A spiking neural network of state transition probabilities in model-based reinforcement learning

by

Mariah Shein

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2017

Abstract

The development of the field of reinforcement learning was based on psychological studies of the instrumental conditioning of humans and other animals. Recently, reinforcement learning algorithms have been applied to neuroscience to help characterize neural activity and animal behaviour in instrumental conditioning tasks. A specific example is the hybrid learner developed to match human behaviour on a two-stage decision task [18]. This hybrid learner is composed of a model-free and a model-based system. The model presented in this thesis is an implementation of that model-based system where the state transition probabilities and Q-value calculations use biologically plausible spiking neurons. Two variants of the model demonstrate the behaviour when the state transition probabilities are encoded in the network at the beginning of the task, and when these probabilities are learned over the course of the task. Various parameters that affect the behaviour of the model are explored, and ranges of these parameters that produce characteristically model-based behaviour are found. This work provides an important first step toward understanding how a model-based system in the human brain could be implemented, and how this system contributes to human behaviour.

Acknowledgements

I would like acknowledge my supervisor, Chris Eliasmith, for providing inspiration and guidance; Terry Stewart, for all the help and infectious enthusiasm; Brent Komer and Sugandha Sharma, for helping me clarify and crystallize my ideas; and Ivana Kajić and Peter Duggins, for your motivation and support. I could not have done this without all of you!

Dedication

This thesis is dedicated to my parents, Elizabeth Shein and Ralph Martin, for their constant love, support, and encouragement.

Table of Contents

List of Tables **ix**

List of Figures **x**

1 Introduction **1**

1.1 Historical background . . . 1

1.2 Reinforcement learning . . . 2

1.2.1 Markov Decision Processes . . . 2

1.2.2 Q-values . . . 3

1.2.3 Temporal-difference learning . . . 3

1.3 Reinforcement learning in neuroscience . . . 4

1.4 Model-based reinforcement learning . . . 5

1.4.1 State transition probabilities . . . 6

1.5 Model-free vs. model-based behaviour . . . 7

1.6 Biological plausibility . . . 8

1.6.1 Neurons . . . 8

1.6.2 Continous time in RL . . . 10

1.6.3 Neural learning . . . 10

1.7 Neural models of model-free RL . . . 11

1.8 Previous neural models of model-based RL . . . 11

1.9 Summary . . . 12

2 Model Implementation and Method **14**

2.1 Two-stage decision task . . . 15

2.2 Neural Engineering Framework . . . 17

2.2.1 Representation . . . 18

2.2.2 Transformation . . . 18

2.3 Static model structure . . . 19

2.3.1 The *Environment* node . . . 22

2.4 Learning model structure . . . 24

2.5 Parameter exploration . . . 25

3 Results **28**

3.1 Static model . . . 28

3.1.1 Learning rate α . . . 29

3.1.2 Number of neurons . . . 29

3.1.3 Individual seeds . . . 31

3.1.4 Time interval . . . 33

3.1.5 Synapse length . . . 35

3.1.6 Static model summary . . . 35

3.2 Learning Model . . . 38

3.2.1 Learning rate α . . . 38

3.2.2 Number of neurons . . . 38

3.2.3 Individual seeds . . . 41

3.2.4 Time interval . . . 42

3.2.5 Learning model summary . . . 45

4 Conclusions and future work **47**

4.1 Discussion . . . 47

4.2 Future work . . . 49

4.2.1 Additional possible tests 49
4.2.2 Model extensions 50
4.3 Closing remarks 51

References **52**

List of Tables

3.1 Individual seed differences in static model 33

3.2 Individual differences in learning model 42

List of Figures

2.1 Tree diagram of the two-stage decision task. Adapted from [2] and reproduced from [45]. 15

2.2 Characteristic two-stage decision task stay probability behaviour for ideal model-free and model-based agents, as well as human data collected by [18]. State transition probabilities are common (C, prob. = 0.7) or rare (R, prob. = 0.3). Trials rewarded by a value of 1 are denoted with +, and − denotes unrewarded trials (rewarded with 0). Adapted from [18] and reproduced from [45]. 16

2.3 Static model diagram. Neural populations are shown as oval shapes. Elements in rectangles use traditional computation. Reproduced from [45]. 20

2.4 Diagram of state transition probability-learning model. As in Fig. 2.3, the components of the model shown in oval shapes are simulated populations of neurons and the rectangular components are directly computed without neurons. The connection between the *State and Action* population and the *Product* population (highlighted in blue) represents the state transition probabilities that are learned over time using the PES rule. The connection highlighted in red has a pure time delay. 26

3.1 Examples of stay probability behaviours of various learning rates α. Adapted from [45]. 30

3.2 Stay probabilities of the different learning rates α. Adapted from [45]. . . 31

3.3 Stay probabilities of different numbers of neurons. Adapted from [45]. . . 32

3.4 Examples of individual differences between Nengo seeds: (a) pure model-based stay probability behaviour, (b) stay probability behaviour suggestive of human data, and (c) stay probabilities opposite to those in (b). Reproduced from [45]. 33

3.5 Stay probabilities of different time intervals. 34

3.6 Individual differences for various time intervals with seed = 1. Adapted from [45]. 34

3.7 Individual differences for time intervals of 0.5s and 0.51s as the synapse length increases. 36

3.8 Individual differences for time intervals of 0.1s, 0.2s, and 0.3s with different synapse lengths. 37

3.9 Stay probabilities of different learning rates for the learning model with a time interval of 0.1s. 39

3.10 Stay probabilities of different learning rates for the learning model with a time interval of 0.05s. 39

3.11 Stay probabilities of different numbers of neurons. 40

3.12 Stay probabilities of different numbers of neurons in the *Error* population. 41

3.13 Comparison of small and default numbers of neurons. 43

3.14 Stay probabilities of different time intervals. 44

3.15 Stay probabilities of different numbers of neurons in the *State and Action* population with a time interval of 0.05s. 44

3.16 “Present” learning model. 46

Chapter 1

Introduction

This thesis presents a novel model of how the state transition probabilities used in model-based reinforcement learning (RL) may be represented and learned by a network of spiking neurons. In this chapter, we will provide a brief overview of the relevant RL background, including Markov Decision Processes and the differences between model-free and model-based RL. Some basic concepts related to modelling spiking neurons will also be introduced. Throughout the chapter, how computational theories of RL have informed psychological and neuroscientific studies of human and animal reinforcement learning will be discussed. By the end of this chapter, it should be apparent that the computational processes underlying biological model-based RL are not as well understood as those of model-free RL, thus producing one motivation for the research presented in this thesis.

1.1 Historical background

The basic idea of reinforcement learning has been around for centuries, for as long as humans have tried to train other creatures to do things they might not otherwise be inclined to do. It was first scientifically formalized by the behaviourists, psychologists in the late 19th to early 20th century who rejected the idea of understanding the mind through introspection, and instead characterized minds as black boxes that could only be understood through a thorough examination of input-output relations [64]. One major paradigm developed by the behaviourists is known as *operant*, or *instrumental*, *conditioning*.

Instrumental conditioning is based on Thorndike's Law of Effect [75], which states that animals are likely to repeat actions that produce pleasant effects and unlikely to repeat actions that produce unpleasant effects. In instrumental conditioning, an animal is trained towards performing specific actions through administering *reinforcers*, which increase the frequency of the preceding behaviour, and *punishers*, which decrease the frequency of the behaviour [46]. More recently, computer scientists have borrowed ideas from instrumental conditioning to develop the field of *reinforcement learning* (RL).

1.2 Reinforcement learning

Central to reinforcement learning (RL) are the terms "state", "action", and "reward". A *state* is a description of an RL agent's environment at a given point in time. *Actions* are the behaviours available to the agent in that environment. *Rewards* are signals from the environment that (roughly) tell the agent if what it is doing is good or bad. Rewards usually take the form of integers or real numbers that can be positive or negative. Positive numbers are analogous to reinforcers and negative numbers are punishers, yet they are both referred to as rewards.

RL has been described as a third type of machine learning, unlike supervised or unsupervised learning. RL is unlike supervised learning because an RL agent is not explicitly told if the action it selects at any point is right or wrong. It is unlike unsupervised learning because it does not look for patterns in unlabelled data [73]. Instead, an RL agent learns through trial-and-error, performing actions, receiving rewards, and using this information to gradually learn over time which actions accrue the most reward in which circumstances. The goal in RL is to choose a course of action (called a *policy*) that maximizes the reward signal [72]. How 'good' an action is in a particular state, or how much it helps towards the goal of reward maximization, is called the *value*. In the branch of RL used in this thesis, these values are calculated according to the Bellman equation, where Q is the value. For this reason, values are often called *Q-values*.

1.2.1 Markov Decision Processes

States, actions, and rewards are three of the four components of a Markov Decision Process (MDP). MDPs are widely used in RL to describe the structure of a task (also known as a model of the world) that an RL agent is learning. An MDP consists of a set S of states s, a set A of actions a, a reward function $R(s, a)$ that describes the expectation of reward given action a performed in state s, and a state transition function $P(s, a, s')$ that describes the probability of transitioning to state s' after performing action a in state s. The state transition probability function is the main focus of the neural model developed in this thesis.

1.2.2 Q-values

Let us now return to the topic of Q-values, as defined by the Bellman equation [12]:

$$Q(s,a) = R(s,a) + \gamma \sum_{s'} P(s,a,s') \max_{a'} Q(s',a'). \tag{1.1}$$

The Q-value of a state s and action a pair is not just the expected reward $R(s,a)$, but also some discounted amount (γ) of the best Q-values of future states multiplied by the probabilities $P(s,a,s')$ of reaching those states. The future states are discounted with γ both to keep the expected values from going to infinity in continuing tasks [71], and to align with the idea that humans and other animals tend to prefer current rewards to rewards received some time in the future [50].

This appears to be a relatively straightforward calculation, but in practice it is complex. First, larger state and action spaces rapidly increase the number of multiplications needed, since the agent must iterate over every state-action pair. Second, if $P(s,a,s')$ and $R(s,a)$ are not known (and they usually are not), it is also necessary to have a method to learn these functions. This problem is again compounded in tasks with large state spaces.

For these reasons, most research in RL has focused on developing methods of approximating Q-values by ignoring the full description of the world model as specified by the MDP. These methods are thus called *model-free* RL. One common model-free RL technique is called temporal-difference learning [71].

1.2.3 Temporal-difference learning

Temporal-difference (TD) learning [73, 55] is a paradigm for how to estimate the value function for a given policy. In general, an agent using TD learning attempts to estimate the Q-values by learning from experience; that is, sampling the environment by trying out actions in states and observing the resulting reward. Without $R(s,a)$ and $P(s,a,s')$ to provide an explicit model of the environment, the only information the agent has about the reward structure is the immediately observed reward r of a state-action pair. The agent also has an internal memory of the Q-values of previously observed state-action pairs. Suppose the agent has observed every state-action pair at least once, and has built up a table of Q-values based on the observed rewards. The Q-values of each state-action pair can be thought of as the sum of the immediate reward and some amount (discounted by a factor of γ) of the future reward:

$$Q(s,a) \approx r + \gamma Q(s', a'), \tag{1.2}$$

where a' is the action taken in state s' according to the policy. To initially build up the Q-value table, $Q(s,a)$ for a previously unobserved s, a is simply defined as r.

As the agent continues sampling the space, it will find some difference between the expected Q-values and those actually observed. This difference is known as the TD error:

$$\delta_{TD}(s,a) \doteq r + \gamma Q(s', a') - Q(s,a). \tag{1.3}$$

This error can be used to signal to the agent how to improve its Q-value estimate after making an observation.

One strategy for learning Q-values from experience is to explore the environment, and after every observation, recalculate $Q(s,a)$, and add this result to a running average of past calculations of $Q(s,a)$:

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha[r + \gamma Q(s', a')], \tag{1.4}$$

where α is a learning rate parameter. This is known as the SARSA [59] update formula. SARSA stands for (s, a, r, s', a') the current state and action, the reward, and the future state and action.

1.3 Reinforcement learning in neuroscience

Aspects of reinforcement learning theory were adopted by psychologists and neuroscientists in their investigation and description of some forms of human and animal learning. The most notable of these approaches has been the reward prediction error hypothesis of dopamine neuron activity [7, 49, 61, 34, 60, 9]. This theory is based on findings that the activity of neurons in the brain that use dopamine as a neurotransmitter can be interpreted as a signal of error in the prediction of reward. Namely, the output activity of dopaminergic neurons reduces when rewards were predicted but not received, and increases when rewards were received but not predicted, i.e., surprises. This reward prediction error (RPE) is likened to the TD error shown in Eq. 1.3.

Many studies have been done comparing human and animal behaviour on various reinforcement learning tasks to that produced by artificial agents using model-free RL algorithms, e.g., [7, 49, 61]. In most cases, and unsurprisingly, humans have been found to exhibit behaviours other than those predicted by purely model-free strategies. Some authors [19, 5] have posited the existence of two systems: one where habits are built up through trial-and-error, and one where actions are planned so as to reach a certain goal. The habit system and the goal-directed system are marked by different behavioural characteristics. In particular, once a habit has formed, it is difficult for behaviour to change even when the reward contingencies change. Goal-directed behaviour is much more flexible. For instance, when a certain state-action pair suddenly stops delivering reward, the goal-directed system reacts much more quickly to compute a new policy to maximize reward, while the habit system would spend a great deal more time attempting to improve its policy through trial-and-error.

A specific example of the differences between habitual and goal-directed learning occurs in conditioning paradigms in which an animal (say, a rat) is trained to perform a task, such as pressing a lever, to gain a reward, such as a drop of sugar water [25]. After a time, the sugar water is replaced by a bitter solution which the rat does not like. If the rat continues to push the lever it is operating under the influence of the habit system, and if it stops pushing the lever, its actions are said to be goal-directed.

The habit-based system can be seen as using a model-free RL strategy, supported by evidence from neural systems, such as the reward prediction error hypothesis of dopamine neuron activity. It is not as well understood how the goal-directed system may be realized in the brain, but it is proposed that it uses a form of what is known as *model-based* RL.

1.4 Model-based reinforcement learning

Model-based RL learns an explicit model of the world (the $P(s, a, s')$ and $R(s, a)$ functions as defined by the theory of MDPs) in order to directly calculate the Q-values according to Eq. 1.1, as opposed to the implicit representation developed through model-free approaches [1, 71]. As noted earlier, learning $P(s, a, s')$ and $R(s, a)$ can be difficult, as iterating over all state-action pairs in large state spaces can be computationally expensive. However, there are benefits of model-based approaches, including their flexibility to changing reward contingencies and the ability to learn from less data [3, 13].

There are many different approaches to implementing model-based RL, including dynamic programming [71], Dyna variants, e.g.. [72], AIXI [79], and R-max [37, 17]. The approach used in this thesis is novel in that it is designed to be implemented with spiking neurons, and focuses on the the representation, calculation, and learning of state transition probabilities.

1.4.1 State transition probabilities

As defined earlier, the probability of transitioning to a given state after performing a specific action in a given previous state is the state transition probability. In an MDP, these probabilities are represented by the function $P(s, a, s')$. In model-based algorithms that explicitly represent this function, the probabilities are generally stored in look-up tables (which can become impractically large in tasks with large state spaces).

In their paper, Gläscher and colleagues pointed out that model-based reasoning does not require the reward prediction error signal from dopamine [33]. Instead, it uses a state prediction error (SPE), defined by:

$$\delta_{SPE}(s, a, s') = 1 - P(s, a, s') \tag{1.5}$$

for a second state s' for which a transition is observed. Gläscher et al. outlined a model-based algorithm that learns state transition probabilities by updating $P(s, a, s')$ after an observed transition to s' by:

$$P(s, a, s') \leftarrow P(s, a, s') + \eta\delta_{SPE}, \tag{1.6}$$

where η is a learning rate, and decreasing all other possible second states s'' according to:

$$P(s, a, s'') \leftarrow P(s, a, s'')(1 - \eta). \tag{1.7}$$

Once learned, these probabilities can then be used to directly compute the Q-values with a variant of Bellman's equation [18]:

$$Q(s, a) = \sum_{s'} P(s, a, s') \max_{a'} Q(s', a'). \tag{1.8}$$

In studies of human goal-directed behaviour, some evidence has been found of state transition probabilities (or action-outcome contingencies as they are often called [20]), being represented in ventromedial prefrontal cortex (vmPFC) [74, 42, 22] or hippocampus [78, 47]. The evidence for hippocampal involvement comes mostly from the well-known literature on spatial and cognitive maps in hippocampus [51, 76]. Humans do appear to learn and store state transition probabilities (though probably not in a look-up table). One goal of this thesis is to investigate how neurons might be able to learn, store, and use these probabilities.

1.5 Model-free vs. model-based behaviour

As discussed earlier, researchers have drawn comparisons between the habit system and model-free RL, and by analogy, the goal-directed system and model-based RL [19]. The differences between model-free and model-based behaviour have been studied by cognitive and computational neuroscientists alike. This section will examine some of the psychological studies that have been done to differentiate these behaviours, as well as some of the work done to characterize how these two systems could work together from a computational perspective.

In the case of humans, it is certain that we use a more sophisticated learning strategy than simple model-free reinforcement learning; we do not require extensive amounts of data in order to learn, and can react quickly to changing goals. Citing evidence from studies on latent learning and cognitive maps, Tolman [76] argued that learning (even non-human) is primarily goal-directed (model-based). Nevertheless, humans still form habits, and when they do, demonstrate insensitivity to changing goals [77], which suggests that humans use a model-free system as well. Given that there are (at least) two learning systems, the question arose as to how these systems might work together to produce a single behavioural output [54]. Some researchers have argued that these systems are mutually exclusive and competitive, with the Q-value of only one system being ultimately used for action selection, e.g., [19, 15, 57]. Others think that humans may use a complementary mixture of both strategies, so that the final Q-value is a weighted average of those produced by both systems [66, 25].

Keramati, Dezfouli, and Piray [40] looked at how to arbitrate between the goal-directed and habit systems, from a computational perspective. They argued that this is done by placing more weight on values calculated by one or the other system based on a speed/accuracy trade-off. In situations where a course of action is needed quickly, it makes more sense to rely on model-free reasoning, but when an accurate response is required, model-based reasoning is preferred.

In fact, it is difficult to experimentally differentiate model-based from model-free behaviours [25]. The most common type of studies in psychology or neuroscience that aim to separate model-free and model-based characteristics are known as sequential decision tasks. Sequential decision tasks consist of multiple stages with one or more possible states in each stage. In each state, a number of actions are possible. Usually, reward is not delivered until an action is performed in a final, or *terminal*, state. Sequential decision tasks can be succinctly expressed as a decision tree, with each node representing a state, and connections representing actions and state transition probabilities.

Daw et al. [18] have done a great deal of work in identifying characteristics of both model-free and model-based approaches in human data.[1] These authors developed a hybrid RL model which incorporates traditional model-free and model-based systems, with behaviour determined by a weighted sum of the values produced by the two systems. They found that through adjusting this weight, the hybrid model could be well fit to human behavioural data, which they claimed supported the idea that humans use a mixture of both strategies.

The hybrid algorithm used by Daw et al. [18] was based on that used by Gläscher et al. [33]. Their algorithm is almost identical, except that the state transition probabilities were predetermined and hardcoded into the Q-value calculations, and not learned in the way decribed in section 1.4.1 [18]. The model-based algorithms used in these two papers provide a foundation for the two versions of the neural model developed in this thesis.

1.6 Biological plausibility

Recently, theoretical and computational neuroscientists have begun combining both the computer science and cognitive neuroscience approaches to reinforcement learning by developing neural models of RL. The aim of such models is to potentially provide insight into how the brain learns the values of state-action pairs in model-free RL, or how a model of the world is learned in model-based RL. In order to do this, it is important to place constraints on the computations and representations available for explaining the behaviour so that the neural model is biologically plausible. Some of the constraints that are incorporated into the model presented in this thesis are: neural spiking and timing effects (including synaptic time constants), distributed representations, and local synaptic plasticity.

1.6.1 Neurons

This section provides a brief overview of concepts relevant to the biologically plausible modelling of neurons. First, a *neuron* consists of dendrites, a soma or cell body, and an axon. *Dendrites* receive information from other neurons and send it to the *soma*, where this information accumulates until a threshold is reached. Once it is reached, an electrical action potential, or *spike*, is released and propagated down the *axon*, which connects to the dendrites of other neurons. A spike can be characterized mathematically as a Dirac delta function, since it is a rapid rise and fall of electrical activity from a baseline level [21]. The

[1]A more extensive description of the task developed by [18] can be found in section 2.1.

pattern of spikes sent by neurons over some time period is often referred to as the neuron's *activity*, and how quickly the neuron sends spikes is called the *firing rate*.

Axons and dendrites are connected through *synapses*, where the arrival of a spike signals the release of a chemical *neurotransmitter* that travels between the axon and the receiving dendrite. Some synapses are *excitatory*, meaning the release of the neurotransmitter increases the firing rate of the receiving neuron, and some are *inhibitory*, meaning the firing rate of the receiving neuron decreases [32]. Information is propagated through the dendrite in an electrical postsynaptic current (PSC), which can be excitatory (EPSC) or inhibitory (IPSC). Neurotransmitters at a synapse are associated with a *time constant*, which is defined as the rate of exponential decay of the EPSC or IPSC. Longer time constants can produce a smoothing effect on the Dirac delta function-like spike of the presynaptic neuron.

The spiking behaviour of neurons has been characterized with various neuron models. These models vary in computational complexity and biological realism, and the choice of neuron model used in simulations affects the type of conclusions that can be drawn. Very simple neuron models may not be biologically plausibly enough to provide insight into the behaviour of real neurons. A model with more realistic neural dynamics may provide more insight, but it may also take an impractically long time to run due to the larger number of calculations required.

The leaky-integrate-and-fire (LIF) [41] neuron model is one of the most commonly used model due to its computational simplicity while maintaining some biological plausibility [56]. LIF neurons are used in this research. LIF neurons integrate their input over time, increasing their voltage until some threshold is reached, when a spike is released and the voltage is reset. The subthreshold dynamics of LIF neurons are given by a simple differential equation [31]:

$$\tau_{RC}\frac{dV}{dt} = -V + J, \tag{1.9}$$

where V is the membrane voltage, J is the input current, and τ_{RC} is a time constant that describes how quickly the voltage changes. LIF neurons are called "leaky" because the voltage is not integrated perfectly as it approaches the firing threshold, approximating the loss of charged neurons through the cell membrane observed in biological neurons. LIF neuron models often incorporate a *refractory period*, where the neuron dynamics are paused for a period of time after the threshold is reached and the voltage is reset, to capture a similar period observed in real neurons..

Historically, there has been significant debate about how neurons represent the world. Some researchers proposed so-called "grandmother cells" [36], where the brain was assumed to have a single neuron for every single thing in the world, so that one neuron would fire whenever, say, a person saw her grandmother's face. There are many issues with this view which we will not discuss here, and it is now generally accepted that a population of neurons can represent a range of inputs through a *distributed representation*. In this case, each neuron in the population has a different *tuning curve*; that is, each neuron will fire at a different rate for each of the possible stimulus values.

1.6.2 Continous time in RL

Real life operates in continuous time, and so, clearly, do biological neural systems. As discussed in the previous section, the timing of spikes and synapses can have a large effect on the functioning of a neural system. However, in traditional MDP reinforcement learning, timing is essentially arbitrary, and usually thought of in terms of discrete time steps, where state presentation, action selection, and the receipt of reward from the previous time step all take place within one time step [71]. Although work has been done to extend RL to continuous time, many RL tasks are defined with discrete time steps, even those used to investigate human behaviour. The model presented in this thesis highlights some of the effects of using a continuous system to approximate a discrete task.

1.6.3 Neural learning

Section 1.6.1 describes how neurons work, how they pass information to other neurons, and how a population of neurons can together represent a range of stimuli. Another important feature of neurons is their ability to learn. This section will provide some background regarding how neural representations can change over time.

Neurons can adjust their representations over time through *synaptic plasticity*, or the ability to change the strength of synapses. This is usually done by changing the amount of neurotransmitter released by the presynaptic neuron or the amount able to be taken up by the postsynaptic neuron [53]. One of the earliest hypotheses for how neurons learn is often paraphrased as "neurons that fire together, wire together." Now known as Hebbian learning, the idea is that two neurons connected by a synapse will strengthen that synapse if the presynaptic neuron's firing frequently leads to the postsynaptic neuron firing [39].

Spike-timing dependent plasticity (STDP) [44] extends Hebbian learning by including a rule for synapses to weaken. As in Hebbian learning, synapses are strengthened when the presynaptic neuron spikes within some short time before the postsynaptic neuron, and additionally, synapses are weakened when the postsynaptic neuron spikes before the presynaptic neuron.

In neural modelling, the strength of a synapse between two neurons is approximated by a weighted connection. Synaptic plasticity, then, is accomplished by adjusting those weights given some learning rule. Neural network learning rules range in biological plausibility, from traditional backpropagation [58], to Prescribed Error Sensitivity (PES) [43], to the Bienenstock-Cooper-Munro (BCM) rule [14]. For the current work, the PES rule was used, which is described in section 2.4.

1.7 Neural models of model-free RL

Many biologically plausible neural models of RL have been developed [6, 28, 29, 52, 63, 69], most of which focus on associative RL, where the model only learns the immediate rewards. These were created mainly to demonstrate the biological plausibility of a neural learning rule, rather than to create an agent capable of learning complex tasks through RL.

Perhaps the most detailed spiking neural model of model-free RL was developed by Rasmussen and Eliasmith [55]. Their model consists of neural populations representing the current state, all possible actions, and an error signal, as well as incorporating a model of the basal ganglia [70] for action selection. The Q-values were represented by the connections between the action populations and the basal ganglia. Learning was done over these connections using the PES rule. Additionally, this model was capable of learning tasks that were defined by Semi-Markov Decision Processes, where reward, state transitions, and action selection are not constrained to take place within one time step. Rasmussen and Eliasmith later extended this model to perform hierarchical reinforcement learning (HRL) [56], which is more effective at handling large problem domains than basic RL [8].

1.8 Previous neural models of model-based RL

As discussed in section 1.7, a large amount of work has been done on the computational basis of the habit system, often using the TD learning paradigm. In contrast, computations underlying goal-directed control are not as well studied [66], with a particular lack of research into how these computations may be performed by biologically plausible neurons. Some neural models of model-based reinforcement learning reframe the problem from a framework of MDPs and Q-values to one of probabilistic inference [16, 66].

To our knowledge, only one other spiking neural model of model-based reinforcement learning using the MDP framework has been published. In Friedrich and Lengyel's [30] model, each state-action pair is represented by a specific neuron[2], with lateral inhibition between neurons that represent the same state but different actions. The resulting neural network structure is very similar to the decision tree structure of the task being considered. In sequential decision tasks, the state transition probabilities were encoded into the weights of excitatory synapses between neurons in different stages. Although the model did learn the state transition probabilities, it did not do so in the normal fashion of trial-and-error and gradually learning from experience. Instead, the appropriate reward was presented simultaneously and continuously to every neuron representing a terminal state, and the connection weights were adjusted according to a local plasticity learning rule.

In their theory, the spiking rate is directly related to the Q-value of that state-action pair, thus, to obtain the behavioural results of their model, Friedrich and Lengyel [30] ran the simulation for some length of time and calculated the policy based on the Q-values indicated by the spiking activity over that time period. This is not especially biologically plausible, because humans and other animals operate in real time, possibly needing to make decisions based on state-action values at any point.

1.9 Summary

This chapter has provided an overview of the two entwined threads of inquiry into reinforcement learning: the psychological/neuroscientific; and the computational. It is clear that these threads have been very successful in characterizing the habit system through model-free reinforcement learning. It is equally clear that how goal-directed behaviours may come about through model-based algorithms implemented in spiking neurons is not well-understood.

Subsequently, this thesis presents a novel spiking neural model of the state transition probabilities often used in model-based reinforcement learning. It is not a complete model-based RL system, but rather focuses on how neurons can learn the portion of the world model specified by the MDP and use that model to calculate the model-based values of state-action pairs. This thesis investigates the effect of a number of parameters on two versions of the system: one in which the state transition probability function is explicitly represented by connection weights in the neural network, and one in which the probabilities are learned from experience over time, by adjusting the connection weights using a learning rule.

[2]or a group of neurons that all represent all states and actions to different degrees, using function approximation to produce a distributed representation.

Chapter 2 provides a complete description of the model developed for this thesis, including a thorough explanation of the task used to evaluate its performance and the parameters that were investigated in order to characterize the model's behaviour. Chapter 3 presents the results of this parameter exploration, and some discussion of the effects of modelling discrete-time tasks using continuous-time components. Chapter 4.3 continues this discussion, proposes areas for future work, and summarizes the overall contributions of this thesis.

Chapter 2

Model Implementation and Method

The model developed for this thesis is based on the model-based algorithm used by Daw et al. [18] to simulate ideal model-based behaviour on their two-stage sequential decision task. The neural components of the model implement the representation of the state transition probabilities and the multiplication of these probabilities by the appropriate Q-values using spiking leaky-integrate-and-fire (LIF) neurons. The remaining parts are implemented using traditional reinforcement learning approaches.

As mentioned in section 1.5, there are two versions of the neural model, a static model in which the state transition probabilities are directly encoded into the connection weights, comparable to the method used by Daw et al. [18], and a learning model in which these probabilities are learned using a state prediction error signal similar to that done by Gläscher et al. [33]. However, neither of these previous models was designed to be implemented using spiking neurons. This chapter lays out the structure of the static model (which does not learn the state transition probabilities) in section 2.3, followed by the extensions necessary for learning the state transition probabilities in section 2.4. The research in this thesis focuses specifically on the neural computations required for model-based RL, performing other computations directly. This approach is justified by the existence of models already thoroughly characterizing model-free RL using similar methods.

Preceding the model descriptions, section 2.1 provides a detailed explanation of the two-stage decision task introduced by [18], which is used to evaluate the models' performance and behaviours. Section 2.2 presents some additional background on the underlying methodology used to create the neural network model.

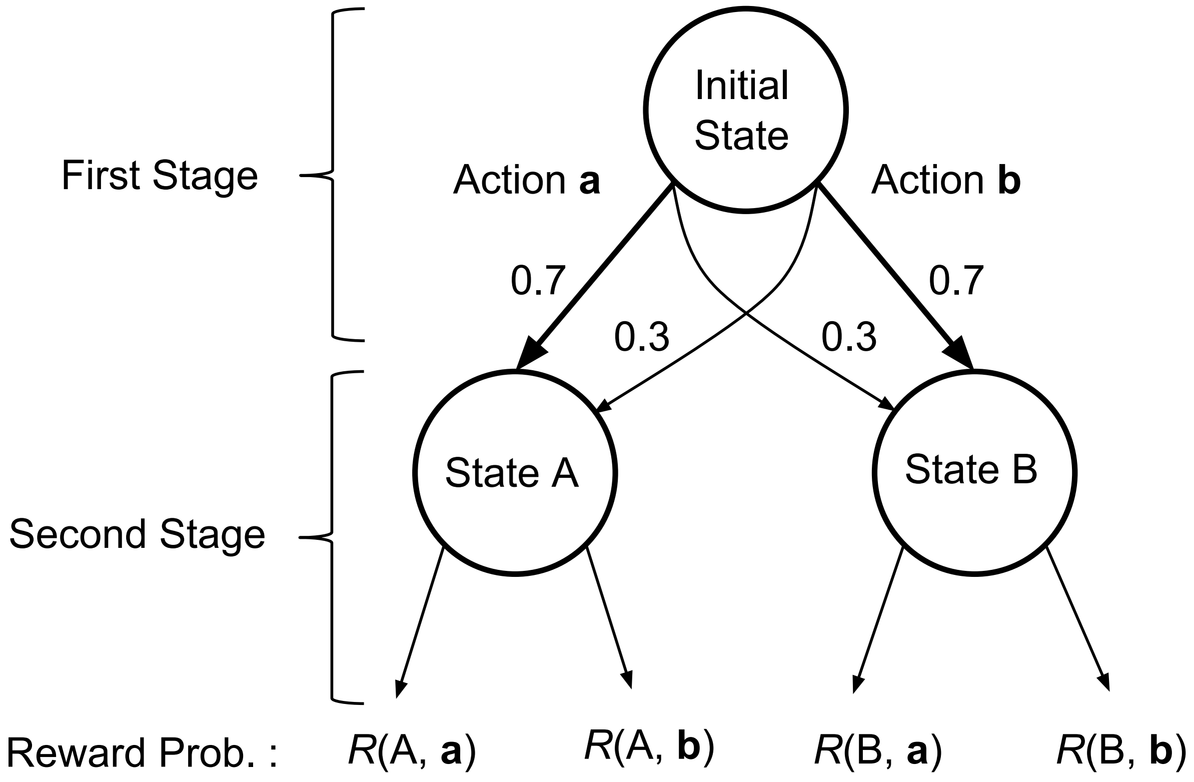
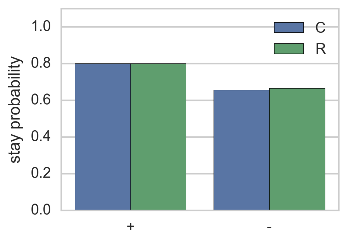


Figure 2.1: Tree diagram of the two-stage decision task. Adapted from [2] and reproduced from [45].

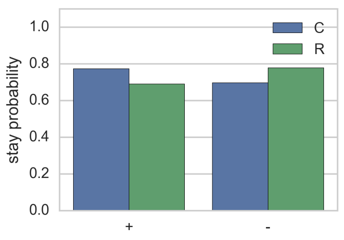
2.1 Two-stage decision task

The task used to evaluate the model's behaviour is the two-stage decision task described in [18], though the model can be easily extended to work on any task with similar characteristics (i.e., sequential decision tasks with discrete time, finite states and actions). The two-stage task was chosen for this thesis because it was designed to produce clearly differentiable behaviours between purely model-free and purely model-based agents.

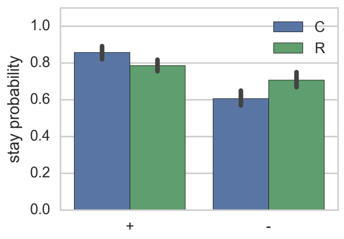
Fig. 2.1 shows a schematic tree diagram of the two-stage task. In the first stage, there is one state, the *initial state*, which has two possible actions (**a** and **b**) [45]. The second stage has two *terminal states* (A and B), which also each have two possible actions. Choosing action **a** in the initial state commonly leads to a state transition to state A, and rarely leads



(a) Ideal model-free behaviour



(b) Ideal model-based behaviour



(c) Human behaviour

Figure 2.2: Characteristic two-stage decision task stay probability behaviour for ideal model-free and model-based agents, as well as human data collected by [18]. State transition probabilities are common (C, prob. = 0.7) or rare (R, prob. = 0.3). Trials rewarded by a value of 1 are denoted with +, and − denotes unrewarded trials (rewarded with 0). Adapted from [18] and reproduced from [45].

to a state transition to state B. Conversely, choosing action **b** in the initial state commonly transitions to state B, and rarely transitions to state A. Common state transitions are set to have a probability of 0.7 and rare state transitions have a probability of 0.3. Actions chosen in the initial state only result in a state transition; they are not rewarded. Actions chosen in the terminal states are rewarded with a value of 1 (reward) or 0 (no reward) with some probability. A single trial consists of a full traversal of this tree, with an agent selecting an action in both stages and receiving reward at the end; then the task resets to the initial state (transitions with probability=1 to initial state), continuing in this fashion for many trials. Whether the agent is rewarded is determined according to reward probabilities in the range $[0.25, 0.75]$, changed slightly each time step by a Gaussian random walk (mean 0, SD 0.025) [18]. Daw and colleagues [18] included this randomness to ensure the agent or human participant learns continually throughout the task, since constantly changing the reward probabilities changes the values of the terminal state-action pairs, and by extension the initial state-action pairs.

In this task, purely model-free behaviours are discriminated from purely model-based behaviours using the stay probability. The *stay probability* looks only at actions chosen in the initial state, and describes the likelihood of choosing the same action in two contiguous trials [18]. Figures 2.2a and 2.2b show the typical model-free and model-based stay probability patterns of behaviour, respectively. For a model-free agent, if a terminal state action is unrewarded (−), the Q-value of choosing the same action in the next trial's first state decreases, and so the stay probability following unrewarded trials is lower than that of the rewarded trials (+). Whether the transition to the terminal state (given the action in the initial state) is rare (R) or common (C) has no effect on this general trend. In contrast, for a model-based agent, the state transition probability has a large effect on the stay probability for rare terminal states. For example, if a model-based agent performs an action in a rare terminal state and is not rewarded (R−), it would decrease the Q-value of that second state, and increase the Q-value of the action it took in the first stage, since that action is actually unlikely to lead to this same terminal state. This would have the effect of increasing the stay probability of R− relative to C−. Similarly, in a rare rewarded state (R+), the model-based agent would increase the Q-value of the second state and decrease the Q-value of the action taken in the first state because it is unlikely to lead back to the high-Q-valued second state.

A specific thought experiment of model-based reasoning may make this more clear. Model-based stay probability behaviour differs from model-free on this task only when there is a rare state transition. So, say the model-based agent is in state B after choosing action **a**. The agent then chooses action **b** and receives a reward, which increases the Q-value of being in that state. Roughly, this Q-value increase (assuming the result is a higher Q-value than state A) will make the agent "want"[1] to return to state B. To do this, the agent knows it should choose action **b** in the initial state, so it will increase the Q-value of action b in the initial state and *decrease* the Q-value of action **a** in that same state. In contrast, a model-free agent in this same situation would *increase* the value of action **a** in the initial state.

As shown in Fig. 2.2c, human stay probability behaviour demonstrates a mixture of model-free and model-based characteristics [18]. Like a model-free agent, the stay probabilities of unrewarded trials are lower than rewarded trials. However, like a model-based agent, there is a statistically significant effect of state transition probability. One of the aims of this thesis is to investigate how a purely model-based system may be implemented in neurons, as a step toward the goal of understanding how model-free and model-based systems in the human brain may produce human behaviours.

2.2 Neural Engineering Framework

The neural components of the model proposed in this thesis are implemented using the principles of the Neural Engineering Framework (NEF) [26]. The three principles comprising

[1]The examples in this chapter will attribute knowledge, desires, and memory to a generic agent for clarity of explanation, and not because agents necessarily have such higher level concepts. That philosophical discussion is beyond the scope of this thesis.

the NEF (representation, transformation, and dynamics) enable the construction of large-scale neural models. Representation and transformation are described in sections 2.2.1 and 2.2.2 respectively. A full explanation of the principle of dynamics is not required for the description of the model implementation.

2.2.1 Representation

In the NEF, populations of neurons represent information as vectors of real numbers that vary over time. Given an n-dimensional stimulus vector $\mathbf{x}(t) = [x_1(t), x_2(t), ..., x_n(t)]$ that varies with time t, a population of neurons encodes this through its activities $a_i(\mathbf{x}(t))$, defined according to the NEF as:

$$a_i(\mathbf{x}) = G_i[\alpha_i \mathbf{x}(t) \cdot e_i + J_i^{bias}], \tag{2.1}$$

where i indexes the neurons in the population, G_i is the nonlinear function describing the neuron's spiking response, α_i is a gain and conversion factor, e_i is the encoding vector which picks out the "preferred stimulus" of the neuron (consistent with the standard idea of a preferred direction vector [62]), and J_i^{bias} is a bias current that accounts for background activity. The terms inside the square brackets describe the current entering the soma of the neuron from the dendrites. In the model presented in this thesis, G_i is the leaky integrate-and-fire (LIF) neuron model [41].

Given the encoding in Eq. 2.1, the original stimulus vector $\mathbf{x}(t)$ can be decoded into an estimate $\hat{\mathbf{x}}(t)$ with:

$$\hat{\mathbf{x}}(t) = \sum_i a_i(\mathbf{x}(t)) * h(t) d_i, \tag{2.2}$$

where d_i are the decoding vectors and $h(t)$ is the postsynaptic current, modelled in LIF neurons as the exponential filter $\frac{1}{\tau}\exp(-t/\tau)$ [26]. The NEF typically finds these decoding vectors, or *representational decoders*, using least squares optimization to minimize the difference between the actual $\mathbf{x}$ and the decoded $\hat{\mathbf{x}}$. The combination of nonlinear encoding in Eq. 2.1 and weighted linear decoding in Eq. 2.2 defines a *population code*, a distributed representation of $\mathbf{x}$ over the population of neurons.

2.2.2 Transformation

Single populations of neurons can represent time varying vectors, but connecting two distinct neural populations allows us to compute functions of these vector variables, through a transformation [26]. That is, if we have a neural population representing x, we can approximate the function $y = f(x)$ through the connections to a second neural population y.

To perform a transformation $f(\mathbf{x})$ in the NEF, the decoding can be reweighted using *transformational decoders* $d_i^{f(\mathbf{x})}$ [26]. In much the same way as representational decoders, transformational decoders can be found using least squares optimization, this time minimizing the difference between the actual $f(\mathbf{x})$ and the decoded estimate $\hat{f}(\mathbf{x})$. The decoding equation is modified slightly when computing transformations:

$$\hat{f}(\mathbf{x}(t)) = \sum_i a_i(\mathbf{x}(t)) d_i^{f(\mathbf{x}(t))}. \tag{2.3}$$

This method can compute both linear and non-linear functions of the encoded vector $\mathbf{x}$. In addition, NEF encoders and decoders can be used to define connection weights between neurons with $w_{ij} = \alpha_j e_j \cdot d_i$, where i indexes neurons in the presynaptic population and j indexes the postsynaptic population.

2.3 Static model structure

This section outlines, in detail, the structure of the static model, with the state transition probabilities directly encoded into a neural transformation using the principle of transformation (see section 2.2.1) of the NEF. The model was constructed using the Nengo neural simulator [10], a Python software package that is based on the NEF principles. Fig. 2.3 shows a schematic diagram of the static model.

The static model uses the NEF to construct two components of a model-based RL agent using spiking LIF neurons [45]. The first component is the representation of the state transition probabilities $P(s, a, s')$, and the second component is the multiplication of $P(s, a, s')$ by the Q-values of future states as in Eq. 1.8. All other necessary components of a model-based agent are implemented using traditional computation. These traditional computations are performed by a Nengo Node (a Python object that allows the non-neural processing of neural outputs and the provision of input signals to a neural model) called *Environment*, shown in rectangles in Fig. 2.3. A full description of the *Environment* node can be found later, in section 2.3.1.

The two neural populations of the model use orthogonal, five-dimensional vectors to represent states and actions [45]. NEF methods allow $\mathbf{x}(t)$ vectors to be non-orthogonal and arbitrarily large, and so this model can be extended in future. The current model uses the

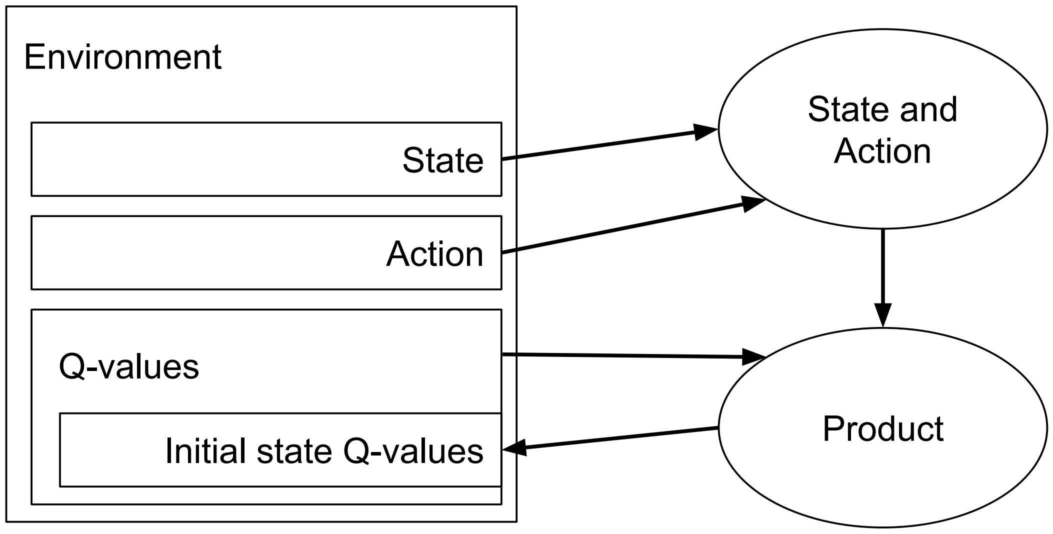


Figure 2.3: Static model diagram. Neural populations are shown as oval shapes. Elements in rectangles use traditional computation. Reproduced from [45].

simplest possible form (given the task structure) for exploration and validation purposes. As such, three dimensions are used to indicate the presence of one of the three states, and two dimensions indicate the two actions. For purposes of explanation in this chapter, three-dimensional vectors are used (i.e., the representation of the considered action is ignored). Let us assume that state A is represented by $S_A = [1, 0, 0]$, state B by $S_B = [0, 1, 0]$, and the initial state by $S_0 = [0, 0, 1]$.

Rather than directly computing traditional model-based equations in neurons, the model gains some efficiency by exploiting the natural parallelism of neural implementations [45]. In particular, state transition probabilities are represented in parallel by a function computed in the connection between two neural populations. This connection maps the state-action pairs represented in the first neural population to a distribution of state transition probabilities with:

$$P(s, a) = [P(s, a, S_A), P(s, a, S_B), P(s, a, S_0)]. \tag{2.4}$$

Given a state-action pair, this function returns the probabilities of reaching each of the three states in the task. For example, in the two stage task, $P(S_0, \mathbf{a}) = [0.7, 0.3, 0.0]$ because when the agent chooses action $\mathbf{a}$ in the initial state S_0, state A is reached with a probability of 0.7, state B with a probabiltiy of 0.3, and the initial state is never immediately reached. Similarly, for any action a, $P(S_A, a) = [0.0, 0.0, 1.0]$. Other tasks with more than three states would have more elements in that probability distribution vector.

The first of the two neural populations is called *State and Action* (see Fig. 2.3) [45]. In the model, it is a 10-dimensional population that represents vectors for both the current state s and the current action a. The second population, called *Product*, is used to calculate the Q-values of the initial state-action pairs.

Traditional model-based agents update the Q-values of the initial state using Eq. 1.8 [18]. Since this neural model uses the parallel representation $P(s, a)$ instead of $P(s, a, s')$, the calculation of the model-based Q-values is done with the following dot product [45]:

$$Q(s, a) = P(s, a) \cdot Q(a'). \tag{2.5}$$

$Q(a')$ is the vector of the best possible action for every state, and is provided to the neural model by the *Environment* node (see section 2.3.1). The first step of this dot product is a multiplication performed by the *Product* population. Generally, multiplications are nonlinear; however, a well-characterized neural implementation was demonstrated by Gosmann [35] that does not require nonlinear interactions (unlike Solway and Botvinick's planning-as-inference model [66]). Gosmann [35] also showed how the NEF can accurately implement this characterization. Using that implementation, the *Product* population performs an element-wise multiply. The final result of Eq. 2.5 is computed by a summation over the output connections from *Product*.

It is worth emphasizing that while the state transition probabilities are computed in parallel, the model-based Q-values for the two actions are not [45]. If the Q-values were computed in parallel, it would require having separate groups of neurons for each possible action, which could become problematic in tasks with large or unknown numbers of actions, because the number of neurons would need to grow accordingly. Instead, to update the Q-values of both possible actions in the initial state, the model uses a serial strategy where both actions are considered one after the other over time. This allows the same group of neurons to be reused for however many actions need to be considered.

Let us look at another specific example to clarify the neural calculation of model-based Q-values [45]. Assume the agent is in the initial state and is currently considering action $\mathbf{a}$, so it "wants" the result of $Q(S_0, \mathbf{a})$ from Eq. 1.8. As earlier, $P(S_0, \mathbf{a}) = [0.7, 0.3, 0.0]$. Say the vector of actions with the highest Q-values in each state $Q(a') = [0.25, 0.75, 0.33]$. The dot product of these two vectors produces a Q-value of $Q(S_0, \mathbf{a}) = 0.4$. Then the agent performs the same procedure to find the Q-value of action $\mathbf{b}$ ($P(S_0, \mathbf{b}) = [0.3, 0.7, 0.0]$) and finds the result of $Q(S_0, \mathbf{b}) = 0.6$. The agent can then use these Q-values to select an action. In this model, action selection and the calculation of $Q(a')$ are done by the *Environment* node.

2.3.1 The ***Environment*** node

The *Environment* node stores the current state, the currently considered action, and the Q-values for each state-action pair, and sends these values as input to the appropriate neural populations. It receives the output from the neural calculation of the initial state's Q-values. The *Environment* node also manages the action selection, delivering the reward, updating the Q-values of the second stage states, changing the reward probabilities as determined by a Gaussian random walk, and doing all of the above at the appropriate time as determined by the *time interval* (the length of time between state transitions).

As a slight aside, "*Environment*" may not be the most accurate name, since in addition to handling the traditionally "environmental" actual state transitions and reward presentation, the node also stores Q-values and performs action selection. However, while these components are not external to the "agent", they are external to the neural aspects of the model, and so referring to all non-neural components as "*Environment*" is a convenient shorthand.

The two-stage decision task is thought of as a discrete task in most simulations. Each state presentation and choice of action are taken as a single time step. However, biological neural systems operate continuously, which is simulated in Nengo by discretizing time into very small steps. Thus, when using a Nengo model to represent the two-stage task, there are two levels of time step: the Nengo continuous approximation (usually around 1ms) and the task's much longer length of time of state presentation (usually more than 50ms). The *Environment* node keeps track of how much time has passed, and selects actions and transitions to an appropriate state at the correct time.

The neural components of the model calculate the Q-values of the initial state in a model-based manner. In contrast, the Q-values of actions in terminal states are learned using a model-free update strategy:

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha r, \tag{2.6}$$

where α is a model-free learning rate parameter, and r is the reward received at the end of a trial. Using model-based calculations for the first stage and model-free calculations for the terminal stage is consistent with previous non-neural models of sequential decision tasks [18, 33, 2]. Eq. 2.6 differs slightly from the standard SARSA update (Eq. 1.4) because there are no future states after the terminal states.

As mentioned earlier, the *Environment* node provides the vector $Q(a')$ (the vector of the Q-values of the best possible action for every state) as input to the *Product* network. This is simply the maximum Q-value for each state.

In order to ensure exploration, the *Environment* node performs action selection by approximating a softmax [45]. That is, for a given state, before selecting the action with the highest Q-value, a small amount of Gaussian noise (mean 0, SD 0.05) is added to all the Q-values of the possible actions in that state.

After a reward is given (at the end of a terminal state), the reward probabilities are updated according to a Gaussian random walk with a standard deviation of 0.025, bounded by $[0.25, 0.75]$ [18]. This is implemented by determining the noise values to be added to each of the reward probabilities. Before the addition is done, it is checked whether the result would fall outside the specified range, and if so, the value is instead subtracted.

Pseudocode for the *Environment* node is shown in Listing 2.1. It anticipates the next section (2.4) slightly, because the version of the model that learns the state transition probabilities requires an extra vector to be provided to the neural components – that of the currently chosen action. Otherwise, the *Environment* node has the same functionality in both versions of the model.

Listing 2.1: *Environment* node pseudocode

```
-pick an action to consider first
-for each Nengo time step:
  -if after the first half of the time interval:
    -set first half of time interval to one time interval later
    -store the Q-value that has been calculated by the neurons
    -switch to considering the other action
  -if after the second half of the time interval:
    -set second half of time interval to one time interval later
    -store the Q-value that has been calculated by the neurons
    -select action
    -if in the second stage, give reward and determine new
     reward probabilities
    -switch back to considering the first action
  -find the max values in each state to form Q(a') vector
  -send to the neural parts the vectors representing the current
   state, the currently considered action, and Q(a')
  -(if learning, also send the currently chosen action)
```

2.4 Learning model structure

The state transition probabilities represented by the connection between the *State and Action* and the *Product* populations can also be learned. Learning the function $P(s, a, s')$, or in the neural formulation, the distribution $P(s, a)$ in Eq. 2.4, can be done in a number of well-studied ways that involve adjusting the connection weights between these two neural populations (see section 1.6.3). As mentioned earlier, the model uses the PES rule to do this.

PES [43] is an online learning rule that minimizes the error E between the response obtained, y_d, and the response required, y. The PES rule changes the decoders d_i by:

$$\Delta d_i = \kappa(y - y_d)a_i = \kappa E a_i, \tag{2.7}$$

where κ is a scalar learning rate and a_i is again the neural activities. The PES rule can be put in terms of the connections weights w_{ij} by multiplying by the encoder e and gain[2] α of the postsynaptic neurons j, the result of which is a form of error-driven Hebbian learning:

$$\Delta w_{ij} = \kappa \alpha_j e_j \cdot E a_i. \tag{2.8}$$

The intuitive way to incorporate learning into this model would be to include an *Error* population that calculates $(y - y_d)$ from Eq. 2.7, the error between the predicted state and the state that was observed, and uses that to modify the decoders on the connection between the *State and Action* population and the *Product* network. However, simply doing this leads to two problems: that of timing; and that of learning over the correct neural activities.

The first problem, timing, arises because of the structure of the static model. The *Environment* node can send the current state to the *State and Action* and the *Error* populations, but the *Product* population sends the predicted *next* states to the *Error* population. The state transition probabilities predict the likelihoods of observing the possible next states given the current state, so they are essentially predictions of the future. Clearly, the error between the future prediction and the current state is not the correct error to minimize. Instead, we need the error between the prediction and the actual next state, which is unknown until the next time interval. The strategy chosen to solve this problem uses a pure time delay [80] of the length of the time interval on the connection between

[2]The gain parameter α_j should not be confused with the model-free learning rate parameter α from Eq. 2.6. When varying the parameter α in chapter 3, we will exclusively refer to the model-free learning rate.

the *Product* network and the *Error* population (highlighted in red in Fig. 2.4). This will then calculate the error between the current state and the previous time interval's state prediction. However, now this error is the error of the previous time interval, and cannot be used to directly modify the decoders on the *State and Action* to *Product* connection. To solve this problem, a similar delay can be set on the learning rule, so that the error modifies the decoders of the past time interval. The result of this is a network that learns to predict the future states [68].

This modification may raise the question of why predicting the future state is necessary, since it seems that learning the present state from past predictions should be mostly equivalent to learning the future state from current predictions. To illustrate why predicting the future state is necessary, a variant of the learning model was created. This variant only includes a pure time delay on the connection between the *Environment* and the *State and Action* population. With this modification, the *Error* population is still calculating the difference between the current state and the previous time interval's state prediction; however, the Q-value calculation done on the connection from *Product* to *Environment* is now one time-step behind. Section 3.2.4 presents the results of this modification, and demonstrates clearly why future state prediction is needed.

One final change must be made to ensure that this network learns over the correct neural activities. In the static model, the action sent from the *Environment* to the *State and Action* population is the currently considered action, not the action that will be ultimately chosen. The *Environment* node of the learning model must keep track of the action that was chosen in the previous time interval and use this information in a new *Gate* node. The *Gate* node allows the error signal to pass through to the learning rule on the *State and Action* to *Product* connection only when the considered action is the same as the chosen action, otherwise it sends an error of 0. The resulting model is shown in Fig. 2.4.

2.5 Parameter exploration

To investigate and characterize the behaviour of the static and learning models on the two-stage decision task, several parameters are explored:

1. The **learning rate** α of the model-free Q-value update of the terminal states, as in Eq. 2.6. This parameter determines the extent to which the receipt of a reward affects the stored Q-values. Higher values of α put more emphasis on the most recently received reward.

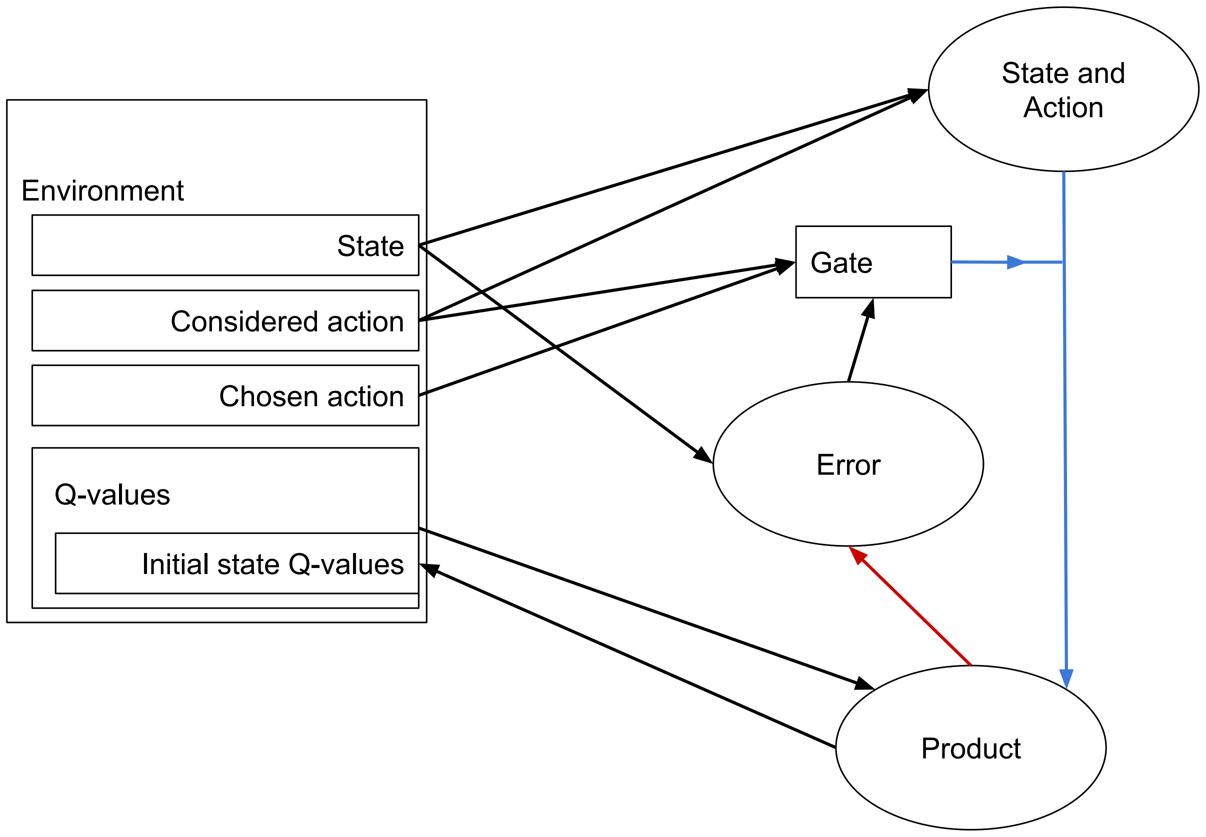


Figure 2.4: Diagram of state transition probability-learning model. As in Fig. 2.3, the components of the model shown in oval shapes are simulated populations of neurons and the rectangular components are directly computed without neurons. The connection between the *State and Action* population and the *Product* population (highlighted in blue) represents the state transition probabilities that are learned over time using the PES rule. The connection highlighted in red has a pure time delay.

2. The **number of neurons** in each of the *State and Action*, *Product*, and *Error* populations (where applicable). In general, increasing the number of neurons should increase the accuracy of the representations and transformations.

3. When instantiating a network, Nengo uses a random seed to determine the random properties of the neurons, such as the neural tuning curves. Therefore, different **individual seeds** may produce different behaviours.

4. The **time interval**[3] between state transitions. Time intervals must be sufficiently long for the neurons to compute the Q-values, otherwise they are expected to produce uniform stay probability behaviour.

5. The **synapse length** on the connection between the *Product* network's value calculation and *Environment*'s Q-value store. Increasing the synaptic time constant effectively takes an average of the neural activity over a longer time period, which can produce a smoothing effect on the decoded signal. This may produce more consistent stay probability behaviour between variations of the other parameters.

Each simulation is run for 10000 trials. Each parameter variation is tested with twenty simulations with different individual seeds. When the individual seed itself is tested, each simulation uses a different random seed for the *Environment* node. The *Environment* random seed is used to make state transitions based on the state transition probability, and to produce random noise for the action selection and the random walks of reward probabilities.

[3]The time intervals used in the simulations are much shorter than that used in human experiments (participants had to respond within 2s, but it was not reported how long participants actually took [18]), however, using human time scales was not practical for the model simulations because they would take an infeasibly long time to run. Also, a human has many more processes happening in that 2s time frame, including sensory processing and motor control, so the shorter time intervals for the model's more limited computations is probably not unreasonable.

Chapter 3

Results

This chapter presents the results of the parameter exploration of the static model implementation in section 3.1 and the learning model in section 3.2. These parameters are the model-free learning rate (α), the number of neurons in the neural populations, the Nengo individual, the time interval between state transitions, and (for the static model only) the synaptic time constant.

The results presented in this chapter are in the form of stay probabilities. The four cases of common-rewarded (C+), common-unrewarded (C−), rare-rewarded (R+), and rare-unrewarded (R−) were calculated for each simulation after all trials in a simulation were run. For every trial, we recorded the action chosen in the initial state, the resulting terminal state, and whether the action in the terminal state was rewarded. These values were used to determine whether a state transition was rare or common and whether there was a *stay*. Then the stay probability was calculated by diving the number of stays in each case by the total number of instances of each case.

The patterns of stay probabilities, or the *stay probability behaviour*, are tested for significance using 95% bootstrap confidence intervals. For all of the data plots presented in this chapter, the error bars show these confidence intervals. In cases where error bars are not visible, the data points are larger than the confidence intervals.

3.1 Static model

While varying the parameters described in the following subsections, default values for the other parameters are used, unless otherwise specified. These values are:

- learning rate *alpha* = 0.3
- number of *State and Action* neurons = 500
- number of *Product* neurons per dimension = 200
- time interval = 0.05s
- synapse time constant = 0.005s
- Nengo time step = 0.001s

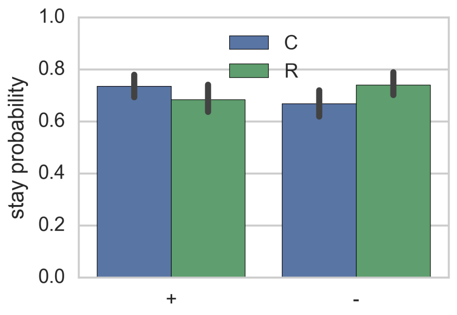
3.1.1 Learning rate α

Fig. 3.1 shows the stay probabilities for four different learning rates [45]. For α values of 0.3 and higher, the stay probability behaviour is characteristically model-based, as expected. These examples also show that higher learning rates produce higher stay probabilities for the C+ and R− cases, and lower stay probabilities for R+ and C− cases. That is, the behaviour becomes more characteristically model-based. This general trend is also shown in Fig. 3.2.

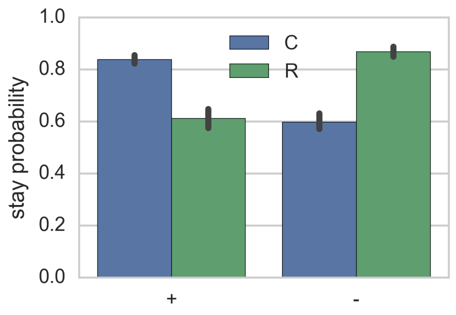
Equation 2.6 shows that higher values of α decrease the model-free learning of the terminal states' Q-values, which suggests that the model-based system may have a stronger influence on the model's behaviour as α increases [45]. This suggestion is supported by the trend found. For instance, when $\alpha = 1.0$, there should be no influence of model-free learning and therefore the most separation between C+, R− and R+, C− cases (see Fig. 3.1d). With a much lower α, the terminal states' Q-values are learned much more slowly, to such an extent that it interferes with the model-based reasoning and produces more uniform stay probabilities (see Fig. 3.1a).

3.1.2 Number of neurons

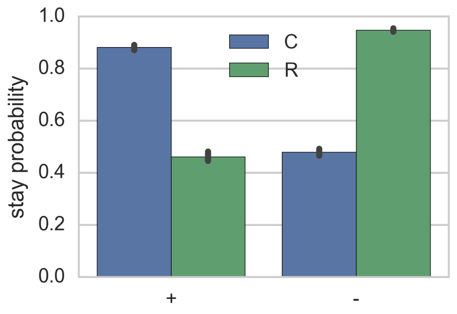
The number of neurons in the *State and Action* population determines the accuracy of the neural approximation of the state transition probabilities [45]. Since increasing the number of neurons in a population usually improves the accuracy of the neural approximation, we would expect that increasing the number of neurons in the *State and Action* population would improve the accuracy of the model-based computations, and therefore have a similar effect to increasing the learning rate α. As shown in Fig. 3.3a, the characteristic separation



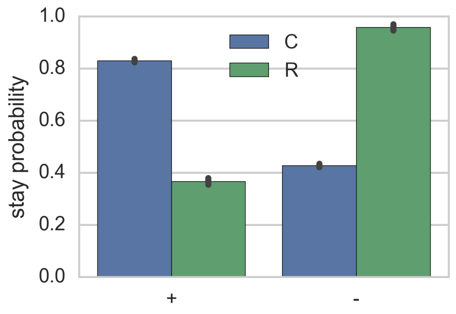
(a) $\alpha = 0.05$



(b) $\alpha = 0.3$



(c) $\alpha = 0.7$



(d) $\alpha = 1$

Figure 3.1: Examples of stay probability behaviours of various learning rates α. Adapted from [45].

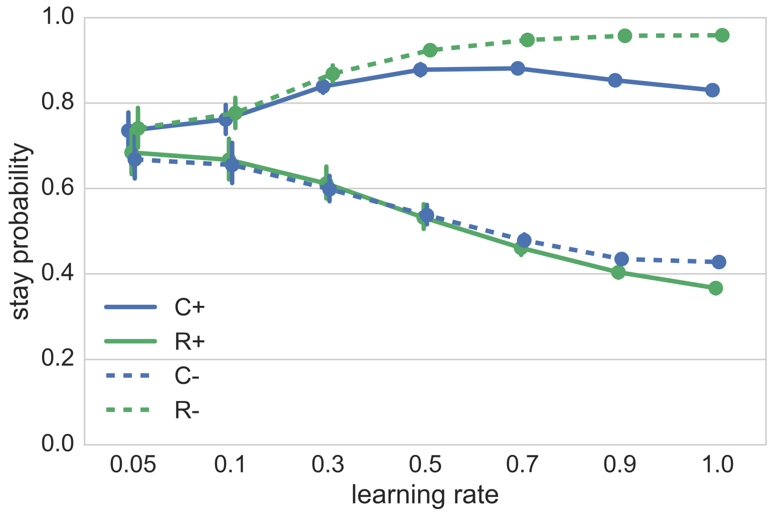


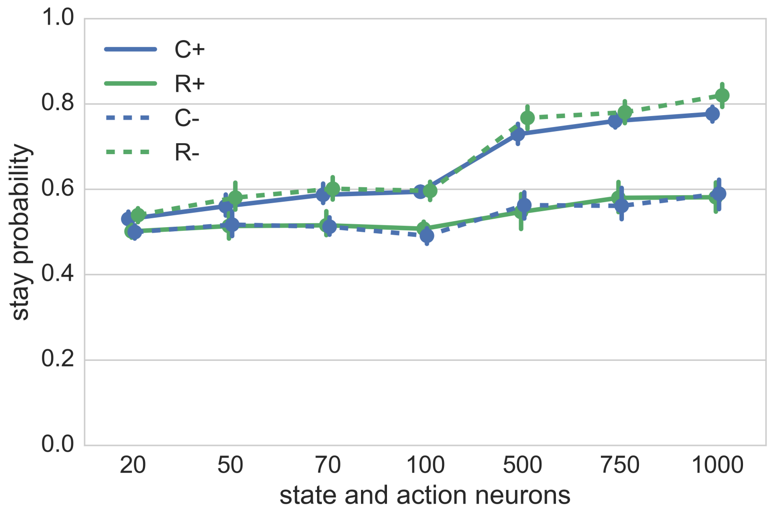
Figure 3.2: Stay probabilities of the different learning rates α. Adapted from [45].

between cases does increase as the number of neurons increases, with populations of more than 100 neurons producing clearly model-based behaviour.

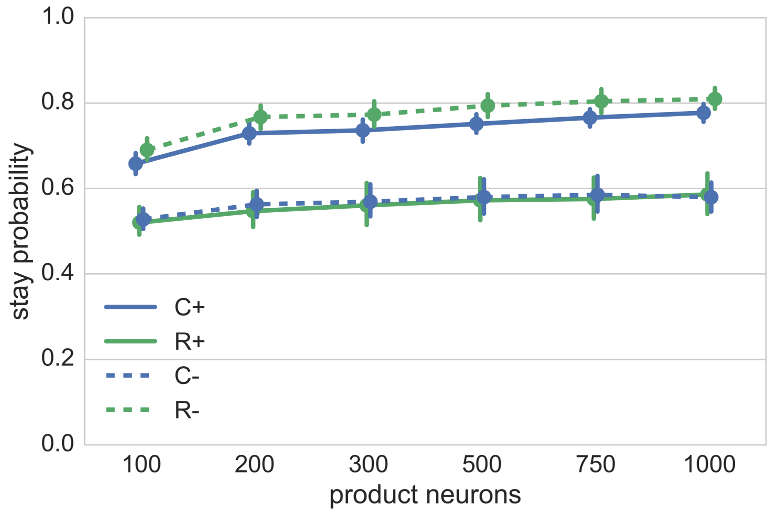
The number of neurons per dimension in the *Product* population does not have much effect on the overall behaviour of the static model, as shown in Fig. 3.3b [45]. This suggests that the precision of this calculation is not critical to the performance of the network.

3.1.3 Individual seeds

When the static model's behaviour is averaged across individual seeds, it appears to be characteristically model-based [45]. However, there are large individual differences between seeds, which fall into one of three categories: *model-based*, *human-like*, and *opposite*. Fig. 3.4 shows examples of each of these categories. Characteristically *model-based* behaviour is shown in Fig. 3.4a. Fig. 3.4b demonstrates a significant differences between the stay probabilities of the R+ and C− cases that is reminiscent of that found in human data by [18] (*human-like*). The third category, shown in Fig. 3.4c, has a significant difference between R+ and C− cases in the *opposite* direction to human data. The number of individuals (out of twenty simulations) that show these three categories of stay probability behaviour is shown in Table 3.1. Most individual seeds show characteristically model-based

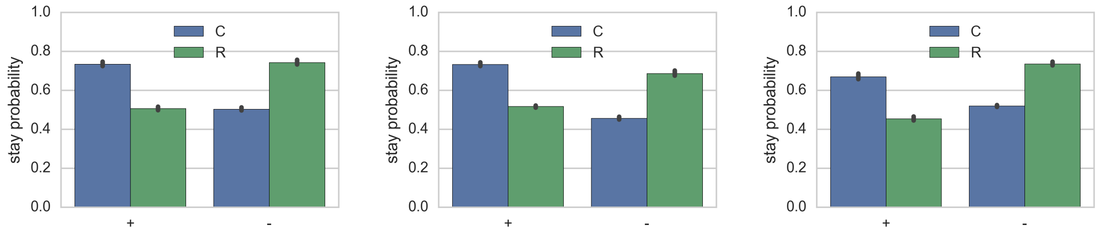


(a) *State and Action* population. Model-based behaviour is clearly distinguishable with 100 or more neurons.



(b) *Product* population.

Figure 3.3: Stay probabilities of different numbers of neurons. Adapted from [45].



(a) seed = 8 (b) seed = 1 (c) seed = 7

Figure 3.4: Examples of individual differences between Nengo seeds: (a) pure model-based stay probability behaviour, (b) stay probability behaviour suggestive of human data, and (c) stay probabilities opposite to those in (b). Reproduced from [45].

behaviour.

Table 3.1: Individual seed differences in static model

Stay probability behaviour	Model-based	Human-like	Opposite
Number of individuals	11	4	5

3.1.4 Time interval

The stay probabilities of different time intervals between state transitions are shown in Fig. 3.5 [45]. Time intervals 0.05s or longer show characteristic model-based behaviour, and as expected, shorter time intervals are insufficient for neurons to compute the Q-values. When averaged over individual seeds, there is no noticeable difference in the stay probability behaviours for time intervals longer than 0.1s.

However, within an individual seed, there are differences in the stay probability behaviour between time intervals that are not dependent on the length of the time interval, nor on any other systematic difference that we have yet investigated [45]. Fig. 3.6 shows examples of the three categories of stay probability behaviour when the individual seed is 1. The almost mirrored stay probability behaviours of Figs. 3.6b and 3.6c is particularly surprising, since there is only a 0.01s time difference between the time intervals.

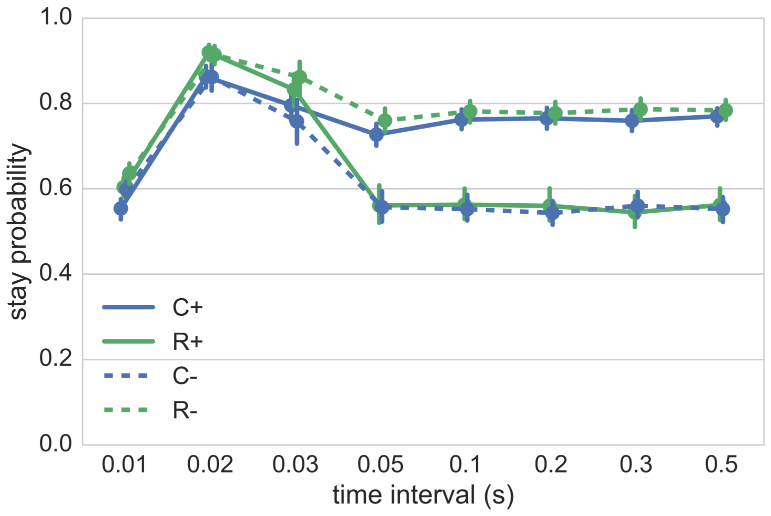
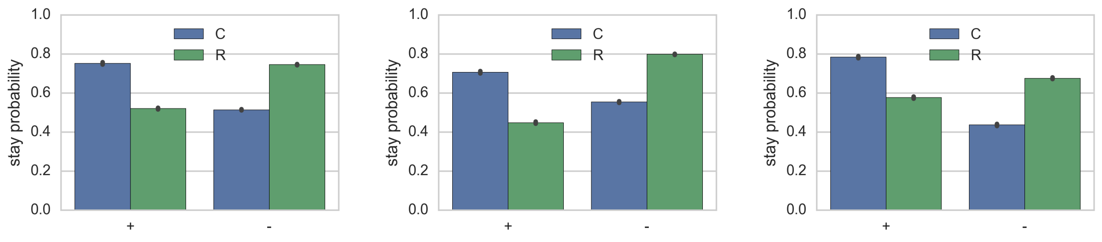


Figure 3.5: Stay probabilities of different time intervals.



(a) time interval = 0.2s (b) time interval = 0.5s (c) time interval = 0.51s

Figure 3.6: Individual differences for various time intervals with seed = 1. Adapted from [45].

3.1.5 Synapse length

Increasing the synaptic time constant between the *Product* population and the *Environment* node can produce a smoothing effect on the decoded signal, effectively taking an average of the neural activity over a longer time period. This means that the state transition probabilities used by the *Product* network are not as influenced by neural noise, and thus could reduce the individual differences between individual seeds and time intervals.

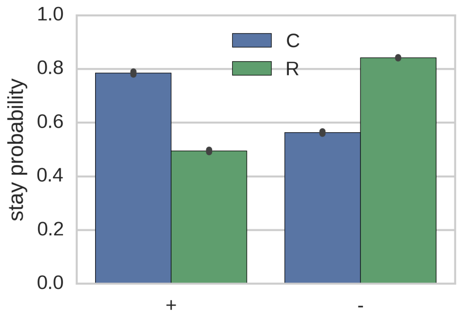
With a long enough time interval, increasing the synapse length greatly diminishes the individual differences between time intervals. For example, Fig. 3.7 shows the stay probabilities with an individual seed of 1 at time intervals of 0.5s and 0.51s, with different synapse lengths. With a synapse length of 0.1s, both time intervals show almost identical, model-based behaviour, unlike those same time intervals with shorter synapse lengths, including the synapse length of 0.005s shown earlier in Figures 3.6b and 3.6c.

However, synapse lengths that are too long relative to the time interval produce behaviour that does not fall into the three general classes defined in section 3.1.3. Fig. 3.8 compares long and short synapse lengths. This data suggests that it may be that a good rule of thumb for the synaptic time constant to be between 10-25% of the time interval length, with the caveat that time constants longer than 0.2s are very uncommon in biological brains [67].

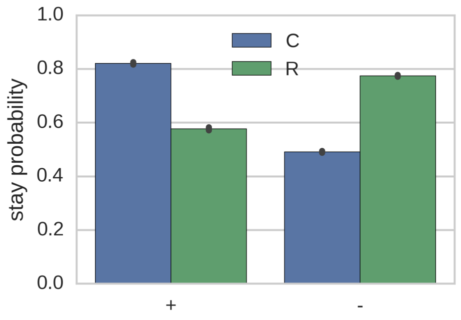
3.1.6 Static model summary

The static model implementation provided some insights into the factors affecting the behaviour of a neural mode-based RL agent. The main finding is that when implemented in neurons, the stay probability behaviour generally matches that of a traditional model-based agent. The individual differences between individual seeds and time intervals found with the default synaptic time constant disappears with longer time constants, suggesting that this variation is mostly due to neural noise.

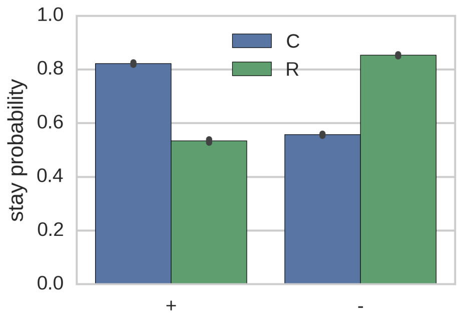
Varying the numbers of neurons in the two neural populations demonstrates the importance of having an accurate calculation of the state transition probabilities. Section 3.2 describes the results of the model implementation that learns the state transition probabilities throughout the task, rather than directly representing them during the initialization of the model.



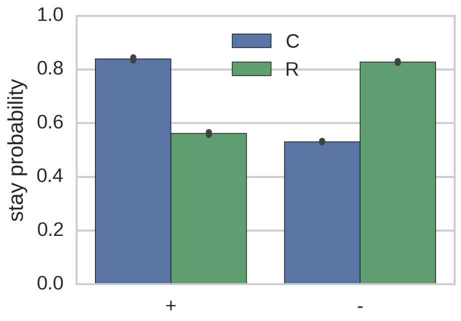
(a) synapse = 0.01s,
time interval = 0.5s



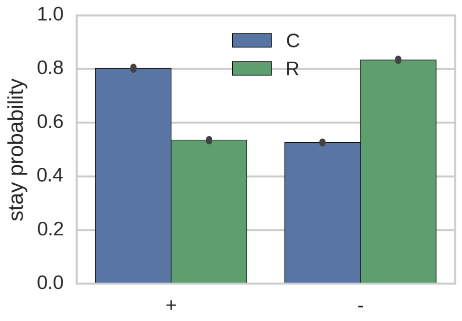
(b) synapse = 0.01s,
time interval = 0.51s



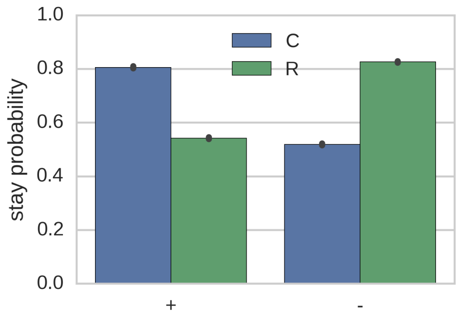
(c) synapse = 0.025s,
time interval = 0.5s



(d) synapse = 0.025s,
time interval = 0.51s

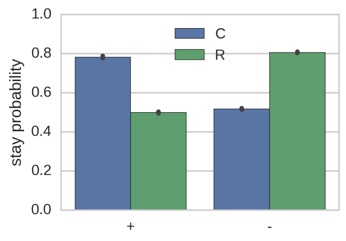


(e) synapse = 0.1s,
time interval = 0.5s

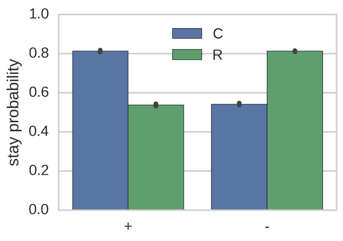


(f) synapse = 0.1s,
time interval = 0.51s

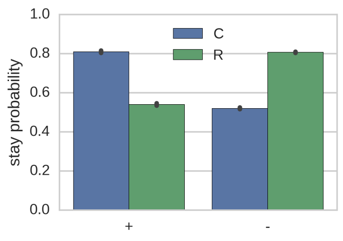
Figure 3.7: Individual differences for time intervals of 0.5s and 0.51s as the synapse length increases.



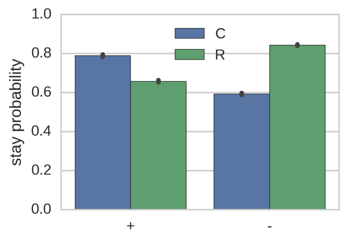
(a) synapse = 0.01s, time interval = 0.1s



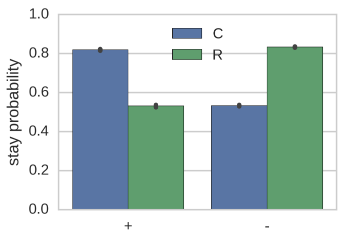
(b) synapse = 0.01s, time interval = 0.2s



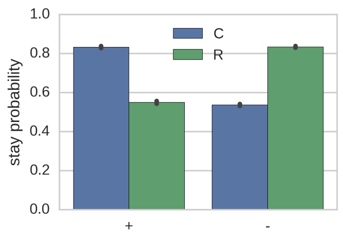
(c) synapse = 0.01s, time interval = 0.3s



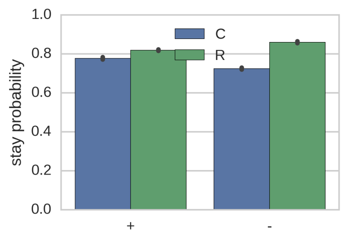
(d) synapse = 0.025s, time interval = 0.1s



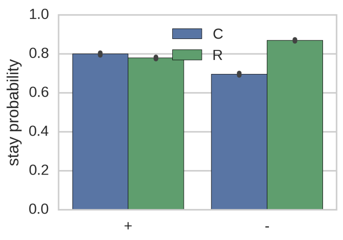
(e) synapse = 0.025s, time interval = 0.2s



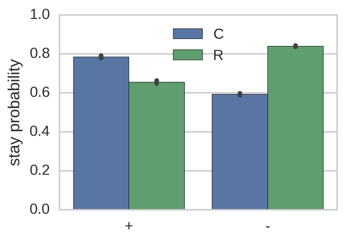
(f) synapse = 0.025s, time interval = 0.3s



(g) synapse = 0.1s, time interval = 0.1s



(h) synapse = 0.1s, time interval = 0.2s



(i) synapse = 0.1s, time interval = 0.3s

Figure 3.8: Individual differences for time intervals of 0.1s, 0.2s, and 0.3s with different synapse lengths.

3.2 Learning Model

The default parameters used in the learning model are the same as those used in the static model, except for the time interval between state transitions, which is 0.1s. This is because there is considerably more variance between individual simulations for the learning model when using a time interval of 0.05s than there is for the static model. Additionally, the learning model has another neural population, *Error*, which has a default of 500 neurons.

3.2.1 Learning rate α

The learning rate again refers to that used by the model-free learning of the Q-values of the terminal states. The general pattern found in the static model is also present in the learning version with the default time interval of 0.1s, shown in Fig. 3.9. As the learning rate increases, the characteristic model-based separation between the stay probabilities of the C+ and R−, and R+ and C− cases is increased. When the time interval is 0.05s, the C+ and R− stay probabilities start high (around 0.9) and remain there as alpha increases (shown in Fig. 3.10). There is also more variance between individual trials in the C− and R+ cases, as shown by the visible 95% bootstrapped confidence intervals for all values of alpha. However, the general effect of the learning rate is not significantly altered by the introduction of a population that learns the transition probabilities.

3.2.2 Number of neurons

The number of neurons in the *State and Action* population does not seem to have as much of an effect on the learning model as on the static model (see Fig. 3.11a). This shows one of the benefits of learning: the behaviour is more robust even when using fewer neurons. Even with only 20 neurons, the model-based effect is clearly visible.

The *Product* network also seems to need fewer neurons to reproduce the stay probability behaviour of the basic model. In fact, using only 50 neurons per dimension produces comparable behaviour to the basic model using 200 neurons per dimension (see Fig. 3.11b). For large numbers of neurons, the stay probabilities of all C/R and +/− cases are higher in the learning model than the static model. This effect is unexpected. The trend may appear because more accurate multiplication leads to higher stay probabilities, or it may be a side effect of the time course of learning. Future work can be done to investigate whether this still trend appears if the stay probabilities are only calculated for the trials in some final time period of the simulation.

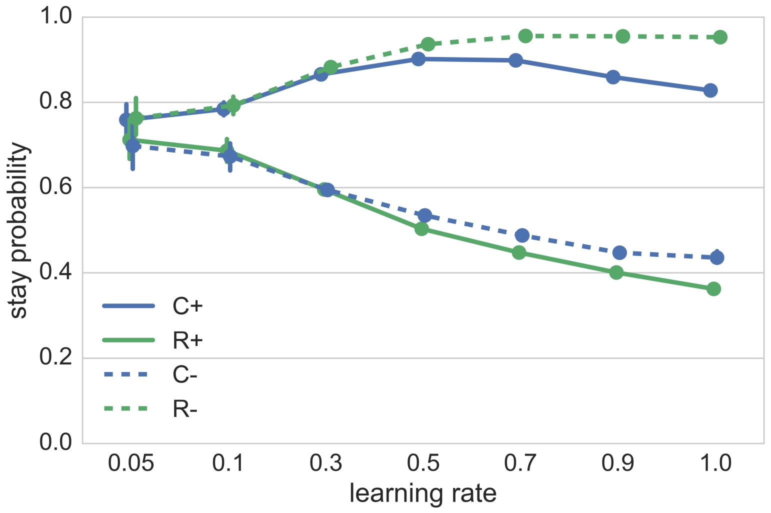


Figure 3.9: Stay probabilities of different learning rates for the learning model with a time interval of 0.1s.

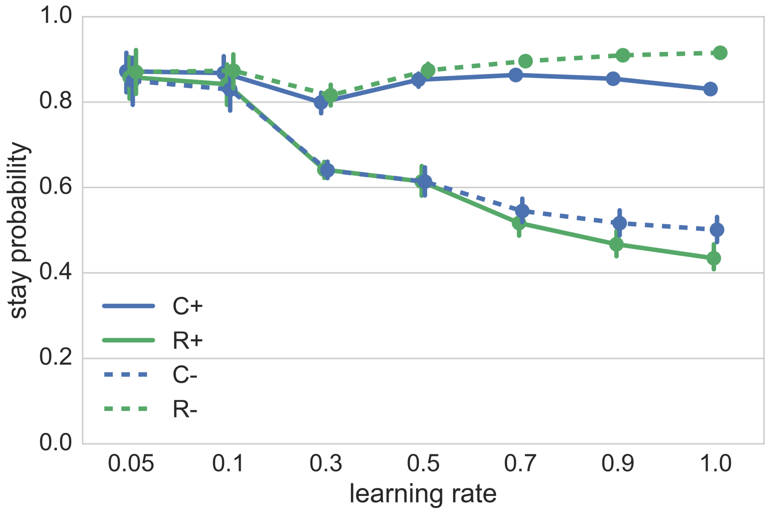
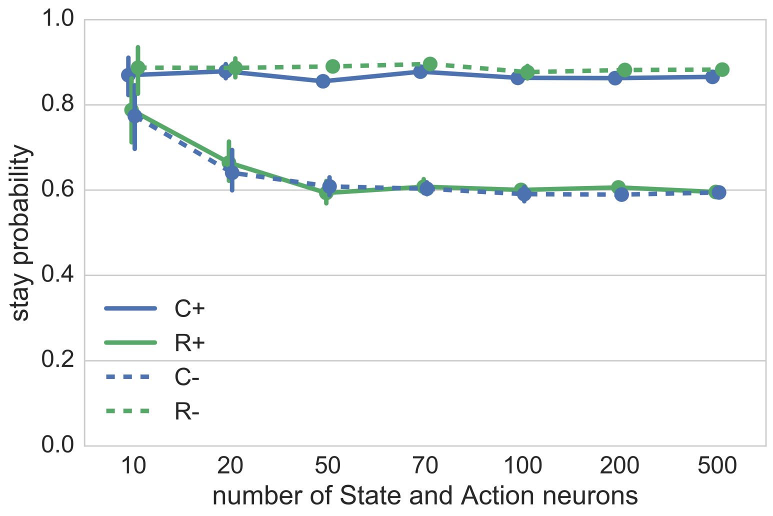
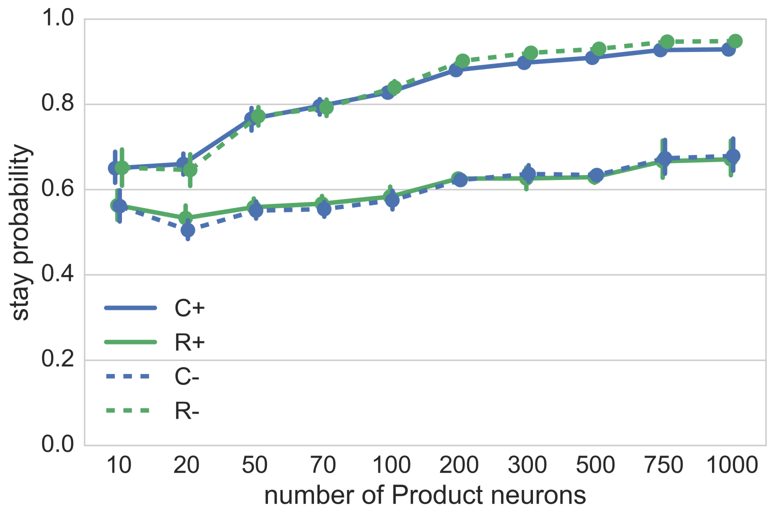


Figure 3.10: Stay probabilities of different learning rates for the learning model with a time interval of 0.05s.



(a) *State and Action* population.



(b) *Product* population.

Figure 3.11: Stay probabilities of different numbers of neurons.

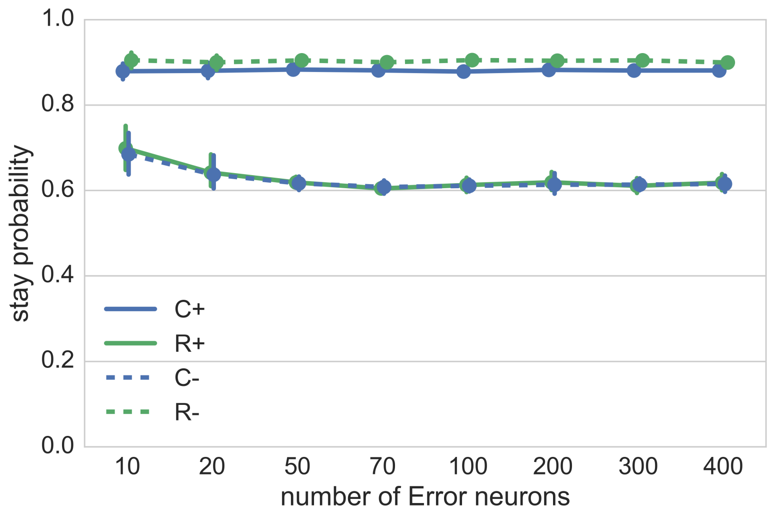


Figure 3.12: Stay probabilities of different numbers of neurons in the *Error* population.

As shown in Fig. 3.12, the number of neurons in the *Error* population does not have a significant effect on the stay probability behaviour when there are more than ten neurons. This trend suggests that accurate error representation is not critical to the performance of the model. There are a number of possible reasons for this, which can be investigated in future work. It may be due to the simplicity of the task, with only four state transition probabilities to learn, or it may be due to the simplicity or orthogonality of the vectors representing the current state and action. It may be that simply representing the sign of the error would be sufficient.

3.2.3 Individual seeds

When using the default neuron numbers, all individuals produced significantly model-based behaviour. Table 3.2 shows the number of each pattern of stay probability behaviour for the learning model with the default number of neurons in each ensemble, as well as with 50 *State and Action* neurons, 50 *Error* neurons, and 70 neurons per dimension in the *Product* network. With the default number of neurons, there are still some small individual differences, as shown in Fig. 3.13a. There are much larger individual differences using the smaller number of neurons, shown in Fig. 3.13b. These results suggest that the default

Table 3.2: Individual differences in learning model

Stay probability behaviour	Model-based	Human-like	Opposite
Number of individuals (default)	20	0	0
Number of individuals (less neurons)	9	3	8

number of neurons should be used in situations when consistent behaviour across seeds is desired, for instance, if this model is incorporated into a larger system.

However, to produce the same behaviour as the static model, the learning model only needs 50 neurons in each of the *State and Action* and *Error* populations, and 70 neurons per dimension in the *Product* population. In terms of total number of neurons in the two models, the learning model requires 10% of the number of neurons used in the static model.

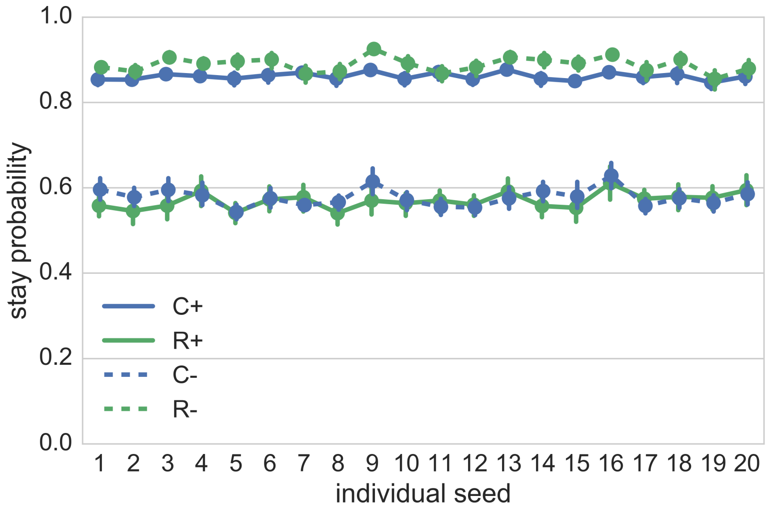
3.2.4 Time interval

Again as expected, longer time intervals tend to produce clearer separation between the stay probabilities of the C+ and R−, and R+ and C− cases, with time intervals that are too short producing uniform stay probabilities, as shown in Fig. 3.14. The main difference between this figure and the one for the static model (Fig. 3.5) is found at the time intervals of 0.05s and 0.1s. In the static model, there is no significant difference between these time intervals, while in the learning model, there is less characteristic separation between the cases at 0.05s, and more separation at 0.1s, leading to a difference in stay probability behaviour. It is unclear whether this difference is statistically significant, since 95% bootstrap confidence intervals do not take multiple comparisons into account. However, it is significant in a practical sense, because simulations run with a 0.05s time interval often produce almost overlapping 95% bootstrap confidence intervals so that characteristic model-based stay probability behaviour is not clearly distinguished, as in Fig. 3.15, while simulations using a time interval of 0.1s are much easier to distinguish.

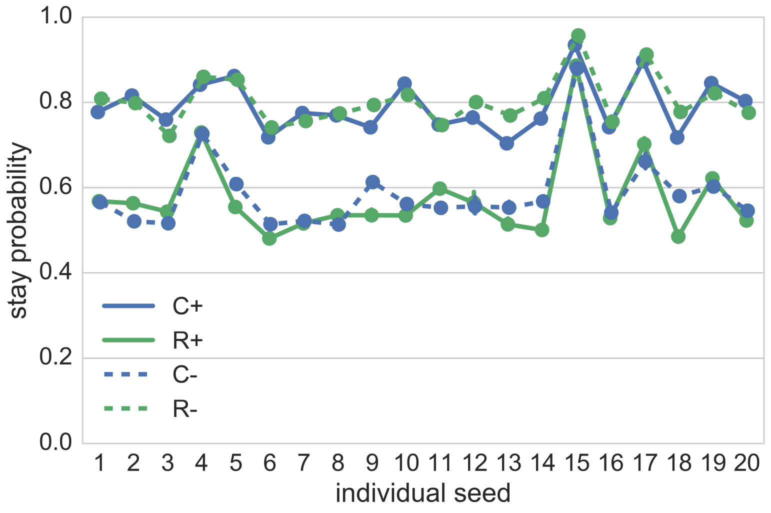
This trend is also shown by the difference between Figures 3.11a, which shows a time interval of 0.1s, and 3.15, which shows a time interval of 0.05s. With the shorter time interval, increasing the number of neurons does not greatly improve the performance.

“Present” learning variant

Surprisingly, the variant of the model that learned the present rather than the future (see section 2.4) displayed model-based stay probability behaviour when the time interval was



(a) Stay probabilities of different individuals with the default number of neurons in each ensemble.



(b) Stay probabilities of different individuals with a small number of neurons in each ensemble.

Figure 3.13: Comparison of small and default numbers of neurons.

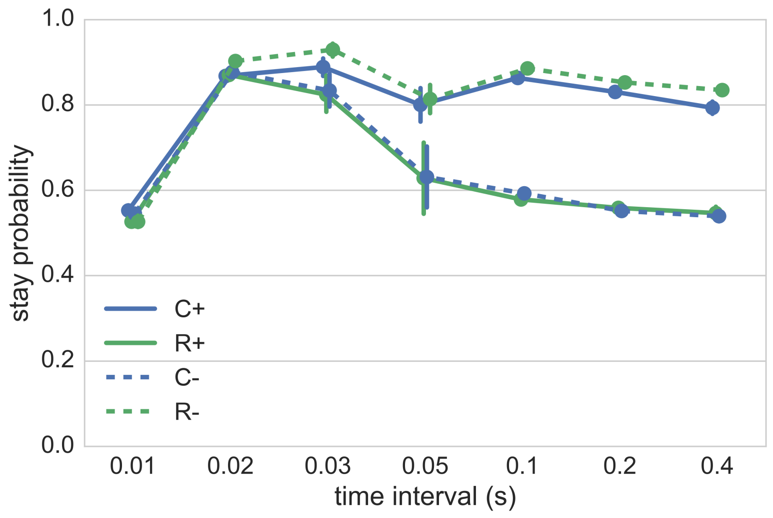


Figure 3.14: Stay probabilities of different time intervals.

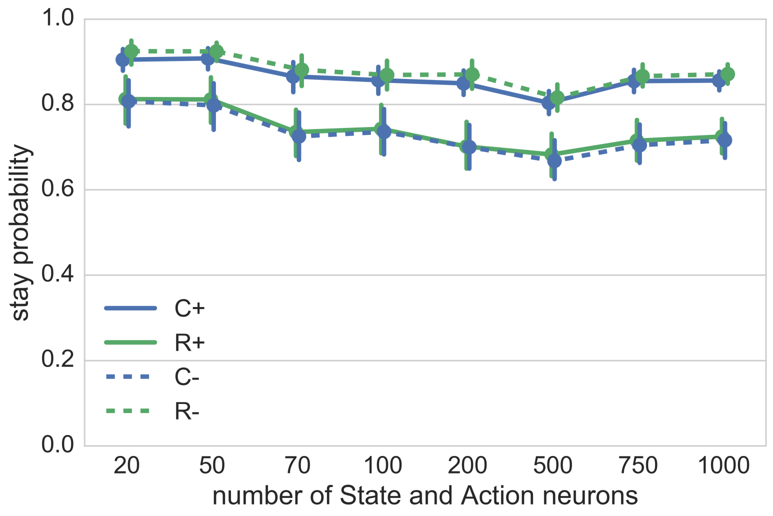


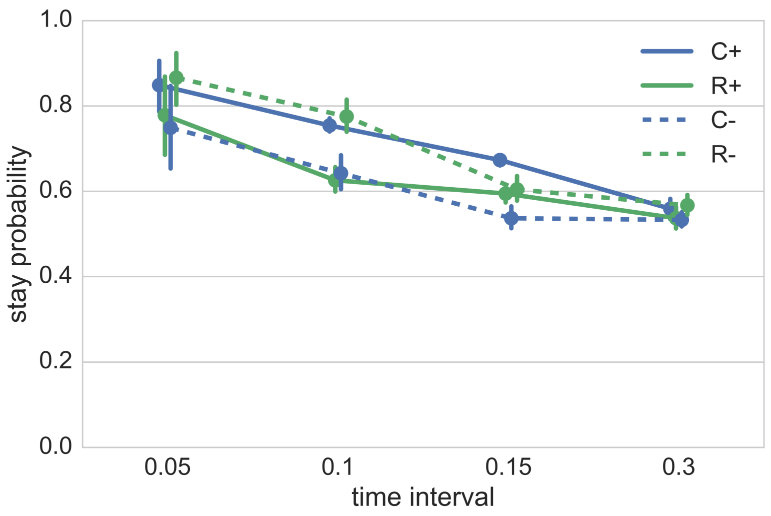
Figure 3.15: Stay probabilities of different numbers of neurons in the *State and Action* population with a time interval of 0.05s.

sufficiently short. Fig. 3.16a shows this significantly model-based behaviour present at 0.1s, followed by a rapid drop-off towards uniform stay probabilities at 0.3s. Interestingly, stay probabilities reminiscent of human data are apparent at 0.15s, shown more clearly in Fig. 3.16.

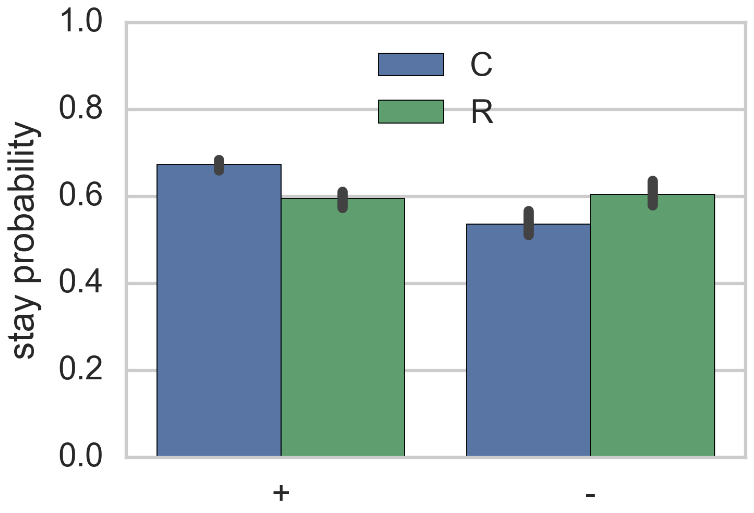
Learning the present rather than the future should have produced uniform stay probabilities for all time intervals. The fact that shorter time intervals displayed model-based behaviours is likely due to a continuous system learning a discrete task. At the beginning of each time interval, due to the synaptic time constant, the spikes sent from the *State and Action* population to the *Product* network would be still representing the state and action of the previous time interval. Thus, there would still be a small amount of information for the PES learning rule to use. With a longer time interval, that small amount of overlap from the previous time interval would become smaller relative to the total length of the time interval. This small overlap would also occur in the standard learning model, but since the remainder of the time interval would contain information relevant to the learning, the overlap likely does not interfere with the overall performance.

3.2.5 Learning model summary

The learning model implementation demonstrated the effects of adding learning to the dynamics of the model. In most cases, the general trends of the parameter explorations were the same as those in the static model, like varying the learning rate α and the time interval between state transitions. However, the advantage of learning the state transition probabilities was shown by varying the number of neurons. The learning model required considerably fewer neurons than the static model to produce comparable performance, and when the default number of neurons was used, the performance was considerably better. Using the default number of neurons produced relatively little variation between individual seeds, such that each seed produced characteristically model-based stay probability behaviour.



(a) Stay probabilities of different time intervals for the present learning model.



(b) Stay probability of the present learning model with a time interval of 0.15s.

Figure 3.16: "Present" learning model.

Chapter 4

Conclusions and future work

This chapter first presents a summary and discussion of the results in section 4.1 that demonstrates the robustness of the model-free behaviour within ranges of the parameters explored. Next, ideas for future work and model extensions that have not been previously discussed are presented in section 4.2, followed by a general conclusion in section 4.3.

4.1 Discussion

This section will summarize and discuss the general trends found in the parameter exploration presented in chapter 3. The main goal of the two model implementations was to produce model-based behaviour in an agent where the state transition probabilities and model-based Q-value computations were performed using biologically plausible spiking neurons. In general, both the static and learning models were very robust to the parameter variations, with characteristic model-based stay probability behaviour appearing in most cases. It is difficult to compare these results to previous work, since no previous models designed for the two-stage decision task were implemented in neurons [18, 2], and neural models of other sequential decision tasks did not perform similar parameter explorations [30]. The one exception to this is the model-free learning rate parameter α.

For both the static and learning models, the clearest model-based stay probability behaviour was found when α was in the range of 0.3 to 0.5. This range agrees well with the α values chosen for other model-based agent implementations on which this model is based [33, 18, 2]. These other models do not present a similar parameter exploration, but instead were fit to human data [33, 18] or were simply set to 0.5 [2]. The range found here also provides a good balance between the model-free learning of the terminal states' Q-values and the model-based computation of the initial state's Q-values.

The general rule of thumb of increasing the number of neurons in a neural population in order to increase the accuracy of the computation was found in both the static and learning models, in that too few neurons in any of the neural populations produced uniform stay probability behaviour. The static model functioned best when the *State and Action* population had at least 100 neurons and *Product* population had at least 100 neurons per dimension. However, increasing the neurons above a certain point produced no significant change in behaviour. The *State and Action* population did not require more than 500 neurons, and the *Product* did not require more than 100 neurons per dimension. The learning model required fewer neurons than the static model, with both *State and Action* and *Error* populations performing well in the range of 20-50 neurons, and *Product* needing around 50-200 neurons per dimension. This leads to the conclusion that the learning model only requires 10% of the neurons of the static model for similar performance, demonstrating that the ability to learn the state transition probabilities is a great improvement. It makes sense that the learning model is more efficient in terms of the number of neurons required, because learning allows for fine-tuning the weights on the neural connections that calculate the state transition probabilities. Learning over time continually refines the accuracy of this calculation, so that even a small number of neurons can produce results with a low error.

However, when the learning model used a number of neurons more comparable to the static model (500 for both *State and Action* and *Error* and 200 per dimension for *Product*), the stay probability behaviour became much more consistent between individual seeds, with all tested seeds showing characteristic model-based stay probability behaviour. For the static model, increasing the synaptic time constant had the same effect of removing individual differences between seeds and time intervals. That the effect of reducing individual differences appears in the two models when different parameters are varied raises some interesting possibilities. Firstly, increasing the number of neurons in the static model much higher than the maximum ranges specified above may have a similar effect of reducing individual differences. Although it did not appear that increasing the number of neurons above a certain point provided any great improvement, these data were averaged across individual seeds, thus obscuring any changes in individual differences that may have occurred as the number of neurons increased. Differences in the 95% bootstrap confidence intervals are also not sufficient for providing evidence for or against this possibility. Secondly, the individual differences between seeds when the learning model used the lower number of neurons may be reduced if a longer synaptic time constant is used. Possible future work could be done to optimize model-based behaviour in the learning model using fewer neurons with longer synaptic time constants.

These two possibilities are worth exploring; however, this exploration would ideally be guided by biological evidence, and not simply model performance and computational efficiency. For eample, the neurotransmitter most commonly used in ventromedial prefrontal cortex (vmPFC), an area associated with model-based reasoning [74, 42, 22] is *gamma*-Aminobutyric acid (GABA), which is associated with a shorter synaptic time constants [38], which suggests that it may be preferable to increase the number of neurons rather than the length of the synaptic time constant.

4.2 Future work

Many questions were raised by the results presented in this thesis, leading to a great deal of future work that can be done. In sections 3.2.2 and 4.1 we discussed some additional tests that could be performed in order to more fully understand the effect of increasing the number of neurons, and its interaction with learning the state transition probabilities. There are many additional avenues for future work, which can be divided into further tests to be performed on the existing models, and extensions to the model. The following sections will propose some possibilities for both of these classes, but it is not intended to be a comprehensive list.

4.2.1 Additional possible tests

While the model-free learning rate parameter *alpha* was varied in this thesis, the learning rate of the PES rule, which controls how quickly the state transition probabilities are learned, was not varied. The model-based agent of Akam et al. [2] had a similar parameter varied, which they called the transition learning rate, with the effect of higher transition learning rates reducing the stay probability in the rare-unrewarded (R$-$) case. This is an unexpected trend, and they do not offer any discussion of why it may be the case. Future work can determine whether varying the PES learning rate would produce a similar result, and possibly provide insight into why higher learning rates affect the R$-$ stay probability.

Other possible parameters to explore are the dimensionality and orthogonality of the vector representations of state and action. As described in section 2.3, the current model implementations use orthogonal vectors of the smallest possible dimensionality, although NEF methods do not impose such constraints. It will be important to investigate the effects of more complicated representations on the overall behaviour of the models.

Undoubtedly, a model-based RL agent should be able to perform any RL task, and not just the two-stage decision task used in this thesis. There are many simple variations to the task that could be used to test the neural model's performance. One example is to alter the reward structure so that instead of a random walk at each trial, the reward probabilities change after some greater number of trials. This approach can mimic studies of change in goal structure, such as reward devaluation (e.g., [24]), and may better indicate the strength of model-based reasoning. Another example of a task variation is to change the state transition probabilities partway through the simulation. This new task could be used to further evaluate the learning model's behaviour. If the current model is tested on modified sequential decision tasks, it will also be important to improve the method of evaluating the performance to a more robust definition of stay probability that takes multiple past trials into account [48].

4.2.2 Model extensions

The most urgent extension to the model presented in this thesis is to remove the parts of the *Environment* node that are more properly a part of the model-based agent (see section 2.3.1), and instead implement them using spiking neurons. Specifically, these components are the action selection and the model-free learning of Q-values in terminal states. Both of these components have well-characterized neural implementations using the NEF (for action selection using the basal ganglia, see e.g., [69]; for model-free RL, see e.g., [55]). Incorporating these components into the current neural model of state transition probabilities would create a completely neural model-based RL agent, which would provide direction for comparisons to neural and behavioural data on model-based systems in the human brain.

Most importantly, the eventual goal is an integration of a complete neural model-based system with a model-free system to produce a hybrid agent similar to that developed in [18]. This will allow for more plausible comparison to human behaviour on the two-stage task, and possibly give insights into human behaviour in more general sequential decision task problems. This is different than implementing the model-free learning of terminal states in neurons, because previous hybrid models combine a model-free learner with a model-based system (which includes model-free learning of select state-action pairs).

4.3 Closing remarks

This thesis demonstrated how the state transition probabilities and Q-value calculations of model-based reinforcement learning can be implemented using simulated populations of spiking neurons. The two variants of the neural model, which respectively represented and learned the state transition probabilities, generally showed characteristically model-based behaviour. Through exploring the effects of various parameters on the stay probability behaviour of the model, parameter values that produce non-model-based behaviours were identified. Although more work can be done to fully characterize the behaviour of the learning model, it was found to greatly improve the consistency of the behaviour between individual seeds, and to require far fewer neurons than the static model. In sum, the work done in this thesis provides a solid foundation for the construction of a completely neurally-implemented model of human behaviour in tasks that require model-based reasoning for optimal performance.

References

[1] Pieter Abbeel, Morgan Quigley, and Andrew Y. Ng. Using inaccurate models in reinforcement learning. *Proceedings of the 23rd International Conference on Machine Learning*, 2006.

[2] Thomas Akam, Rui Costa, and Peter Dayan. Simple plans or sophisticated habits? State, transition and learning interactions in the two-step task. *PLoS Computational Biology*, 11(12), 2015.

[3] Christopher G. Atkeson and Juan Carlos Santamaria. A Comparison of direct and model-based reinforcement learning. *IEEE International Conference on Robotics and Automation*, pages 3557–3564, 1997.

[4] Sean Aubin, Aaron R. Voelker, and Chris Eliasmith. Improving with practice: A neural model of mathematical development. *Topics in Cognitive Science*, 2016.

[5] Bernard W. Balleine and John P. O'Doherty. Human and rodent homologies in action control: Corticostriatal determinants of goal-directed and habitual action. *Neuropsychopharmacology*, 35:48–69, 2010.

[6] Dorit Baras and Ron Meir. Reinforcement learning, spike-time-dependent plasticity, and the BCM rule. *Neural Computation*, 19(8):2245–2279, 2007.

[7] Andrew G. Barto. *Adaptive critics and the basal ganglia.* MIT Press, Cambridge, MA, 1995.

[8] Andrew G. Barto and Sridhar Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems: Theory and Applications*, 13:343–379, 2003.

[9] Hannah M. Bayer and Paul W. Glimcher. Midbrain dopamine neurons encode a quantitative reward prediction error signal. *Neuron*, 47:129–141, 2005.

[10] Trevor Bekolay, James Bergstra, Eric Hunsberger, Travis DeWolf, Terrence C. Stewart, Daniel Rasmussen, Xuan Choo, Aaron R. Voelker, and Chris Eliasmith. Nengo: A python tool for building large-scale functional brain models. *Frontiers in Neuroinformatics*, 7(48), 2014.

[11] Trevor Bekolay, Carter Kolbeck, and Chris Eliasmith. Simultaneous unsupervised and supervised learning of cognitive functions in biologically plausible spiking neural networks. *35th Annual Conference of the Cognitive Science Society*, pages 169–174, 2013.

[12] R. E. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, 1957.

[13] Hugues Bersini and Vittorio Gorrini. Three connectionist implementations of dynamic programming for optimal control: A preliminary comparative analysis. In *Proceedings of International Workshop on Neural Networks for Identification, Control, Robotics and Signal/Image Processing*, pages 428–437, Aug 1996.

[14] Elie L. Bienenstock, Leon N. Cooper, and Paul W. Munro. Theory for the development of neuron selectivity: Orientation specificity and binocular interation in visual cortex. *The Journal of Neuroscience*, 2(1):32–48, 1982.

[15] Rafal Bogacz. Optimal decision-making theories: Linking neurobiology with behaviour. *Trends in Cognitive Sciences*, 11(3):118–125, 2007.

[16] Matthew Botvinick and Marc Toussaint. Planning as inference. *Trends in Cognitive Sciences*, 16(10):485–488, 2012.

[17] Ronen I. Brafman and Moshe Tennenholtz. R-max: A general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3:213–231, 2002.

[18] Nathaniel D. Daw, Samuel J. Gershman, Ben Seymour, Peter Dayan, and Raymond J. Dolan. Model-based influences on humans' choices and striatal prediction errors. *Neuron*, 69(6):1204–1215, 2011.

[19] Nathaniel D. Daw, Yael Niv, and Peter Dayan. Uncertainty-based competition between prefrontal and dorsolateral striatal systems for behavioral control. *Nature Neuroscience*, 8(12):1704–1711, 2005.

[20] Nathaniel D. Daw and John P. O'Doherty. Multiple systems for value learning. In P. Glimcher and E. Fehr, editors, *Neuroeconomics: Decision Making and the Brain*, chapter 21, pages 393–410. Academic Press, New York, 2 edition, 2013.

[21] Peter Dayan and L. F. Abbott. *Theoretical neuroscience: Computational and mathematical modeling of neural systems.* MIT Press, Cambridge, 2001.

[22] Sanne de Wit, Poppy Watson, Helga A Harsay, Michael X Cohen, Irene van de Vijver, and K Richard Ridderinkhof. Behavioral/Systems/Cognitive Corticostriatal Connectivity Underlies Individual Differences in the Balance between Habitual and goal-directed action control. *Journal of Neuroscience*, 32(35):12066–12075, 2012.

[23] Amir Dezfouli, Bernard W. Balleine, and Tim Behrens. Actions, action sequences and habits: Evidence that goal-directed and habitual action control are hierarchically organized. *PLoS Computational Biology*, 9(12):1–14, 2013.

[24] A Dickinson. Actions and habits: The development of behavioural autonomy. *Philosophical Transactions of the Royal Society of London. Series B, Biological Sciences*, 308(308):67–78, 1985.

[25] Bradley B. Doll, Dylan A. Simon, and Nathaniel D. Daw. The ubiquity of model-based reinforcement learning. *Current Opinion in Neurobiology*, 22(6):1075–1081, 2012.

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

[27] Daniel E. Feldman. The Spike-timing dependence of plasticity. *Neuron*, 75:556–571, 2012.

[28] Răzvan V. Florian. Reinforcement learning through modulation of spike-timing-dependent synaptic plasticity. *Neural Computation*, 19(6):1468–1502, 2007.

[29] Michael J. Frank and David Badre. Mechanisms of hierarchical reinforcement learning in corticostriatal circuits 1: Computational analysis. *Cerebral Cortex*, 22(3):509–526, 2012.

[30] Johannes Friedrich and Máté Lengyel. Goal-directed decision making with spiking neurons. *Journal of Neuroscience*, 36(5):1529–1546, 2016.

[31] Wulfram Gerstner and Werner M. Kistler. *Spiking neuron models: Single neurons, populations, plasticity.* Cambridge University Press, Cambridge, 2002.

[32] Wulfram Gerstner, Werner M. Kistler, Richard Naud, and Liam Paninski. *Neuronal dynamics: From single neurons to networks and models of cognition.* Cambridge University Press, Cambridge, 2014.

[33] Jan Gläscher, Nathaniel Daw, Peter Dayan, and John P. O'Doherty. States versus rewards: Dissociable neural prediction error signals underlying model-based and model-free reinforcement learning. *Neuron*, 66(4):585–595, 2010.

[34] Paul W. Glimcher. Understanding dopamine and reinforcement learning: the dopamine reward predicition error hypothesis. *PNAS*, 108:15647–15654, 2011.

[35] Jan Gosmann. Precise multiplications with the NEF. Technical report, University of Waterloo, Waterloo, Ontario, Canada, 2015.

[36] Charles G. Gross. Genealogy of the "grandmother cell". *Neuroscientist*, 8(5):512–518, 2002.

[37] Marek Grzes and Jesse Hoey. Efficient planning in R-max. *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems*, 2011.

[38] Anirudh Gupta, Yu Wang, and Henry Markram. Organizing principles for a diversity of GABAergic interneurons and synapses in the neocortex. *Science*, 287(5451):273–278, 2000.

[39] Donald O. Hebb. *The Organization of behavior.* Wiley & Sons, New York, 1949.

[40] Mehdi Keramati, Amir Dezfouli, and Payam Piray. Speed/accuracy trade-off between the habitual and the goal-directed processes. *PLoS Computational Biology*, 7(5), 2011.

[41] M. Louis Lapicque. Recherches quantitatives sur l'excitation électrique des nerfs traitée comme une polarisation. *Journal de Physiologie et de Pathologie Générale*, 9:620–635, 1907.

[42] Mimi Liljeholm, Elizabeth Tricomi, John P. O'Doherty, and Bernard W. Balleine. Neural correlates of instrumental contingency learning: Differential effects of action-reward conjunction and disjunction. *Journal of Neuroscience*, 31(7):2474–2480, 2011.

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

[44] Henry Markram, Joachim Lubke, Michael Frotscher, and Bert Sakmann. Regulation of synaptic efficacy by coincidence of postsynaptic APs and EPSPs. *Science*, 275(5297):213–215, 1997.

[45] Mariah Martin Shein, Terrence C. Stewart, and Chris Eliasmith. Parameter exploration of a neural model of state transition probabilities in model-based reinforcement learning. *15th Annual Meeting of the International Conference on Cognitive Modelling*, 2017.

[46] James E. Mazur. *Learning and behavior.* Pearson, 7 edition, 2013.

[47] K. J. Miller, M. M. Botvinick, and C. D. Brody. Dorsal hippocampus plays a causal role in model-based planning. *bioRxiv*, page 096594, 2016.

[48] Kevin J Miller, Carlos D Brody, and Matthew M Botvinick. Identifying model-based and model-free patterns in behavior on multi-step tasks. *bioRxiv*, page 096339, 2016.

[49] P. Read Montague, Peter Dayan, and Terrence J. Sejnowski. A framework for mesencephalic dopamine systems based on predictive Hebbian learning. *Journal of Neuroscience*, 16(5):1936–1947, 1996.

[50] Amy L. Odum. Delay discounting: I'm a k, you're a k. *Journal of the Experimental Analysis of Behavior*, 96(3):427–439, 2011.

[51] John O'Keefe and Lynn Nadel. *The hippocampus as a cognitive map.* Clarendon Press, Oxford, 1978.

[52] Weibke Potjans, Abigail Morrison, and Markus Diesmann. A Spiking neural network model of an actor-critic. *Neural Computation*, 21:301–339, 2009.

[53] Kandel. Eric R., James H. Schwartz, Thomas M. Jessell, Steven A. Siegelbaum, and A. J. Hudspeth. *Principles of neural science.* McGraw-Hill, New York, 5 edition, 2013.

[54] Antonio Rangel, Colin Camerer, and P. Read Montague. A Framework for studying the neurobiology of value-based decision making. *Nature Reviews Neuroscience*, 9:545–556, 2008.

[55] Daniel Rasmussen and Chris Eliasmith. A neural reinforcement learning model for tasks with unknown time delays. *Proceedings of the 35th Annual Conference of the Cognitive Science Society*, pages 3257–3262, 2013.

[56] Daniel Rasmussen and Chris Eliasmith. A neural model of hierarchical reinforcement learning. *Proceedings of the 36th Annual Conference of the Cognitive Science Society*, pages 1252–1257, 2014.

[57] Roger Ratcliff and Gail McKoon. The diffusion decision model: Theory and data for two-choice decision tasks. *Neural Computation*, 20(4):873–922, 2008.

[58] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[59] G A Rummery and M Niranjan. On-line Q-learning using connectionist systems. Technical report, Cambridge University, 1994.

[60] Wolfram Schultz, Paul Apicella, and Tomas Ljungbergb. Responses of monkey dopamine neurons to reward and conditioned stimuli during successive steps of learning a delayed response task. *Journal of Neuroscience*, 13(3):900–913, 1993.

[61] Wolfram Schultz, Peter Dayan, and P. Read Montague. A Neural substrate of prediction and rxeward. *Science*, 275:1593–1599, 1997.

[62] Andrew B. Schwartz, Ronald E. Kettner, and Apostolos P. Georgopoulos. Primate motor cortex and free arm movements to visual targets in three-dimensional space. I. Relations between single cell discharge and direction of movement. *Journal of Neuroscience*, 8(8):2913–2927, 1988.

[63] H. Sebastian Seung. Learning in spiking neural networks by reinforcement of stochastic synaptic transmission. *Neuron*, 40(6):1063–1073, 2003.

[64] B. F. Skinner. *The behavior of organisms.* Appleton-Century-Crofts, New York, 1938.

[65] B. F. Skinner. Two “synthetic social relations”. *Journal of the Experimental Analysis of Behavior*, 5(4):531–533, 1962.

[66] Alec Solway and Matthew M. Botvinick. Goal-directed decision making as probabilistic inference: A Computational framework and potential neural correlates. *Psychological Review*, 119(1):120–154, 2012.

[67] Terrence C. Stewart. A technical overview of the Neural Engineering Framework. Technical report, University of Waterloo, Waterloo, Ontario, Canada, 2012.

[68] Terrence C. Stewart. Predicting the future, 2017.

[69] Terrence C. Stewart, Trevor Bekolay, and Chris Eliasmith. Learning to select actions with spiking neurons in the basal ganglia. *Frontiers in Neuroscience*, 6(2):1–14, 2012.

[70] Terrence C. Stewart, Xuan Choo, and Chris Eliasmith. Dynamic behaviour of a spiking model of action selection in the basal ganglia. *Proceedings of the 32nd Annual Conference of the Cognitive Science Society*, pages 235–240, 2010.

[71] R. S. Sutton and A. G. Barto. *Reinforcement learning: An Introduction.* MIT Press, Cambridge, 1998.

[72] Richard S. Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *SIGART Bulletin*, 2(4):160–163, 1991.

[73] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: an introduction.* MIT Press, Cambridge, Mass., 1998.

[74] Saori C. Tanaka, Bernard W. Balleine, and John P. O'Doherty. Calculating consequences: Brain systems that encode the causal effects of actions. *Journal of Neuroscience*, 28(26):6750–6755, 2008.

[75] Edward L. Thorndike. *Animal intelligence: an experimental study of the associative processes in animals*, volume 2. Columbia University?, New York, 1898.

[76] Edward C. Tolman. Cognitive maps in rats and men. *Psychological Review*, 55(4):189–208, 1948.

[77] Elizabeth Tricomi, Bernard W. Balleine, and John P. O'Doherty. A specific role for posterior dorsolateral striatum in human habit learning. *European Journal of Neuroscience*, 29:2225–2232, 2009.

[78] Matthijs A. A. van der Meer, Adam Johnson, Neil C. Schmitzer-Torbert, and A. David Redish. Triple dissociation of information processing in dorsal striatum, ventral striatum, and hippocampus on a learned spatial decision task. *Neuron*, 67(1):25–32, 2010.

[79] Joel Veness, Kee Siong Ng, Marcus Hutter, William Uther, and David Silver. A Monte-Carlo AIXI approximation. *Journal of Artificial Intelligence Research*, 40:95–142, 2011.

[80] Aaron R. Voelker and Chris Eliasmith. Improving spiking dynamical networks: Accurate delays, higher-order synapses, and time cells. *Neural Computation*, in press.