Neural Networks 107 (2018) 48–60

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

2018 Special Issue

Neural circuits for learning context-dependent associations of stimuli

Henghui Zhu[a], Ioannis Ch. Paschalidis[b,*], Michael E. Hasselmo[c]

[a] Division of Systems Engineering, Boston University, 15 Saint Mary's Street, Brookline, MA 02446, United States
[b] Department of Electrical and Computer Engineering, Division of Systems Engineering, and Department of Biomedical Engineering, Boston University, 8 Saint Mary's Street, Boston, MA 02215, United States
[c] Center for Systems Neuroscience, Kilachand Center for Integrated Life Sciences and Engineering, Boston University, 610 Commonwealth Ave., Boston, MA 02215, United States

ARTICLE INFO

Article history:
Available online 13 August 2018

Keywords:
Neural circuit model
Neural networks
Reinforcement learning

ABSTRACT

The use of reinforcement learning combined with neural networks provides a powerful framework for solving certain tasks in engineering and cognitive science. Previous research shows that neural networks have the power to automatically extract features and learn hierarchical decision rules. In this work, we investigate reinforcement learning methods for performing a context-dependent association task using two kinds of neural network models (using continuous firing rate neurons), as well as a neural circuit gating model. The task allows examination of the ability of different models to extract hierarchical decision rules and generalize beyond the examples presented to the models in the training phase. We find that the simple neural circuit gating model, trained using response-based regulation of Hebbian associations, performs almost at the same level as a reinforcement learning algorithm combined with neural networks trained with more sophisticated back-propagation of error methods. A potential explanation is that hierarchical reasoning is the key to performance and the specific learning method is less important.

1. Introduction

The understanding of mechanisms for flexible cognition in a range of different tasks will benefit from the understanding of the mechanisms by which neural circuits can perform symbolic processing. One aspect of symbolic processing is the application of rules to a range of different combinations of task stimuli in different contexts. In this work, we examine how different neural network models can learn the application of specific rules involving the effect of repeated exposure to different spatial contexts on the associations between different stimuli and responses.

We compare traditional reinforcement learning algorithms using neural network function approximators to a framework using gating of neural activity. These models contain simplified representations of neuronal input–output functions that use standard but relatively abstract models of neuronal activity. They can be seen as general models of the interaction of populations of neurons in neocortical structures. Extensive evidence indicates a role of neural circuit activity in the prefrontal cortex (PFC) in the learning of task rules (Miller & Cohen, 2001; Wallis, Anderson, & Miller, 2001). Similarly, activity in the prefrontal cortex has been implicated in learning and implementation of hierarchical rules (Badre & Frank, 2012; Badre, Kayser, & D'Esposito, 2010). Thus, the neural representations of context-dependent structure used in the models presented here are relevant to the mechanisms of neural circuits mediating rule-based function in the prefrontal cortex.

Considerable recent research has focused on the strength of deep reinforcement learning. This recent work returns to the use of neural networks as function approximators in the context of reinforcement learning first considered by Tesauro (1994), but introduces deep architectures instead of the earlier multilayer perceptron with one hidden layer used in Tesauro (1994). Arguably, this represents a break from a body of work considering function approximators as linear combinations of hard-to-engineer nonlinear feature functions (Bertsekas & Tsitsiklis, 1996; Estanjini, Li, & Paschalidis, 2012; Konda & Tsitsiklis, 2003; Pennesi & Paschalidis, 2010; Tsitsiklis & Van Roy, 1997; Wang, Ding, Lahijanian, Paschalidis, & Belta, 2015; Wang & Paschalidis, 2017a).

We study whether traditional reinforcement learning methods, such as Q-learning (Tsitsiklis, 1994; Watkins & Dayan, 1992) and Q-learning using linear function approximation, can learn to generalize beyond what they have seen during training for context-dependent association tasks. We prove that this is not possible, thus revealing significant limitations of these methods.

* Corresponding author.
E-mail addresses: henghuiz@bu.edu (H. Zhu), yannisp@bu.edu (I.Ch. Paschalidis), hasselmo@bu.edu (M.E. Hasselmo).
URLs: http://sites.bu.edu/paschalidis/ (I.Ch. Paschalidis), http://www.bu.edu/hasselmo (M.E. Hasselmo).

https://doi.org/10.1016/j.neunet.2018.07.018
0893-6080/© 2018 Elsevier Ltd. All rights reserved.

It is therefore worthwhile to consider the mechanisms by which deep reinforcement learning could be used to perform context-dependent rules for associations between stimuli and responses. Recent work has shown that deep learning techniques coupled with reinforcement learning algorithms can learn decision making strategies over a high-dimensional state space such as images in a video game (Hausknecht & Stone, 2015; Mnih et al., 2016, 2015). In Mnih et al. (2015), the deep Q-network is used to train a reinforcement learning agent playing Atari games using game images. Mnih et al. (2016) further develops this idea into actor–critic learning and also proposes a parallel computing scheme for reinforcement learning. Besides the deep Q-network, deep neural networks have been used to directly approximate the policy, which can be optimized using either a policy gradient method (Peters & Schaal, 2008), a Newton-like method (Wang & Paschalidis, 2017b), or trust region policy optimization (Liu, Wu, & Sun, 2018; Schulman, Levine, Abbeel, Jordan, & Moritz, 2015). Finally, neural networks have been shown to exhibit good performance in some general control tasks, see, e.g., Levine, Finn, Darrell, and Abbeel (2016) and Watter, Springenberg, Boedecker, and Riedmiller (2015).

The neural networks in the models outlined above enable the learning agent to make decisions hierarchically. All these models use the traditional neural network elements with continuous values representing the mean firing rate across a population of neurons that increases when the input crosses a threshold (Rumelhart, McClelland, 1986). These papers also use multi-layer networks to code the relevant features of the sensory input images, training the synaptic connections between the layers with the traditional back-propagation of error algorithm (Rumelhart, Hinton, & Williams, 1986). The multi-layer networks are then combined with the Q-learning algorithm to learn the value of different actions for a given state (Dayan & Watkins, 1992; Sutton & Barto, 1998), resulting in sophisticated game-playing of the simulated agent.

In this paper, we test a similar framework to determine how well it can perform in the specific learning task we consider, which involves detecting the hierarchical structure of the correct response to stimuli in different contexts. In addition, we test how this framework can generalize to produce correct responses when encountering novel combinations of context and stimuli. Different than the existing work, we also examine the use of recurrent neural network architectures (Xu et al., 2015) in the reinforcement learning algorithms. As we will see, these networks perform significantly better than the feedforward networks in our learning task.

The neural network-based reinforcement learning methods discussed above are compared to another neural circuit approach using the generation and selection of gating elements that regulate the spread of activity between different populations of neurons (Hasselmo & Stern, 2018). This framework for modeling effects of gating has precursors in a number of previous models. In particular, the use of neuronal interactions to gate the spread of activity within a network was used in previous models of the prefrontal cortex that focused on modeling goal-directed action selection using interacting populations of neurons (Hasselmo, 2005; Koene & Hasselmo, 2005). This resembles features of a general theory of prefrontal function focused on routing of activity (Miller & Cohen, 2001). The use of cortical gating units is related but different in implementation from the use of basal ganglia to gate activity into working memory (O'Reilly & Frank, 2006). In addition, the selective learning of internal memory actions that manipulate memory buffers was used in previous models in which reinforcement learning algorithms were employed to regulate the use of working memory or episodic memory for solving simple behavioral tasks in the form of non-Markov decision processes (Hasselmo, 2005; Hasselmo & Eichenbaum, 2005; Zilli & Hasselmo, 2008c). Different types of memory buffers can be used in different ways to solve different behavioral problems (Zilli & Hasselmo, 2008a, b).

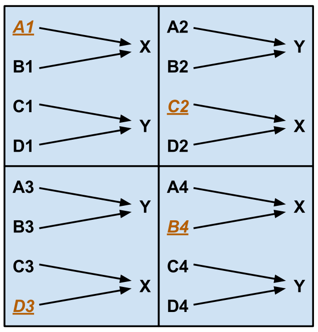


Fig. 1. Mapping between individual stimuli (A, B, C, D) and the spatial context (quadrants 1, 2, 3, 4) onto correct actions X or Y, providing 16 state–action pairs. The underlined (red) state–action pairs are not seen during training but presented during testing.

In this paper, we compare previous neural network models with the framework of using simple models of gating in neural circuits to learn the context-dependent stimulus–output map. Since the decision rule for the task is hierarchical, we use a sequential input for this model based on sequential attention to different input components. The learning rule for the gated weights in this neural circuit model is Hebbian, with plasticity of the synapses regulated by correct responses. We compare the performance of neural network models with the performance of the neural circuit gating model in both learning the hierarchical decision rules and in making successful generalizations in order to generate correct responses to novel combinations of context and stimuli.

The remainder of this paper is organized as follows. Section 2 presents the learning task, the negative results on traditional reinforcement learning algorithms, the neural network-based reinforcement learning algorithms, and the neural circuit gating model. Section 3 presents results from training these models and applying the trained models to test examples. In Section 4, we examine the way the various models make decisions and seek to qualitatively understand the performance results. Conclusions are in Section 5.

Notational conventions. Bold lower case letters are used to denote vectors and bold upper case letters are used for matrices. Vectors are column vectors, unless explicitly stated otherwise. Prime denotes transpose. For the column vector $\mathbf{x} \in \mathbb{R}^n$ we write $\mathbf{x} = (x_1, \ldots, x_n)$ for economy of space. Vectors or matrices with all zeros are written as $\mathbf{0}$, the identity matrix as $\mathbf{I}$, and $\mathbf{e}$ is the vector with all entries set to 1. We will use script letters to denote sets.

2. Methods

In this section, we present a number of different neural network-based models for learning context-dependent behavior. We start by presenting a specific learning task where one has to associate responses with inputs consisting of a stimulus and a context. We describe how to train the proposed models and test how effective they are in generalizing based on context, that is, whether they can use context information to generate correct responses to previously unseen inputs.

2.1. The learning task

The basic learning task was considered in our previous work in Raudies, Zilli, and Hasselmo (2014). The task aims to evaluate the ability of a learning agent to associate specific stimuli with appropriate responses in particular spatial contexts. Fig. 1 shows the mapping between input and responses. The input consists of stimuli, denoted by letters A, B, C, and D, and a spatial context

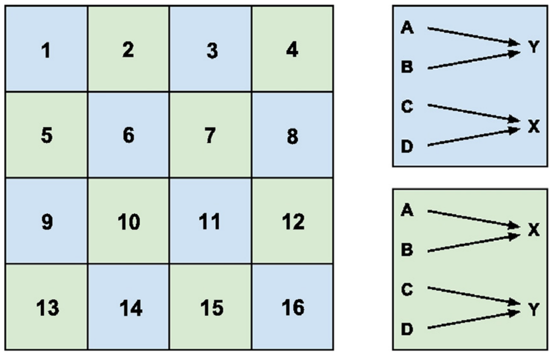


Fig. 2. A larger context association task with 16 contexts. Different contexts map to one out of two different stimulus–response association rules. Specifically, contexts 1, 3, 6, 8, 9, 11, 14, and 16 (blue) correspond to the top right rule and the remaining contexts (green) to the lower right rule. Mapping between individual stimuli and the spatial context onto correct actions X or Y yields 64 state–action pairs.

corresponding to four quadrants and denoted by numbers 1, 2, 3, and 4. We will use the term *state* to refer to the stimulus–context pairs. The two legal responses (or actions) are X and Y. To test the ability of the algorithms to handle more complex tasks, we also consider a similar task with 16 contexts shown in Fig. 2. In this task, we have the same two rules for mapping stimuli to responses but a larger number of contexts than the task in Fig. 1.

Notice that for both tasks the mapping from states to actions exhibits symmetry, in the sense that the association rule learned in one spatial context is shared with another context. To test the generalization ability of the various models, we hide some context-stimulus pairs during training. For the basic task, the hidden states are underlined and in red in Fig. 1. For the larger task, we hide pairs involving a specific stimulus in every context but in a way so that for each hidden context-stimulus pair there exists another pair with the same stimulus and response in another context which is not hidden. This is done so that sufficient information to infer responses for every context-stimulus pair is presented during training.

We are interested in learning these association rules from past examples, that is, in a *reinforcement learning* framework. We are particularly interested in the ability of methods to learn the symmetry in the state–action map and produce correct actions for previously unseen states, effectively generalizing from past examples.

In Raudies et al. (2014) we proposed a classification approach using a deep belief network model to learn the correct actions (labels) to given states (data points). One can also develop alternative supervised learning approaches including different classification methods and regression. Our objective in this work, however, is not necessarily to learn the best possible decision function but rather to investigate the power of neural circuit models and evaluate the effectiveness of different learning methods. We will work in a reinforcement learning framework.

We will use two different representations (codings) for the state, shown in Fig. 3. The first method is a vector presentation, introduced in Raudies et al. (2014). It uses an $(\kappa + l)$-dimensional binary vector to code the state, where l is the number of contexts and κ the number of stimuli. We will use $n = \kappa + l$ to denote the dimension of the entire vector encoding the state. For the task in Fig. 1, we have $n = 8$, while for the task in Fig. 2, $n = 20$. The first κ bits correspond to the stimuli. Specifically, for both tasks we consider $\kappa = 4$ and the first four bits correspond to the stimuli A, B, C, and D, respectively, as shown in Fig. 3(a). The last l bits correspond to the contexts $1, \ldots, l$. In the basic task of Fig. 1 for example, $l = 4$ and the last four bits of the vector correspond to contexts 1, 2, 3, and 4, respectively. Fig. 3(b) provides an example of this type of encoding for the task of Fig. 1.

Our second representation of the state, uses a sequence of two n-dimensional binary vectors. The first vector in the sequence represents the stimulus; for our tasks, bits 1 through κ $(= 4)$ being 1 correspond to stimuli A through D, respectively, while the last l bits are set to zero. The second vector in the sequence has its first κ bits set to zero and the last l bits, κ through $\kappa + l$, being set to 1 to represent contexts 1 through l, respectively. Fig. 3(c) shows an example of representing state B3 in the task of Fig. 1 using a two vector sequence $\mathbf{s}_1, \mathbf{s}_2$. This type of representation is similar to attention mechanisms used in *Recurrent Neural Network (RNN)* models (Xu et al., 2015), in which the agent is assumed to first pay attention to a stimulus and then to the context.

To introduce some of our various learning methods, it would be convenient to assume a learning agent who is continuously being presented with states (the stimuli–context pairs) and produces an action (response) that can be either X or Y. Given a state and the selected action, the agent *transitions* to a next state which simply corresponds to the next state at which the agent is asked to produce a response. We will use a discrete-time *Markov Decision Process (MDP)* (Bertsekas, 1995; Bertsekas & Tsitsiklis, 1996) to represent this learning process.

(a) The encoding of different stimuli and contexts in the task of Figure 1.

Stimuli and contexts		Encoding
Stimulus	A	(1, 0, 0, 0, 0, 0, 0 ,0)
	B	(0, 1, 0, 0, 0, 0, 0 ,0)
	C	(0, 0, 1, 0, 0, 0, 0 ,0)
	D	(0, 0, 0, 1, 0, 0, 0 ,0)
Context	1	(0, 0, 0, 0, 1, 0, 0 ,0)
	2	(0, 0, 0, 0, 0, 1, 0 ,0)
	3	(0, 0, 0, 0, 0, 0, 1 ,0)
	4	(0, 0, 0, 0, 0, 0, 0 ,1)

(b) An example of vector encoding for stimulus-context pairs.

Stimulus-context pair: B3,
Stimulus B $\longrightarrow (0, 1, 0, 0, 0, 0, 0, 0)$,
Context 3 $\longrightarrow (0, 0, 0, 0, 0, 0, 1, 0)$,
Encoded vector:
$\mathbf{s} = (0, 1, 0, 0, 0, 0, 1, 0)$.

(c) An example of sequential encoding for stimulus-context pairs.

Stimulus-context pair: B3,
Stimulus B $\longrightarrow (0, 1, 0, 0, 0, 0, 0, 0)$,
Context 3 $\longrightarrow (0, 0, 0, 0, 0, 0, 1, 0)$,
Encoded time series:
$\mathbf{s}_1 = (0, 1, 0, 0, 0, 0, 0, 0)$,
$\mathbf{s}_2 = (0, 0, 0, 0, 0, 0, 1, 0)$.

Fig. 3. The vector encoding and the sequential encoding for the stimulus–context pairs.

The MDP has a finite state space $\mathcal{S}$, consisting of the states and an action space $\mathcal{U}$, consisting of the actions X and Y. Let $\mathbf{s}_k \in \mathcal{S}$ and $u_k \in \mathcal{U}$ be the state and the action taken at time k, respectively, and let $\mathbf{s}_0$ be an initial state of the MDP. Let $p(\mathbf{s}_{k+1}|\mathbf{s}_k, u)$ denote the probability that the next state is $\mathbf{s}_{k+1}$, given the current state is $\mathbf{s}_k$ and action u is taken. We assume, without loss of generality in our setting, that these transition probabilities are uniform in all non-hidden states for all states and actions.

When the agent selects a correct action, it receives a reward; otherwise, it gets penalized. Let $g(\mathbf{s}_k, u_k)$ be the one-step reward at time k when action u_k is selected at state $\mathbf{s}_k$. We define the one-step reward to be 1 if u_k is the correct response at state $\mathbf{s}_k$ and -4 otherwise. We seek a *policy*, which is a mapping from states to actions, to maximize the long-term discounted reward

$$\bar{R} = \sum_{k=0}^{\infty} \gamma^k g(\mathbf{s}_k, u_k), \tag{1}$$

where we will use a discount rate of $\gamma = 0.9$.

We next introduce some basic concepts from reinforcement learning; see Bertsekas (1995) for a more comprehensive treatment. The *value function* $V(\mathbf{s})$ for a state $\mathbf{s}$ is the maximum long-term reward obtained starting from $\mathbf{s}$. The value function satisfies the following intuitive recursive equation:

$$V(\mathbf{s}) = \max_u \left(g(\mathbf{s}, u) + \gamma \sum_{\mathbf{q}\in\mathcal{S}} p(\mathbf{q}|\mathbf{s}, u)V(\mathbf{q}) \right), \tag{2}$$

which is known as Bellman's equation. Solving the MDP amounts to finding a value function, say $V^*(\cdot)$, which is a solution to (2). Given such a function $V^*(\cdot)$, one can easily find the optimal action u^* at each state $\mathbf{s}$ as the maximizing u in (2) and that u^* will necessarily be the correct action.

Define now the so-called Q-value function which is a function $Q(\mathbf{s}, u)$ of state–action pairs and is equal to the maximum long-term reward obtained starting from $\mathbf{s}$ and selecting as first action u. The Q-value function also satisfies a recursive equation, namely

$$Q(\mathbf{s}, u) = g(\mathbf{s}, u) + \gamma \sum_{\mathbf{q}\in\mathcal{S}} p(\mathbf{q}|\mathbf{s}, u) \max_v Q(\mathbf{q}, v). \tag{3}$$

From a solution, say $Q^*(\cdot, \cdot)$, of (3) one can also obtain the optimal action at each state $\mathbf{s}$ as $u^*(\mathbf{s}) = \arg\max_{u\in\mathcal{U}} Q(\mathbf{s}, u)$.

2.2. Traditional reinforcement learning and linear function approximation

The traditional reinforcement learning methods we consider in this work are the reinforcement learning algorithms that use a look-up table to store the value or Q-value function during training. Some typical examples are value iteration, TD-learning and Q-learning (Bertsekas, 1995). We will show that these methods are not suitable for the context association tasks we consider. For simplicity, we will establish this result for Q-learning; it can be easily extended to the other methods.

Q-learning (Watkins & Dayan, 1992) is a method for solving (3) and can be used even in the absence of an explicit model for the MDP. Essentially, Q-learning solves (3) using a value iteration (successive approximation) method but with the expectation with respect to the next state being approximated by sampling. The original Q-learning algorithm iterates over the Q-values at all states and actions, which is computationally intractable for large MDPs. Approximate versions of the method have been introduced (see Bertsekas, 1995; Bertsekas & Tsitsiklis, 1996), where the Q-value function is approximated using a set of features of the state–action pairs. This makes it possible to derive good policies for MDPs with a large state–action space. Traditionally, linear function approximation is used, since it is simple and leads to convergence results (Bertsekas, 1995; Bertsekas & Tsitsiklis, 1996; Xu, Zuo, & Huang, 2014). However, the effectiveness of the policy depends greatly on the selection of appropriate features and the latter is more of an art and very much problem specific.

The original Q-learning algorithm updates the Q-factors as follows:

$$Q_{k+1}(\mathbf{s}_k, u_k) = Q_k(\mathbf{s}_k, u_k) - \lambda_k \mathrm{TD}_k$$
$$\mathrm{TD}_k = Q_k(\mathbf{s}_k, u_k) - \gamma \max_{u\in\mathcal{U}} Q_k(\mathbf{s}_k, u) - g(\mathbf{s}_k, u_k), \tag{4}$$

where λ_k is a Square Summable but Not Summable (SSNS) step-size sequence, which means $\lambda_k > 0$, $\sum_{k=0}^{\infty}\lambda_k = \infty$, and $\sum_{k=0}^{\infty}\lambda_k^2 < \infty$. The actions u_k can be chosen according to an ε-random policy, i.e., with probability ε, choose a random action and with probability $1-\varepsilon$, choose an action maximizing $Q_k(\mathbf{s}_k, \cdot)$. The algorithm maintains all Q-factor estimates for all state–action pairs during the training processes. Hence, it requires a large amount of memory and a long training time if the number of the state–action pairs is excessive (Bertsekas, 1995).

Although convergence proofs for the original Q-learning algorithm under some conditions can be found in Bertsekas and Tsitsiklis (1996) and Tsitsiklis (1994), the Q-factors obtained by this algorithm do not work perfectly for the learning tasks we are considering. A negative result is shown in the next theorem.

Theorem 2.1. *The Q-factors obtained by the algorithm in (4) do not converge to the optimal Q-factor $Q(\mathbf{s}, u)$ for the MDP of the learning task in Section 2.1 if the initial $Q_0(\mathbf{s}, u) \neq Q(\mathbf{s}, u)$ for any $\mathbf{s}$ included in the hidden states.*

Proof. According to the Q-learning updating rule, the Q-factor for state–action pair $(\mathbf{s}, u)$ can be updated if $\mathbf{s}_k = \mathbf{s}$ and $u_k = u$ for some k. However, hidden states are not shown during training. Hence, Q-factors for state–action pairs $(\mathbf{s}, u)$, where $\mathbf{s}$ is a hidden state are never updated from their initial states. Thus, if their initial values are not optimal, the Q-factors obtained by the algorithm in (4) do not converge to their optimal values for all state–action pairs. ■

The traditional reinforcement learning algorithms fail in the context association tasks since they use a look-up table presentation and lack the ability to make a generalization. Therefore, using some function approximation for Q-factors is necessary. Q-learning using a linear function approximation is one of the simplest choices and leads to convergence results (Bertsekas, 1995; Bertsekas & Tsitsiklis, 1996). The algorithm approximates the Q-factors as

$$\tilde{Q}(\mathbf{s}, u) = \boldsymbol{\phi}(\mathbf{s}, u)'\boldsymbol{\theta} \tag{5}$$

where $\boldsymbol{\phi}(\mathbf{s}, u)$ is a feature vector of the state–action pair of the MDP and $\boldsymbol{\theta}$ is a parameter vector obtained iteratively. The performance of this method depends heavily on the selection of the features. As we show later, even when using a set of features that contain sufficient information regarding the future rewards associated with a state–action pair, the linear architecture fails to find an optimal policy.

For ease of exposition, we consider the task in Fig. 1; however, the discussion below and Theorem 2.2 can be readily extended to the larger task. We use the vector encoding of the state, shown in Fig. 3, to construct features. Since this is only a feature for the state, it needs augmentation to account for actions as well. To that end, we use the following Q-factor estimate:

$$\begin{bmatrix} \tilde{Q}(s, \mathrm{X}) \\ \tilde{Q}(s, \mathrm{Y}) \end{bmatrix} = \boldsymbol{\Theta}\mathbf{x}(s), \tag{6}$$

where $\boldsymbol{\Theta} \in \mathbb{R}^{2\times 8}$ is a parameter matrix and $\mathbf{x}(s) \in \mathbb{R}^8$ is the vector encoding of the state s. Notice that this approximation is a special case of (5).

Theorem 2.2. *The Q-factors obtained by the Q-learning algorithm under the linear function approximation in (6) are not optimal for the MDP of the learning task in Fig. 1. Moreover, the policy obtained by this algorithm is not the optimal.*

Proof. We will show that the policy obtained from the Q-factors derived by this algorithm is not optimal. Suppose we find a Q-factor function in the form (6) that selects the optimal actions for all states. From Fig. 1, it follows:

$$\tilde{Q}(\mathrm{A1}, \mathrm{X}) > \tilde{Q}(\mathrm{A1}, \mathrm{Y}), \tag{7}$$

$$\tilde{Q}(\mathrm{A2}, \mathrm{X}) < \tilde{Q}(\mathrm{A2}, \mathrm{Y}), \tag{8}$$

$$\tilde{Q}(\mathrm{C1}, \mathrm{X}) < \tilde{Q}(\mathrm{C1}, \mathrm{Y}), \tag{9}$$

$$\tilde{Q}(\mathrm{C2}, \mathrm{X}) > \tilde{Q}(\mathrm{C2}.\mathrm{Y}), \tag{10}$$

From Fig. 3, we obtain

$$\mathbf{x}(\mathrm{A2}) - \mathbf{x}(\mathrm{A1}) + \mathbf{x}(\mathrm{C1}) = \mathbf{x}(\mathrm{C2}). \tag{11}$$

Using (11), the linearity of the Q-factor estimates (cf. (6)) and (7)–(9), it follows $\tilde{Q}(\mathrm{C2}, \mathrm{X}) < \tilde{Q}(\mathrm{C2}, \mathrm{Y})$. This contradicts (10). ■

So, even if one uses a meaningful feature mapping that contains all relevant information regarding a state, Q-learning may not always produce the correct answer. This is because the linear function approximation does not have the ability to make decisions hierarchically. This result can be easily generalized to affine functions as well. Hence, Q-learning relies heavily on the feature selection, with the latter being more of an art and very much problem specific.

2.3. Q-learning using neural networks

Neural networks offer an alternative to feature engineering and learning approximations of the value function, or the Q-value function, or even the policy directly. Deep learning is making major advances in solving problems that have resisted the best attempts of the artificial intelligence community for many years (LeCun, Bengio, & Hinton, 2015). The advance of deep learning makes it possible to use deep neural networks to approximately solve the MDP efficiently. One such method is the deep Q-network (DQN), proposed in Mnih et al. (2015). The main idea is to use a deep neural network to approximate the Q-value function and obtain the neural network weights using Q-learning. Following this line of work, Mnih et al. (2016) proposed an asynchronous method for Q-learning and actor–critic learning, on which the Q-learning method we use for our learning task is based. Though the neural networks used for our tasks are not deep, they provide some insight on the ability of neural networks to generalize as is needed in the tasks we consider.

The algorithm for deep Q-learning is shown in Algorithm 1 in Mnih et al. (2016). It uses a deep neural network to approximate the Q-value function of the MDP. The neural network takes as input the state and outputs the estimated Q-value function of the state and each possible action. Optimizing the neural network parameters is not an easy task, since reinforcement learning using a nonlinear approximator is known to be unstable (Tsitsiklis & Van Roy, 1997). Mnih et al. (2015) uses a biologically inspired mechanism, termed experience replay, and maintains two neural networks with the parameters of one of them being updated in a slower time-scale to mitigate the instability. Further work (Mnih et al., 2016) simplifies the reinforcement learning algorithm, replacing the replay mechanism with multiple agents. For the tasks in the current paper, the instability of the algorithm can be overcome by the periodical updating rule of Mnih et al. (2015). We only use the algorithm in Mnih et al. (2016) with a single agent.

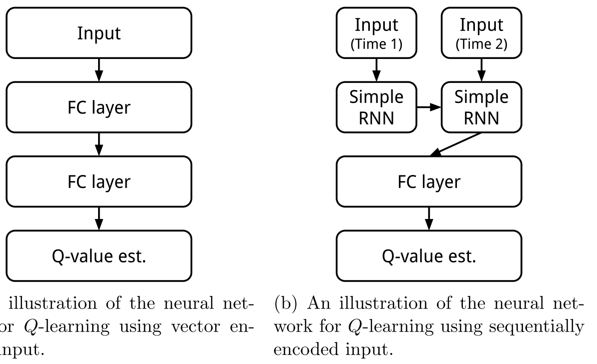

Input
FC layer
FC layer
Q-value est.
(a) An illustration of the neural network for Q-learning using vector encoded input.
Input (Time 1)
Input (Time 2)
Simple RNN
Simple RNN
FC layer
Q-value est.
(b) An illustration of the neural network for Q-learning using sequentially encoded input.

Fig. 4. An illustration of the neural networks used in Q-learning. The FC layer in the diagram represents a Fully-Connected feedforward layer. RNN indicates a Recurrent Neural Network. Q-value est. in the diagram denotes the estimate of the Q-value at the input state for all possible actions.

In particular, in this paper we compare two kinds of neural networks (Goodfellow, Bengio, & Courville, 2016), both of which use units with continuous firing rates, to approximate the Q-value function. The first one is a feedforward neural network, which consists only of fully-connected layers and inputs in the vector encoding form (cf. Fig. 3(b)). The second neural network is a so-called *Recurrent Neural Network (RNN)*, which accepts inputs in the sequential encoding (cf. Fig. 3(c)) and produces an estimate of the Q-value function. The structure of these neural networks is shown in Fig. 4.

In the feedforward network (Fig. 4(a)), the input is the vector encoded state $\mathbf{s}$; an $n = \kappa + l$-dimensional vector as we indicated earlier. The output is the Q-value function at each possible action in that state, that is, the 2-dimensional vector $(Q(\mathbf{s}, X), Q(\mathbf{s}, Y))$. The activation functions of the neurons are all Rectified Linear Units (ReLU), except the output layer (Nair & Hinton, 2010). In particular, $\mathrm{ReLU}(\mathbf{x}) = \max(\mathbf{x}, 0)$, for some vector $\mathbf{x}$, where the maximum is taken element-wise. There is no activation function in the output layer, since the Q-value function should not be restricted. Letting $\mathbf{s}$ be the input state, and $\mathbf{h}, \mathbf{q}$ the outputs of the two FC layers in Fig. 4(a), we have

$$\begin{aligned} \mathbf{h} &= \mathrm{ReLU}(\mathbf{W}_1\mathbf{s} + \mathbf{b}_1), \\ \mathbf{q} &= \mathbf{W}_2\mathbf{h} + \mathbf{b}_2, \end{aligned} \tag{12}$$

where $\mathbf{W}_1 \in \mathbb{R}^{m\times n}$ is the weight matrix of the first FC layer, $\mathbf{W}_2 \in \mathbb{R}^{2\times m}$ is the weight matrix of the second FC layer, and $\mathbf{b}_1 \in \mathbb{R}^m$, $\mathbf{b}_2 \in \mathbb{R}^2$, are additional parameters of the first and second layers we need to learn. Here, m is the number of hidden neurons in the first FC layer and $\mathbf{q}$ is the estimate of $(Q(\mathbf{s}, X), Q(\mathbf{s}, Y))$.

Next, we turn to the RNN architecture in Fig. 4(b). Recall that the input, in this case, uses the sequential encoding and is a sequence of two vectors $\mathbf{s}_1, \mathbf{s}_2 \in \mathbb{R}^n$ (cf. Fig. 3(c)). The output, similar as above, is the Q-value function at each possible action in that state, i.e., $(Q(\mathbf{s}, X), Q(\mathbf{s}, Y))$. For simplicity, we use a neural network with a simple RNN layer and a fully-connected layer to approximate the Q-value function. We let $\mathbf{s}_1, \mathbf{s}_2$ be the sequential encoding of the input state $\mathbf{s}$. We denote by $\mathbf{h}_1, \mathbf{h}_2$ the outputs of the first and the second RNN layers, and by $\mathbf{q}$ the output of the FC layer. We have

$$\begin{aligned} \mathbf{h}^1 &= \mathrm{ReLU}(\mathbf{W}_{11}\mathbf{s}_1 + \mathbf{W}_{12}\mathbf{h}^0), \\ \mathbf{h}^2 &= \mathrm{ReLU}(\mathbf{W}_{11}\mathbf{s}_2 + \mathbf{W}_{12}\mathbf{h}^1), \\ \mathbf{y} &= \mathbf{W}_2\mathbf{h}^2 + \mathbf{b}_2, \end{aligned} \tag{13}$$

where the initial state of the RNN is $\mathbf{h}^0 = \mathbf{0}$, $\mathbf{W}_{11} \in \mathbb{R}^{m\times n}$, $\mathbf{W}_{12} \in \mathbb{R}^{m\times m}$, $\mathbf{W}_2 \in \mathbb{R}^{2\times m}$ and $\mathbf{b}_2 \in \mathbb{R}^2$ are parameters we wish to learn, m is the number of the hidden states in the simple RNN layers, and $\mathbf{y}$ is the estimate of $(Q(\mathbf{s}, X), Q(\mathbf{s}, Y))$.

2.4. Actor–critic learning using neural networks

The actor–critic algorithm is also a type of reinforcement learning algorithm. It posits a parametric Randomized Stationary Policy (RSP) and rather than seeking an optimal policy, it seeks an optimal parameter vector for the RSP. Traditionally, actor–critic learning uses a logistic function for the policy which leads to a linear function approximation for the Q-value function (Estanjini et al., 2012; Grondman, Busoniu, Lopes, & Babuska, 2012; Konda & Tsitsiklis, 2003; Wang et al., 2015; Wang & Paschalidis, 2017a). In particular, the policy is specified through a probability for selecting action u at state $\mathbf{s}$ given by

$$\mu_{\boldsymbol{\theta}}(u|\mathbf{s}) = \frac{\exp\{\boldsymbol{\theta}'\boldsymbol{\phi}(\mathbf{s},u)\}}{\sum_v \exp\{\boldsymbol{\theta}'\boldsymbol{\phi}(\mathbf{s},v)\}}, \tag{14}$$

where $\boldsymbol{\theta}$ is a parameter vector and $\boldsymbol{\phi}(\mathbf{s},u)$ is a vector of features of the state and the action. The operation on the right hand side of (14) which assigns the highest probability to the action that maximizes the exponent $\boldsymbol{\theta}'\boldsymbol{\phi}(\mathbf{s},u)$ is often referred to as *Softmax*. Specifically, for some vector $\mathbf{x} = (x_1,\ldots,x_k) \in \mathbb{R}^k$, Softmax($\mathbf{x}$) $\in \mathbb{R}^k$ and the ith element is given by

$$\text{Softmax}(\mathbf{x})_i = \frac{\exp(x_i)}{\sum_{j=1}^k \exp(x_j)},$$

$i = 1,\ldots,k$. It can be shown (Estanjini et al., 2012; Konda & Tsitsiklis, 2003; Wang et al., 2015; Wang & Paschalidis, 2017a), that given an RSP as in (14), a good linear approximation of the Q-value function is $Q_{\boldsymbol{\theta}}(\mathbf{s},u) = \mathbf{r}'\boldsymbol{\psi}_{\boldsymbol{\theta}}(\mathbf{s},u)$ where $\boldsymbol{\psi}_{\boldsymbol{\theta}}(\mathbf{s},u) = \nabla \ln \mu_{\boldsymbol{\theta}}(u|\mathbf{s})$.

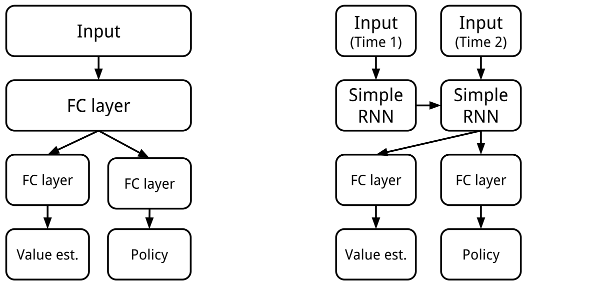
The actor–critic method alternates between an *actor* step which is a gradient update of the parameter vector $\boldsymbol{\theta}$ using the gradient of the long-term reward, and a *critic* step which, given the current $\boldsymbol{\theta}$, uses Temporal-Difference (TD) learning (Pennesi & Paschalidis, 2010) to learn the appropriate parameter $\mathbf{r}$ in the Q-value function approximation in addition to the long-term reward and its gradient with respect to $\boldsymbol{\theta}$. As we commented earlier when discussing Q-learning, these methods have been shown to converge (Estanjini et al., 2012; Konda & Tsitsiklis, 2003) but depend on proper selection of feature functions in order to be effective.

If instead one uses a neural network to approximate the value function and the policy, the actor–critic updating steps should be modified. For example, Hausknecht and Stone (2015) proposed a deep actor–critic learning similar to the DQN. Mnih et al. (2016) used a simpler way, updating a loss function that combines the policy advantage and a temporal-difference term for the value function.

In this paper, we use the actor–critic learning algorithm of Mnih et al. (2016) to handle the learning tasks we introduced in Section 2.1. The neural network takes as input the state $\mathbf{s}$ of the MDP, and outputs both a policy and a value function estimate. As we did with Q-learning, we will use both a feed-forward neural network and an RNN version of the algorithm. Again, we only use the actor–critic learning algorithm in Mnih et al. (2016) with a single agent.

In the feed-forward network case, the neural network structure for actor–critic learning is shown in Fig. 5. For the policy, we use a fully-connected layer with a Softmax activation function. For the value function, we use an additional fully-connected layer without an activation function. Letting $\mathbf{s}$ be the vector encoded state, $\mathbf{h}$ the output of the first FC layer, v the output of the FC layer producing the value function estimate, and $\boldsymbol{\mu}$ the output of the FC layer producing the policy estimate, we have

$$\begin{aligned} \mathbf{h} &= \text{ReLU}(\mathbf{W}_1\mathbf{s} + \mathbf{b}_1), \\ \boldsymbol{\mu} &= \text{Softmax}(\mathbf{W}_\mu \mathbf{h} + \mathbf{b}_\mu), \\ v &= \mathbf{W}_v \mathbf{h} + b_v, \end{aligned} \tag{15}$$



(a) An illustration of the neural network for actor-critic learning using the vector encoding of the state. (b) An illustration of the neural network for actor critic learning using the sequential encoding of the state.

Fig. 5. An illustration of the neural networks used in actor–critic learning. The FC layer in the diagram represents a fully-connected layer. Q-value est. in the diagram denotes the estimate of the Q-value of the state at all possible actions.

where $\mathbf{W}_1 \in \mathbb{R}^{m\times n}$, $\mathbf{b}_1 \in \mathbb{R}^m$, $\mathbf{W}_\mu \in \mathbb{R}^{2\times m}$, $\mathbf{b}_\mu \in \mathbb{R}^2$, $\mathbf{W}_v \in \mathbb{R}^{1\times m}$ and $b_v \in \mathbb{R}$ are the parameters in the neural networks we need to learn, and m is the number of the hidden neurons in the first FC layer.

In the RNN case, and similar to the Q-learning case, we let $\mathbf{s}_1$, $\mathbf{s}_2$ be the sequential encoding of the input state $\mathbf{s}$, $\mathbf{h}_1$, $\mathbf{h}_2$ the outputs of the first and the second RNN layers, v the output of the FC layer producing the value function estimate, and $\boldsymbol{\mu}$ the output of the FC layer producing the policy estimate, which leads to

$$\begin{aligned} \mathbf{h}^1 &= \text{ReLU}(\mathbf{W}_{11}\mathbf{s}_1 + \mathbf{W}_{12}\mathbf{h}^0), \qquad (16) \\ \mathbf{h}^2 &= \text{ReLU}(\mathbf{W}_{11}\mathbf{s}_2 + \mathbf{W}_{12}\mathbf{h}^1), \\ \boldsymbol{\mu} &= \text{Softmax}(\mathbf{W}_\mu \mathbf{h}^2 + \mathbf{b}_\mu), \\ v &= W_v \mathbf{h}^2 + b_v, \qquad (17) \end{aligned}$$

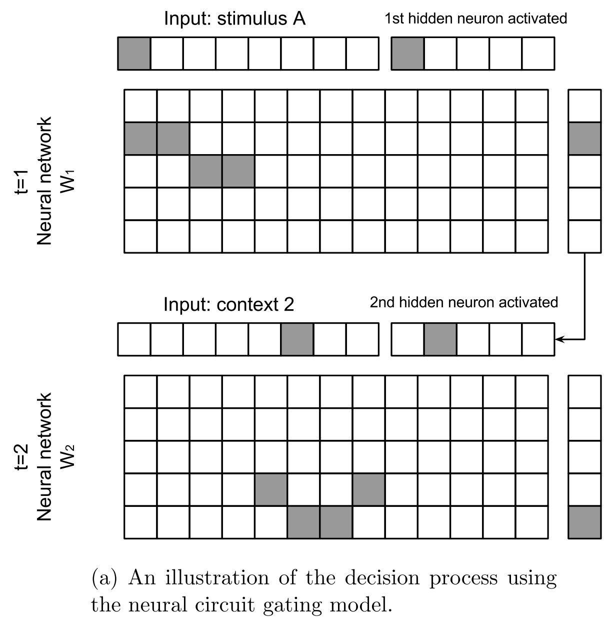
where the initial state of the RNN is $\mathbf{h}^0 = \mathbf{0}$, $\mathbf{W}_{11} \in \mathbb{R}^{m\times n}$, $\mathbf{W}_{12} \in \mathbb{R}^{m\times m}$, $\mathbf{W}_\mu \in \mathbb{R}^{2\times m}$, $\mathbf{W}_v \in \mathbb{R}^{1\times m}$, $\mathbf{b}_\mu \in \mathbb{R}^2$ and $b_v \in \mathbb{R}$ are parameters to learn and m is the number of hidden neurons in the RNN layers.

2.5. Neural circuit gating model

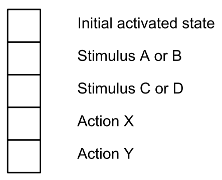
The neural network implementations in Q-learning and actor–critic learning use standard firing-rate neuron models (Dayan & Abbott, 2001) with threshold-linear input–output functions (rectified linear unit, ReLU). Next we present an alternative approach for the tasks we outlined in Section 2.1. This alternate approach uses neurons with simpler step-function threshold dynamics that could be considered similar to the generation of single spikes in individual neurons. These single spikes then gate the spread of activity between other neurons by altering the weight matrix (Hasselmo & Stern, 2018).

With regard to biological justification, this gating mechanism is based on the nonlinear effects between synaptic inputs on adjacent parts of the dendritic tree that are due to voltage-sensitive conductances such as the N-Methyl-D-Aspartate (NMDA) current (Katz, Kath, Spruston, & Hasselmo, 2007; Poirazi, Brannon, & Mel, 2003). These interactions could allow synaptic input from a spiking neuron to determine whether adjacent neurons have a significant influence on the membrane potential. Here, this is represented by the spiking of hidden neurons directly gating the weight matrix. Alternatively, these effects could be due to axo-axonic inhibition gating the output of individual neurons.

Learning parameters for this model are based on the Hebbian rule for plasticity of synaptic connections. Previously, Hasselmo (2005) presented a neurobiological circuit model with gating of the spread of neural activity combined with local Hebbian learning



(a) An illustration of the decision process using the neural circuit gating model.



(b) An interpretation of the hidden states.

Fig. 6. An illustration of the decision process by the simplest version of the neural circuit model after successful learning for the task of Fig. 1. In this example, we use the stimulus–context pair A2 as input, provided in the form of the sequential encoding $\mathbf{s}_1, \mathbf{s}_2$. Gray entries denote 1 and empty (white) entries denote zero. For illustrative proposes, we ignore the noise (setting $\epsilon = 0$). At $t = 1$, the first hidden neuron is activated by default. After Hebbian learning, this neuron gates the weight matrix $\mathbf{W}_1$. The encoded input for stimulus A spreads across this weight matrix $\mathbf{W}_1$ gated by the hidden neuron (cf. (18)), resulting in the output pattern in which the second hidden neuron is activated, which gates the weight matrix $\mathbf{W}_2$. Thus, at $t = 2$, the network weight matrix $\mathbf{W}_2$ is applied to $\mathbf{a}^2 = (0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0)$. With the coded input of context 2, the activity spreads across the weight matrix to generate an output in which the fifth hidden neuron is activated, corresponding to action Y.

rules and suggested it could have functions similar to TD learning. In this paper, we present another neural circuit model based mostly on Hebbian rules (Hasselmo & Stern, 2018), which has a performance comparable to the more abstract neural network models. Our neural circuit model is defined next. An illustrative example of this model and how it operates for the task of Fig. 1 can be found in Fig. 6.

We will use the sequential encoding of the state as input, where a state $\mathbf{s}$ is presented as a sequence of two n-dimensional vectors $\mathbf{s}_1$ and $\mathbf{s}_2$. We have n neurons to receive these signals. In addition to the input neurons, we use $m = 5$ hidden neurons to process the information. Three of these neurons store an internal state and two are used to output actions X and Y (see Fig. 6(b)). We denote by a_i^t the activation of neuron i at time t; $a_i^t = 1$ if the neuron gets activated and is zero otherwise. Here, $i = 1, \ldots, n$ corresponds to the input neurons and $i = n+1, \ldots, n+m$ corresponds to the hidden neurons. We let $\mathbf{a}^t = (a_1^t, \ldots, a_{n+m}^t)$. For simplicity, we assume that there is only one hidden neuron spiking at each time.

Spiking of different hidden neurons induces a different structure of the neural network weight matrix between input neurons and hidden neurons. As noted above, this reflects the nonlinear interaction of synapses on the dendritic tree, in which activation of one synapse can allow an adjacent synapse with voltage-sensitive conductances to have an effect. The weight matrix of the neural network is denoted as $\mathbf{W}_j \in \mathbb{R}^{m\times(n+m)}$, when hidden neuron j is spiking. Let

$$\mathbf{f}^t = \mathbf{W}_j \mathbf{a}^t, \tag{18}$$

where $j = \{i \mid a_i^t = 1\}$ is the index of the activated hidden neuron at time t. Notice that the activated hidden neuron determines the weight matrix to be used. We assume that these iterations start with the first hidden neuron being activated at $t = 1$, namely $\mathbf{a}^1 = (\mathbf{s}_1, 1, 0, 0, 0, 0)$.

To determine the state of the hidden neurons $(a_{n+1}^{t+1}, a_{n+2}^{t+1}, a_{n+3}^{t+1})$ at time $t+1$, we use the following probabilistic model. With a small probability ε, we randomly pick one of these hidden neurons to emit a spike at time $t+1$. Otherwise, we let the neuron $k = n+1, n+2, n+3$ with the highest f_k^t to emit a spike. This procedure can be interpreted as a balance of exploration and exploitation in the reinforcement learning context (Bertsekas, 1995; Bertsekas & Tsitsiklis, 1996).

The last two hidden neurons (a_{n+4}^t, a_{n+5}^t) represent selection of either action X or action Y. Specifically, $(a_{n+4}^t, a_{n+5}^t) = (1, 0)$ selects action X and $(a_{n+4}^t, a_{n+5}^t) = (0, 1)$ action Y. Otherwise, no action is implemented.

To learn the weight matrices $\mathbf{W}_i$ we follow the properties of the Hebbian learning rule. The synapses are updated by reward-dependent Hebbian Long-Term Potentiation (LTP), in which active synapses are tagged based on the presence of joint pre-synaptic and post-synaptic activity, and then, the synapse is strengthened if the output action matches the correct action. Long-Term Depression (LTD) provides an activity-dependent reduction in the efficacy of neuronal synapses to serve as a regularization of the learning process.

The LTP rule is mostly based on the basic Hebb rule in Dayan and Abbott (2001), in which simultaneous pre- and post-synaptic activity increases synaptic strength. In particular, suppose the i_t hidden neuron is activated at time t. Let $\mathbf{a}_h^t$ denote the vector consisting of the last m components of $\mathbf{a}^t$, corresponding to the hidden neurons. Then the LTP term is

$$\Delta \mathbf{W}_{i_t,\mathrm{LTP}} = \mathbf{a}_h^{t\prime} \mathbf{o}^t,$$

where $\mathbf{o}^t \in \mathbb{R}^m$ indicates the spiking of hidden neuron at time t. In particular, we let $\mathbf{o}_i^t = 1$ if hidden neuron i is activated at time t, and $\mathbf{o}_i^t = 0$ otherwise. The LTD term is

$$\Delta \mathbf{W}_{i_t,\mathrm{LTD}} = -(\mathbf{a}_h^t)'\mathbf{e}$$

where $\mathbf{e} \in \mathbb{R}^m$ is the vector of all 1's.

Finally, the weight matrices $\mathbf{W}_i$ are updated as follows. For a given input signal, when the output of the neural circuit model coincides with the correct output, we update the weight matrices $\mathbf{W}_{i_1}$ and $\mathbf{W}_{i_2}$ as

$$\mathbf{W}_{i_t,t+1} = \mathbf{W}_{i_t,t} + \alpha_{LTP} \Delta \mathbf{W}_{i_t,\mathrm{LTP}} + \alpha_{LTD} \Delta \mathbf{W}_{i_t,\mathrm{LTD}} \tag{19}$$

where α_{LTP} and α_{LTD} are appropriate stepsizes. Since this updating rule is not necessarily stable, we project the elements of $\mathbf{W}_i$ to $[0, 1]$ after every update.

3. Results

In this section, we investigate the performance of the proposed algorithm in the context association task of Fig. 1. We also test our algorithms on the larger task of Fig. 2 to assess how well they scale.

Most neural models encode information in a distributed manner across a population of neurons (Dayan & Abbott, 2001). This manner of encoding information has many advantages, such as graceful degradation when individual neurons are lost. However, the distributed representation makes it difficult to interpret the activity patterns in trained models. In order to overcome this difficulty, this paper uses two approaches to investigate the performance of different methods. The first one is a minimalist approach, which tries to use the simplest model to train the reinforcement learning agents and the neural circuit gating model. Using the least number of units and parameters, it becomes easier to understand how these methods solve the learning tasks of Section 2.1. The other approach is to use a larger numbers of neurons to learn the tasks since the real neural system encodes the information in a distributed manner. We will compare the performance of each model across different numbers of units and find some potential relationships among these methods.

3.1. Q-learning with function approximation by a neural network

First, we test the Q-learning algorithm with function approximation on the task of Fig. 1 using a feedforward neural network (cf. (12)). For minimalism, we use only one hidden layer with 2 hidden neurons. For training the network we use Algorithm 1 in Mnih et al. (2016). We observe, that the policy we obtain can find the correct action related to each label without having seen the 4 unseen states shown in Fig. 1. One set of learned parameters which allows the model to successfully perform the task in Fig. 1 is shown in (A.1) of Appendix A. It is interesting to observe in $\mathbf{W}_1$ the similarity of the columns corresponding to contexts 1 and 4 as well as contexts 2 and 3, which reflects the symmetry of the mapping in Fig. 1.

Next, we test the Q-learning algorithm on the task of Fig. 1 using sequentially-encoded input. Again, the neural network we use has 2 hidden states in each simple RNN layer. Using the same setting, the learned model successfully makes each generalization and finds the correct answer. One set of learned parameters which succeed in the task of Fig. 1 is shown in (B.1) of Appendix B. Again, the desired symmetry can be observed in the columns of $\mathbf{W}_{11}$ corresponding to the various contexts.

Finally, we tested the Q-learning algorithm on the task of Fig. 1 using different numbers of hidden neurons. It is known that biological neural systems encode information in a distributed manner. One pattern of information may be encoded by many neurons. This encoding may not be very efficient (Dayan & Abbott, 2001), but increased numbers of units may help in the learning procedure. We changed the number of the hidden neurons to assess its effect on the performance of the neural network. We used the following settings for the various parameters of the Q-learning and the actor–critic learning algorithms. We used the Adam optimizer (Kingma & Ba, 2014) with a learning rate of 0.05, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$. The discount factor was $\gamma = 0.9$. We let the algorithm update every 100 actions. The maximum learning step was set to 50,000 actions. Shown in Fig. 7 is the performance of all of our algorithms on the task of Fig. 1 as a function of the number of hidden neurons. Notice that by increasing the number of hidden neurons, we improve the performance of the learning.

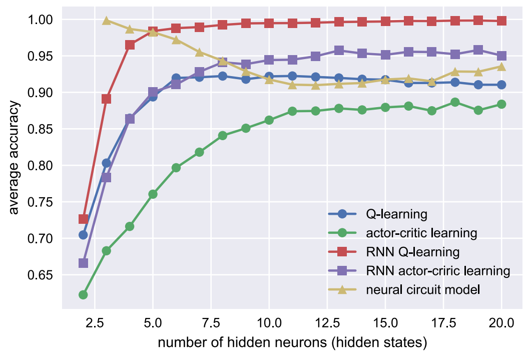


Fig. 7. The average accuracy (% of correct actions in a test set of input states) of different models. We tested each algorithm 1000 times and averaged the test results.

3.2. Actor–critic learning

We first tested the actor–critic algorithm on the task of Fig. 1 using the vector-encoded input. As a minimal example, we again used only one hidden layer with 2 hidden neurons. Using Algorithm S3 in Mnih et al. (2016), the actor–critic agent converges to parameters that succeed in performing the learning task. The learned parameters are shown in (C.1) of Appendix C. Again, the columns of the weight matrix $\mathbf{W}_1$ corresponding to contexts 1 and 4 are similar, and the same is the case for the columns corresponding to contexts 2 and 3.

Next, we investigated actor–critic learning by using sequentially-encoded inputs on the task of Fig. 1. Each RNN uses 2 hidden neurons. We use the same learning parameters as in Q-learning and set the entropy regularization parameter (β in Mnih et al., 2016) to 1. The trained model successfully makes the appropriate generalizations and succeeds in finding the correct action. One set of learned parameters is shown in (D.1) of Appendix D, where again the correct context symmetry is evident.

Finally, we tested the actor–critic learning algorithm on the task of Fig. 1 using different numbers of hidden neurons. We used the same settings as in the last subsection to evaluate the performance as a function of the number of hidden neurons. We did not include the entropy regularization in Algorithm S3 in Mnih et al. (2016) for simplicity.

3.3. Neural circuit gating model

In this section, we used the neural circuit model with gating described in Section 2.5 and applied to the task of Fig. 1. As we described, we use five hidden neurons in this model, three of which are used to store the internal state. The neural circuit gating model uses the sequentially-encoded input. We let $\varepsilon = 0.01$, $\alpha_{\text{LTP}} = 0.8$, and $\alpha_{\text{LTD}} = 0.1$. This neural circuit gating model can successfully generalize what it learned and find the correct actions. One set of learned parameters that succeeded in the task in Fig. 1 is shown in (E.1)–(E.4) of Appendix E. Again, these matrices reflect the symmetry exhibited in the mapping from states to actions.

We also tested the performance of the neural circuit gating model on the task of Fig. 1 with different numbers of hidden neurons. We set the maximum number of iterations to 50,000 actions. The performance of this model is shown in Fig. 7. It can be seen that the neural circuit gating model attains relatively high performance with a small number of hidden neurons and then decreases gradually as the number of the hidden neurons increases. Its performance is comparable to the RNN versions of the Q-learning and actor–critic models. We discuss these observations further in Section 4.3.

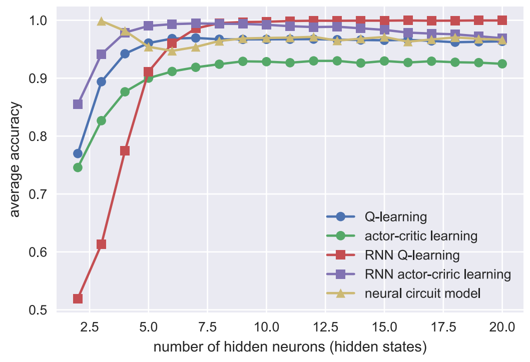


Fig. 8. The average accuracy (% of correct actions in a test set of input states) of different models. We tested each algorithm 1000 times and averaged the test results.

3.4. Learning to perform the larger task

In this subsection, we consider the larger context association task (cf. Fig. 2) in order to assess the scalability of the methods we examined. We test the performances of different algorithms as we increase the size of the model. The results are shown in Fig. 8. We find that the performances of the different algorithms in this larger task are qualitatively similar to the smaller task we discussed earlier. The accuracy in this larger task is slightly higher than the basic task under the same model complexity. This is because this task provides more training examples than the basic task, which makes the models less likely to overfit.

4. Discussion

4.1. Comparison between models with sequentially-encoded inputs

The development of different models allowed us to compare the mechanisms with which the trained neural networks and the spiking neural circuit model make decisions for the context-dependent association task. We first analyze the neural circuit model applied to the task of Fig. 1, with the decision rule shown in Fig. 9. Suppose the input is A1. Recall that hidden neuron 1 is activated before the first input. So the neural network structure used at time 1 is $\mathbf{W}_1$. From the matrices in Appendix E, it can be seen that stimulus A or B will activate hidden neuron 2 while stimulus C or D will activate hidden neuron 3. Thus, at time 2, if hidden neuron 2 is activated, the network uses weight matrix $\mathbf{W}_2$, whereas if hidden neuron 3 is activated, the network uses weight matrix $\mathbf{W}_3$ (shown in the Appendix). From the structure of $\mathbf{W}_2$, context 1 or 4 will activate hidden neuron 4 and context 2 or 3 will activate hidden neuron 5. Thus, for stimulus A or B in context 1 or 4, the network generates action X, and for stimulus A or B in context 2 or 3 it will generate action Y. If stimulus C or D is presented, then activation of neuron 3 results in the network using weight matrix $\mathbf{W}_3$ and the network selects the opposite actions in response to the contexts (e.g., contexts 1 & 4 result in generation of action Y). The neural circuit model makes this decision hierarchically. It also utilizes the similarities, say between context 1 and context 2, to make decisions. Thus, based on our mechanism of gating, the trained neural circuit gating model has the capacity to abstract the learning rules and properly select actions for previously unseen pairings of contexts and stimuli.

Then we consider the RNN version of the Q-learning and the actor–critic model applied to the task of Fig. 1. We will argue that these RNN models learn similar decision making processes as the neuron circuit model, even though this is not immediately evident by observing the trained parameters. Fig. 10 shows the evolution of the hidden states of the RNN in Q-learning and actor–critic learning, respectively. The RNN represents stimuli A and B or stimuli C and D using similar encodings. From the RNN weight matrices learned by these two methods, we observe the similarity between contexts 1 and 4 and 2 and 3, respectively. It follows that the RNN discovers the appropriate symmetry in the mapping and can successfully select actions for previously unseen pairings of context and stimulus.

4.2. Comparison of feedforward neural networks and recurrent neural networks

As we have seen, there are similarities in decision making by the neural circuit gating model and the RNN models learned by either Q-learning or actor–critic learning. Both are able to discover the symmetry of the map between stimulus–context pairs and actions. Both make decisions hierarchically. The functional properties of feed-forward neural networks are, however, quite different.

We show the states of the hidden layers of each of the feed-forward neural networks used for learning to perform the task of Fig. 1 in Fig. 11. Both hidden layers separate the different inputs in one shot. Based on the weight matrices learned by the two reinforcement learning algorithms, they capture the symmetry in the state–action map. Yet, we can no longer claim that the feed-forward neural networks make the decision hierarchically, since the linear structure of the fully-connected layer cannot compose a hierarchical decision rule.

4.3. Relationship of neural circuit gating model to other models

We have seen some similarities between the reinforcement learning methods using neural networks and the neural circuit gating model. This raises the following question: why are the two types of methods similar?

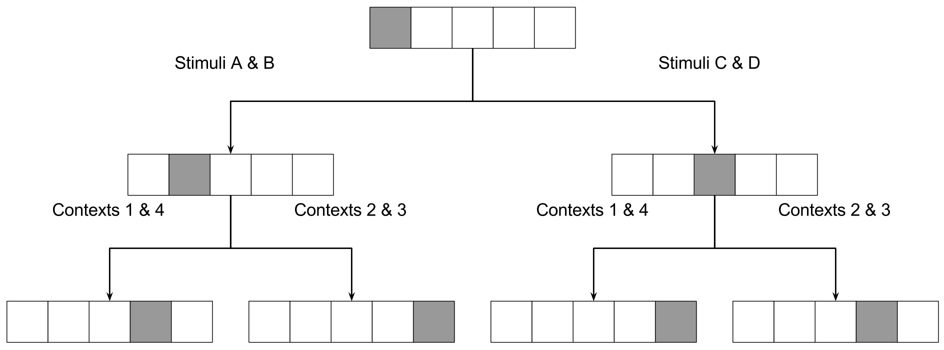
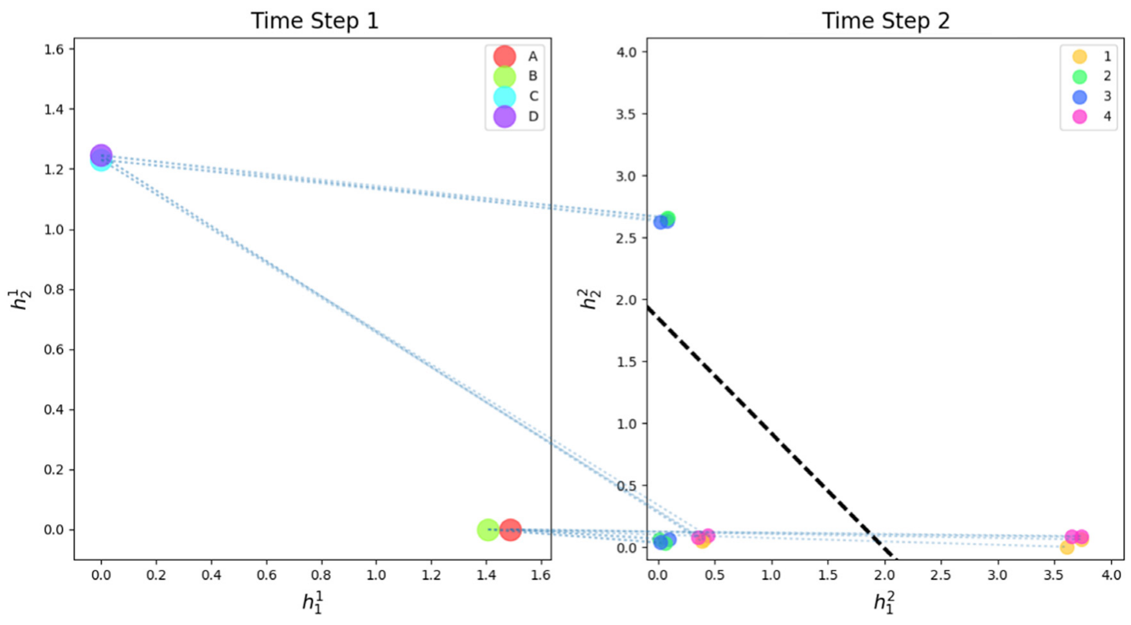
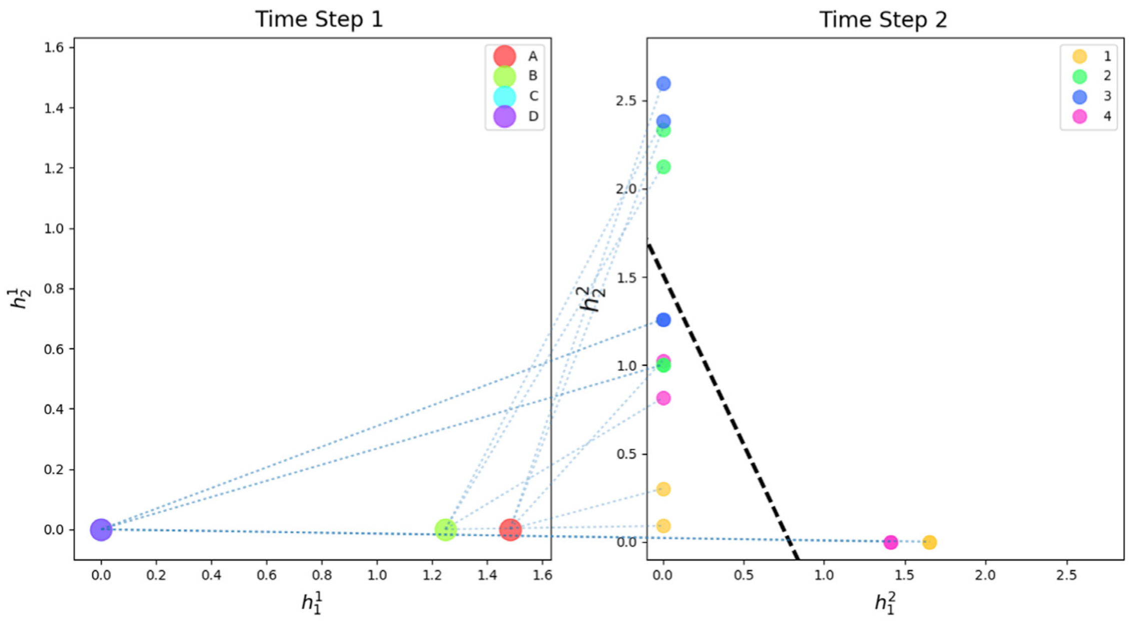


Fig. 9. The hierarchical decision rule of the neural circuit model demonstrated for the task of Fig. 1. The blocks at the three different levels show the state of the hidden neurons at each time instant. The gray block indicates the spiking of the corresponding hidden neuron.



(a) Q-learning using the simple RNN.

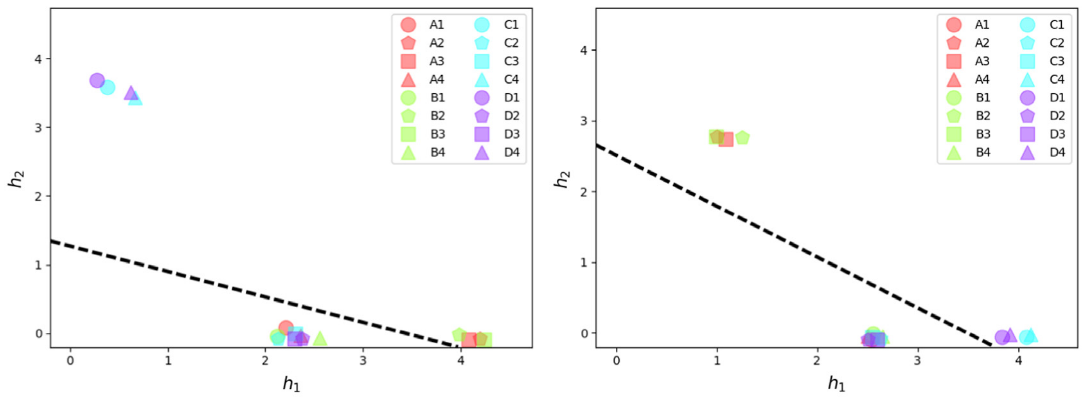


(b) Actor-critic learning using the simple RNN.

Fig. 10. The hidden state of the simple RNN for the task of Fig. 1. The activation function is ReLU, so the hidden state variables are nonnegative. The linkage between the two time steps in the corresponding figures indicates that a state $\mathbf{s}_1$ in time step 1 is precursor of a state $\mathbf{s}_2$ in time step 2. The black dashed line in the figures in time step 2 is the decision boundary: points above the line correspond to X and points below the line correspond to Y. For example, note in Figure (a), stimulus C in the left figure is connected to contexts 2 and 3 in the right figure and map to X. To make the dots easier to differentiate, a small amount of noise was added to the position of the dots in time step 2.

First, the neural circuit gating model does not involve an explicit method of optimization nor does it employ temporal difference learning and back-propagation of the error. We think the key is in the way the weight matrices of the neural circuit get updated by Eq. (19). Notice that this update reinforces the weight matrix involved in a correct decision through the LTP term. On the other hand, the LTD term reduces these weights in order to avoid "over-fitting" to a correct decision. Furthermore, the noise introduced in translating the $\mathbf{f}^t$ signals of (18) into neuron firings allows sufficient exploration of the state–action space.

This neuro-computational model uses the effect of internal activity on gating units that regulate the spread of activity within the circuit. These models are based on earlier models using interactions of current sensory input states with backward spread of activity from future desired goals (Hasselmo, 2005; Koene & Hasselmo, 2005). These earlier models were able to simulate the learning of goal-directed action selection based on the interaction of feedback from goal with current input, rather than using temporal difference learning. The use of gating in these early models resembles the gating properties used in models known as LSTM (Long Short



(a) Q-learning using the vector-encoded input.

(b) Actor-critic learning using vector-encoded input.

Fig. 11. The hidden state of the feedforward neural networks for the task of Fig. 1. The activation function is ReLU, so the hidden state variables are nonnegative. The black dashed line in the figures is the decision boundary determined by the output layer; points above the line correspond to action Y and below the line to action X.

Term Memory) in which gating was regulated by back-propagation through time (Gers, Schmidhuber, & Cummins, 2000; Graves & Schmidhuber, 2005; Hochreiter & Schmidhuber, 1997). The use of gating in these models also resembles the models developed by O'Reilly and his group in which the prefrontal cortex interacts with basal ganglia to gate the flow of information into and out of working memory (O'Reilly & Frank, 2006). However, the models differ in that the O'Reilly and Frank model used gating regulated by reinforcement learning mechanisms (termed PVLV) similar to temporal difference learning for regulating dopaminergic activity for reward or expectation of reward (O'Reilly, Frank, Hazy, & Watz, 2007), whereas the Hasselmo model (Hasselmo, 2005) focused on the internal spread from representations of desired output or goals within cortical structures (Koene & Hasselmo, 2005). The O'Reilly and Frank framework has been used to effectively store symbol-like role-filler interactions (Kriete, Noelle, Cohen, & O'Reilly, 2013) and to model performance in an n-back task (Chatham et al., 2011) and a hierarchical rule learning task (Badre & Frank, 2012).

These gating models can be thought of as extending the use of actions in reinforcement learning models. In most of these models, actions change the state of an agent in its external network (Sutton & Barto, 1998). In contrast, the model presented here builds on the use of internal processes or "memory actions" (Zilli & Hasselmo, 2008c) that modify internal activity by performing tasks such as loading a working memory buffer, or loading an episodic memory buffer (Zilli & Hasselmo, 2008c). The use of memory actions allows solution of non-Markov decision processes based on retention of memory from prior states (Zilli & Hasselmo, 2008a, b). The modeling of gating in neural circuits could provide different potential mechanisms for rule learning.

5. Conclusions

We simulated performance of a contextual association task from Raudies et al. (2014) involving inputs consisting of a stimulus and context and exhibiting particular symmetry in the stimulus–action map. In particular, under some contexts one type of mapping from input to actions is applicable, while in other contexts the mapping is reversed. Human and animals, when presented with enough examples that demonstrate both maps, have the ability to generalize and make correct decisions even on inputs that have never been presented to them.

We first established that traditional reinforcement learning algorithms, such as Q-learning or Q-learning using a linear function approximation architecture, do not have the ability to generalize beyond the examples presented to them in a training phase. We then examined a variety of neural network-based models. We considered two reinforcement learning algorithms, Q-learning and actor–critic, analyzed in the approximate dynamic programming literature (Bertsekas & Tsitsiklis, 1996) and recently tested by training on a series of computer games (Hausknecht & Stone, 2015; Mnih et al., 2016, 2015). We employed both feedforward neural networks and recurrent neural networks as function approximators in these learning methods. We found that the recurrent neural networks perform better, which could potentially be explained by their use of a hierarchical (rather than a "flat") decision making process. We also devised a custom-made neural circuit gating model, which uses hidden neurons to determine how the input is being processed. This model was trained using Hebbian-type updating of the weight matrices. Surprisingly, the simple neural circuit performs similarly to the more sophisticated reinforcement learning algorithms. A potential explanation is that hierarchical reasoning is the key to performance and the specific learning method is of less importance.

Acknowledgments

Research partially supported by the National Science Foundation, United States under grants DMS-1664644, CNS-1645681, CCF-1527292 and IIS-1237022, by the Army Research Office, United States under grant W911NF-12-1-0390, and by the Office of Naval Research, United States under grant MURI N00014-16-1-2832.

Appendix A. Parameters: Q-learning with vector-encoded state

Eq. (A.1) is given in Box I.

Appendix B. Parameters: Q-learning with sequentially-encoded state

Eq. (B.1) is given in Box II.

Appendix C. Parameters: actor–critic learning with vector-encoded state

Eq. (C.1) is given in Box III.

Appendix D. Parameters: actor–critic learning with sequentially-encoded state

Eq. (D.1) is given in Box IV.

$$\mathbf{W}_1 = \begin{array}{c} \begin{array}{cccccccc} \multicolumn{4}{c}{\text{stimulus}} & \multicolumn{4}{c}{\text{context}} \\ \text{A} & \text{B} & \text{C} & \text{D} & 1 & 2 & 3 & 4 \end{array} \\ \left[\begin{array}{cccccccc} 1.70 & 1.66 & -0.17 & -0.17 & -0.29 & 1.57 & 1.59 & -0.11 \\ -1.57 & -2.01 & 1.89 & 1.95 & 2.02 & -1.77 & -1.91 & 1.86 \end{array}\right] \end{array}$$

$$\mathbf{b}_1 = \begin{bmatrix} 1.01 \\ -0.27 \end{bmatrix}, \ \mathbf{W}_2 = \begin{bmatrix} 0.25 & -1.21 \\ 1.17 & 1.29 \end{bmatrix}, \ \mathbf{b}_2 = \begin{bmatrix} -2.40 \\ -5.57 \end{bmatrix}. \tag{A.1}$$

Box I.

$$\mathbf{W}_{11} = \begin{array}{c} \begin{array}{cccccccc} \multicolumn{4}{c}{\text{stimulus}} & \multicolumn{4}{c}{\text{context}} \\ \text{A} & \text{B} & \text{C} & \text{D} & 1 & 2 & 3 & 4 \end{array} \\ \left[\begin{array}{cccccccc} 1.49 & 1.41 & -0.85 & -0.89 & 1.95 & -2.18 & -2.18 & 1.96 \\ -0.14 & -0.21 & 1.23 & 1.25 & -2.00 & 1.84 & 1.85 & -2.03 \end{array}\right] \end{array}$$

$$\mathbf{W}_{12} = \begin{bmatrix} 1.15 & -1.30 \\ -1.82 & 0.62 \end{bmatrix}, \ \mathbf{W}_2 = \begin{bmatrix} 1.33 & 1.43 \\ -1.26 & -1.37 \end{bmatrix}, \ \mathbf{b}_2 = \begin{bmatrix} 4.64 \\ 9.82 \end{bmatrix}. \tag{B.1}$$

Box II.

$$\mathbf{W}_1 = \begin{array}{c} \begin{array}{cccccccc} \multicolumn{4}{c}{\text{stimulus}} & \multicolumn{4}{c}{\text{context}} \\ \text{A} & \text{B} & \text{C} & \text{D} & 1 & 2 & 3 & 4 \end{array} \\ \left[\begin{array}{cccccccc} -0.03 & -0.00 & 1.42 & 1.38 & 1.45 & 0.04 & 0.04 & 1.50 \\ 1.27 & 1.27 & -1.58 & -1.68 & -1.66 & 1.31 & 1.28 & -1.62 \end{array}\right] \end{array}$$

$$\mathbf{b}_1 = \begin{bmatrix} 1.24 \\ 0.22 \end{bmatrix}, \ \mathbf{W}_p = \begin{bmatrix} -2.10 & -2.95 \\ 2.12 & 2.92 \end{bmatrix}, \ \mathbf{b}_p = \begin{bmatrix} 7.36 \\ -7.36 \end{bmatrix}$$

$$\mathbf{W}_v = \begin{bmatrix} 1.02 & 0.77 \end{bmatrix}, \ b_v = 5.95. \tag{C.1}$$

Box III.

$$\mathbf{W}_{11} = \begin{array}{c} \begin{array}{cccccccc} \multicolumn{4}{c}{\text{stimulus}} & \multicolumn{4}{c}{\text{context}} \\ \text{A} & \text{B} & \text{C} & \text{D} & 1 & 2 & 3 & 4 \end{array} \\ \left[\begin{array}{cccccccc} 1.48 & 1.25 & -1.19 & -1.08 & 1.65 & -1.60 & -1.39 & 1.41 \\ -0.78 & -1.20 & -0.68 & -0.56 & -1.03 & 1.00 & 1.26 & -0.31 \end{array}\right] \end{array}$$

$$\mathbf{W}_{12} = \begin{bmatrix} -1.69 & 0.20 \\ 0.90 & -0.18 \end{bmatrix}, \ \mathbf{W}_p = \begin{bmatrix} -4.01 & -2.06 \\ 4.01 & 2.10 \end{bmatrix},$$

$$\mathbf{b}_p = \begin{bmatrix} 3.15 & -3.15 \end{bmatrix}, \ \mathbf{W}_v = \begin{bmatrix} -0.01 & -0.04 \end{bmatrix}, \ b_v = 9.92. \tag{D.1}$$

Box IV.

Appendix E. Parameters: neural circuit gating model

$$\mathbf{W}_1 = \begin{array}{c} \begin{array}{ccccccccccccc} \multicolumn{4}{c}{\text{stimulus}} & \multicolumn{4}{c}{\text{context}} & \multicolumn{5}{c}{\text{hid. neur.}} \\ \text{A} & \text{B} & \text{C} & \text{D} & 1 & 2 & 3 & 4 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \left[\begin{array}{ccccccccccccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right] \end{array} \tag{E.1}$$

$$\mathbf{W}_2 = \begin{array}{c} \begin{array}{ccccccccccccc} \multicolumn{4}{c}{\text{stimulus}} & \multicolumn{4}{c}{\text{context}} & \multicolumn{5}{c}{\text{hid. neur.}} \\ \text{A} & \text{B} & \text{C} & \text{D} & 1 & 2 & 3 & 4 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \left[\begin{array}{ccccccccccccc} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right] \end{array} \tag{E.2}$$

$$\mathbf{W}_3 = \begin{array}{c} \\ \\ \\ \\ \\ \\ \end{array}\begin{array}{ccccccccccccc} \multicolumn{4}{c}{\text{stimulus}} & \multicolumn{4}{c}{\text{context}} & \multicolumn{5}{c}{\text{hid. neur.}} \\ A & B & C & D & 1 & 2 & 3 & 4 & 1 & 2 & 3 & 4 & 5 \end{array}$$
$$\mathbf{W}_3 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (E.3)$$

$$\mathbf{W}_4 = \mathbf{W}_5 = \mathbf{0}. \quad (E.4)$$

References

Badre, D., & Frank, M. J. (2012). Mechanisms of hierarchical reinforcement learning in cortico–striatal circuits 2: Evidence from fMRI. *Cerebral Cortex*, *22*(3), 527–536.

Badre, D., Kayser, A. S., & D'Esposito, M. (2010). Frontal cortex and the discovery of abstract action rules. *Neuron*, *66*(2), 315–326.

Bertsekas, D. P. (1995). *Dynamic programming and optimal control. Vol. I and II*. Belmont, MA: Athena Scientific.

Bertsekas, D., & Tsitsiklis, J. (1996). *Neuro-dynamic programming*. Belmont, MA: Athena Scientific.

Chatham, C. H., Herd, S. A., Brant, A. M., Hazy, T. E., Miyake, A., O'Reilly, R., et al. (2011). From an executive network to executive control: a computational model of the *n*-back task. *Journal of Cognitive Neuroscience*, *23*(11), 3598–3619.

Dayan, P., & Abbott, L. F. (2001). *Theoretical neuroscience. Vol. 10*. Cambridge, MA: MIT Press.

Dayan, P., & Watkins, C. (1992). Q-learning. *Machine Learning*, *8*(3), 279–292.

Estanjini, R. M., Li, K., & Paschalidis, I. C. (2012). A least squares temporal difference actor–critic algorithm with applications to warehouse management. *Naval Research Logistics (NRL)*, *59*(3–4), 197–211 URL http://dx.doi.org/101002/nav.21481.

Gers, F. A., Schmidhuber, J., & Cummins, F. (2000). Learning to forget: Continual prediction with LSTM. *Neural Computation*, *12*(10), 2451–2471.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press, URL http://www.deeplearningbook.org.

Graves, A., & Schmidhuber, J. (2005). Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks*, *18*(5), 602–610.

Grondman, I., Busoniu, L., Lopes, G. A., & Babuska, R. (2012). A survey of actor-critic reinforcement learning: Standard and natural policy gradients. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, *42*(6), 1291–1307.

Hasselmo, M. E. (2005). A model of prefrontal cortical mechanisms for goal-directed behavior. *Journal of Cognitive Neuroscience*, *17*(7), 1115–1129.

Hasselmo, M. E., & Eichenbaum, H. (2005). Hippocampal mechanisms for the context-dependent retrieval of episodes. *Neural Networks*, *18*(9), 1172–1190.

Hasselmo, M. E., & Stern, C. E. (2018). A network model of behavioural performance in a rule learning task. *Philosophical Transactions of the Royal Society B: Biological Sciences*, *373*, 20170275.

Hausknecht, M., & Stone, P. (2015). Deep reinforcement learning in parameterized action space. arXiv preprint arXiv:1511.04143.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*(8), 1735–1780.

Katz, Y., Kath, W. L., Spruston, N., & Hasselmo, M. E. (2007). Coincidence detection of place and temporal context in a network model of spiking hippocampal neurons. *PLoS Computational Biology*, *3*(12), e234.

Kingma, D., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

Koene, R. A., & Hasselmo, M. E. (2005). An integrate-and-fire model of prefrontal cortex neuronal activity during performance of goal-directed decision making. *Cerebral Cortex*, *15*(12), 1964–1981.

Konda, V. R., & Tsitsiklis, J. N. (2003). On actor-critic algorithms. *SIAM Journal on Control and Optimization*, *42*(4), 1143–1166.

Kriete, T., Noelle, D. C., Cohen, J. D., & O'Reilly, R. C. (2013). Indirection and symbol-like processing in the prefrontal cortex and basal ganglia. *Proceedings of the National Academy of Sciences*, *110*(41), 16390–16395.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*(7553), 436–444.

Levine, S., Finn, C., Darrell, T., & Abbeel, P. (2016). End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research (JMLR)*, *17*(1), 1334–1373.

Liu, H., Wu, Y., & Sun, F. (2018). Extreme trust region policy optimization for active object recognition. *IEEE Transactions on Neural Networks and Learning Systems*, *29*(6), 2253–2258.

Miller, E. K., & Cohen, J. D. (2001). An integrative theory of prefrontal cortex function. *Annual Review of Neuroscience*, *24*(1), 167–202.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., & Lillicrap, T. P., et al. (2016). Asynchronous Methods for Deep Reinforcement Learning. arXiv 48, 1–28. URL http://arxiv.org/abs/1602.01783.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. a., Veness, J., Bellemare, M. G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529–533. URL http://dx.doi.org/101038/nature14236.

Nair, V., & Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International conference on machine learning (ICML-10)* (pp. 807–814)..

O'Reilly, R. C., & Frank, M. J. (2006). Making working memory work: a computational model of learning in the prefrontal cortex and basal ganglia. *Neural Computation*, *18*(2), 283–328.

O'Reilly, R. C., Frank, M. J., Hazy, T. E., & Watz, B. (2007). PVLV: the primary value and learned value Pavlovian learning algorithm. *Behavioral Neuroscience*, *121*(1), 31.

Pennesi, P., & Paschalidis, I. C. (2010). A distributed actor-critic algorithm and applications to mobile sensor network coordination problems. *IEEE Transactions on Automatic Control*, *55*(2), 492–497.

Peters, J., & Schaal, S. (2008). Reinforcement learning of motor skills with policy gradients. *Neural Networks*, *21*(4), 682–697.

Poirazi, P., Brannon, T., & Mel, B. W. (2003). Arithmetic of subthreshold synaptic summation in a model CA1 pyramidal cell. *Neuron*, *37*(6), 977–987.

Raudies, F., Zilli, E. A., & Hasselmo, M. E. (2014). Deep belief networks learn context dependent behavior. *PLoS One*, *9*(3).

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986a). Learning representations by back-propagating errors. *Nature*, *323*(6088), 533–536 URL http://dx.doi.org/101038/323533a0.

Rumelhart, D. E., McClelland, . J. L., et al. (1986b). *Parallel distributed processing*. Cambridge, MA: MIT Press.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., & Moritz, P. (2015). Trust region policy optimization. In *International conference on machine learning* (pp. 1889–1897)..

Sutton, R., & Barto, A. (1998). *Reinforcement learning*. Cambridge, MA: MIT Press.

Tesauro, G. (1994). TD-Gammon, a self-teaching backgammon program, achieves master-level play. *Neural Computation*, *6*(2), 215–219.

Tsitsiklis, J. N. (1994). Asynchronous stochastic approximation and *q*-learning. *Machine Learning*, *16*(3), 185–202.

Tsitsiklis, J. N., & Van Roy, B. (1997). An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, *42*(5), 674–690.

Wallis, J. D., Anderson, K. C., & Miller, E. K. (2001). Single neurons in prefrontal cortex encode abstract rules. *Nature*, *411*(6840), 953–956.

Wang, J., Ding, X., Lahijanian, M., Paschalidis, I. C., & Belta, C. A. (2015). Temporal logic motion control using actor–critic methods. *International Journal of Robotics Research*, *34*(10), 1329–1344.

Wang, J., & Paschalidis, I. C. (2017a). An actor-critic algorithm with second-order actor and critic. *IEEE Transactions on Automatic Control*, *62*(6), 2689–2703.

Wang, J., & Paschalidis, I. C. (2017b). An actor-critic algorithm with second-order actor and critic. *IEEE Transactions on Automatic Control*, *62*(6), 2689–2703. http://dx.doi.org/10.1109/TAC.2016.2616384.

Watkins, C. J., & Dayan, P. (1992). Q-learning. *Machine Learning*, *8*(3–4), 279–292.

Watter, M., Springenberg, J., Boedecker, J., & Riedmiller, M. (2015). Embed to control: A locally linear latent dynamics model for control from raw images. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, & R. Garnett (Eds.), *Advances in neural information processing systems. Vol. 28* (pp. 2746–2754). Curran Associates, Inc..

Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., & Salakhutdinov, R., et al. (2015). Show, attend and tell: Neural image caption generation with visual attention. arXiv preprint arXiv:1502.03044 2 (3), 5.

Xu, X., Zuo, L., & Huang, Z. (2014). Reinforcement learning algorithms with function approximation: Recent advances and applications. *Information Sciences*, *261*, 1–31.

Zilli, E. A., & Hasselmo, M. E. (2008a). Analyses of markov decision process structure regarding the possible strategic use of interacting memory systems. *Frontiers in Computational Neuroscience*, *2*, 6.

Zilli, E. A., & Hasselmo, M. E. (2008b). The influence of markov decision process structure on the possible strategic use of working memory and episodic memory. *PLoS One*, *3*(7), e2756.

Zilli, E. A., & Hasselmo, M. E. (2008c). Modeling the role of working memory and episodic memory in behavioral tasks. *Hippocampus*, *18*(2), 193–209.