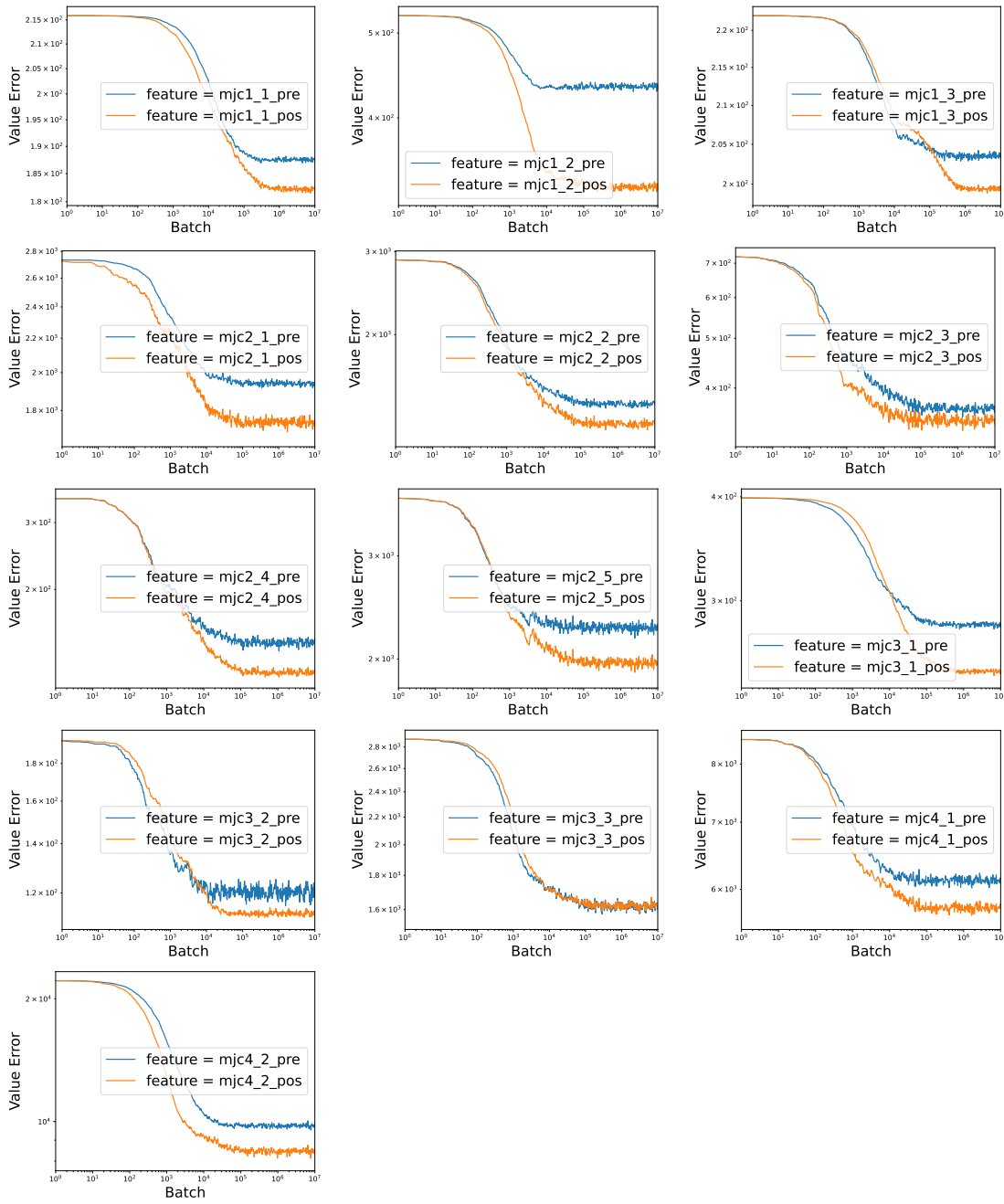


Figure 3.4: Learning curves of TD Learning evaluated on the post-learning policy (π_{pos}) with no normalization. The learning rate is set to 1.0. Shaded area denotes 95% confidence interval. A total of 5 seeds were used. Experimental data collected by Boccara et al., 2019.

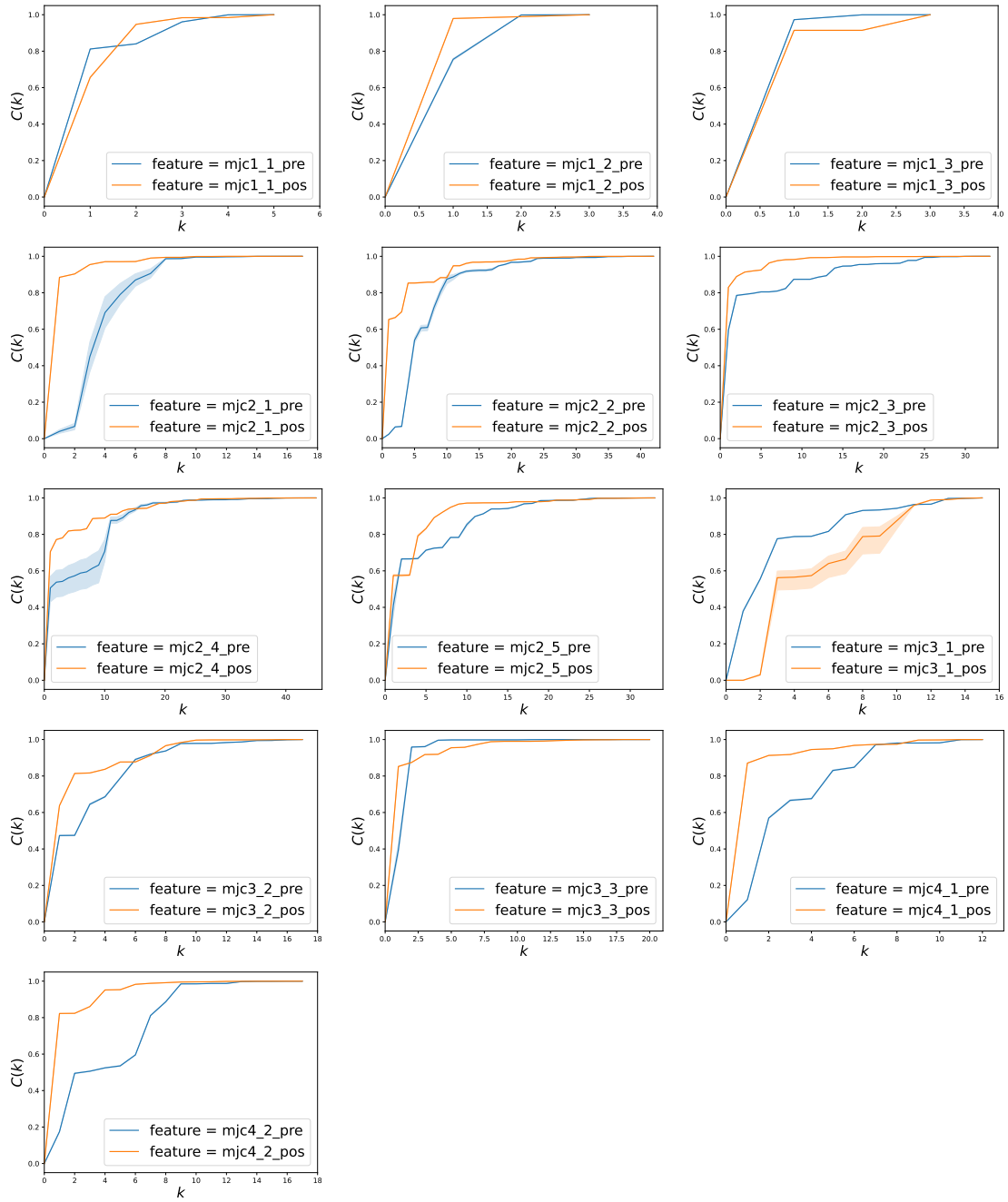


Figure 3.5: Spectral perspective of TD Learning. The features where $C(k)$ rises faster have better feature-reward matches. The functions were evaluated on the post-learning policy (π_{pos}) with no normalization. Shaded area denotes 95% confidence interval. A total of 5 seeds were used. Experimental data collected by Boccarda et al., 2019.

4

Discussion

In this thesis, we presented a novel hypothesis and theory to explain spatial receptive field remapping in rats. Our hypothesis is that these receptive fields form features in the learning process and change to improve learning speed or convergence error. Our model is based on TD learning, a policy evaluation method by which the agent learns the value function of a policy. First, we evaluated the theory of TD learning to understand which parameters influence learning dynamics. Second, using an evolutionary algorithm, we found that place cell features that have faster learning speeds typically

have centers near the reward locations. This is consistent with the qualitative findings in Boccara et al., 2019. Finally, using features generated from neural data for TD learning, we also observed results that supported our hypothesis. In all but one of the trials, the features after learning allowed for faster learning and lower convergence value errors. In the sections below, we discuss some our results, limitations of the theory, and future work.

4.1 CONVERGENCE SPEED AND ERROR DEPENDS ON HYPERPARAMETERS AND FEATURES

From Figure 3.1, we learned that even in complex environments, convergence TD value error depends on hyperparameters such as learning rate and batch size. Higher learning rates lead to faster learning, but higher convergence value errors. This provides insight that whenever comparing two different learning curves, seeing such phenomenon may be due to higher learning rates.

Furthermore, from Figure 3.4, we learned that features affect the convergence speed and error of a TD learning agent. Some features may be more suitable for learning a value function compared to others. We observed that using features generated from experimental data, the features after learning (ψ_{pos}) provide better convergence speed and errors compared to the features before learning (ψ_{pre}).

4.2 FEATURES MATCH THE VALUE FUNCTION

From Figure 3.2, we saw that the features evolved to have better convergence speed and error usually congregate near rewards. This means that the features might be trying to match the shape of the value function, which is higher near reward areas. From Figures A.4-A.6, we can see that the value function is high in the areas that have reward.

4.3 FEATURE FIT DEPENDS ON POLICY

Comparing Figures 3.3-3.4, we saw that the learning speeds of a TD learner depend on the policy being evaluated. For the same reward function, a certain set of features may be better for one policy over another. We observed that even though ψ_{pos} is better for evaluating the value function of π_{pos} , it is not necessarily better for evaluating π_{rand} or π_{pre} . From the value functions of π_{pos} , we can also see much higher value near the reward areas compared to that of π_{rand} and π_{pre} . This suggests that the rats have learned features that would improve learning on specific policies that are useful to them, such as π_{pos} .

4.4 DISCREPANCY BETWEEN LEARNING CURVES AND SPECTRAL ANALYSIS

In Figure 3.5, we observed that not all curves for π_{pos} rose faster than that of π_{pre} , though learning curves for π_{pos} suggest otherwise. This may be due to limitations for our TD learning theory to explain our results, as one of the main assumptions of the theory is high dimensional features, i.e., adequate power for the features to represent reward and value functions. We discuss in Section 4.7 for future work on this area.

4.5 MODELING POLICY ITERATION

We had only modeled policy evaluation in our theory and experiments. Policy evaluation constitutes of half the learning process. Ultimately, the rat must be able to learn how to get to the reward locations. Therefore, in the complete learning process, the agent not only needs to update their evaluation of the current policy (policy evaluation), they also need to update their policy based on the evaluation (policy iteration). For a complete explanation of the rat's behavior, it would be necessary to model both parts of the learning process.

There are many types of RL algorithms that contains policy iteration. The analogous algorithms similar to TD learning are Actor Critic methods (Sutton & Barto, 2018; Mnih et al., 2016).

4.6 REPRESENTATION LEARNING

In Section 3.2, we showed that the center of the place cells move to locations closer to the rewards by evolving place cell features to optimize for learning speed. This is reminiscent of representation learning, where transformations of data are learned to enable faster learning in supervised learning contexts (Bengio et al., 2013). There are also similar representation learning techniques for RL tasks (Uehara et al., 2022).

Recently, there have been significant progress towards understanding the representation/feature learning aspect of deep neural networks (Bordelon & Pehlevan, 2022; Yang & Hu, 2022). As the animal learns the task, could the receptive field remapping observed in Boccarda et al., 2019 be explained as deep neural networks trying to conduct representation learning? This is left for future work.

4.7 LARGE-SCALE NEURAL RECORDINGS

The experimental data obtained from Boccarda et al., 2019 may be limited due to insufficient features for learning. As we saw in the analysis of the data (Table 3.1), we have at most 45 neurons in a trial. This means that there may not be enough features created in each trial to accurately verify our hypothesis. This is exacerbated by the fact that the TD learning dynamics theory relies on the assumption of high dimension features on the order of about the same as the number of states.

Fortunately, with the advent of new high-throughput neural recording techniques, it may be feasible to conduct the same experiment as in Boccarda et al., 2019 with more neurons recorded at the same time. This has been successful in verifying other theories, such as the continuous attractor

network hypothesis of grid cells (Gardner et al., 2022). In this case, Neuropixel (Jun et al., 2017) was used to record from 960 neurons simultaneously. The second version of Neuropixel was able to simultaneously record from 5120 sites (Steinmetz et al., 2021), providing enough neural recording data for our purposes.

4.8 VERIFICATION OF OTHER EXPERIMENTAL FINDINGS

In addition to observing place cell and grid cell fields moving closer to reward locations, Boccara et al., 2019 had other findings such as activation fields closer to the reward locations are attracted more towards the reward locations than fields that are further away. Furthermore, grid score degradation was also observed after the rat learns the reward locations. Findings such as these can not be verified by our cross-entropy algorithm as all the features are RBFs. Our theory could be more strongly supported if some other qualitative changes observed experimentally could be found in artificial settings.

4.9 CONCLUSION

Overall, our findings support the hypothesis that the perception of rats change to facilitate learning. While our theory is pertinent to policy evaluation, we believe that this theory could be extended to fully explain spatial receptive field remapping phenomena in animals.

A

Appendix

A.1 ADDITIONAL CROSS-ENTROPY BASED BASIS ADAPTATION RESULTS

Figure A.1 shows additional results for the cross-entropy based basis adaptation algorithm. We can see that regardless of which π_{pos} being used, the place cells evolve to locations near the rewards.

A.2 ADDITIONAL TD LEARNING CURVES WITH EXPERIMENTAL DATA

Figure A.2 shows TD Learning curves for π_{pre} with no normalization. Figure A.3 shows TD learning curves for π_{pos} with peak normalization.

A.3 PLOTS OF TRUE VALUE FUNCTIONS

Figures A.4-A.6 depicts the different target true value functions on different policies. Notice that the value function is higher closer to reward locations. We can also see that the value function of π_{pre} is more similar to that of π_{rand} . This means that the animal’s behavior pre-learning is more like a random walk.

The white portions in the figures represent invalid states in the rate maps.

A.4 PLOTS OF LEARNED VALUE FUNCTIONS

Figures A.7 depicts the learned value functions on π_{pos} with different features.

A.5 EQUIPMENT AND TOOLS

Experiments were conducted on a PC with the following specification: AMD Ryzen Threadripper 3960X 24-Core Processor 3.80 GHz, 128GB RAM, NVIDIA GeForce RTX 4090 GPU. The Harvard FAS computing cluster was also used to parallel compute across different seeds. The experiments were optimized with the JAX library (Bradbury et al., 2018).

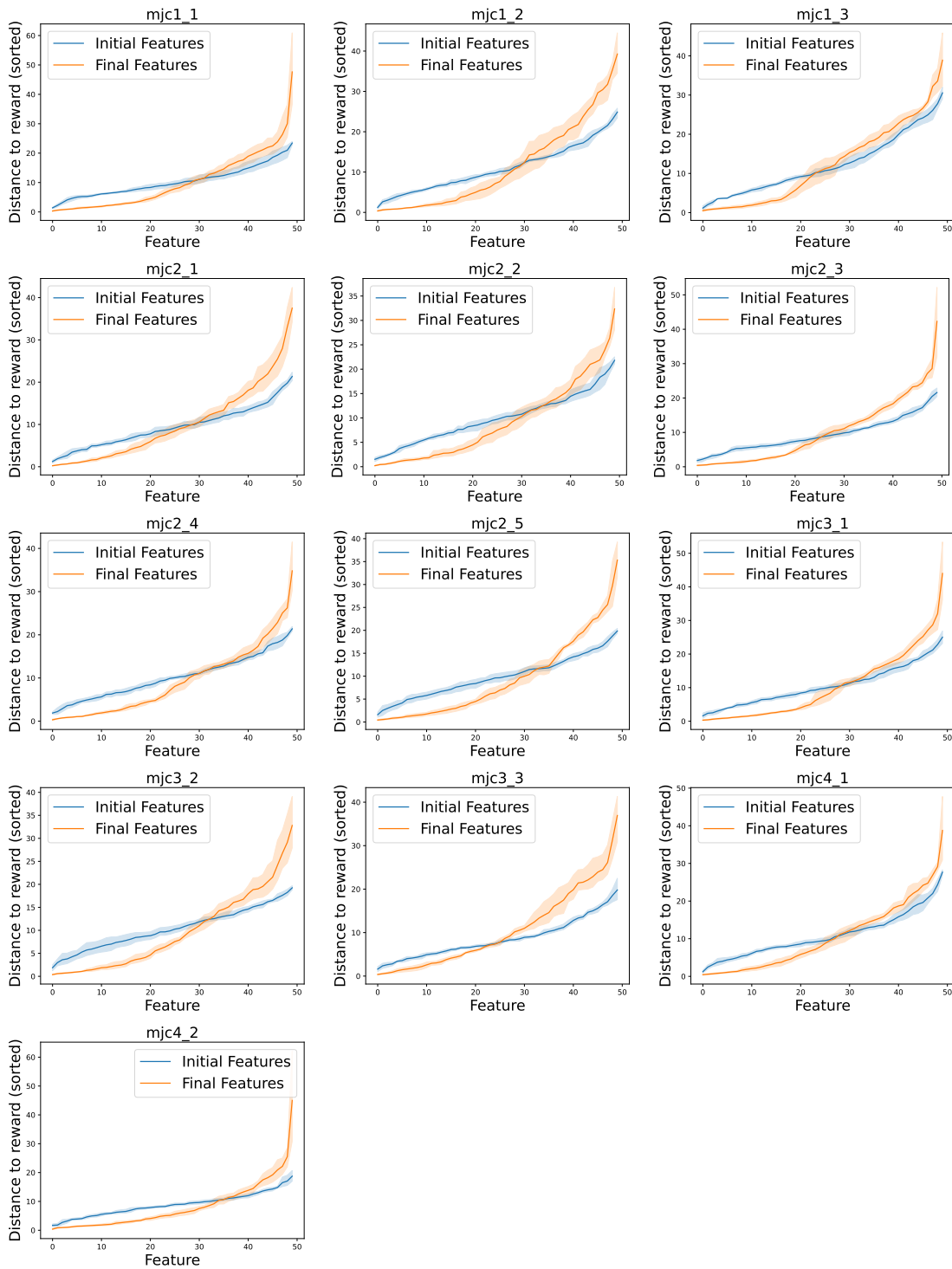


Figure A.1: Additional results of the cross-entropy based basis adaptation algorithm (Algorithm 2) for different π_{pos} . Shaded area denotes 95% confidence interval. A total of 5 seeds were used.

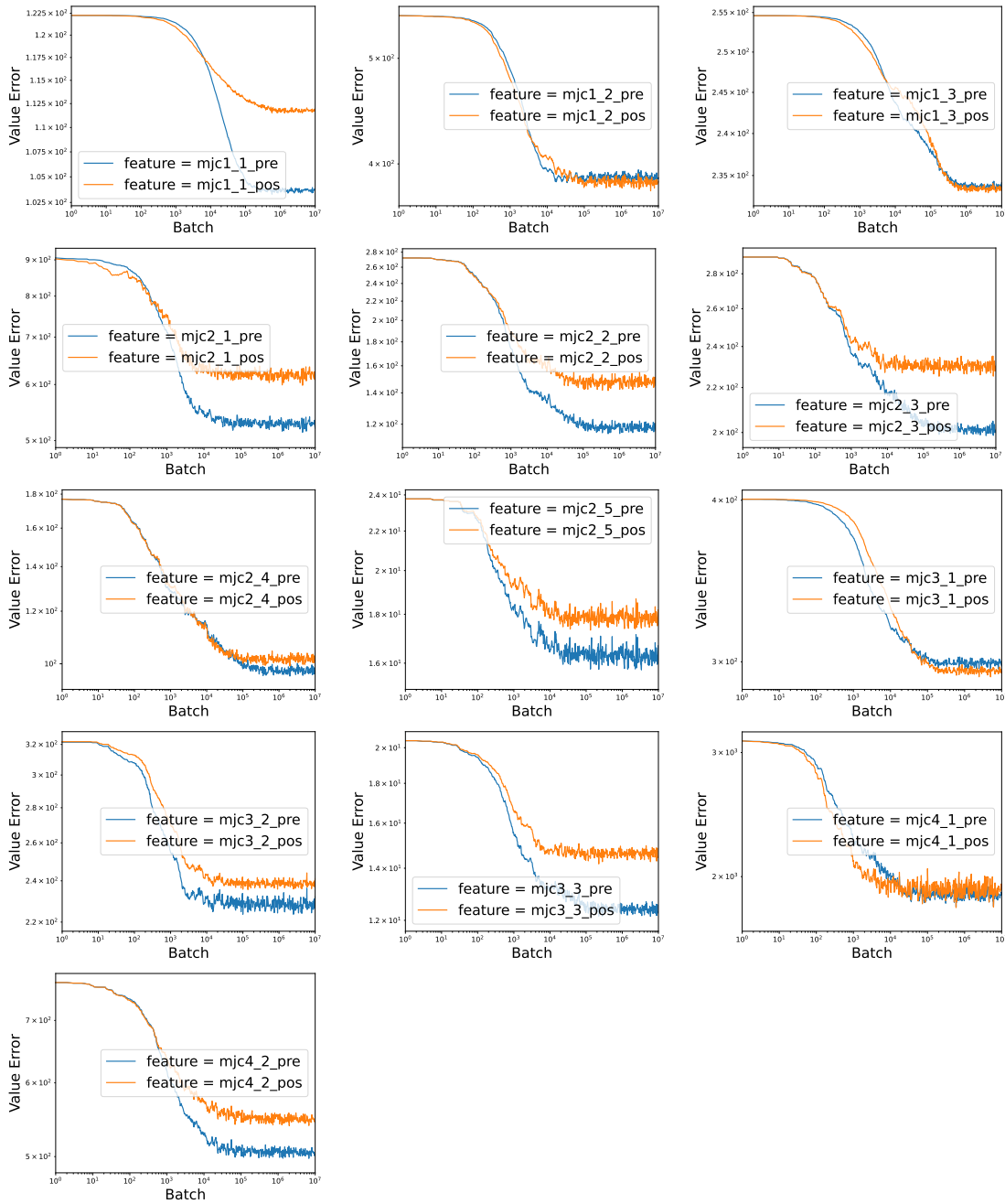


Figure A.2: Learning curves of TD Learning evaluated on the pre-learning policy (π_{pre}) with no normalization. The learning rate is set to 1.0. Shaded area denotes 95% confidence interval. A total of 5 seeds were used. Experimental data collected by Boccarda et al., 2019.

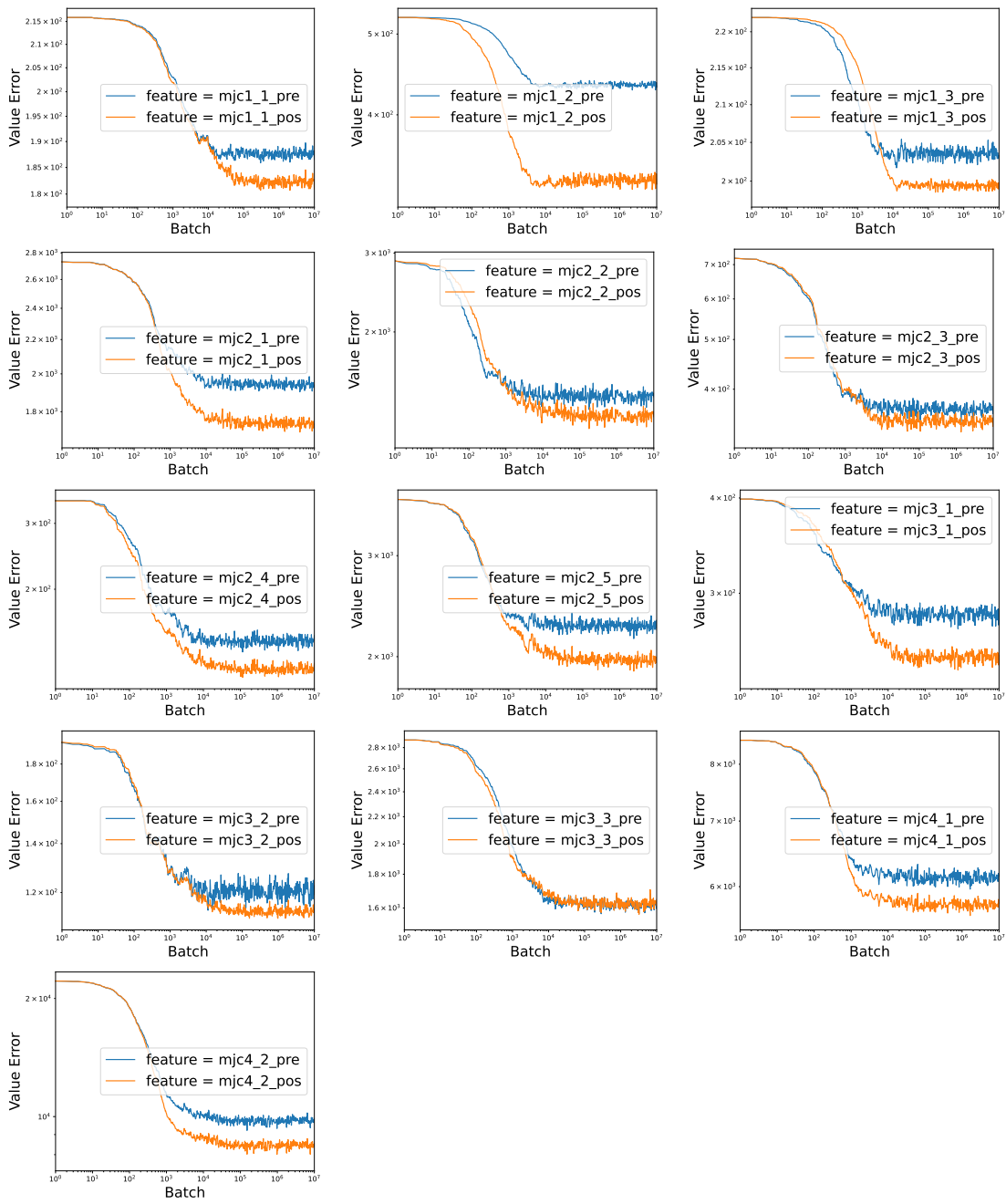


Figure A.3: Learning curves of TD Learning evaluated on the post-learning policy (π_{pos}) with peak normalization. The highest firing rate location for each neuron are normalized to 1. The learning rate is set to 1.0. Shaded area denotes 95% confidence interval. A total of 5 seeds were used. Experimental data collected by Boccarda et al., 2019.

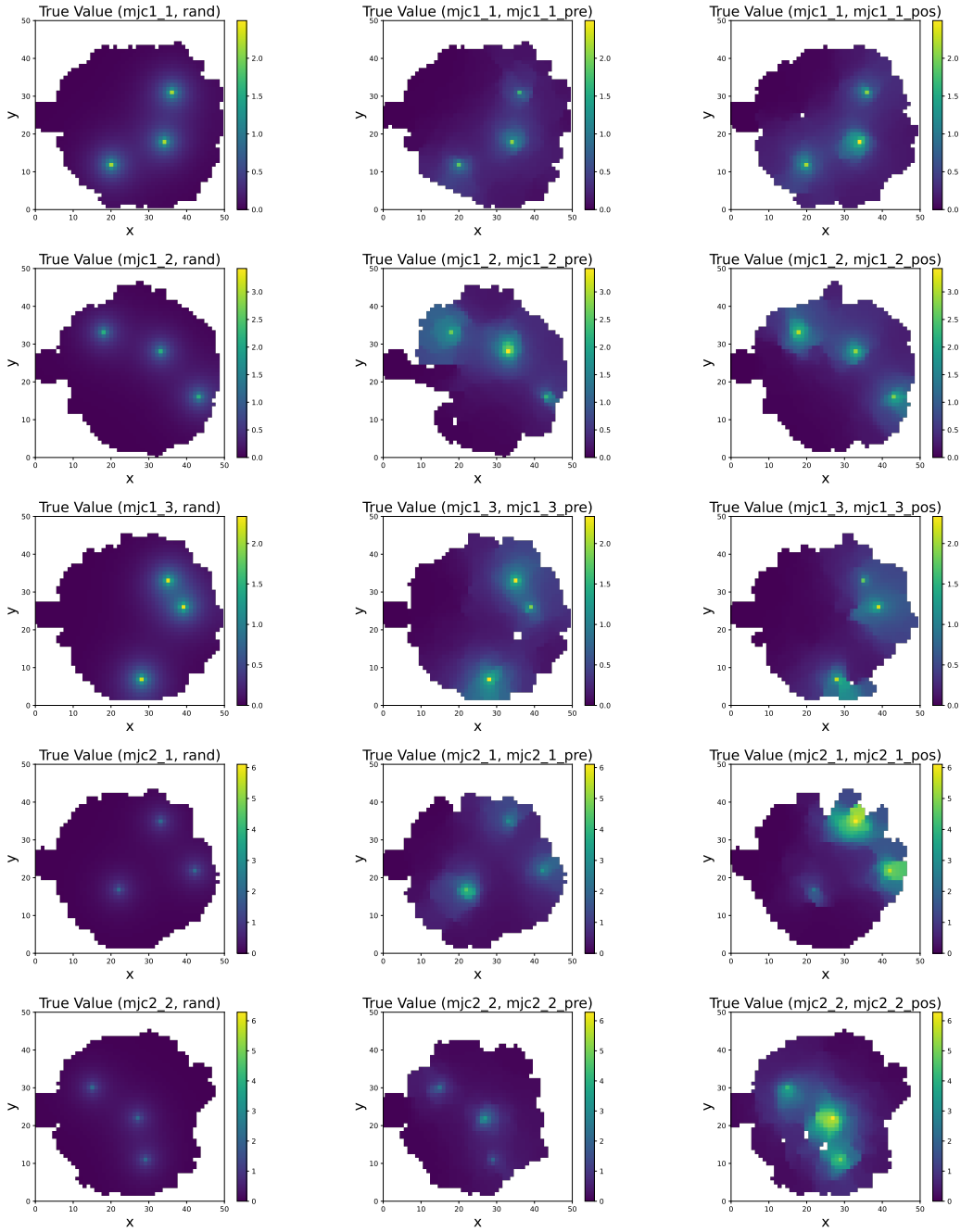


Figure A.4: True value functions of different trials and polices, computed with Equation 2.9. (left) Random policy with different reward locations based on the trial. (middle) The true value function based on the probability transition matrix of π_{pre} . (right) The true value function based on the probability transition matrix of π_{pos} . Experimental data collected by Boccarda et al., 2019. (Continued in Figure A.5.)

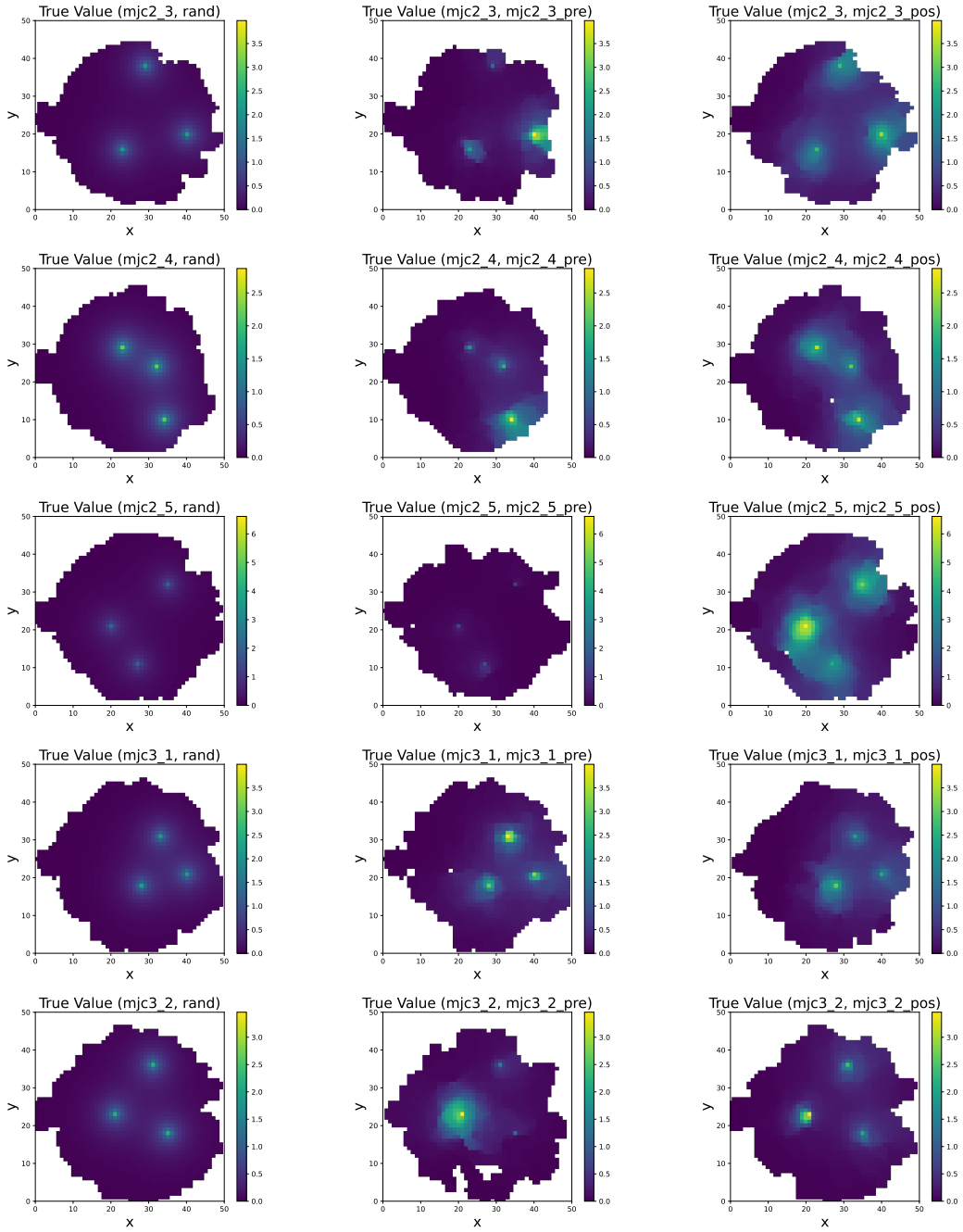


Figure A.5: True value functions of different trials and polices, computed with Equation 2.9. (left) Random policy with different reward locations based on the trial. (middle) The true value function based on the probability transition matrix of π_{pre} . (right) The true value function based on the probability transition matrix of π_{pos} . Experimental data collected by Boccara et al., 2019. (Continued in Figure A.6.)

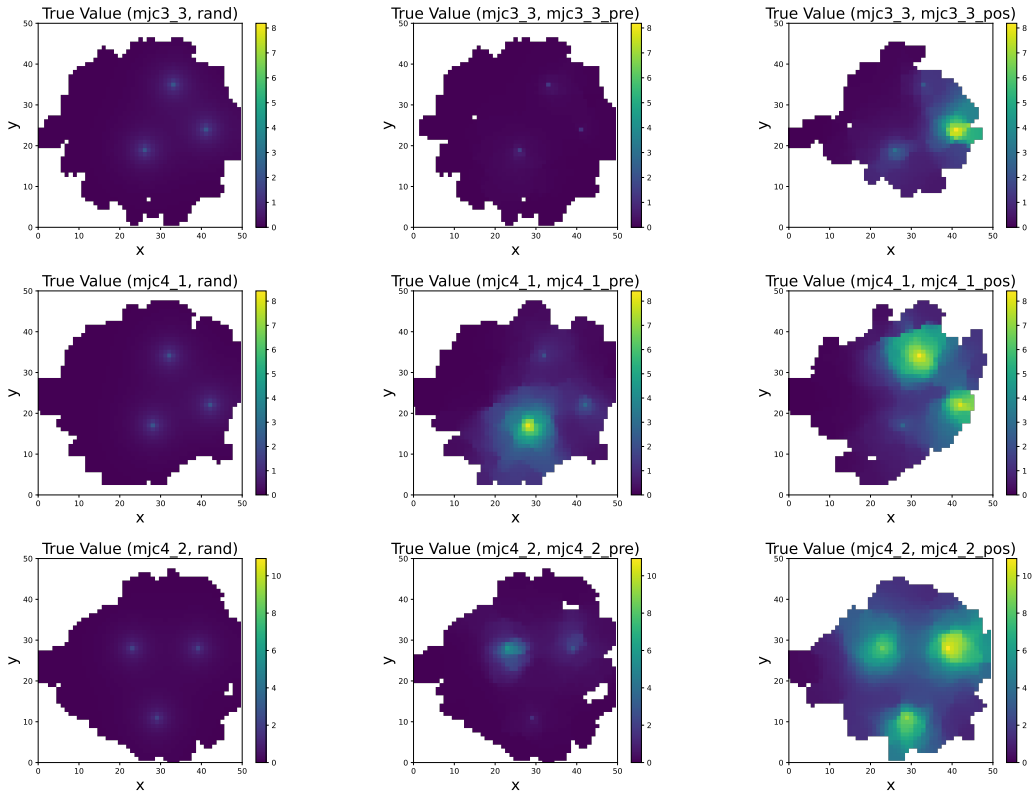


Figure A.6: True value functions of different trials and polices, computed with Equation 2.9. (left) Random policy with different reward locations based on the trial. (middle) The true value function based on the probability transition matrix of π_{pre} . (right) The true value function based on the probability transition matrix of π_{pos} . Experimental data collected by Boccara et al., 2019.

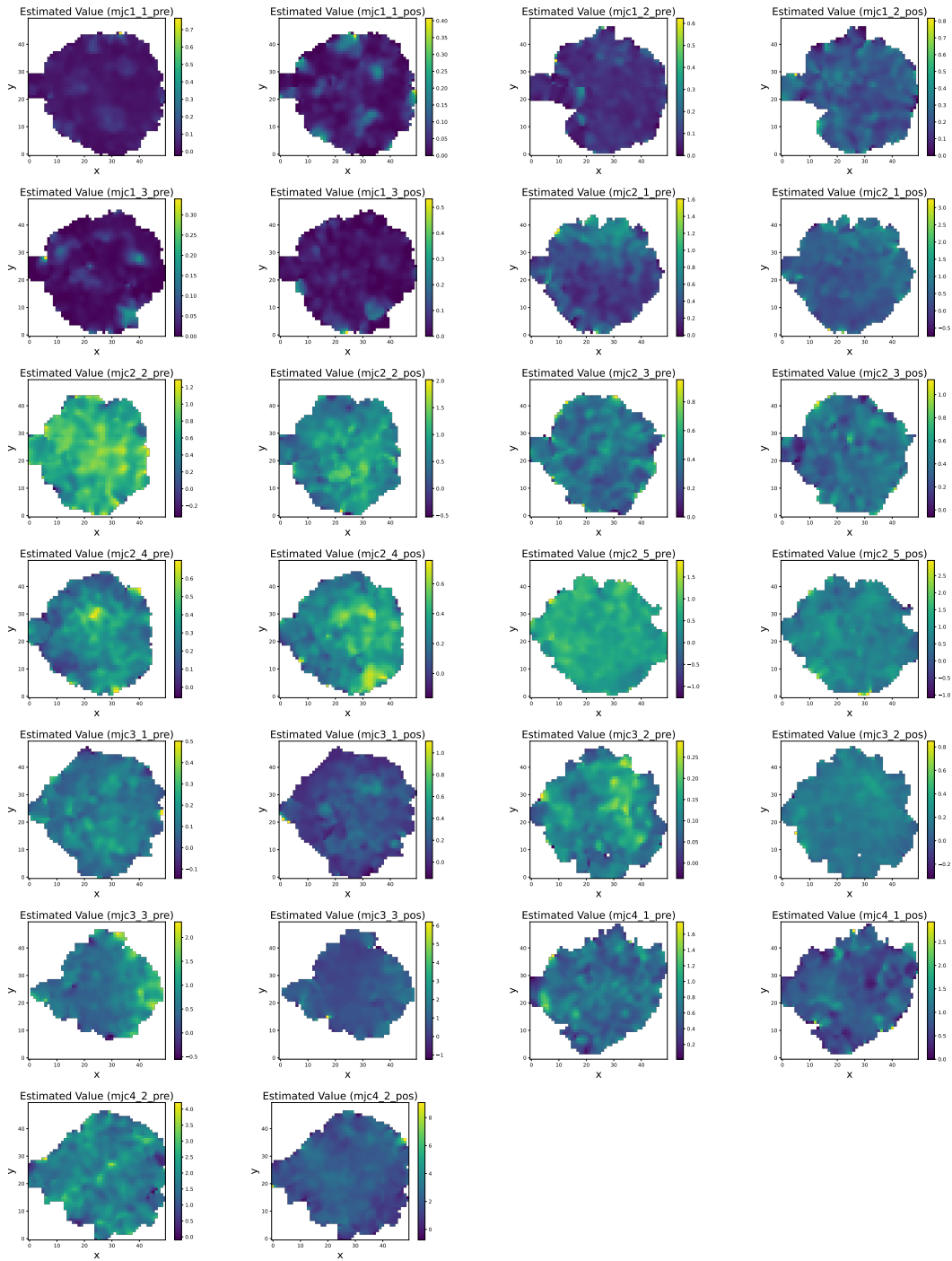


Figure A.7: Learned value functions of different trials on π_{pos} with different features. The title of the subfigures denote the feature used. Experimental data collected by Boccarda et al., 2019.

Bibliography

- Boccaro, C. N., Nardin, M., Stella, F., O'Neill, J., & Csicsvari, J. (2019). The entorhinal cognitive map is attracted to goals. *Science*, 363(6434), 1443–1447. <https://doi.org/10.1126/science.aav4837>
- Sanguinetti-Scheck, J. I., & Brecht, M. (2020). Home, head direction stability, and grid cell distortion. *Journal of Neurophysiology*, 123(4), 1392–1406. <https://doi.org/10.1152/jn.00518.2019>
- Ginosar, G., Aljadeff, J., Las, L., Derdikman, D., & Ulanovsky, N. (2023). Are grid cells used for navigation? On local metrics, subjective spaces, and black holes. *Neuron*, 111(12), 1858–1875. <https://doi.org/10.1016/j.neuron.2023.03.027>
- O'Keefe, J., & Dostrovsky, J. (1971). The hippocampus as a spatial map. Preliminary evidence from unit activity in the freely-moving rat. *Brain Research*, 34(1), 171–175. [https://doi.org/10.1016/0006-8993\(71\)90358-1](https://doi.org/10.1016/0006-8993(71)90358-1)
- Hafting, T., Fyhn, M., Molden, S., Moser, M.-B., & Moser, E. I. (2005). Microstructure of a spatial map in the entorhinal cortex. *Nature*, 436(7052), 801–806. <https://doi.org/10.1038/nature03721>
- Jeffery, K. J. (2007). Self-localization and the entorhinal–hippocampal system. *Current Opinion in Neurobiology*, 17(6), 684–691. <https://doi.org/10.1016/j.conb.2007.11.008>
- Bush, D., Barry, C., Manson, D., & Burgess, N. (2015). Using Grid Cells for Navigation. *Neuron*, 87(3), 507–520. <https://doi.org/10.1016/j.neuron.2015.07.006>
- Stensola, T., Stensola, H., Moser, M.-B., & Moser, E. I. (2015). Shearing-induced asymmetry in entorhinal grid cells. *Nature*, 518(7538), 207–212. <https://doi.org/10.1038/nature14151>
- Dang, S., Wu, Y., Yan, R., & Tang, H. (2021). Why grid cells function as a metric for space. *Neural Networks*, 142, 128–137. <https://doi.org/10.1016/j.neunet.2021.04.031>
- Schultz, W., Dayan, P., & Montague, P. R. (1997). A Neural Substrate of Prediction and Reward. *Science*, 275(5306), 1593–1599. <https://doi.org/10.1126/science.275.5306.1593>

- Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction* (Second edition). The MIT Press.
- Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., & Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529–533. <https://doi.org/10.1038/nature14236>
- Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., & Kavukcuoglu, K. (2016). Asynchronous Methods for Deep Reinforcement Learning. *International Conference on Machine Learning*, 1928–1937. Retrieved October 18, 2020, from <http://proceedings.mlr.press/v48/mniha16.html>
- Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., & Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, *529*(7587), 484–489. <https://doi.org/10.1038/nature16961>
- Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., van den Driessche, G., Graepel, T., & Hassabis, D. (2017). Mastering the game of Go without human knowledge. *Nature*, *550*(7676), 354–359. <https://doi.org/10.1038/nature24270>
- Amo, R., Matias, S., Yamanaka, A., Tanaka, K. F., Uchida, N., & Watabe-Uchida, M. (2022). A gradual temporal shift of dopamine responses mirrors the progression of temporal difference error in machine learning. *Nature Neuroscience*, *25*(8), 1082–1092. <https://doi.org/10.1038/s41593-022-01109-2>
- Robbins, H., & Monro, S. (1951). A Stochastic Approximation Method. *The Annals of Mathematical Statistics*, *22*(3), 400–407. Retrieved January 28, 2024, from <https://www.jstor.org/stable/2236626>
- Bordelon, B., Masset, P., Kuo, H., & Pehlevan, C. (2023). Loss Dynamics of Temporal Difference Reinforcement Learning. Retrieved November 28, 2023, from <https://openreview.net/forum?id=Tj0eXVPnRX>
- Moore, A. W. (1990). *Efficient memory-based learning for robot control* (tech. rep.). University of Cambridge.
- Towers, M., Terry, J. K., Kwiatkowski, A., Balis, J. U., Cola, G. d., Deleu, T., Goulão, M., Kallinteris, A., KG, A., Krimmel, M., Perez-Vicente, R., Pierré, A., Schulhoff, S., Tai, J. J., Shen, A. T. J., & Younis, O. G. (2023). Gymnasium. <https://doi.org/10.5281/zenodo.8127026>

- Watkins, C. (1989). Learning from delayed rewards. *PhD thesis, Cambridge University, Cambridge, England.*
- Watkins, C. J. C. H., & Dayan, P. (1992). Q-learning. *Machine Learning*, 8(3), 279–292. <https://doi.org/10.1007/BF00992698>
- Menache, I., Mannor, S., & Shimkin, N. (2005). Basis Function Adaptation in Temporal Difference Reinforcement Learning. *Annals of Operations Research*, 134(1), 215–238. <https://doi.org/10.1007/s10479-005-5732-z>
- de Boer, P.-T., Kroese, D. P., Mannor, S., & Rubinstein, R. Y. (2005). A Tutorial on the Cross-Entropy Method. *Annals of Operations Research*, 134(1), 19–67. <https://doi.org/10.1007/s10479-005-5724-z>
- Rubinstein, R. Y., & Kroese, D. P. (2004). *The Cross-Entropy Method*. Springer. <https://doi.org/10.1007/978-1-4757-4321-0>
- Dunn, B., Wennberg, D., Huang, Z., & Roudi, Y. (2017, January 17). *Grid cells show field-to-field variability and this explains the aperiodic response of inhibitory interneurons*. arXiv: 1701.04893 [q-bio]. <https://doi.org/10.48550/arXiv.1701.04893>
- Tarjan, R. (1972). Depth-First Search and Linear Graph Algorithms. *SIAM Journal on Computing*, 1(2), 146–160. <https://doi.org/10.1137/0201010>
- Hagberg, A. A., Schult, D. A., & Swart, P. J. (2008). Exploring network structure, dynamics, and function using NetworkX. In G. Varoquaux, T. Vaught, & J. Millman (Eds.), *Proceedings of the 7th python in science conference* (pp. 11–15).
- Canatar, A., Bordelon, B., & Pehlevan, C. (2021). Spectral bias and task-model alignment explain generalization in kernel regression and infinitely wide neural networks. *Nature Communications*, 12(1), 2914. <https://doi.org/10.1038/s41467-021-23103-1>
- Graham, R. L., Knuth, D. E., & Motzkin, T. S. (1972). Complements and transitive closures. *Discrete Mathematics*, 2(1), 17–29. [https://doi.org/10.1016/0012-365X\(72\)90057-X](https://doi.org/10.1016/0012-365X(72)90057-X)
- Bengio, Y., Courville, A., & Vincent, P. (2013). Representation Learning: A Review and New Perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8), 1798–1828. <https://doi.org/10.1109/TPAMI.2013.50>
- Uehara, M., Zhang, X., & Sun, W. (2022, January 5). *Representation Learning for Online and Offline RL in Low-rank MDPs*. arXiv: 2110.04652 [cs, stat]. <https://doi.org/10.48550/arXiv.2110.04652>
- Bordelon, B., & Pehlevan, C. (2022). Self-Consistent Dynamical Field Theory of Kernel Evolution in Wide Neural Networks. *Advances in Neural Information Processing Systems*, 35, 32240–32256. Retrieved January 5, 2024, from https://proceedings.neurips.cc/paper_files/paper/2022/hash/d027a5c93d484a4312cc486d399c62c1-Abstract-Conference.html

- Yang, G., & Hu, E. J. (2022, July 15). *Feature Learning in Infinite-Width Neural Networks* (3). arXiv: 2011.14522 [cond-mat]. <https://doi.org/10.48550/arXiv.2011.14522>
- Gardner, R. J., Hermansen, E., Pachitariu, M., Burak, Y., Baas, N. A., Dunn, B. A., Moser, M.-B., & Moser, E. I. (2022). Toroidal topology of population activity in grid cells. *Nature*, 602(7895), 123–128. <https://doi.org/10.1038/s41586-021-04268-7>
- Jun, J. J., Steinmetz, N. A., Siegle, J. H., Denman, D. J., Bauza, M., Barbarits, B., Lee, A. K., Anastassiou, C. A., Andrei, A., Aydın, Ç., Barbic, M., Blanche, T. J., Bonin, V., Couto, J., Dutta, B., Gratiy, S. L., Gutnisky, D. A., Häusser, M., Karsh, B., ... Harris, T. D. (2017). Fully integrated silicon probes for high-density recording of neural activity. *Nature*, 551(7679), 232–236. <https://doi.org/10.1038/nature24636>
- Steinmetz, N. A., Aydın, C., Lebedeva, A., Okun, M., Pachitariu, M., Bauza, M., Beau, M., Bhagat, J., Böhm, C., Broux, M., Chen, S., Colonell, J., Gardner, R. J., Karsh, B., Kloosterman, F., Kostadinov, D., Mora-Lopez, C., O’Callaghan, J., Park, J., ... Harris, T. D. (2021). Neuropixels 2.0: A miniaturized high-density probe for stable, long-term brain recordings. *Science*, 372(6539), eabf4588. <https://doi.org/10.1126/science.abf4588>
- Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., & Zhang, Q. (2018). *JAX: Composable transformations of Python+NumPy programs* (Version 0.3.13). <http://github.com/google/jax>