A neural model of hierarchical reinforcement learning

Daniel Rasmussen (drasmuss@uwaterloo.ca)
Chris Eliasmith (celiasmith@uwaterloo.ca)
Centre for Theoretical Neuroscience, University of Waterloo
Waterloo, ON, Canada, N2J 3G1

Abstract

We present the first model capable of performing hierarchical reinforcement learning in a general, neurally detailed implementation. We show that this model is able to learn a spatial pickup and delivery task more quickly than one without hierarchical abilities. In addition, we show that this model is able to leverage its hierarchical structure to transfer learned knowledge between related tasks. These results point towards the advantages to be gained by using a hierarchical RL framework to understand the brain's powerful learning ability.

Keywords: neural model; reinforcement learning; hierarchical reinforcement learning; Neural Engineering Framework

Introduction

Reinforcement learning (RL) has a long history of rich interaction between computational theories and neuroscientific understanding. This interaction has led to new understandings of neural data, as well as new biologically inspired computational theories. However, basic RL techniques have a number of challenges, two of the most critical being difficulty scaling up to complex problem domains and difficulty transferring knowledge between tasks (Barto & Mahadevan, 2003; Taylor & Stone, 2009). These are two areas in which the brain excels, which presents a dilemma for neural reinforcement learning models. However, new theories have been proposed to overcome these challenges on the computational side, which may again prove fruitful when we apply them in the effort to understand and model the brain's learning ability.

One active area of research in RL is the field of Hierarchical Reinforcement Learning (HRL; Barto & Mahadevan, 2003). HRL introduces higher level actions into the RL framework, where selecting one of those actions may drive a whole sequence of decisions. For example, a high level action might be "go to the grocery store" or "go to work", and selecting one of those actions then guides a sequence of "right turn" or "left turn" sub-actions. This helps to address the scaling problem by imposing more structure on the problem space. A long sequence of decisions can now be encapsulated in a single choice ("go to the grocery store"), and the value of that choice can be calculated in a single learning update (somewhat) independently of the intervening choices. HRL also addresses the knowledge transfer problem, as the high-level actions represent natural, modular components to transfer between tasks. For example, once we have learned how to navigate to work, it is easy to see how we could reuse that skill as a subcomponent in many other work-related tasks.

In this paper we present a biologically plausible neural model capable of performing hierarchical reinforcement learning. This allows us to bring the enhanced power of HRL into a theory of neural function, providing a hypothesis for how the brain could achieve its strengths in scaling and knowledge transfer. In the next section we give a brief introduction to the mathematical underpinnings of HRL. We then describe the implementation of the model we have developed, and demonstrate the ability of this model to speed learning and transfer knowledge between related tasks. We conclude with a discussion of some of the testable predictions that arise from this model, and the next directions for its continued development.

Background

Reinforcement learning

Reinforcement learning is concerned with maximizing overall reward across a sequence of decisions. It is usually formulated as a Markov Decision Process (MDP) where the task has some state space S, available actions A, transition function $T(s,a)$ (which describes how the agent will move through the state space given a current state s and selected action a), and reward function $r(s,a)$ (which describes the feedback the agent will receive after selecting action a in state s).

Temporal difference (TD) learning describes a popular family of methods for solving the reinforcement learning problem.[1] It uses the concept of Q values, where $Q(s,a)$ indicates the long term reward to be expected when selecting action a in state s. This can be expressed recursively as the immediate reward $r(s,a)$ plus the value of the next state, that is:

$$Q(s,a) = r(s,a) + \gamma Q(s',a') \tag{1}$$

(γ is a discount factor applied to future rewards).

TD learning is a method for learning those Q values in an environment where the transition and reward functions are unknown, and can only be sampled by exploring the environment. It accomplishes this by taking advantage of the fact that a Q value is essentially a prediction, which can be compared against observed data. That is, as the agent moves through the state space it is acquiring samples of $r(s,a)$ and $Q(s',a')$, and it can use the difference between that observed data and

[1] TD learning is also one of the classic examples of cross-fertilization between computational theory and neuroscience, as it presented a new framework to understand data on dopamine function (Schultz, 1998).

$Q(s,a)$ to update the prediction.[2] Specifically,

$$\Delta Q(s,a) = r(s,a) + \gamma Q(s',a') - Q(s,a) \quad (2)$$

There are many approaches to building neural models of reinforcement learning, ranging from more abstract artificial neural networks to detailed spiking neural models (Potjans et al., 2009; Frémaux et al., 2013). In Rasmussen & Eliasmith (2013) we present our own model of reinforcement learning, which extended previous efforts in ways that will become important in the next section. We will not go into detail on these different approaches here, as we wish to focus on the new development of hierarchical reinforcement learning.

Hierarchical reinforcement learning

As RL is based on the MDP framework, HRL is based on the Semi-MDP (SMDP) framework. SMDPs extend MDPs by adding time into the equation. Mathematically, the transition and reward functions—$T(s,a,t)$ and $r(s,a,t)$—now depend on time as well as the state/action selected. This means that, for example, if the agent selects an action they may not get a reward until several seconds or minutes later (or they could receive several rewards throughout that period), and they may not arrive in a new state until some time after that.

The SMDP framework is important for HRL, as the time delays can be used to encapsulate the activity of the subpolicy. That is, after selecting the action of "go to the grocery store", the results cannot be observed immediately. The pertinent information needs to be preserved over time while that subpolicy executes, so that the TD error can be computed once the grocery store is reached. The SMDP framework allows us to represent that type of environment.

Under the SMDP framework, Q values can be re-expressed as:

$$Q(s,a) = \sum_{t=0}^{\tau-1} \gamma^t r(s,a,t) + \gamma^\tau Q(s',a') \quad (3)$$

where τ is the time at which the state transition occurs. This leads to a modified TD update of:

$$\Delta Q(s,a) = \sum_{t=0}^{\tau-1} \gamma^t r(s,a,t) + \gamma^\tau Q(s',a') - Q(s,a) \quad (4)$$

Although computationally simple (the main difference being that rewards are summed over the delay period), the SMDP framework significantly complicates a neural implementation. The main problem is that information on reward, discount, and Q values now needs to be preserved over a potentially unknown, variable, and lengthy delay period. Almost all neural RL models rely on some type of "eligibility trace" to preserve information between states, which imposes a fixed and short[3] time window on any learning update.

[2]Note that what we describe here is the SARSA (Rummery & Niranjan, 1994) implementation of TD learning. The other main approach is Q-learning (Watkins & Dayan, 1992), which operates on a similar principle but searches over possible future $Q(s',a')$ values rather than waiting to observe them.

[3]Assuming the common biological explanation for eligibility traces based on neurotransmitter decay.

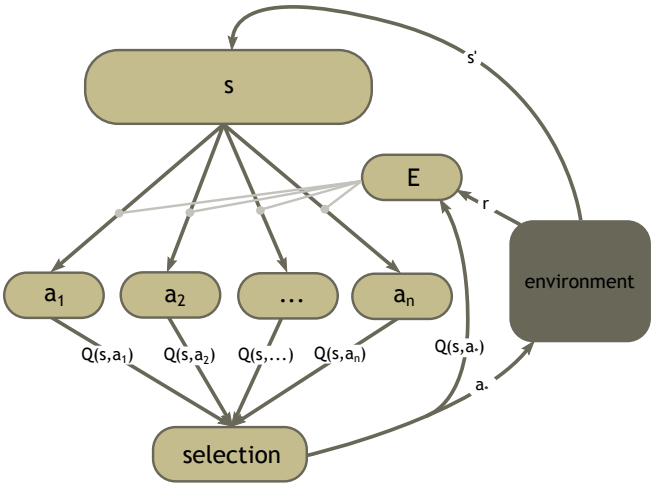

Figure 1: Architecture of a model for performing non-hierarchical reinforcement learning. See text and Rasmussen & Eliasmith (2013) for details.

However, in previous work (Rasmussen & Eliasmith, 2013) we demonstrated a model capable of learning in an arbitrary SMDP environment, thus laying the groundwork for a model of HRL.

There has been little work attempting to integrate this computational theory with neural modelling. Botvinick et al. (2009) discuss how the actor-critic architecture could be extended to support HRL, along with the neurophysiological evidence supporting the plausibility of such extensions. However, their model itself was not implemented at the neural level. In Frank & Badre (2012) the authors modified their previous model of corticostriatal action selection to allow for a hierarchy of actions. However, theirs was a model of a specific hierarchical task, rather than a general model of hierarchical reinforcement learning that can be applied across tasks. For example, their model was unable to solve tasks involving temporally extended sequences of actions. We are not aware of any other work that presents a general model of how the brain could perform hierarchical reinforcement learning.

Model

Previous work

This work is based on a previous model of SMDP reinforcement learning, described in Rasmussen & Eliasmith (2013). We will briefly review the important features of that model, but for the sake of brevity focus primarily on how we extend the model to perform HRL.

The model's architecture is shown in Figure 1. At the top is a population of neurons that represent the current state, s. This state can represent any desired information; it is simply an abstract vector, which is encoded into neural activities using the principles of the Neural Engineering Framework (NEF; Eliasmith & Anderson, 2003). The state can vary continuously over time and space, or, if desired, the system can approximate discrete states by restricting the state to fixed

points in the vector space. All components of this model are implemented using leaky integrate-and-fire (LIF) neurons; in the case of the state population, these neurons, combined with the principles of the NEF, take the input state and convert it into firing activity.

The output activity of the state neurons is passed to a second set of neural populations, corresponding to the different actions available to the agent, a_n.[4] Each of those populations attempts to output the value of its associated action given the current state (as represented by the activity of the s population). Using the NEF we can interpret the output of the a neurons as estimated Q values.

Next, the system needs to select an action to perform based on those Q values. The "selection" network performs this function. The core of this component is a neural model of the basal ganglia and thalamus (for more detail see Stewart et al. 2010), along with several memory components needed to preserve information across the SMDP time delay. The end result is that the highest valued action and the Q value of that action are produced as output.

The action is delivered to the environment, which computes a new state and reward. The system is designed to treat the environment as a black box; all of its processing occurs in a general, task-independent fashion, so that different environments can be swapped out without affecting the rest of the model.

The value of the selected action is delivered to the error calculation network, E. This network computes the error shown in Equation 4 through several interconnected neural dynamical systems (its implementation is described in more detail in Rasmussen & Eliasmith 2013). The output of this network is used to drive an error-modulated local learning rule (MacNeil & Eliasmith, 2011) on the connections between the s and a populations, so that over time the the output of the a populations will come to represent the correct Q values.

Hierarchical model

In order to extend this model for hierarchical reinforcement learning, the first step is to allow the model to represent several different policies. That is, it needs to be able to represent one set of Q values if it is in the "go to the grocery store" context, and flexibly switch to a different set of Q values if the context changes to "go to work".

One approach would be to have multiple sets of connections between the s and a populations, and switch between them using some gating mechanism. However, this is impractical for a number of reasons: it greatly increases the number of connections needed, it introduces the new problem of how to switch between connections, and it is inflexible in that the contexts must be directly encoded into the structure of the model. Thus in our model we instead accomplish this by including a representation of the current context in the vector input to the s population. The output of the s neurons then represents context-dependent activity, allowing the system to produce different Q values in different contexts with a single set of connection weights. This allows the system to represent and swap between different policies simply by changing the context representation in the s input, without changing any of the structural aspects of the model.

The next question is how to organize the model into a hierarchy, so that higher level decisions (e.g., "go to the grocery store") can control the lower level decisions. Given the structure laid out above, this can be accomplished by allowing high level systems to set the context in lower level systems. This architecture is shown in Figure 2. The key feature is that the action selected by the higher level system, rather than affecting the environment, is used to set the context of the lower level system.[5] Thus if the higher level system were to select the "go to the grocery store" action, it would set the lower level system to be in the "grocery store" context. The lower level system would then choose actions according to its "grocery store" policy, and the selected actions would be delivered to the environment to control the movement of the agent.

Note that we have shown a system here with two levels, but there is no theoretical restriction on the depth of the hierarchy. These systems could be chained together in this way to provide as many levels as desired, the only constraint being the number of neurons required (the model used in this work uses approximately 35 000 neurons per level, but that value is affected by the complexity of the task). In addition, we have shown the architecture here such that all levels of the hierarchy receive the same environmental state and reward information, which is the simplest case. However, this system can also operate in the case where different hierarchical levels use different (e.g., internally generated) state/reward values, an example of which is demonstrated in the results section.

In regard to the neuroanatomical mapping, it is important to note that although we have separate selection networks in the different levels, neuroanatomically these are all based in the same basal ganglia. The different selection networks in each layer correspond to different corticostriatal loops, which have been shown to be organized in a hierarchical manner (Frank & Badre, 2012). In addition, the label for the state representations, "cortex", is intentionally ambiguous. As mentioned, this model is designed as a general reinforcement learning system that can operate across many tasks. Thus the state could take on many forms; it could be visual input, hippocampal place cell activations, or more abstract prefrontal representations, all of which have efferent projections to the basal ganglia.

It is not necessary *a priori* that different hierarchical levels should have different structural components. For example, an alternate implementation would be to have one system,

[4] The system we describe here uses a discrete action space, where the agent chooses one of n possible actions. However, that is not an intrinsic requirement of this architecture; this type of system could represent a continuous action space through a weighted sum of the available actions.

[5] This general style of architecture has been employed in several hierarchical systems throughout the years; for example, it can be traced back to work on feudal RL (Dayan & Hinton, 1993).

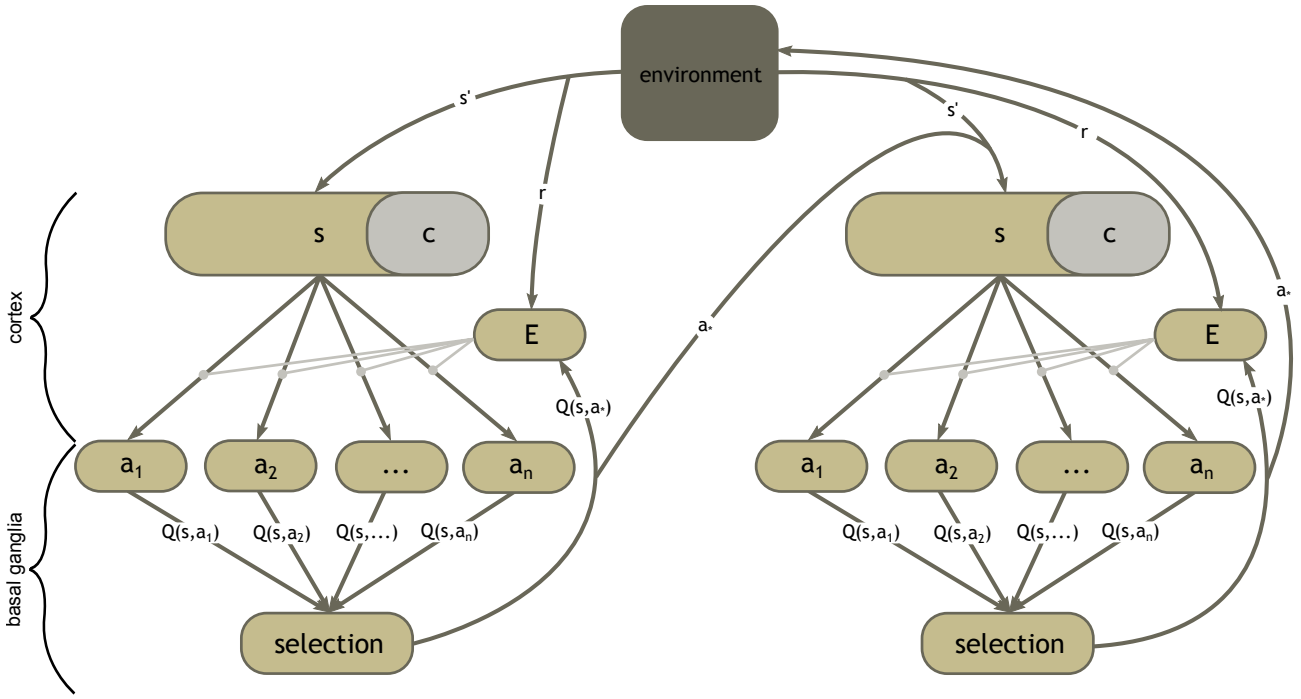

Figure 2: Hierarchical reinforcement learning architecture, wherein the actions of the higher level system modify the context of the lower level system. See text for details.

with both the low and high level actions available to it, and the high level actions would recursively modify the system's own context. However, the implementation we have chosen is consistent with empirical data on reinforcement learning in hierarchical tasks from Badre et al. (2010). They showed that learning in the hierarchical setting showed structurally distinct activations, with more abstract contexts associated with increasingly anterior activation in the prefrontal cortex. In addition, they showed that subjects were able to learn at multiple levels of the hierarchy simultaneously, which is an important advantage of the implementation we have chosen. That is, if low and high level actions are combined into a single system, then that system can only learn at one level at a time (corresponding to the currently selected action). Separating out the levels allows this system to learn in parallel at all levels of the hierarchy.

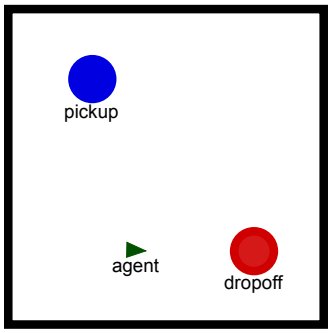

Figure 3: Schematic representation of the environment in the delivery task. The agent must navigate to the pickup location to retrieve the package and then move to the dropoff location to receive reward.

Results

Task

In order to demonstrate the performance of the model, we have chosen to use a delivery task. In this task the agent must go to one location to pick up a package, and then a second location to drop it off. This task is commonly used in HRL, both in computational and experimental settings, as it naturally lends itself to hierarchical learning: at the low level the system simply learns to navigate to a given location, and the high level learns which location should be the current target.

The model operates in continuous time and space. A schematic representation of the environment is shown in Figure 3. The agent begins at a random location, empty-handed. It can move in any of the four cardinal directions, unless blocked by a wall (shown in black) in which case it will stay still. Upon entering the blue region the agent will pick up the package. Upon entering the red region with the package in hand, the agent receives a constant reward of 1.5 for 500ms, at which point the package is reset and the agent is moved to another random location. At all other times the agent receives a base reward of -0.05. This penalty increases by -0.1 for every second the agent attempts to move into a wall, to encourage it to complete the task quickly and move throughout the environment.

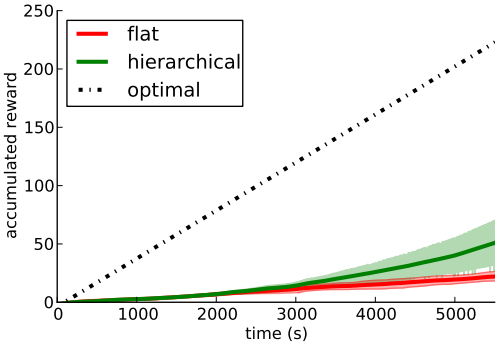

Figure 4: Total reward accumulated by a flat versus hierarchical reinforcement learning model over the course of the delivery task, demonstrating the advantages of a hierarchical system. Displaying 95% confidence intervals.

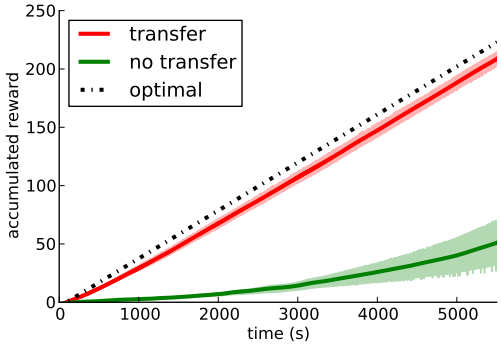

Figure 5: Total reward accumulated by a model initialized with skills learned on a simpler task versus a model without prior information, demonstrating the ability of the model to transfer knowledge between tasks. Displaying 95% confidence intervals.

The state representation output from the environment is constructed to mimic the output of hippocampal place cells. Simulated place cells are tuned to random locations throughout the environment. Each cell has a Gaussian activation corresponding to the distance of the agent from that location. These activations are concatenated into an n-dimensional vector (where n is the number of place cells), which becomes the state signal input to the model. The environment represents whether the agent has the package in hand or not by appending one of two orthogonal 2-dimensional vectors to the state representation.

Hierarchical learning

The first result to demonstrate is that a model with hierarchical learning ability performs better than a standard "flat" reinforcement learning system. For this we trained two systems on the delivery task. One had the structure shown in Figure 1 and the other had the structure shown in Figure 2. In the hierarchical system, the lower level receives an internally generated reward rather than reward from the environment—a reward of 1.5 whenever it achieves the goal set by the high level system. Other than the structural differences, the two systems were identical: they had the same parameters, and the same initial conditions (we initialized all Q values to 0.1).

Figure 4 shows the results of the two systems. We have plotted the total reward accumulated by each system relative to the reward accumulated by a randomly moving agent (used as a baseline). We are also showing an upper bound on performance, determined by simulating an agent that performed optimally, always selecting the correct action. It can be seen that while both the flat and hierarchical systems begin with near-random performance, after approximately 2000 seconds[6] the hierarchical system has begun to markedly improve its performance, as evidenced by the increasing slope of reward accumulation. By the end of the training period the average rate of reward accumulation is 52% of optimal for the hierarchical system and 13% for the flat system.

Knowledge transfer

The next important aspect of HRL is its ability to support knowledge transfer. To test this ability, we pretrained a flat model (as in Figure 1) for 2000 seconds on a simpler task, where the agent was rewarded just for moving to one of the two targets in Figure 3 (randomly chosen every 60 seconds). Thus the system learns the low-level skills it needs (how to navigate to the targets), but not the high-level policy for how to put those skills together to accomplish the delivery task. We then took the knowledge learned in that system, represented by the connection weights between the s and a populations, and loaded it into the corresponding actions in the lower layer of a hierarchical model as shown in Figure 2.

As can be seen in Figure 5, the model is able to successfully transfer knowledge between the two tasks. Even though the model has never seen the delivery task before, it is able to begin with quite high performance due to its previous experience on the simpler task. It is worth noting that this benefit goes beyond a simple 2000 second head start; even after 2000 seconds, the untrained model has still not achieved the performance (as measured by the slope of reward accumulation) of the transfered model. This is because the learning in the high level system is significantly aided by the fact that the lower level system can reliably perform the actions selected by the high level system. In the untrained system both layers must train up simultaneously, which is a more difficult task. By the end of the training period the average rate of reward accumulation is 98% of optimal for the transfered system and 52% for the untrained system.

[6]Note that all times shown are simulation time; the model itself takes much longer to run, due to the challenges of simulating large-scale neural models in current software and hardware. Improving the simulation speed of NEF models is a focus of ongoing work (see Bekolay et al., 2014).

Discussion

We have demonstrated the ability of the model to perform hierarchical learning, as well as the enhanced reinforcement learning abilities of such a model. Specifically, we have shown that the model is able to take advantage of the hierarchical structure of a task to speed its learning versus a flat reinforcement learning model. Another important advantage of the HRL approach is that it naturally lends itself to knowledge transfer, which we demonstrated through the model's ability to benefit from knowledge gained on a related task to speed its learning on the delivery task. This is the the first model to present a general and neurally detailed implementation of hierarchical reinforcement learning.

With the functional capabilities of the model established, it is now possible to begin to compare it in detail to experimental data. One of the important goals of these models is to create predictions, which can be used to verify the model as well as to generate new hypotheses for experimental investigation. One implication of this model is its hierarchical structure; namely, that different layers of the hierarchy are separated into structurally distinct regions, and that they interact by the output of higher level regions modifying the state representations in lower level regions. As mentioned, there is already support for this hypothesis in the work of Badre et al. (2010). However, this model allows us to generate much more specific predictions, such as the timecourse and magnitude of error signals in each level over the course of a task.

Another important prediction from this model is the presence of time-delayed representations. This arises from the switch to the SMDP framework, which requires the model to preserve information on the identity and value of the previous state. This suggests that we should be able to find neural activity that correlates not just with the current state (which is already fairly well established), but also, simultaneously, activity representing the value of the previous decision point.

We will also continue to expand the functional capabilities of this model. One of the important open questions in HRL research is how to learn the hierarchical structure (e.g., learning which states should become subgoals), as opposed to having that structure built in as it is in this model. We are particularly interested in the work of Singh et al. (2005) on intrinsic motivation. This is the idea that the brain has internal mechanisms that make novel or surprising states/events rewarding, independent of the external task reward, and that that internally generated reward signal can be used to guide the development of useful subpolicies. Extending the model presented here to include that ability would allow it to provide a more fully-featured account of the brain's flexible reinforcement learning ability.

Acknowledgements

This work was supported by the Natural Sciences and Engineering Research Council of Canada, Canada Research Chairs, the Canadian Foundation for Innovation, and Ontario Innovation Trust.

References

Badre, D., Kayser, A. S., & D'Esposito, M. (2010). Frontal cortex and the discovery of abstract action rules. *Neuron*, *66*(2), 315–26.

Barto, A. G., & Mahadevan, S. (2003). Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 1–28.

Bekolay, T., Bergstra, J., Hunsberger, E., DeWolf, T., Stewart, T. C., Rasmussen, D., ... Eliasmith, C. (2014). Nengo: a Python tool for building large-scale functional brain models. *Frontiers in Neuroinformatics*, *7*(48), 1–13.

Botvinick, M. M., Niv, Y., & Barto, A. G. (2009). Hierarchically organized behavior and its neural foundations: a reinforcement learning perspective. *Cognition*, *113*(3), 262–80.

Dayan, P., & Hinton, G. (1993). Feudal reinforcement learning. In *Advances in Neural Information Processing Systems* (pp. 271–278).

Eliasmith, C., & Anderson, C. (2003). *Neural engineering: Computation, representation, and dynamics in neurobiological systems*. Cambridge: MIT Press.

Frank, M. J., & Badre, D. (2012). Mechanisms of hierarchical reinforcement learning in corticostriatal circuits 1: computational analysis. *Cerebral Cortex*, *22*(3), 509–26.

Frémaux, N., Sprekeler, H., & Gerstner, W. (2013). Reinforcement Learning Using a Continuous Time Actor-Critic Framework with Spiking Neurons. *PLoS Computational Biology*, *9*(4), e1003024.

MacNeil, D., & Eliasmith, C. (2011). Fine-tuning and the stability of recurrent neural networks. *PloS ONE*, *6*(9), e22885.

Potjans, W., Morrison, A., & Diesmann, M. (2009). A spiking neural network model of an actor-critic learning agent. *Neural Computation*, *339*, 301–339.

Rasmussen, D., & Eliasmith, C. (2013). A neural reinforcement learning model for tasks with unknown time delays. In M. Knauff, M. Pauen, N. Sebanz, & I. Wachsmuth (Eds.), *Proceedings of the 35th Annual Meeting of the Cognitive Science Society* (pp. 3257–3262). Austin: Cognitive Science Society.

Rummery, G., & Niranjan, M. (1994). *On-line Q-learning using connectionist systems* (Tech. Rep. No. September).

Schultz, W. (1998). Predictive reward signal of dopamine neurons. *Journal of Neurophysiology*, *80*, 1–27.

Singh, S., Barto, A. G., & Chentanez, N. (2005). Intrinsically motivated reinforcement learning. In L. K. Saul, Y. Weiss, & L. Bottou (Eds.), *Advances in Neural Information Processing Systems* (pp. 1281–1288). MIT Press.

Stewart, T. C., Choo, X., & Eliasmith, C. (2010). Dynamic behaviour of a spiking model of action selection in the basal ganglia. In S. Ohlsson & R. Catrambone (Eds.), *Proceedings of the 32nd Annual Conference of the Cognitive Science Society* (pp. 235–240). Austin: Cognitive Science Society.

Taylor, M., & Stone, P. (2009). Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, *10*, 1633–1685.

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