The Effect of Prior Knowledge on Training Reinforcement Learning Models

by

Pratyush Menon

Supervised by Professor Asaf Gilboa

April 2022

The Effect of Prior Knowledge on Training Reinforcement Learning Models

by

Pratyush Menon

1004282661

Supervised by Professor Asaf Gilboa

April 2022

Abstract

Learning occurs in the context of prior knowledge, with new information being integrated into existing knowledge structures. Indeed, flexible behavior relies on the ability to draw upon past information. Important facets of this are the recall of similar prior experiences (known as episodic memory), and the capacity to form knowledge structures across several similar experiences (known as schematic memory). The research presented in this thesis attempts to address the implicit assumption that the brain starts in a tabula rasa (blank slate) state by exploring the interplay between flexible behavior and prior knowledge from a computational perspective. It builds upon research conducted on modeling memory consolidation with computational tools such as deep neural networks and reinforcement learning. Specifically, it analyzes the performance of reinforcement learning agents equipped with either episodic or schematic decision-making systems in a foraging task when prior knowledge is provided and taken away. Experiments were performed under two specific conditions: one in which agents were provided prior knowledge of the environment, and another in which agents started under a tabula rasa premise. Results show that prior knowledge does not heavily impact episodic decision-making systems, but a significant improvement in performance for schematic decision-making systems was observed. Consequently, results suggest that the relative importance of schematic decision-making heavily increases when prior knowledge is provided. These results make testable biological predictions suggesting that cortical systems would play a larger role guiding behavior for familiar tasks, while hippocampal systems would be more important for new tasks.

Acknowledgements

This research would not have been possible without a lot of help and guidance. I would like to thank Professor Asaf Gilboa for supervising me throughout this process, and providing me with plenty of advice and direction during my first foray into independent research. I would also like to express my gratitude to Benjamin Nealy for helping me start this process and putting me on the path towards researching this topic. His guidance on experiment design, and his willingness to share his expertise and contacts to help me were also invaluable. Finally, I would like to thank my parents and family for supporting me through not only this process, but also my academic career as a whole. I hope this makes you proud!

Contents

List of Figures

1 Introduction

Flexible behavior relies on the ability to apply previously learned information to new experiences. This ability is facilitated through both the act of recalling similar prior experiences (episodic memory) and the capacity to form generalizable knowledge structures across these experiences (schematic memory). A new method of studying the cognitive methods behind this ability is by using deep reinforcement learning (RL) models. Indeed, previous work has demonstrated the interplay between episodic and schematic memory systems in RL agents (Santoro et al., 2016) and the important role of prior knowledge in the ability to efficiently perform new tasks, both for humans (Dubey et al., 2018) and RL agents (Badreddine and Spranger, 2019).

Despite the important role of prior knowledge, earlier work studying flexible behavior in RL models implicitly assumes a *tabula rasa* premise (Santoro et al., 2016). Indeed, a known deficiency in the cognitive neuroscience literature is the study of how prior knowledge affects and interacts with new information, as many experiments are designed under the assumption that learning and memory take place within a *tabula rasa* state of the brain (Brod et al., 2013). To gain further insight into how prior knowledge affects the systems involved in flexible behavior, such as episodic and schematic memory, this thesis aims to connect these two disparate areas of research with computational modeling.

To study the importance of prior knowledge towards flexible behavior, a model endowed with both episodic and semantic memory (Santoro et al., 2016) will be trained to find rewards in a foraging task under two conditions: with and without prior knowledge representing contextual information about the environment. In addition to testing full model performance, tests will also be conducted on the performance of each memory system in isolation. This will provide further insight into the role that prior knowledge plays in forming each type of memory representation individually. Finally, the relative importance of schematic and episodic memory across conditions will be compared by determining the performance of the two memory systems in each condition. A larger increase in performance for schematic decision-making when prior knowledge is provided would imply a greater importance of schematic knowledge in that condition, and vice versa for episodic knowledge.

The first hypothesis is that prior knowledge will improve learning efficiency of the RL agents with episodic and schematic memory systems. This can be tested by comparing agent performance, as measured by the cumulative reward received across a fixed number of episodes, across conditions and is supported by research done on more conventional RL agent architectures (Badreddine and Spranger, 2019). The second hypothesis is that the presence of prior knowledge will impact the representations formed by each memory system – specifically, the relative importance of episodic memory will decrease in the condition where prior knowledge is provided (and vice versa for schematic memory). This is consistent with results shown by Santoro et al., where the relative importance of schematic memory increased as the agent gained more experience in its environment (Santoro et al., 2016). This can be tested by comparing performance across conditions when the model only has access to one memory system.

To conclude, this research provides an opportunity to improve understanding of the cognitive mechanism of flexible behavior, via machine learning modelling. Specifically, the results will help address "questions regarding the cognitive and neural (inter)dependencies among forms of memory, and the role that these forms of memory play in support of cognition more broadly" (Duff et. al, 2014, Overview).

2 Background & Literature Review

2.1 Memory Systems

The brain relies on multiple memory systems to guide flexible behavior (Doll et al., 2015). While there are theorized to be many forms of memory, this research focuses on declarative memory – ie. the capacity to recall everyday facts and events (Eichenbaum, 2004). This includes systems that capture specific experiences (episodic memories), as well as systems that capture more general patterns (schematic memories) (Santoro et al., 2016).

2.1.1 Episodic Memory

Episodic memory is a form of declarative memory (Burgess et al., 2002). It refers to the ability to recognize and remember previously encountered objects, events, locations, etc. and to discriminate them from each other as well as ones which were not experienced (Xue, 2018). Furthermore, it can be characterized as a representation of relations between events and the context in which they happened, together composing a single specific experience (Eichenbaum, 2017).

Episodic memory is theorized to be mediated by the hippocampus, but cognitive map theory also suggests that the hippocampus is also heavily involved in spatial memory (Burgess et al., 2002). Although seemingly unrelated, these two separate functions may be reconciled by recognizing that space is a central component of all episodic memories (Buzsáki & Moser, 2013).

The key regions of the hippocampus which are theorized to be involved in episodic memory are the dentate gyrus, CA3, and CA1 (Rolls, 2010). The computational purposes of each region were theorized by Rolls in his 2010 paper and will be expanded upon below (Rolls, 2010).

The dentate gyrus is a region of the hippocampus that takes in input from the entorhinal cortex region of the brain. The entorhinal cortex is not located in the hippocampus but contains grid cells which encode spatial information about the organism's environment and location (Lipton & Eichenbaum, 2008). The existence and purpose of grid cells will be further discussed in a later section. The dentate gyrus takes this input and provides an output to the CA3 region of the hippocampus. The CA3 region has far fewer neurons than the Dentate Gyrus, so the Dentate Gyrus produces a sparse and efficient representation of its input. In producing a sparse

representation, it also performs pattern separation/orthogonalization, enabling the hippocampus to store different memories even of very similar events.

The CA3 itself is a sparse, recurrent, autoassociative network in the hippocampus. Its purpose is theorized to be pattern completion – ie. it allows for the retrieval of a whole representation to be initiated by the activation of some small part of the same representation. It can operate effectively as a standalone network, and as such can also allow for arbitrary associations between different inputs. These properties enable it to work well as a storage network for episodic memories.

Finally, the CA1 region of the hippocampus is connected to the CA3 region as well as other parts of the brain. Its purpose is thought to be pattern retrieval – ie. it can retrieve and complete entire episodic memories from the sparse representations that are stored in the CA3.

Together, these three regions are theorized to carry out most of the functions expected for an episodic memory store. Replicating these functions was an important consideration for a computational model of an episodic decision-making system.

2.1.2 Schematic Memory

Schematic memory, also known as semantic memory (Ghosh & Gilboa, 2014), is another form of declarative memory (Eichenbaum, 2004). It refers to a more generalized statistical model formed from multiple events (Lengyel & Dayan, 2007). As a memory structure, schemas are capable of guiding behavior and influencing the encoding and retrieval of episodic memory (Ghosh & Gilboa, 2014). The main region of the brain implicated in schema functions is the ventromedial prefrontal cortex (Ghosh & Gilboa, 2014).

As defined by Ghosh and Gilboa, and illustrated in *Figure 1*, the four necessary features of a schema include an associative network structure, a basis on multiple episodes, a lack of unit detail, and adaptability (Ghosh & Gilboa, 2014).

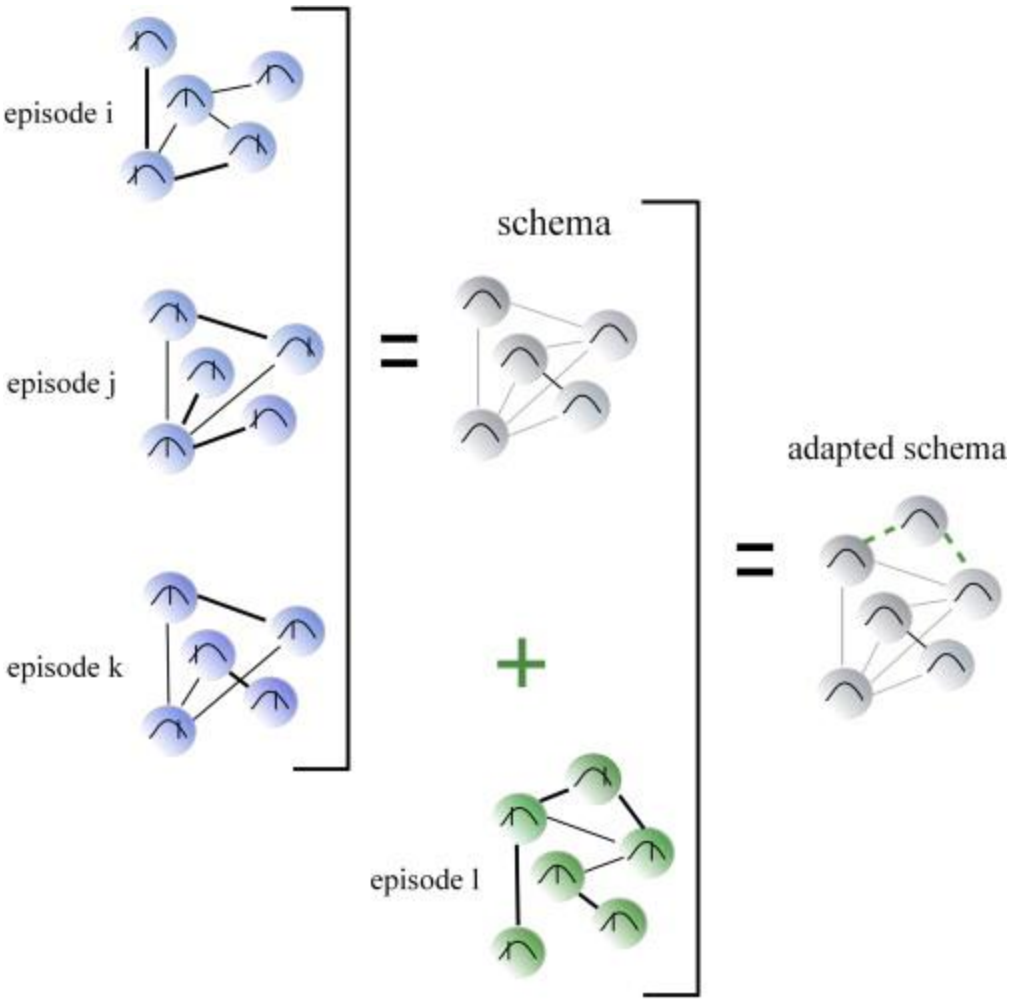

Figure 1: Necessary features of schema structure. *The schema's associative network structure is represented by circles (individual units) and lines connecting them (interrelations). Differences in line connections and thickness indicate variability across episodes. The basis on multiple episodes is depicted by the formation of the schema from episodes i-k. Each episode is unique, but all share the same general structure that is learned by the schema. The lack of unit detail is illustrated by the fact that each unit in each episode takes a specific value sampled from a probability distribution, while the schema learns only the distribution and not any of the specific values. Finally, the adaptability is indicated by the inclusion of new information from episode l, as illustrated by the green lines. (Ghosh & Gilboa, 2014)*

The associative network structure is a consequence of the fact that schemas are composed of units (to hold information) and their interrelations (to integrate different information). Similarly, the requirement that schemas are based on multiple episodes is a result of the definition that

schemas encompass representations of the similarities across multiple events. While this allows schemas to guide behavior in new, unseen situations, it also follows that schemas must lack unit detail since no two constituent episodes will be entirely alike, resulting in differences being lost. Finally, schemas must be adaptable to support the continued acquisition of new information and experiences.

2.1.3 Memory Consolidation

Animals often base their decision making on recent episodic memories, but these episodic memories are encoded and consolidated into schematic memories over time (Santoro et al., 2016). This was tested by Richards et al. in a water maze experiment with rodents, where platform locations were stochastically sampled. 30 days after training, rodent swim patterns more closely matched the probability distribution of platform locations, despite weaker memory for specific platforms – this is consistent with the idea that schematic memory forms as episodic memory is gradually forgotten (Richards et al., 2014). Prior work also shows that neural patterns in the human hippocampus and medial prefrontal cortex have some overlap across associative memories after week, further supporting the gradual organization of discrete episodes into schematic knowledge (Tompary & Davachi, 2017).

This process has been theorized to have several benefits. First and foremost, McClelland et al. suggest that it reduces memory interference between hippocampal and cortical systems (McClelland et al., 1995). While the hippocampal system allows for rapid learning of new items, the cortical systems learn slowly from an ensemble of items (as mentioned in previous sections). Memory consolidation allows new items to be quickly saved without disrupting schema structure, where the new items will be gradually introduced later. Additionally, it is suggested that rapid modifications of cortical representations would induce instability, so memory consolidation allows for new events to be saved immediately while also ensuring that cortical representations can be modified at a more gradual pace (Squire & Alvarez, 1995).

Computational models studying memory consolidation also suggest that it may improve decision making, as it can allow animals to exploit temporal statistics of the environment (Santoro et al., 2016). For example, if fruit trees commonly grow in a certain bounded area, it may be advantageous for an animal to repeatedly visit the same tree while it still has fruit on it (episodic

memory), but in the long run, it is more useful for the animal to know the general location of those fruit trees as the specific tree may die.

2.1.4 Prior Knowledge

The vast majority of learning in animals occurs in the context of pre-existing knowledge (prior knowledge), as new information from the environment is integrated with prior experiences (Brod et al., 2013). As such, prior knowledge directly influences the cognitive processes that are important for learning and retaining new information in our memory systems (Brod et al., 2013). Despite the important role played by prior knowledge, surprisingly little is known about how pre-existing information and new information interact within the brain (Brod et al., 2013). This is likely because most cognitive neuroscience experiments are designed with the implicit assumption that learning and memory take place within a *tabula rasa* (blank) state of the brain (Brod et al., 2013).

This makes the interaction between prior knowledge and established memory systems within the brain an exciting research area.

2.2 Spatial Navigation

Spatial navigation tasks provide a perfect common ground between the cognitive neuroscience and machine learning fields. As mentioned earlier, the hippocampus is theorized to be heavily involved in mediating both episodic and spatial memory (Burgess et al., 2002). Additionally, navigation tasks such as Grid World environments (where the agent is tasked with finding rewards in a 2D grid) are common in reinforcement learning literature. This makes spatial navigation an exciting point of intersection within which research on cognitive mechanisms can be done via artificial intelligence models (Bermudez-Contreras et al., 2020).

2.2.1 Place Cells

A place cell is a type of neuron located in the hippocampus that fires selectively at one or more locations in an environment, as pictured in *Figure 2* (Moser et al., 2015). Different place cells have different firing locations (known as place fields) and are organized non-topographically – ie. the firing fields of neighboring place cells are no more related than the firing fields of place cells in different locations (Moser et al., 2015).

Place cells also have a role in expressing episodic memories; it is suggested that place cells express the location of the animal in combination with information about events that take/took place there (Leutgeb et al., 2005). Further, place cells can express multiple different spatial maps, with maps for different locations being completely uncorrelated. This showed that place cells can express and store multiple orthogonal representations, as theorized for an episodic memory store (Moser et al., 2015).

This close relationship between spatial and episodic memory is why spatial navigation is an apt context within which to study cognitive mechanisms of memory.

2.2.2 Grid Cells

A grid cell is a type of neuron with a periodic firing field that forms a hexagonal grid pattern (Bermudez-Contreras et al., 2020), as pictured in *Figure 2* (Moser et al., 2015). Grid cells are organized non-topographically just like place cells (Moser et al., 2015). The locations at which grid cells fire are dependent on environmental factors such as landmarks and environment shape (Bermudez-Contreras et al., 2020).

Grid cells are located in the entorhinal cortex, which provides input to the hippocampus where place cells are located (Rolls, 2010). As such, it is theorized that place cells are generated by transformations from grid cells as well as other similar classes of spatially modulated cells (Moser et al., 2015).

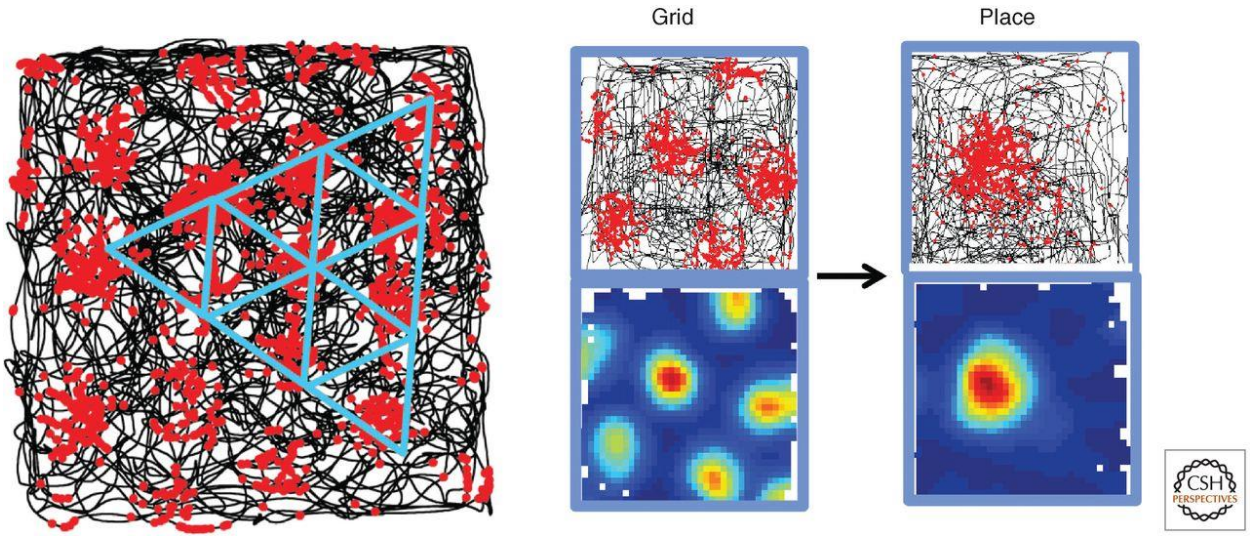

Figure 2: Grid cells and place cells. *On the left, the spike locations of a grid cell from a rat brain are superimposed on the rat's trajectory (black). Blue equilateral triangles are used to show the regular hexagonal structure of the grid pattern. On the top right, trajectories with superimposed spike locations are shown respectively for a grid cell and place cell. The bottom right shows the corresponding rate map for the grid and place cell, with red showing high activity and blue showing low activity. (Moser et al., 2015)*

2.3 Computational Methods

2.3.1 Deep Learning

Machine learning is the field studying algorithms that can "learn" to accomplish a desired task without being explicitly programmed to do so. Deep learning specifically refers to a sub-field of machine learning, focusing on a common algorithm known as neural networks.

In general, neural networks are composed of successive layered affine transformations and non-linear activation functions. "Layers" include the initial vector input to the neural network, as well as the output of each successive affine/linear transformation and activation function. The size/width/number of units in a layer corresponds to the dimensionality of the activation function's output.

For the most basic case, a fully connected neural network, these computations can be mathematically represented by the following equation, where l_i represents the i^{th} layer of the neural network, l_{i+1} represents the subsequent layer, $\boldsymbol{W}$ represents a weight matrix, b represents

a bias vector of the same dimensionality as l_{i+1}, and σ represents a non-linear activation function.

$$l_{i+1} = \sigma(\boldsymbol{W} l_i + b)$$

A commonly used activation function is the sigmoid function, $\sigma(x) = \frac{1}{1+e^{-x}}$.

Neural networks are a widely used technique in machine learning since fully connected neural networks can approximate any function to any arbitrary level of precision, given enough depth (number of successive layers) and width. This is known as the universal approximation theorem (Kratsios, 2019).

Neural networks are trained by assigning a loss/error value to their outputs, then using an algorithm called backpropagation to adjust the network weights and biases to minimize the loss value. A common loss function that is used for supervised learning (the case where we want to predict a specific output value/label for a given input) is the squared error loss. The equation for this loss function is represented below, where y represents a vector of neural network outputs corresponding to all input data, and t represents the correct/desired label for each input.

$$L = \|y - t\|^2$$

2.3.2 Restricted Boltzmann Machines

A restricted Boltzmann machine (RBM) is a type of generative model – ie. a machine learning model which can extract features from unlabeled data and estimate the underlying probability density of samples (Hwang et al., 2020). RBMs reproduce the underlying sample density of data by using a pair of layers as described above – one is "visible" and one is "hidden" (Hinton, 2012). As seen in *Figure 3*, visible and hidden units are fully connected to each other, but share no intra-layer connections.

The training and use of RBMs was described by Hinton (Hinton, 2012). Essentially, the visible units are set to be the input to the network, and the hidden units are allowed to update via an affine transformation + activation function. The output is the probability at which each hidden unit will be set to 1 (else, it will be set to 0).

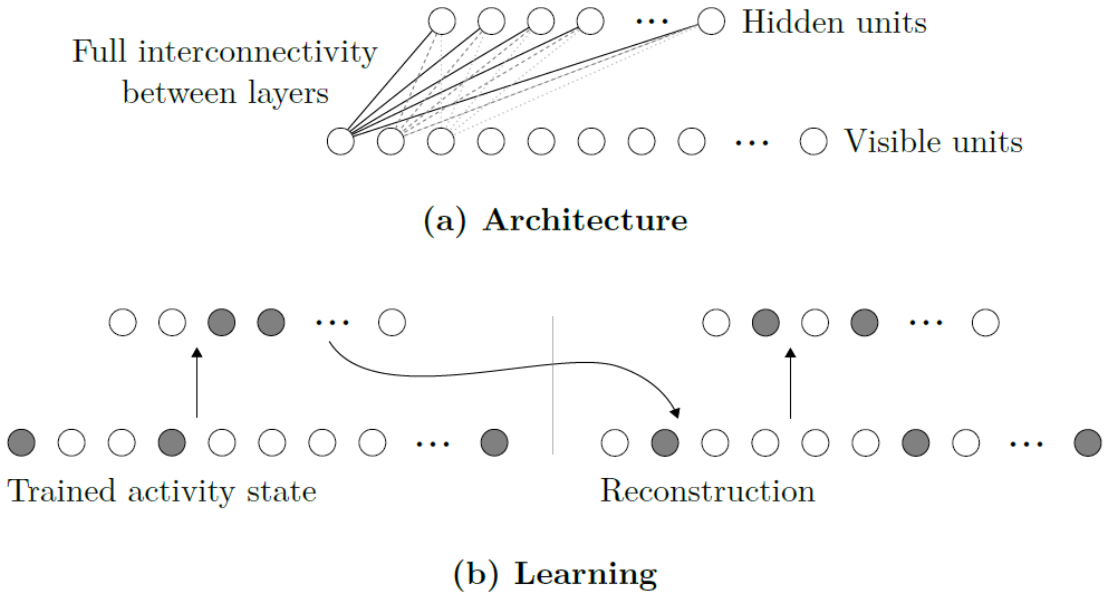

Figure 3: Restricted Boltzmann Machines. *(a) shows the architecture of a RBM. (b) shows how an input vector is clamped onto the visible units, which then update the hidden units. These hidden units then reconstruct the visible units. (Santoro, 2015)*

These probabilities are used to sample the values of each hidden unit. Then, the hidden units are used to reconstruct the visible units using a transposed affine transformation + activation function.

$$P\{h_i = 1|v\} = \sigma(\boldsymbol{W}v + b_h)$$

$$P\{v_j = 1|h\} = \sigma(\boldsymbol{W}^T h + b_v)$$

The equations above describe those two steps process required to calculate the hidden unit. h_i denotes each unit in the hidden layer (while h denotes the hidden layer itself), v_j denotes each unit in the visible layer, (while v denotes the visible layer itself), $\boldsymbol{W}$ is a weight matrix, b_h is a bias vector with the same dimensionality as the hidden layer, b_v is a bias vector with the same dimensionality as the visible layer, and σ is the sigmoid function. Note that the hidden layer values are sampled from the probabilities, and are all either 0 or 1, while the visible layer values can either be sampled or remain as probabilities. In this implementation of the RBM, probabilities were used for the visible layer.

RBMs can be trained with backpropagation using the below loss function, where H is the size of the hidden layer, $\{v\}$ is the entire set of inputs, and $\{v'\}$ is the corresponding set of reconstructed visible units (Hwang et al., 2020).

$$L = \sum_{\{(v,v')\}} \left(b_v\left(v^T - v'^T\right) + \sum_{i=1}^{H} log \frac{1 + e^{\sigma(\boldsymbol{W}v + b_h)_i}}{1 + e^{\sigma(\boldsymbol{W}v' + b_h)_i}} \right)$$

2.3.3 Reinforcement Learning

Of all forms of machine learning, Reinforcement Learning (RL) is the most like the type of learning seen in humans and other animals, so it provides a solid framework within which to explore cognitive mechanisms of memory seen in animals (Sutton et al., 2018).

As seen in *Figure 4*, RL involves training a computerized learner (known as an agent) to take actions and interact with an environment to achieve a certain goal (Sutton et al., 2018). From a mathematical perspective, this can be stated as the problem of learning a mapping between environment states and optimal actions that the agent can take in order to maximize a numerical reward signal that reflects the environment state/progress towards a specified goal (Sutton et al., 2018).

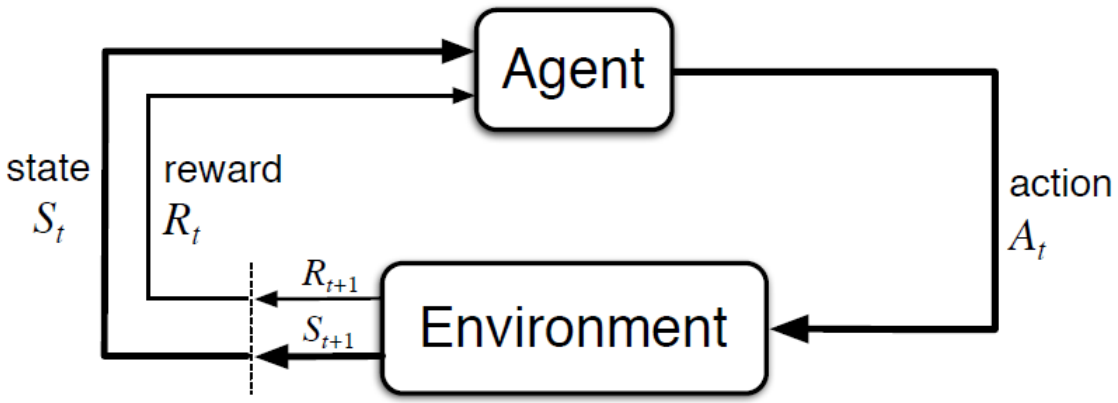

Figure 4: The agent-environment interaction. *Based on a previous/initial state and reward, the agent chooses an action. This action affects the environment, which outputs a new state as well as a reward value in the next step. These are then fed back to the agent. This loop continues until the goal, or another predetermined stop condition, is met. (Sutton et al., 2018)*

Beyond the agent and environment, Sutton & Barto describe the 4 major elements of a RL system as the policy, the reward signal, the value function, and the model (Sutton et al., 2018).

The policy defines the agent's behavior – ie. the (possibly stochastic) mapping between current state and the action which the agent is likely to take. The goal of a reinforcement learning system is to arrive at the optimal policy which maximizes reward.

The reward signal defines the goal of the RL system. The agent can receive a (positive or negative) reward at each time step, depending on whether it is progressing towards/achieving a desired goal or not. The agent's sole objective is to maximize the reward signal, and the policy is tailored for that alone (as such, designing reward signals that result in desired behavior is an important aspect of reinforcement learning research).

The value function is a measure of the long-term reward an agent is likely to receive from a specified environment state. This differs from a reward as the reward determines the immediate desirability of a state, while the value function considers the long-term desirability of a state based on the states and rewards that are likely to follow.

A model is a learned approximation of environment behavior which the agent can use to make inferences about how the environment may behave in the future. Models allow agents to engage in planning (ie. making decisions regarding the current state based on predictions of possible future states). RL methods that rely on models are known as model-based learners, as opposed to model-free learners which learn by trial and error, without explicitly modeling environment behavior.

In the brain, both hippocampus and the prefrontal cortex appear to be involved in model-based (goal-directed) learning and decision making (Sutton et al., 2018). In fact, Doll, Simon, and Daw wrote that "model-based influences appear ubiquitous more or less wherever the brain processes reward information" (Doll et al., 2012).

A major obstacle in RL is that many complex tasks have intractably large state spaces, making it nigh impossible to find an optimal policy or optimal value function, even in the limit of infinite time and data (Sutton et al., 2018). As such, we need to rely on approximations. A popular approximation technique is using neural networks, due to the universal approximation theorem which was mentioned earlier. This leads to the sub-field of Deep RL.

Deep RL is a highly active field of research, with its application to spatial navigation tasks being especially important. Recent results in this field have provided insight into both cognitive function in the brain as well as optimizing computational agent performance (Bermudez-Contreras et al., 2020). For example, researchers have found evidence that computational agents can use grid cell-like activity patterns as an intermediate representation for spatial navigation tasks (Banino et al., 2018), while theories of memory consolidation in the brain have inspired techniques such as experience replay that result in improved reinforcement learning performance (Cazé et al., 2018).

An especially relevant result is that reinforcement learning agents with both episodic and schematic decision-making systems, and the ability to transition between the two, appear to perform better in spatial navigation tasks than reinforcement learning agents with a sole decision-making system (Santoro et al., 2016). As an application of flexible behavior to reinforcement learning, this result provides a basis for the research conducted in this thesis. However, these experiments were designed without allowing for the reinforcement learning agent to gain prior knowledge of its environment. Prior knowledge is an extremely important factor in the cognitive processes behind learning and memory (Brod et al., 2013), so exploring this topic provides an opportunity to gain insight into more realistic cognitive models of flexible behavior where prior knowledge is a factor – just as it is in real organisms.

2.3.4 Computational Representations of Prior Knowledge

In order to design an experiment which considers prior knowledge, a valid computational representation of prior knowledge in RL agents must be implemented. There have been a few earlier implementations of RL agents with prior knowledge. One is directly transferring neural network weights, similar to transfer learning in supervised tasks. A major issue with this approach is that this limits network architectures to be the same as the one which has the prior knowledge. If not, a mapping between the two networks' weights would be needed, and this would be extremely difficult to come up with in most cases (Dixon et al., 2000).

An improvement on this method which allows for flexibility in architecture design would be from directly pretraining the agent on similar environments. However, it would be important to ensure that the environments and tasks were extremely similar, or else the agent could learn an optimal policy for an earlier task which results in unintended behavior in the task of interest.

Imitation learning can also be used to give the agent prior knowledge, based on policies used by an expert mentor – ie. a human and/or heuristic algorithm for the intended task (Osa et al., 2018). The agent can simulate behavior on a dataset of state, action, reward tuples and use that information to start with a good policy. However, this can also be problematic as it limits exploration, possibly resulting in an agent that simply clones the behavior of its mentor, which may or may not be optimal. Furthermore, it may also lead to unpredictable behavior in states that were unseen in the dataset used to train the agent.

A simple implementation of prior knowledge which may work well would be augmenting state inputs for the agent with vector representations of the prior knowledge which the agent is expected to have. This representation works for studying the effect of prior knowledge, but it would not be relevant if the agent's prior knowledge itself is under study.

There is no single accepted computational representation of prior knowledge across reinforcement learning tasks. Different representations have been used to explore different research goals, or for different tasks (Dixon et al., 2000). As such, it will be necessary to test different implementations and decide on the best course of action for this research problem.

3 Experiment Design

A foraging task in a grid environment to run tests on the reinforcement learning agent was created using the SilicoLabs Experimenter tool, and built upon the Unity game engine and the Unity ML-Agents Toolkit (Juliani et al., 2020).

3.1 Environment Design

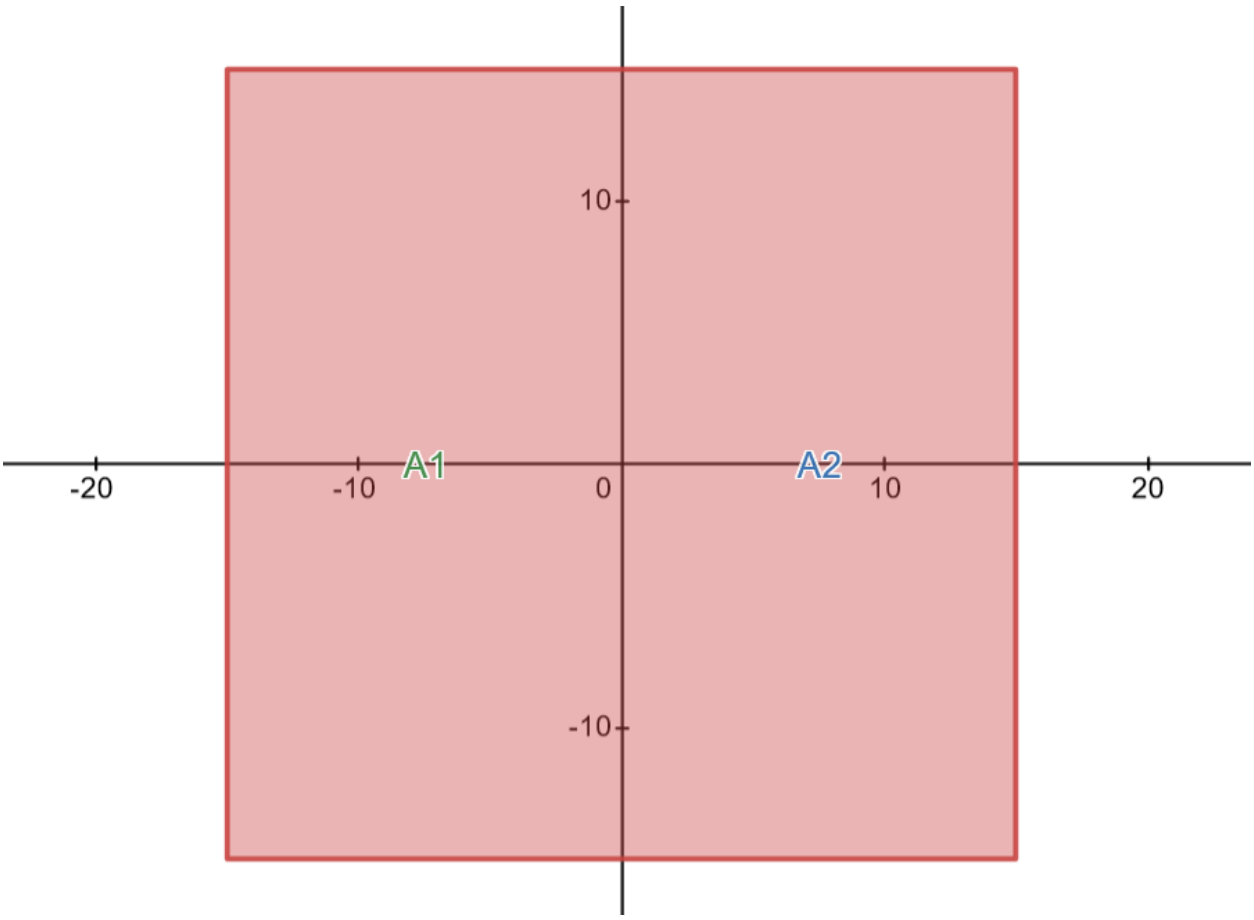

Figure 5: A wireframe of the environment. *A1 and A2 denote the center of the random distributions responsible for spawning the two apples, respectively.*

As shown in Figure 5, the environment consists of a simple square grid 30 units wide, centered at (0, 0). The agent is then randomly spawned according to a uniform random distribution covering the entire area of the grid. After spawning, the agent is able to take 4 actions at any given time: it can move 1 unit up, down, left, or right.

Two apples are randomly spawned, one using a normal distribution centered at (-7.5, 0) and the other using a normal distribution centered at (7.5, 0). These locations were chosen such that, when the environment is vertically divided into two halves, the apple closest to the agent is always expected to be the reward currently in the same half as the agent.

When running experiments, the agent is only provided with state information consisting of its two-dimensional position within the grid. When provided to the agent, the position coordinates were normalized between 0 and 1 as follows: $\left(\frac{x+15}{30}, \frac{y+15}{30}\right)$.

Furthermore, finding either apple corresponded to giving the agent a reward valued at 1, and ended the episode. Additionally, each step corresponded to a -0.001 reward, in order to encourage the agent to end the episode as soon as it could.

3.2 Prior Knowledge Implementation

Since the focus of this research is to study the effect of prior knowledge, and not the implementation itself, the goal with this prior knowledge implementation was simplicity. Furthermore, it had to give the agent some useful information that would help maximize its reward. However, to ensure fairness between the two conditions (with and without prior knowledge) the given prior knowledge had to represent information that the agent could learn on its own, even without it directly being provided.

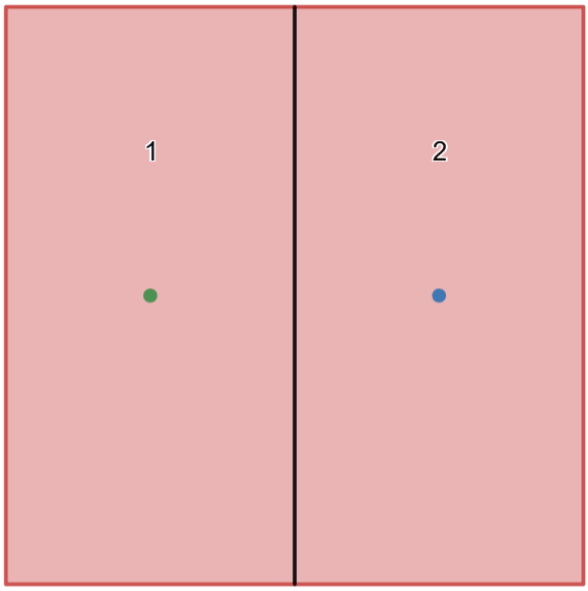

Figure 6: Visualizing the two "zones" in the environment.

Based on these considerations, prior knowledge was implemented by vertically dividing the environment into two halves or “zones” and providing the agent with knowledge regarding which half of the environment it currently occupies, as seen in Figure 6.

This implementation provides the agent the opportunity to learn to use this knowledge to target the closer reward, based on the zone it currently occupies. While this information is useful, the agent without prior knowledge is not at an unfair disadvantage, as it can also learn to consider the two halves of the environment separately based on the location information it is provided. This is because the prior knowledge information is solely based on the x-coordinate of the agent’s current location, which is part of the regular state information it gets in both conditions.

To provide the agent with the prior knowledge information, the state vector was augmented with a component corresponding to the labeled zone the agent currently is in. In the condition where the agent was not provided with prior knowledge, this extra component was kept, but always set to 0.

3.3 Tests Conducted

In total, 6 tests were conducted using this environment, as 3 different agent variants were used across 2 conditions. The 3 agents were as follows: a solely episodic decision-making agent, a solely schematic decision-making agent, and a memory consolidation agent which used both

decision making systems. The 2 conditions were as follows: a condition where the agents were provided prior knowledge, and a condition where the agents were not provided with any prior knowledge.

A complete test consisted of 5 trials 60 episodes long, where each episode ended when the agent found the reward.

4 Agent Architecture

To test the desired hypotheses, reinforcement learning agents were implemented based on work described by Santoro (Santoro et al., 2016). The four major components were models of place cells, episodic memory, schematic memory, and navigation, which were then used in conjunction for the different agents. All machine learning models were built using the open-source Pytorch framework (Paszke et al., 2019). Code snippets of the implementations for all components of the agent are included in Appendix A.

4.1 Place Cells Model

To better model biological systems, input state information (spatial location and zone information) was transformed into a representation of place cell activity at that location. As in Santoro's model, 980 place cells were used (Santoro et al., 2016), with the following formula representing the activity of the i-th place cell, m_i.

$$m_i(x) = e^{\frac{-\|x-s_i\|^2}{2\sigma_m^2}}$$

x is the three-dimensional state vector, s_i is a three-dimensional vector where all three values are randomly sampled from the uniform distribution with bounds $[0, 1]$, and the constant $\sigma_m = 0.16$ represents the breadth of the place field.

4.2 Episodic Memory Model

4.2.1 Episodic Network

The episodic network was designed to model hippocampus function and is illustrated in Figure 7. Like the hippocampus, it receives a spatial input. Instead of grid cells from the entorhinal cortex, it receives the current state information (including agent position and zone information) as a 3-dimensional vector.

A fully connected layer representing the dentate gyrus transforms the initial spatial input into a higher-dimensional representation.

Next, the CA3 was implemented as an auto-encoder with 3 fully connected layers, in order to represent its autoassociative nature. To represent the recurrent nature of the network, weight sharing was used as well – ie. each layer of the CA3 used the same weights instead of different

weights as seen in conventional autoencoder architectures. The input to the CA3 was the output from the dentate gyrus layer.

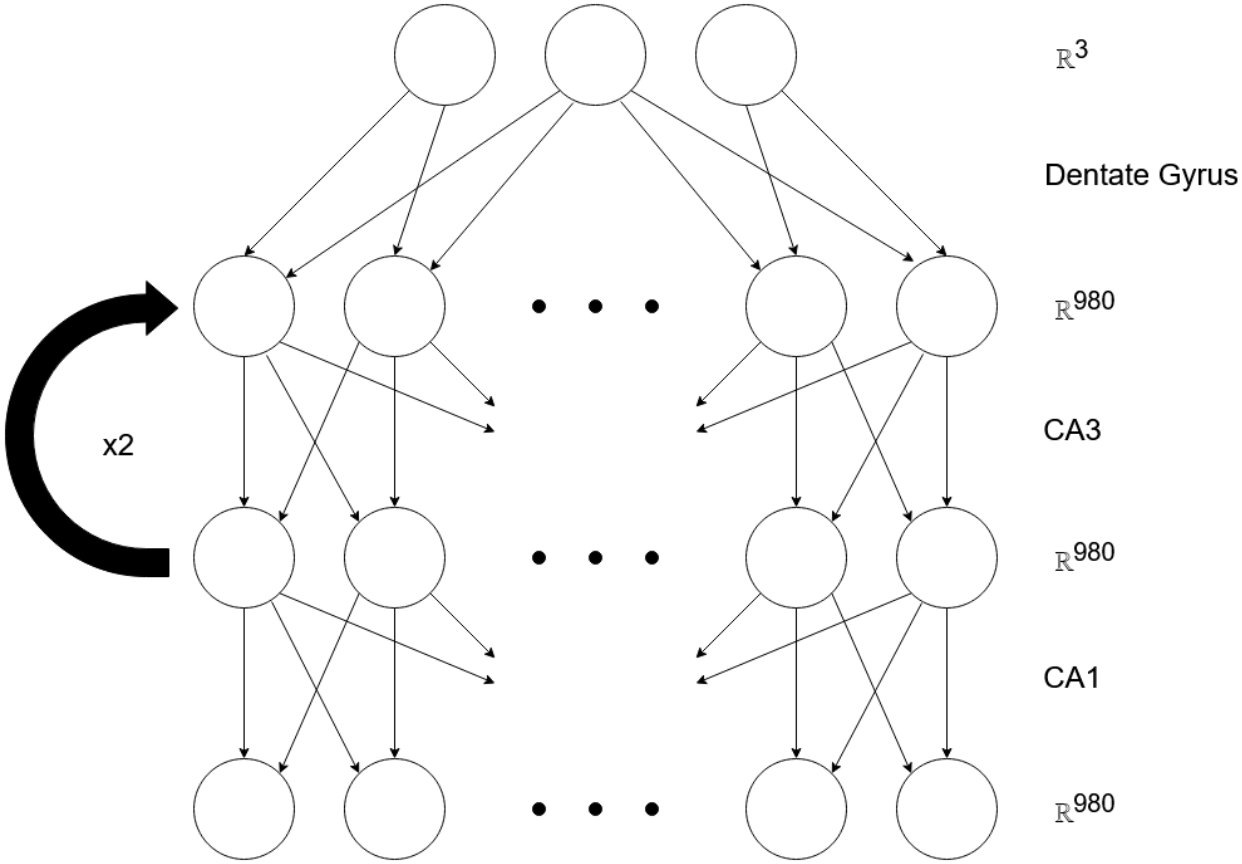

Figure 7: Architecture of the episodic network.

The encoded representation output from the CA3 autoencoder was then used as input to a fully connected layer representing the CA1. This output was meant to represent a retrieved memory of place cell activity.

As such, the final output of the episodic network is a retrieved memory of place cell activity at the location where the episodic network thinks the reward is, given the current state (ie. an episodic memory goal). Each of the layers in the episodic network used a sigmoid activation function.

The components of the episodic network were trained separately, at the end of each timestep. The CA3 was trained to reconstruct its input, so the loss that was used for backpropagation was $L = \|e_{DG} - e_{CA3}\|$, where e_{DG} was the output of the dentate gyrus layer (input to the CA3 layer)

and e_{CA3} was the output of the CA3. This was used to update the weights of the CA3 and dentate gyrus layers.

The CA1 was trained to reconstruct place cell activity from the CA3 representation. As such, it was trained to match the current state's place cell activity, with the loss function $L = \|m(x) - g_E\|$, where $m(x)$ was the place cell activity and g_E was the output of the episodic network. This was used to update the weights of the CA1 layer.

To modulate training such that memories of reward locations were stored more strongly, the learning rate used was dependent on the value of the current state. Specifically, $\alpha_E = 0.1V(m(x))$. Furthermore, when the episode ended (ie. an apple was found), the learning process was repeated 50 times to encourage the episodic network to output the location of the last found reward.

4.2.2 Critic Network

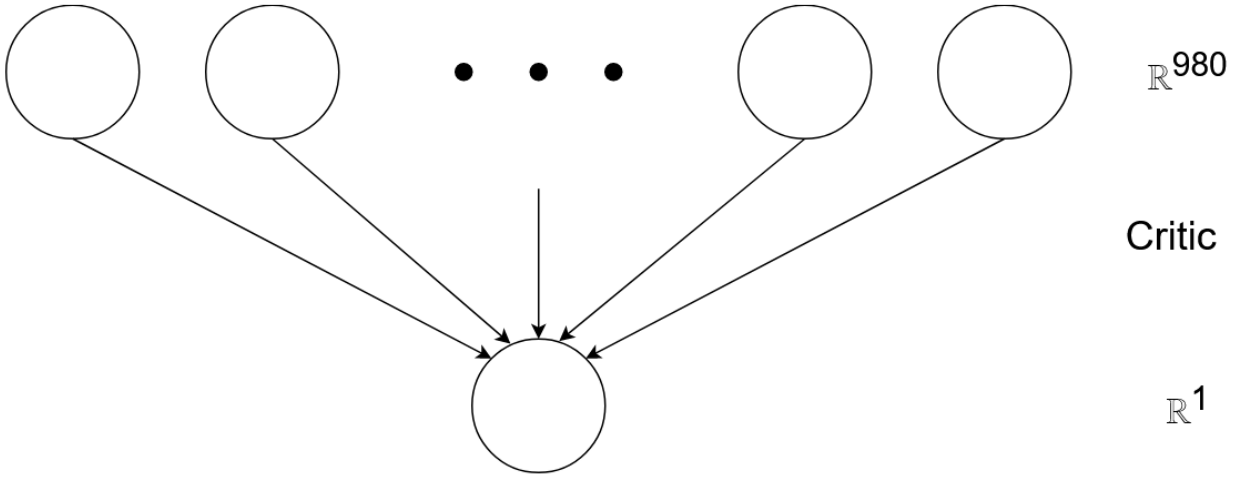

Figure 8: Architecture of the critic network.

In order to estimate the value of each state, a critic network was needed as part of the episodic memory model. It took the place cell representation of the current state as input and computed the value of the current state. A tanh activation function was used to ensure that the estimated value was bounded between -1 and 1.

The critic network was trained via a temporal difference algorithm. The prediction error δ_t was computed at each timestep with the following formula, and backpropagation was used to minimize it at each timestep.

$$\delta_{t-1} = R_{t-1} + \gamma V(m(x_t)) - V(m(x_{t-1}))$$

V is the critic function, x_{t-1} is the last state, x_t is the current state, the discount rate $\gamma = 0.95$, and R_{t-1} is the reward at the last timestep. For the final state of an episode, $\delta_t = R_t - V(m(x_t))$.

Learning was modulated so that "unexpected" states (with a larger temporal difference error) were more strongly remembered. To do this, the learning rate was dependent on the temporal difference error, with $\alpha_C = 0.04\delta_t$.

4.3 Schematic Memory Model

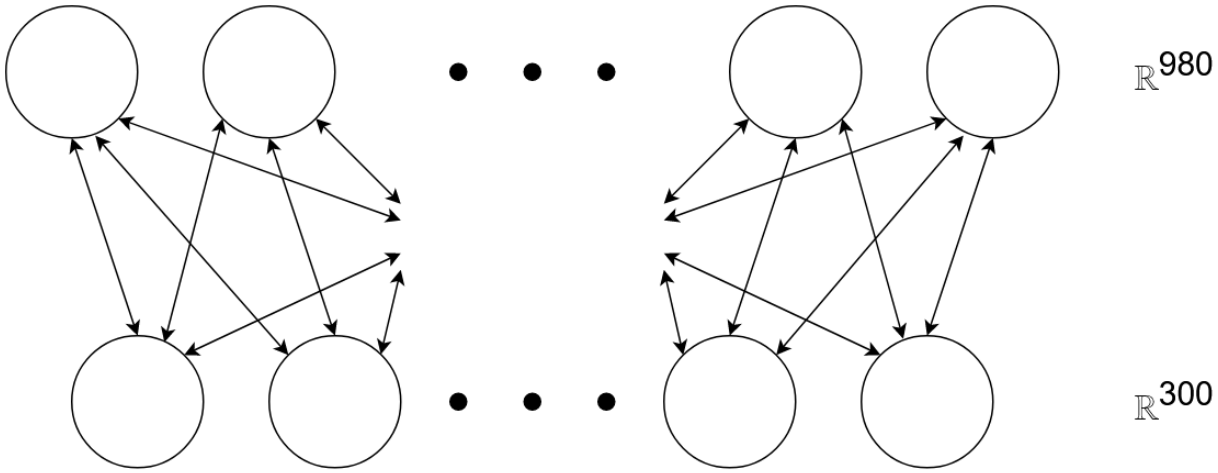

Figure 9: Architecture of the schematic network.

The schematic network was designed to learn a general statistical model of past reward locations. Towards this end, it was implemented as a two-layer RBM that was trained to reconstruct its input, as shown in Figure 9. This allows it to learn a general probability distribution over multiple previously experienced reward locations. In addition to achieving the required purpose, this design meets all requirements of a schema as described earlier.

The overall output of the schematic network was a retrieved memory of place cell activity at the location where the episodic network thinks the reward is, given the current state (ie. a schematic memory goal).

The schematic network was trained at the end of each episode, after the episodic network was trained. 200 random states were input to the episodic network, and then the schematic network

was trained to reconstruct the output episodic memory goals. Repeating this process after each reward is found allows the schematic network to build a model of the probability distribution of reward locations. This training method resembles the biological process of episodic replay (Santoro et al., 2016).

4.4 Navigation Model

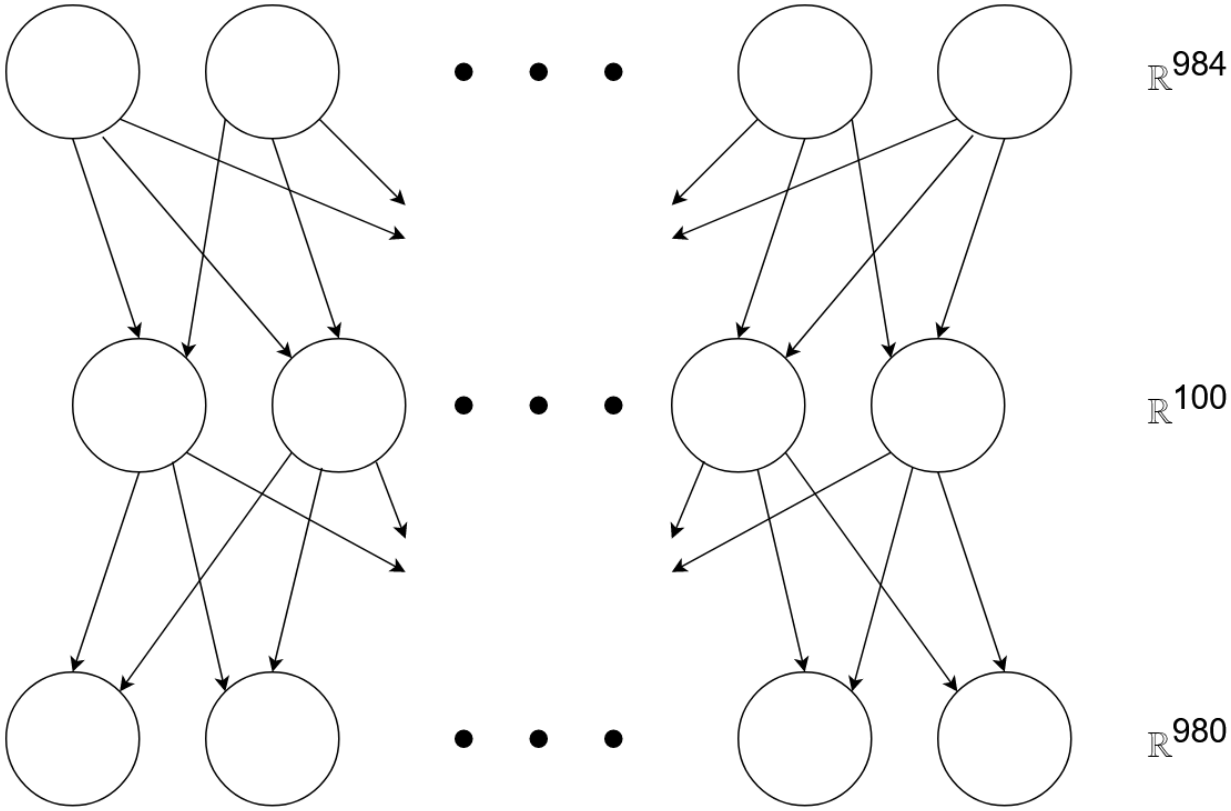

Figure 10: Architecture of the navigation network.

The navigation network was used to help the agent choose an action (ie. a movement direction) based on an input memory goal. The total input to the navigation network was a concatenation of the place cell activity for the current state as well as a one-hot encoding of a possible action choice (of which there are 4). This input was passed through a two-layer fully connected neural network that predicts the place cell activity at the next state if the given action is taken, as shown in Figure 10.

To choose an action, the current state's place cell activity is sequentially concatenated with all possible action choices, and all outputs are collected. The predicted next state's place cell

activity is then compared with the memory goal's place cell activity to create a similarity metric based on the below equation, where d_i is the difference for action i, g_M is the memory goal, and p_i is the predicted next state for action i.

$$d_i = \|g_M - p_i\|$$

Then, the chosen action a is sampled from a probability distribution created such that actions with a lower difference are more likely to be selected. Specifically, the probability of selecting an action is inversely proportional to the difference metric for that action.

$$\Pr\{a = a_i\} \propto \frac{1}{d_i}$$

The navigation network was trained with the goal that it would predict the next state correctly. As such, it was trained after each timestep, with the loss function designed to minimize the difference between the place cell activity of the next state as compared to the place cell activity predicted by the navigation network for the chosen action.

$$L = \|p_t - p_a\|^2$$

4.5 Agent Design

The four above components were all combined to build the agent, as shown in Figure 11.

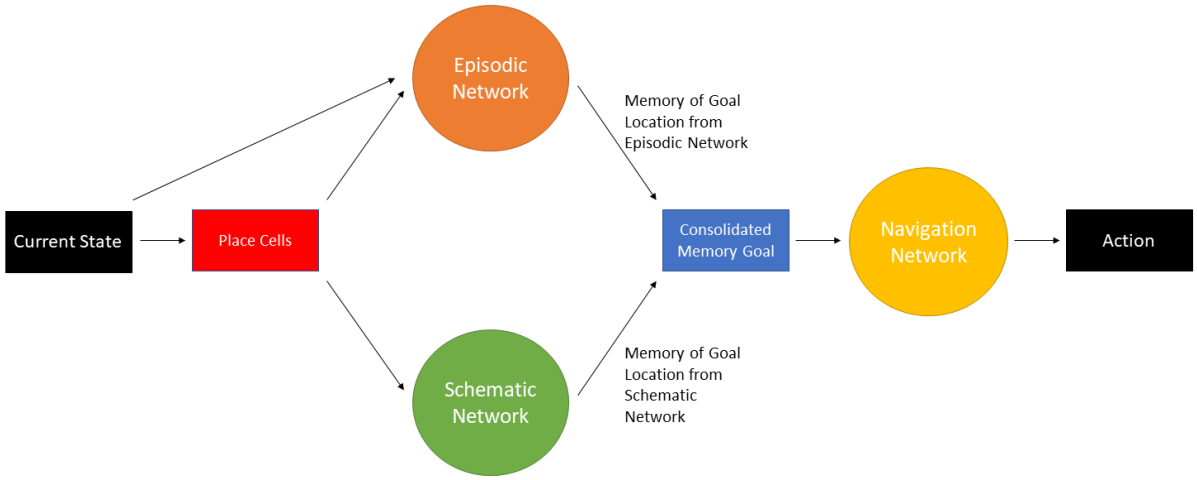

Figure 11: Architecture of the agent as a whole.

The current state and its place cell representation were the input for the episodic and schematic networks, which output their respective memory goals. These two goals were combined into a consolidated memory goal, which was input to the navigation network to choose the next action to take.

To allow for different weightings for schematic and episodic memory goals, the consolidated memory goal that was passed to the navigation network was based on a convex combination of both.

$$g_M = \varphi g_E + (1 - \varphi) g_S$$

The schematic decision-making agent used $\varphi = 0$ so the consolidated goal solely matched the schematic goal, and the episodic decision-making agent used $\varphi = 1$ for the opposite effect.

In order to gradually transition from episodic to schematic decision-making, the memory consolidation agent used a variable φ that was updated at each timestep of an episode according to the following rule (Santoro, 2016).

$$\varphi = e^{-\frac{t}{\beta}}$$

The constant β was set to 2000. This results in an agent that relies more heavily on its episodic memory to make decisions for the first 1000 steps, before schematic decision making starts to take over.

5 Results & Discussion

The following results were based on running the tests shown in Table 1, with performance being measured based on reward rate (ie. the cumulative reward divided by the total number of steps taken so far). As 5 trials were performed as part of each test, the results were averaged to reduce noise in the data. All plots analyzed in this section use episode number on the x-axis; however, analogous plots with number of steps on the x-axis are included in Appendix B.

5.1 Effect of Prior Knowledge on Same Agent Performance

The first aim of the research was to study the effect of prior knowledge on each agent individually, to test whether it improves learning. Towards this end, the reward rates of agents on both conditions were plotted and compared.

5.1.1 Effect of Prior Knowledge on Episodic Decision-Making Agent

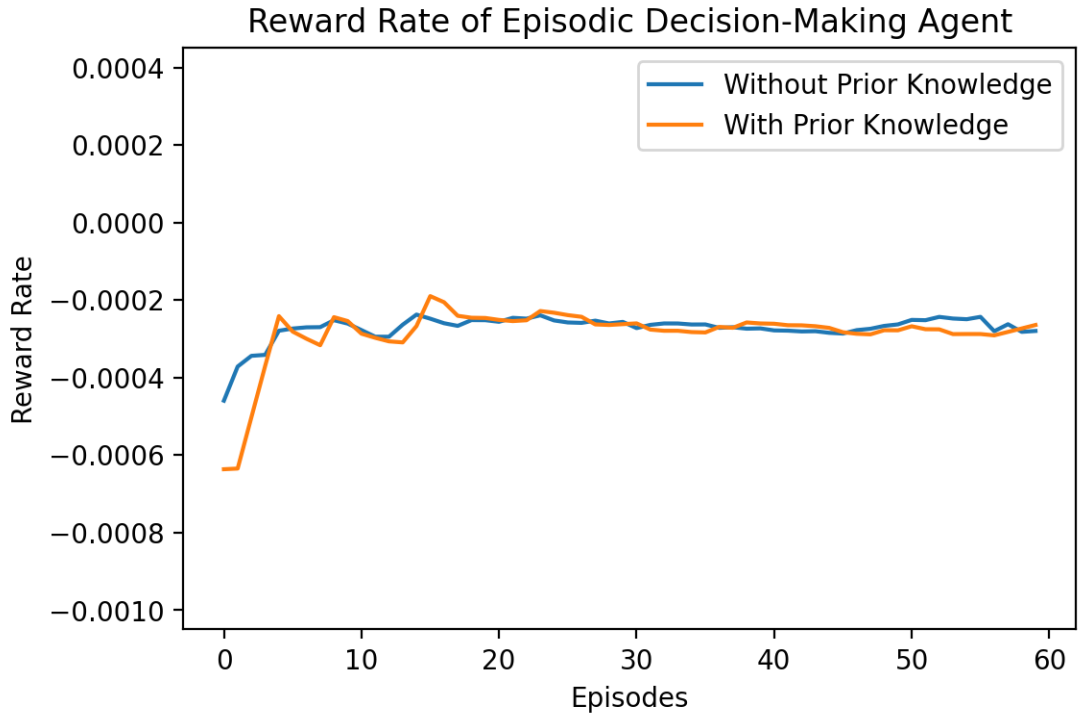

Figure 12: Reward rate vs episodes for episodic decision-making agent.

For the solely episodic decision-making agent, prior knowledge does not affect the performance too much. Both agents converge to a similar level of performance, as visualized in Figure 12. Specifically, the agent without prior knowledge is bounded by a reward rate of -0.00028, while

the agent with prior knowledge is bounded by a reward rate of -0.00026. This represents an improvement of 0.00002 in the reward rate, which is small enough to be negligible.

While the hypothesis was that prior knowledge would result in a performance improvement for all agents, the results seen in Figure 12 make sense. The lack of improvement seen is likely because the episodic network is designed to remember only the last reward location, so it guides the agent to navigate to the last reward location, even if the agent spawns in another zone with a closer reward. To make better use of prior knowledge information, an episodic memory architecture which could store two memories to remember the last reward locations for both zones would be needed.

5.1.2 Effect of Prior Knowledge on Schematic Decision-Making Agent

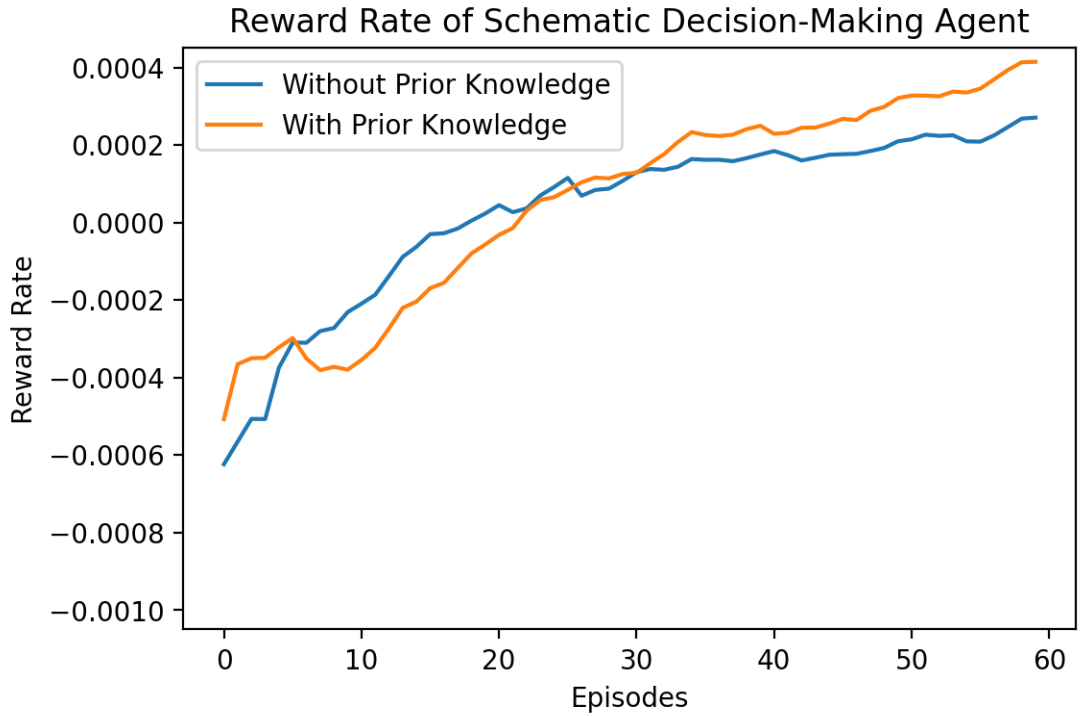

Figure 13: Reward rate vs episodes for schematic decision-making agent.

Unlike the episodic agent, the schematic decision-making agent has a significant final performance improvement when prior knowledge is provided, as seen in Figure 13. This result supports the initial hypothesis that prior knowledge would help improve agent performance. The agent without prior knowledge is bounded by a reward rate of 0.00027, while the agent with

prior knowledge is bounded by a reward rate of 0.00041. This represents an improvement of 0.00014 in the reward rate, which at this scale, is quite a major improvement.

However, it is also noticeable that the schematic agent performs noticeably worse for the first 20 episodes when prior knowledge is provided. Since prior knowledge adds complexity to the state representation, this likely makes it more difficult for the schematic network as it needs to learn a different statistical distribution for each zone instead of one for the entire environment. This could lead to the schematic network needing more time to create an adequate statistical model.

5.1.2 Effect of Prior Knowledge on Memory Consolidation Agent

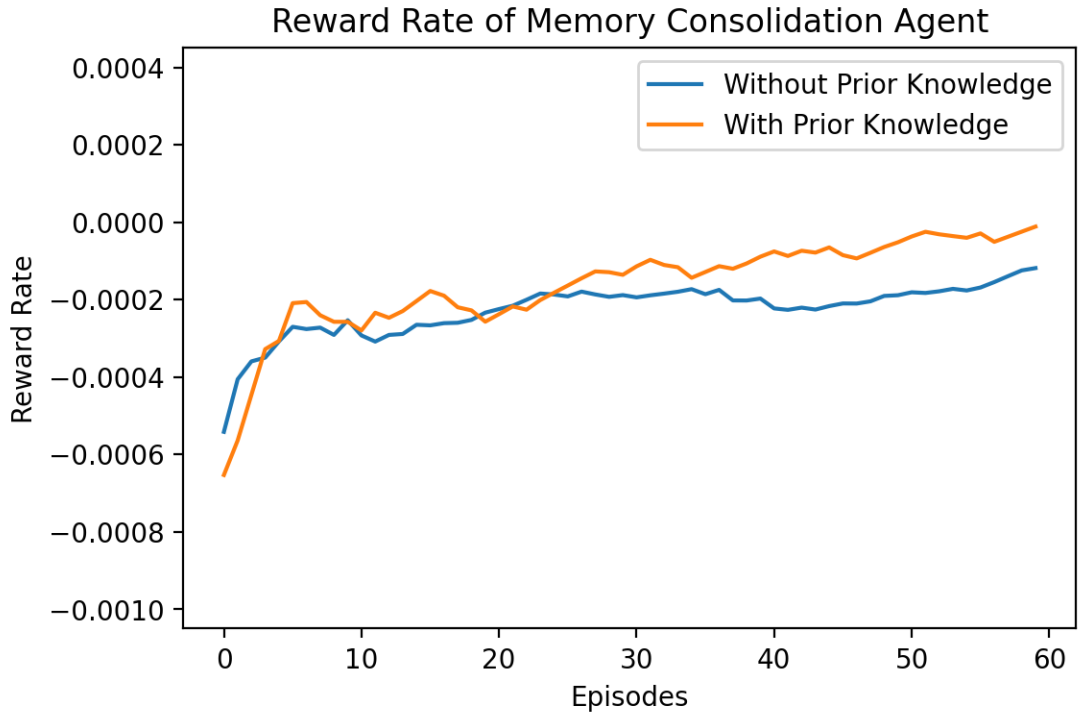

Figure 14: Reward rate vs episodes for memory consolidation agent.

As seen in Figure 14, there is a noticeable performance improvement for the memory consolidation agent as well. The agent without prior knowledge is bounded by a reward rate of -0.00012, while the agent with prior knowledge is bounded by a reward rate of -0.00001. This represents an improvement of 0.00011 in the reward rate, which is certainly discernable, but not as significant as the one that existed for the schematic agent..

This is likely because the consolidation agent is limited by the fact that it heavily relies on episodic decision making at first, until the weightage of both memory goals becomes near equal after around 1000 steps. Since the episodic network does not take advantage of prior knowledge as much, this somewhat limits the performance improvement of the consolidation agent, as its performance begins to converge at a reward rate near 0, which corresponds to an average of 1000 steps taken per episode.

5.2 Effect of Prior Knowledge Across Different Agents

The other aim of the research was to study the relative impact of prior knowledge on the performance of different memory systems at different points during the training process.

5.2.1 Effect of Prior Knowledge at Performance Convergence

As seen in section 5.1 and visualized in Figure 15, prior knowledge had the greatest effect on schematic knowledge at convergence.

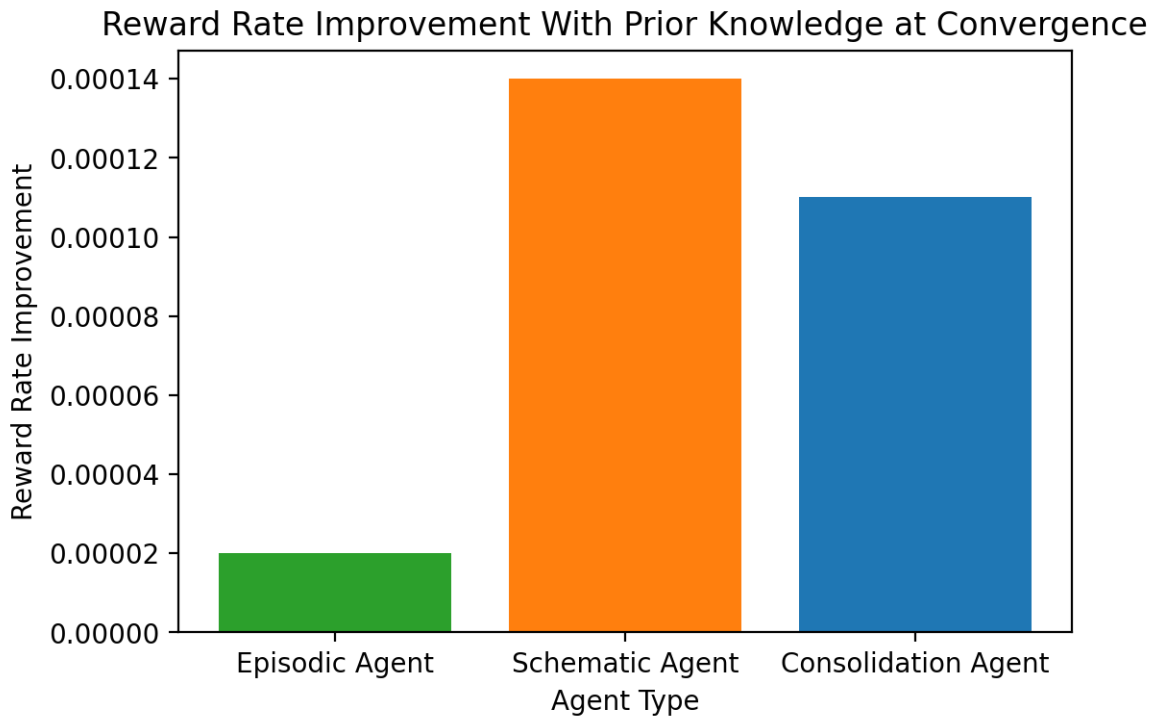

Figure 15: Comparison between the performance improvements exhibited by different agent types when prior knowledge was provided.

The improvement shown by the schematic decision-making agent was 7 times greater relatively compared to the improvement exhibited by the episodic decision-making agent, and the improvement exhibited by the memory consolidation agent was 5.5 times greater. These figures are consistent with the hypothesis that the relative importance of episodic memory to decision-making decreases as prior knowledge is provided.

5.2.2 Effect of Prior Knowledge During Training

When prior knowledge is not provided to the agents, episodic decision-making initially proves more useful, before schematic decision-making overtakes it later into training. These results are seen in Figure 16, and are consistent with the results of Santoro's experiment, where prior knowledge also was not provided (Santoro et al., 2016).

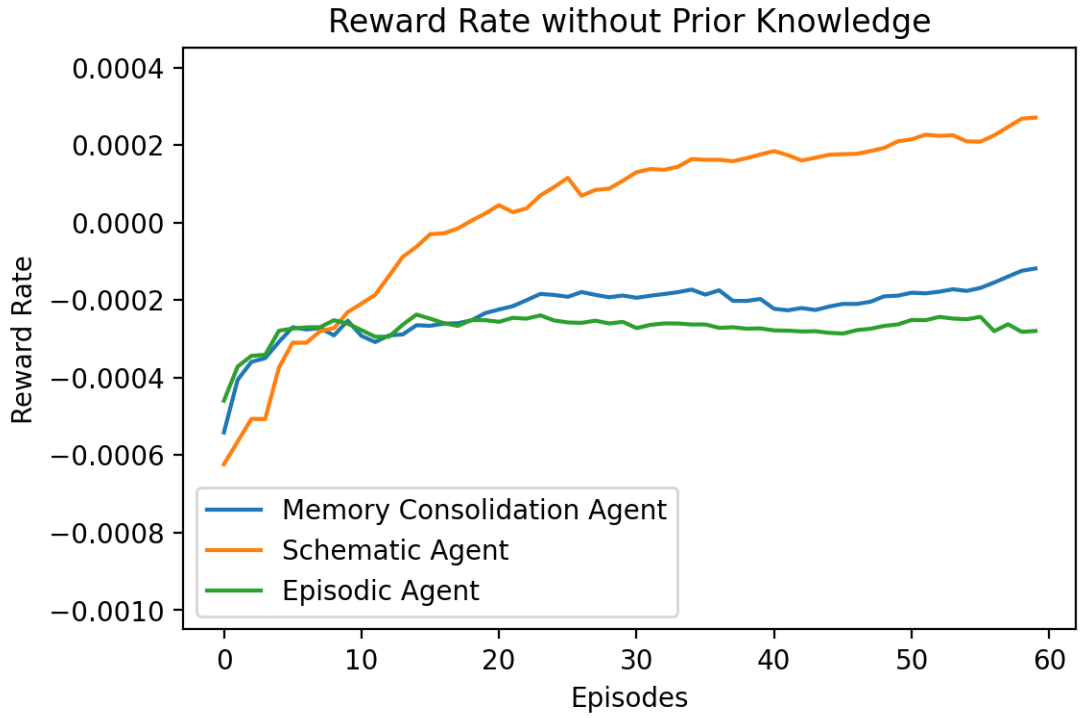

Figure 16: Reward rates of all agents when prior knowledge was not provided.

It is also interesting to note that the memory consolidation agent performs similarly to the episodic agent early in the training process, but overtakes it later, just after the schematic agent does so. This suggests that, to some extent, it succeeds in utilizing the best qualities from both decision-making systems.

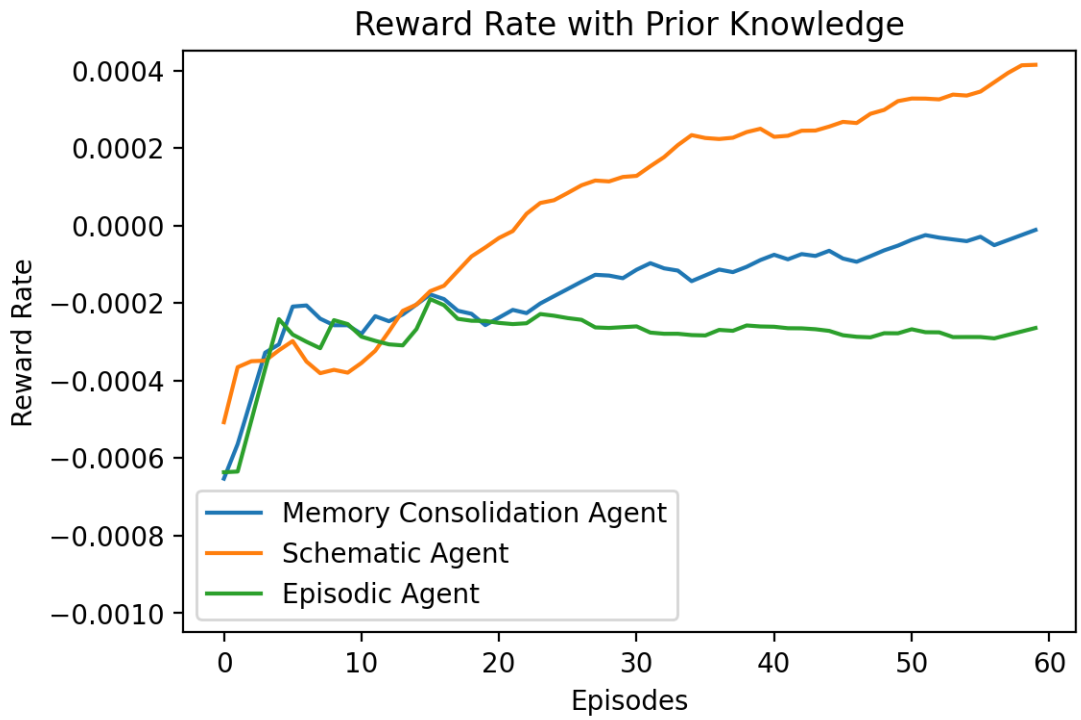

Figure 17: Reward rates of all agents when prior knowledge was provided.

The hypothesis was that this relationship would stay true when prior knowledge was provided, but that schematic decision-making would more quickly overtake episodic decision-making in that case. However, as seen in Figure 17, while the overall relationship stays true, it takes longer for schematic decision-making to become advantageous when prior knowledge is provided.

As discussed in section 5.1.2, this delay may be because the schematic network needs to learn two separate statistical distributions when prior knowledge is provided in this case, so it requires more episodes (data points) to do this satisfactorily.

The observation that the memory consolidation agent can utilize the best aspects of both decision-making networks to some extent remains true. In order to improve the final converged performance of the memory consolidation agent to be closer to that of the schematic agent, it may have been helpful to reduce the constant β so that φ would decrease and switch over to schematic decision-making earlier in an episode. However, it remains to be seen whether this would reduce the performance of the memory consolidation agent in the early episodes where schematic decision-making was not as effective.

6 Conclusions & Future Directions

6.1 Summary

Prior research has studied the importance of flexible behavior through machine learning models. In particular, it has been demonstrated that episodic and schematic memories have unique yet complementary advantages for guiding behavior in the context of a *tabula rasa* agent, showing a possible reason for the evolution of memory consolidation in the mammalian brain (Santoro et al., 2016).

This research builds on those results by examining the effect and importance of prior knowledge on flexible behavior in such a framework. Specifically, the aims were to test whether prior knowledge would improve the performance of both episodic and schematic decision-making systems, and whether prior knowledge would change the relative importance of episodic and schematic decision-making systems. The initial hypotheses were that prior knowledge would improve agent performance, and that it would increase the relative importance of schematic memory for decision making. In order to ensure that results of this research were consistent with the prior work, both the agent architecture and task design were based on the experiments run by Santoro et al.

The results of this research mostly support the initial hypotheses made. The presence of prior knowledge appears to improve agent performance with the sole exception of the episodic decision-making agent. This exception is likely by design, as the episodic decision-making agent is created to follow the information it learned from the last episode, disregarding the information given to it. As such, prior knowledge does not make any noticeable impact on the performance of the episodic decision-making agent. However, the other two agent types (schematic decision-making agent and memory consolidation agent) showed significant improvements when prior knowledge was provided, as expected.

Furthermore, the relative importance of schematic memory increased when prior knowledge was provided, as initially hypothesized. Other work in the field showed that schematic decision-making outperformed episodic decision-making when the agent had more experience in its environment (Santoro et al., 2016). As such, it was a logical extension that directly provided prior knowledge would have a similar effect, and this is supported by the results of this research.

These results offer biological insight regarding the interplay of episodic and schematic decision-making systems in the brain in real world scenarios. Specifically, these results in computational models suggest that cortical systems play a large role in guiding behavior during a task for which a large amount of prior knowledge already exists. On the other hand, they suggest that hippocampal systems would play a larger role when guiding behavior while learning how to complete an unfamiliar task. While this specific prediction remains untested, prior research supporting this has shown that hippocampal firing in rats occurs when faced with the task of navigating through an unfamiliar maze (Yu & Frank, 2015). This suggests that predictions made with this computational model may indeed be biologically relevant and useful for guiding future research with biological models.

6.2 Future Work

These results offer plenty of opportunities for further research that builds upon what has already been explored.

To start, further experiments could be carried out within the current framework. For example, further experimentation with the memory consolidation agent could be carried out. Currently, the memory consolidation agent was based solely on the implementation by Santoro. However, section 5.2.2 mentioned that a different update schedule for $\varphi(t)$ could have improved the final performance at convergence for the memory consolidation agent. As such, further research could be done to find an optimal $\varphi(t)$ that maximizes the reward rate which the agent converges to. This could provide biological insight regarding the optimal decision-making strategy for organisms foraging for food.

Experiments aiming to find the limits of the observations in this work could also be carried out. For example, is there a limit to the amount of prior knowledge provided to the agent before improvements to agent performance cease, or schematic memory stops becoming relatively more important than episodic memory for decision making? Answering this question could provide insights regarding the limitations of prior knowledge in biological systems.

Additionally, the effect of anomalies or misleading knowledge could provide insight. What happens if the prior knowledge provided to the agent is incorrect? Is the effect greater on the episodic or schematic agent? This could be implemented by simply switching the zone labels or

providing the agent with random zone labels for an episode. This experiment would model the effect of the biologically plausible situation of outdated or misleading knowledge on a computational agent.

Another avenue for further research is experimenting with the agent architecture. A possible improvement that could be made is pre-training the navigation network. Since the navigation network is trained to predict the next state given the current state and a chosen action, it would be simple to programmatically generate a large amount of data which could be used to pre-train the network. Then, using the same pre-trained navigation network weights for all agents would reduce any variance in performance, and isolate the experiment further to solely depending on the episodic and schematic memory systems.

A more major change would be to experiment with different models for episodic and schematic memory systems, instead of relying solely on the Santoro models. A wealth of literature exists on modeling memory systems in reinforcement learning (Ramani, 2019), so it is possible that a combination of different models may result in even more biologically applicable results.

Future research could also build upon these results by increasing the complexity of the task. Currently, the experiment involved a foraging task in a simple grid world with 2 zones and 2 corresponding rewards. A simple method to experiment with increased complexity would be to increase the number of zones and rewards in the environment. Then, tests could be run to observe the effect on performance with and without prior knowledge.

Presently, spatial coordinates and zone information were directly provided to the agent. Instead of directly providing the spatial coordinates as state information, a method of increasing the complexity of the task could involve using a 3D environment and providing visual observations instead. Visual observations could be transformed into a place cell representation using a combination of convolutional neural network architectures and mathematical functions like the method used in this paper. The prior knowledge information could also be combined into the visual representation by using a different appearance of the environment in each zone.

Future research could also focus on testing to see whether the observations in computational models hold true in humans. The SilicoLabs Experimenter tool makes it trivial to run the same experiment with a human game player instead of computerized agent. However, to ensure the

task remains non-trivial, a few changes could be made. The environment size could be significantly increased, and a first-person view could be used, so the player doesn't automatically get full knowledge of the environment. Instead of being a blank grid, the environment would have to be styled with visual landmarks. A similar form of prior knowledge could be provided by using a different environment appearance for each zone. Then, the results obtained from human participants could be compared with the results seen here on computational models, providing an indication of the biological accuracy of these observations.

Finally, direct biological research on animal models could be carried out to examine the relative importance of cortical and hippocampal systems in guiding behavior for familiar tasks as compared to new experiences. As mentioned earlier, these computational models suggest that cortical systems would play a larger role for familiar tasks, while hippocampal systems would be more heavily involved in guiding behavior when a new task is being learned. If the results of these biological experiments are consistent with the observations in computational models, this would provide the best indication of the biological relevance of these models.

7 References

Badreddine, S., & Spranger, M. (2019). Injecting Prior Knowledge for Transfer Learning into Reinforcement Learning Algorithms using Logic Tensor Networks. *CoRR, abs/1906.06576.*

Banino, A., Barry, C., Uria, B., Blundell, C., Lillicrap, T., Mirowski, P., Pritzel, A., Chadwick, M. J., Degris, T., Modayil, J., Wayne, G., Soyer, H., Viola, F., Zhang, B., Goroshin, R., Rabinowitz, N., Pascanu, R., Beattie, C., Petersen, S., … Kumaran, D. (2018). Vector-based navigation using grid-like representations in artificial agents. *Nature*, *557*(7705), 429–433. https://doi.org/10.1038/s41586-018-0102-6

Bermudez-Contreras, E., Clark, B. J., & Wilber, A. (2020). The neuroscience of spatial navigation and the relationship to Artificial Intelligence. *Frontiers in Computational Neuroscience*, *14*. https://doi.org/10.3389/fncom.2020.00063

Brod, G., Werkle-Bergner, M., & Shing, Y. L. (2013). The influence of prior knowledge on memory: A developmental cognitive neuroscience perspective. *Frontiers in Behavioral Neuroscience, 7*. https://doi.org/10.3389/fnbeh.2013.00139

Burgess, N., Maguire, E. A., & O'Keefe, J. (2002). The human hippocampus and spatial and episodic memory. *Neuron*, *35*(4), 625–641. https://doi.org/10.1016/s0896-6273(02)00830-9

Buzsáki, G., & Moser, E. I. (2013). Memory, navigation and Theta rhythm in the hippocampal-entorhinal system. *Nature Neuroscience*, *16*(2), 130–138. https://doi.org/10.1038/nn.3304

Cazé, R., Khamassi, M., Aubin, L., & Girard, B. (2018). Hippocampal replays under the scrutiny of Reinforcement Learning Models. *Journal of Neurophysiology*, *120*(6), 2877–2896. https://doi.org/10.1152/jn.00145.2018

Dixon, K.R., Malak, R.J., & Khosla, P.K. (2000). Incorporating Prior Knowledge and Previously Learned Information into Reinforcement Learning Agents.

Doll, B. B., Shohamy, D., & Daw, N. D. (2015). Multiple memory systems as substrates for multiple decision systems. *Neurobiology of Learning and Memory*, *117*, 4–13. https://doi.org/10.1016/j.nlm.2014.04.014

Doll, B. B., Simon, D. A., & Daw, N. D. (2012). The ubiquity of model-based reinforcement learning. *Current Opinion in Neurobiology*, *22*(6), 1075–1081. https://doi.org/10.1016/j.conb.2012.08.003

Dubey, R., Agrawal, P., Pathak, D., Griffiths, T., & Efros, A. (2018). Investigating Human Priors for Playing Video Games. *ICML*.

Duff, M. C., Covington, N. V., Hilverman, C., & Cohen, N. J. (2020). Semantic memory and the hippocampus: Revisiting, reaffirming, and extending the reach of their critical relationship. *Frontiers in Human Neuroscience*, *13*. https://doi.org/10.3389/fnhum.2019.00471

Eichenbaum, H. (2004). Hippocampus: Cognitive Processes and Neural Representations that Underlie Declarative Memory. *Neuron*, *44*(1), 109–120. https://doi.org/10.1016/j.neuron.2004.08.028

Eichenbaum, H. (2017). Prefrontal–hippocampal interactions in episodic memory. *Nature Reviews Neuroscience*, *18*(9), 547–558. https://doi.org/10.1038/nrn.2017.74

Ghosh, V. E., & Gilboa, A. (2014). What is a memory schema? A historical perspective on current neuroscience literature. *Neuropsychologia*, *53*, 104–114. https://doi.org/10.1016/j.neuropsychologia.2013.11.010

Hinton, G. E. (2012). A practical guide to training restricted boltzmann machines. *Neural Networks: Tricks of the Trade*, 599-619. https://doi.org/10.1007/978-3-642-35289-8_32

Hwang, J., Hwang, W., & Jo, J. (2020). Tractable loss function and color image generation of multinary restricted Boltzmann machine. *CoRR, abs/2011.13509*.

Juliani, A., Berges, V., Teng, E., Cohen, A., Harper, J., Elion, C., Goy, C., Gao, Y., Henry, H., Mattar, M., Lange, D. (2020). Unity: A General Platform for Intelligent Agents. *arXiv preprint* arXiv:1809.02627. https://github.com/Unity-Technologies/ml-agents.

Kratsios, A. (2021). The Universal Approximation Property. *Annals of Mathematics and Artificial Intelligence, 89(5-6),* 435–469.

Lengyel, M., & Dayan, P. (2007). Hippocampal Contributions to Control: The Third Way. *NIPS*.

Leutgeb, S., Leutgeb, J. K., Barnes, C. A., Moser, E. I., McNaughton, B. L., & Moser, M.-B. (2005). Independent codes for spatial and episodic memory in hippocampal neuronal ensembles. *Science*, *309*(5734), 619–623. https://doi.org/10.1126/science.1114037

Lipton, P. A., & Eichenbaum, H. (2008). Complementary roles of hippocampus and medial entorhinal cortex in episodic memory. *Neural Plasticity*, *2008*, 1–8. https://doi.org/10.1155/2008/258467

McClelland, J. L., McNaughton, B. L., & O'Reilly, R. C. (1995). Why there are complementary learning systems in the hippocampus and neocortex: Insights from the successes and failures of connectionist models of learning and memory. *Psychological Review*, *102*(3), 419–457. https://doi.org/10.1037/0033-295x.102.3.419

Moser, M.-B., Rowland, D. C., & Moser, E. I. (2015). Place cells, grid cells, and memory. *Cold Spring Harbor Perspectives in Biology*, *7*(2). https://doi.org/10.1101/cshperspect.a021808

Osa, T., Pajarinen, J., Neumann, G., Bagnell, J. A., Abbeel, P., & Peters, J. (2018). An algorithmic perspective on imitation learning. *Foundations and Trends in Robotics*, *7*(1-2), 1–179. https://doi.org/10.1561/2300000053

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., … Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. *Advances in Neural Information Processing Systems 32*, 8024–8035. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

Ramani, D. (2019). A Short Survey On Memory Based Reinforcement Learning. *arXiv e-prints*, arXiv:1904.06736.

Richards, B. A., Xia, F., Santoro, A., Husse, J., Woodin, M. A., Josselyn, S. A., & Frankland, P. W. (2014). Patterns across multiple memories are identified over time. *Nature Neuroscience*, *17*(7), 981–986. https://doi.org/10.1038/nn.3736

Rolls, E. T. (2010). A computational theory of episodic memory formation in the hippocampus. *Behavioural Brain Research*, *215*(2), 180–196. https://doi.org/10.1016/j.bbr.2010.03.027

Santoro, A. (2015). *A computational model of systems consolidation: Optimizing Memory-guided behavior using abstracted hippocampal and cortical networks* (thesis). University of Toronto, Toronto.

Santoro, A., Frankland, P. W., & Richards, B. A. (2016). Memory transformation enhances reinforcement learning in Dynamic Environments. *Journal of Neuroscience*, *36*(48), 12228–12242. https://doi.org/10.1523/jneurosci.0763-16.2016

Squire, L. R., & Alvarez, P. (1995). Retrograde amnesia and memory consolidation: A neurobiological perspective. *Current Opinion in Neurobiology*, *5*(2), 169–177. https://doi.org/10.1016/0959-4388(95)80023-9

Sutton, R. S., Bach, F., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT Press Ltd.

Tompary, A., & Davachi, L. (2017). Consolidation promotes the emergence of representational overlap in the hippocampus and medial prefrontal cortex. *Neuron*, *96*(1). https://doi.org/10.1016/j.neuron.2017.09.005

Xue, G. (2018). The neural representations underlying human episodic memory. *Trends in Cognitive Sciences*, *22*(6), 544–561. https://doi.org/10.1016/j.tics.2018.03.004

Yu, J. Y., & Frank, L. M. (2015). Hippocampal–cortical interaction in decision making. *Neurobiology of Learning and Memory*, *117*, 34–41. https://doi.org/10.1016/j.nlm.2014.02.002

Appendices

Appendix A: Code Snippets

The full code written for this project is available publicly at https://github.com/pratyushmn/pytorch-episodic-semantic-network.

```python
class AutoEncoder(nn.Module):
    def __init__(self, units: int) -> None:
        """Initializes a nn module "autoencoder" which puts the input through the same layer twice.
        """
        super(AutoEncoder, self).__init__()

        self.CA3W = nn.Linear(units, units)

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        """Returns output of the "autoencoder" for a given input. 1 is subtracted elementwise everytime before the
sigmoid function is applied in the original code; not sure why.
        """
        step1 = torch.sigmoid(self.CA3W(x))
        step2 = torch.sigmoid(self.CA3W(step1))
        return step2
```

Figure 18: Implementation of CA3 network.

```python
class Critic(nn.Module):
    def __init__(self, units: int, gamma: float) -> None:
        """Initializes a nn module "critic" which acts as a value function for the input state.
        """
        super(Critic, self).__init__()

        self.critic_layer = nn.Linear(units, 1)
        self.gamma = gamma

        self.last_val = None
        self.curr_val = None
        self.TDdelta = 0

        self.optim = torch.optim.SGD(self.critic_layer.parameters(), lr=0.04)

    def activateCritic(self, spatial_CA1: torch.Tensor) -> None:
        """Saves the critic's valuation of the input spatially cued CA1.
        """
        self.last_val = self.curr_val
        self.curr_val = torch.tanh(self.critic_layer(spatial_CA1))

    def temporalDifference(self, reward: int) -> None:
        """Computes critic prediction error based on the reward and past predictions, using temporal difference
learning strategies.
        """
        if self.last_val is not None:
            self.TDdelta = reward + self.gamma * self.curr_val - self.last_val

    def updateCriticW(self) -> None:
        """Updates weights for critic based on TD delta.
        """
        # can only do TD learning if at least 2 states have been experienced
        if self.last_val is None: return

        # update the learning rate based on the TD delta
        for g in self.optim.param_groups: g['lr'] = 0.04*self.TDdelta.item()

        # backprop
        self.optim.zero_grad()
        critic_loss = nn.functional.l1_loss(self.last_val, self.TDdelta + self.last_val)
        critic_loss.backward(retain_graph=True)
        self.optim.step()
```

Figure 19: Implementation of critic network.

```python
class EpisodicLearner(nn.Module):
    def __init__(self, device, units: int, lr: float = 0.01, gamma: float = 0.95, place_field_breadth: float = 0.16,
numContext: int = 0) -> None:
        """Initializes an episodic learner class.
        """
        super(EpisodicLearner, self).__init__()

        self.CA3Units = units
        self.CA1Units = units
        self.device = device

        self.CA1Fields = [torch.empty(self.CA1Units, device=self.device).uniform_(0, 1) for i in range(numContext + 2)]

        # NN Layers
        self.input_to_autoencoder = nn.Linear(numContext + 2, self.CA3Units)
        self.autoencoder = AutoEncoder(self.CA3Units)
        self.autoencoder_to_linear = nn.Linear(self.CA3Units, self.CA1Units)
        self.critic = Critic(self.CA1Units, gamma)

        self.input_to_autoencoder.to(self.device)
        self.autoencoder.to(device)
        self.autoencoder_to_linear.to(self.device)
        self.critic.to(self.device)

        self.lr = lr

        self.place_field_breadth = place_field_breadth

        self.loss = nn.L1Loss()

        self.CA3_optimizer = torch.optim.SGD(chain(self.input_to_autoencoder.parameters(),
self.autoencoder.parameters()), lr=self.lr, momentum=0.5)
        self.CA1_optimizer = torch.optim.SGD(self.autoencoder_to_linear.parameters(), lr=self.lr)

    def forward(self, state: torch.Tensor) -> torch.Tensor:
        """Return CA1 output cued from CA3 for given state; also update critic-based state values using CA1 cued from
spatial input
        """
        # CA3
        x = torch.sigmoid(self.input_to_autoencoder(state))
        CA3 = self.autoencoder(x)

        # CA1 cued from the CA3
        CA1_from_CA3 = torch.sigmoid(self.autoencoder_to_linear(CA3))

        return CA1_from_CA3.detach()

    def predict_state_val(self, state: torch.Tensor) -> None:
        """Compute the critic prediction of state value and save it.
        """
        # CA1 cued from space
        CA1_from_spatial = self.probeCA1(state)

        # Critic value for state based on CA1 cued from space
        self.critic.activateCritic(CA1_from_spatial)

    def backward(self, state: torch.Tensor) -> None:
        """Train the weights of all neural network layers (except the critic) based on the input state.
        """

        # updating learning rates (ie. self.lr is 0.1 if reward was found, else 0.1*self.curr_val)
        for g in self.CA3_optimizer.param_groups: g['lr'] = self.lr
        for g in self.CA1_optimizer.param_groups: g['lr'] = self.lr

        x = torch.sigmoid(self.input_to_autoencoder(state))

        # CA3 learning
        self.CA3_optimizer.zero_grad()
        CA3 = self.autoencoder(x)
        CA3_loss = self.loss(CA3, x)
        CA3_loss.backward()
        self.CA3_optimizer.step()

        # CA1 learning
        self.CA1_optimizer.zero_grad()
        CA1_from_CA3 = torch.sigmoid(self.autoencoder_to_linear(CA3.detach()))
        CA1_from_spatial = self.probeCA1(state)
        CA1_loss = self.loss(CA1_from_CA3, CA1_from_spatial)
        CA1_loss.backward()
        self.CA1_optimizer.step()

    def probeCA1(self, state: torch.Tensor) -> torch.Tensor:
        """Outputs the spatially cued CA1 (place cells) based on input state.
        """
        return (torch.exp(-(sum([(state[i] - self.CA1Fields[i])**2 for i in range(len(state))])) / (2 *
(self.place_field_breadth**2))))
```

Figure 20: Implementation of episodic memory, including place cells.

```python
class SemanticLearner(nn.Module):
    def __init__(self, device, vUnits: int, hUnits: int = 300, lr: float = 0.00001, k: int = 1) -> None:
        """Initializes a semantic learner class (based on Restricted Boltzmann Machines).
        """
        super(SemanticLearner, self).__init__()

        self.device = device

        self.num_visible = vUnits
        self.num_hidden = hUnits
        self.lr = lr
        self.k = k # for contrastive divergence

        self.v = nn.Parameter(torch.Tensor(np.random.normal(0, 0.01, (1, self.num_visible))))
        self.h = nn.Parameter(torch.Tensor(np.random.normal(0, 0.01, (1, self.num_hidden))))
        self.W = nn.Parameter(torch.Tensor(np.random.normal(0, 0.01, (self.num_hidden, self.num_visible))))

        self.optimizer = torch.optim.SGD(self.parameters(), lr=self.lr)

        self.to(self.device)

    def visible_to_hidden(self, v: torch.Tensor) -> torch.Tensor:
        """Conditional sampling of a hidden variable given a visible variable.
        """
        return torch.sigmoid(F.linear(v, self.W, self.h)).bernoulli()

    def hidden_to_visible(self, h: torch.Tensor) -> torch.Tensor:
        """Conditional distribution of a visible variable given a hidden variable.
        """
        return torch.sigmoid(F.linear(h, self.W.t(), self.v))

    def forward(self, state: torch.Tensor) -> torch.Tensor:
        """Return the generated state from the input state.
        """
        state = state.view(-1, self.num_visible)
        h = self.visible_to_hidden(state)

        # Contrastive Divergence
        for _ in range(self.k):
            v = self.hidden_to_visible(h)
            h = self.visible_to_hidden(v.bernoulli())

        return v.detach().view(-1)

    def gibbs_sampling(self, x: torch.Tensor) -> torch.Tensor:
        """Returns output of RBM (ie. visible units after passing through RBM k times).
        """
        h = self.visible_to_hidden(x)

        # Contrastive Divergence
        for _ in range(self.k):
            v = self.hidden_to_visible(h)
            h = self.visible_to_hidden(v.bernoulli())

        return v

    def free_energy(self, v: torch.Tensor) -> torch.FloatTensor:
        """Free energy function from Hwang et al. 2020
        """
        v_term = torch.matmul(v, self.v.t()) # matrix multiplication of input v and bias for visible layer
        sum_term = torch.sum(F.softplus(F.linear(v, self.W, self.h)), dim=1)
        return torch.mean(-v_term - sum_term)

    def backward(self, x: torch.Tensor) -> None:
        """Train the weights of the RBM based on the input example state x.
        """
        self.optimizer.zero_grad()

        x = x.view(-1, self.num_visible)

        loss = self.free_energy(self.gibbs_sampling(x)) - self.free_energy(x)

        loss.backward()
        self.optimizer.step()
```

Figure 21: Implementation of schematic memory.

```python
class NavigationLearner(nn.Module):
    def __init__(self, device, units: int, lr: float = 0.0075, actionSpace: int = 8) -> None:
        """Initializes a navigation learner class that predicts the next state of the environment given a combination
of episodic + semantic memory outputs of the current state of the environment, for all possible actions.
        """
        super(NavigationLearner, self).__init__()

        self.device = device

        # In the original code, there's no bias in the linear layers.
        self.evaluator = nn.Sequential(
            nn.Linear(units + actionSpace, 100),
            nn.Sigmoid(),
            nn.Linear(100, units),
            nn.Sigmoid()
        )

        self.evaluator.to(self.device)

        self.lr = lr
        self.eps = 1

        self.optimizer = torch.optim.SGD(self.parameters(), lr=self.lr)
        self.loss = nn.MSELoss()

        self.actionSpace = actionSpace

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        """Returns output of the neural network for a given input.
        """
        return self.evaluator(x)

    def choose(self, memoryGoal: np.ndarray, state: np.ndarray, trialTime: int) -> int:
        """Returns an action to choose based on CA1 state and current memory of goal.
        """
        hamming = torch.zeros(self.actionSpace, device=self.device)
        norms = torch.zeros(self.actionSpace, device=self.device)
        actions = torch.zeros(self.actionSpace, device=self.device)

        for i in range(self.actionSpace):
            actions.fill_(0)
            actions[i] = 1

            x = torch.cat((state, actions))

            output = self.forward(x).detach()
            hamming[i] = torch.sum(torch.abs(memoryGoal - output))
            norms[i] = 1/torch.sum(torch.abs(memoryGoal - output))

        if torch.all(hamming == hamming[0]):
            print("All Hammings are equal.")
            probs = np.array([1/self.actionSpace for i in range(self.actionSpace)])
        else: probs = convertToProbability(1 - hamming, np.clip(trialTime/2000, 0, 0.99), 1)

        return np.random.choice(np.arange(self.actionSpace), p=probs)

    def learn(self, state: torch.Tensor, next_state: torch.Tensor, action: int) -> None:
        """Updates neural network weights.
        """
        self.optimizer.zero_grad()

        actions = torch.zeros(self.actionSpace, device=self.device)
        actions[action] = 1

        x = torch.cat((state, actions))

        state_prediction = self.forward(x)

        loss = self.loss(state_prediction, next_state)

        loss.backward()
        self.optimizer.step()
```

Figure 22: Implementation of navigation network.

```python
class Agent():
    def __init__(self, episodicUnits: int = 980, contextDimension: int = 0, actionSpace: int = 8, episodic: bool =
True, semantic: bool = True, priorKnowledge: bool = True) -> None:
        """Initializes an agent class that can learn to correctly choose actions given an input state, using a
combination of
        episodic and semantic memory.
        """
        self.device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
        print("Device: {}".format(self.device))

        self.learners = []
        self.learners.append(EpisodicLearner.EpisodicLearner(self.device, episodicUnits, numContext=contextDimension))
        self.learners.append(SemanticLearner.SemanticLearner(self.device, episodicUnits))
        self.navigation = NavigationLearner.NavigationLearner(self.device, episodicUnits, actionSpace=actionSpace)

        self.priorKnowledge = priorKnowledge

        # Episodic, semantic, and overall
        self.goals = [np.zeros(episodicUnits), np.zeros(episodicUnits), np.zeros(episodicUnits)]

        # Policy to switch between episodic and semantic network
        self.tau = 2000
        self.policyN = 1
        self.episodic = episodic
        self.semantic = semantic

        # last action
        self.action = 0

    def act(self, state: np.ndarray, trialTime: int) -> int:
        """Returns an action to choose based on input state and current number of steps taken.
        """
        state = torch.Tensor(state).to(self.device)
        self.ca1Now = self.probeCA1(state)
        self.currState = state

        self.goals[0] = self.learners[0].forward(state)
        self.learners[0].predict_state_val(state)
        self.goals[1] = self.learners[1].forward(self.ca1Now)

        # Overall goal
        if self.episodic and self.semantic:
            self.goals[2] = (self.policyN * self.goals[0] + (1 - self.policyN)  * self.goals[1])
        elif self.episodic:
            self.goals[2] = self.goals[0]
        elif self.semantic:
            self.goals[2] = self.goals[1]

        # Choose an action based on goal and predictions from nav network
        self.action = self.navigation.choose(self.goals[2], self.ca1Now, trialTime)

        # Decay policy
        self.decayPolicy()

        return self.action

    def learn(self, state: np.ndarray, reward: float) -> None:
        """Updates weights of component networks.
        """
        state = torch.Tensor(state).to(self.device)
        self.ca1Next = self.probeCA1(state)
        if reward > 0:
            self.navigation.learn(self.ca1Now, self.ca1Next, self.action)
            self.learners[0].critic.temporalDifference(reward)
            self.learners[0].critic.updateCriticW()
            self.learners[0].lr = 0.1

            for i in range(50): self.learners[0].backward(state)

            randomState = state
            for i in range(200):
                self.ca1Next = self.learners[0].forward(randomState)
                self.learners[1].backward(self.ca1Next)
                if self.priorKnowledge is True:
                    x, y = np.random.uniform(-15, 15), np.random.uniform(-15, 15)
                    c = 1 if x <= 0 else 2
                    randomState = [x, y, c]
                else: randomState = [np.random.uniform(-15, 15), np.random.uniform(-15, 15), 0]
                randomState = torch.Tensor([(randomState[0] + 15)/30, (randomState[1] + 15)/30,
randomState[2]/2]).to(self.device)
        else:
            self.navigation.learn(self.ca1Now, self.ca1Next, self.action)
            self.learners[0].critic.temporalDifference(reward)
            self.learners[0].critic.updateCriticW()
            self.learners[0].lr = self.learners[0].critic.curr_val.item() * 0.1
            self.learners[0].backward(self.currState)

    def probeCA1(self, state: torch.Tensor) -> torch.Tensor:
        """Returns place cell representation of current state.
        """
        return self.learners[0].probeCA1(state)

    def decayPolicy(self) -> None:
        """Updates the balance between episodic and semantic memory goal every step.
        """
        self.policyN = activatePolicy(self.policyN, self.tau)
```

Figure 23: Implementation of overall agent.

Appendix B: Result Plots (with step count on x-axis)

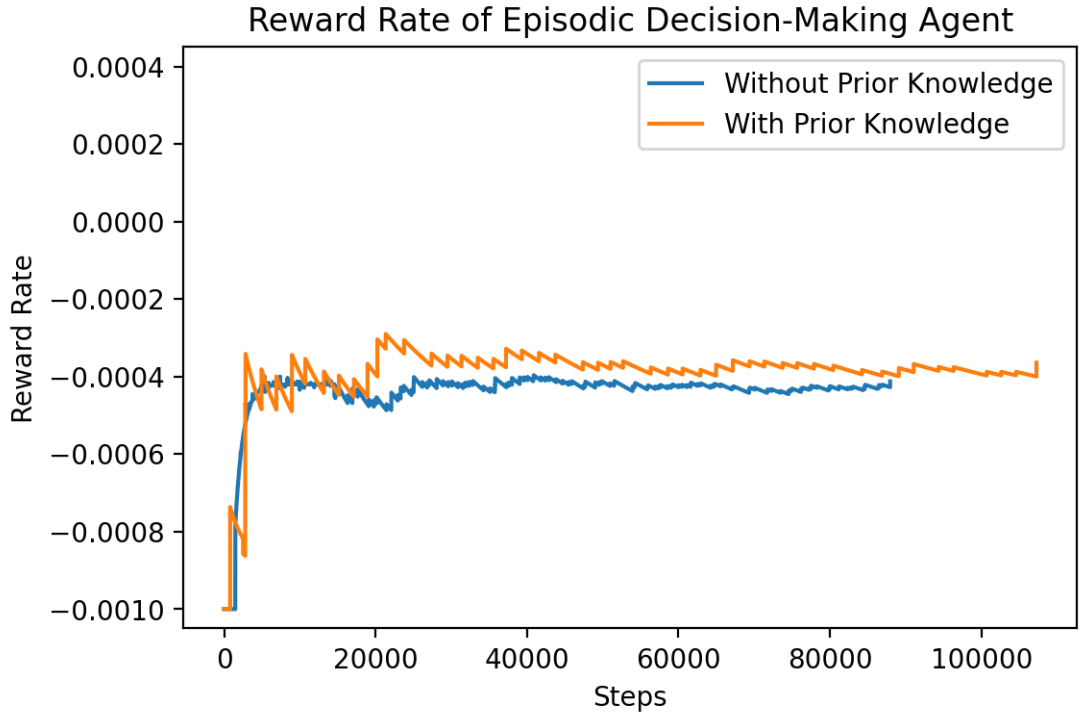

Figure 24: Reward rate vs steps for episodic decision-making agent.

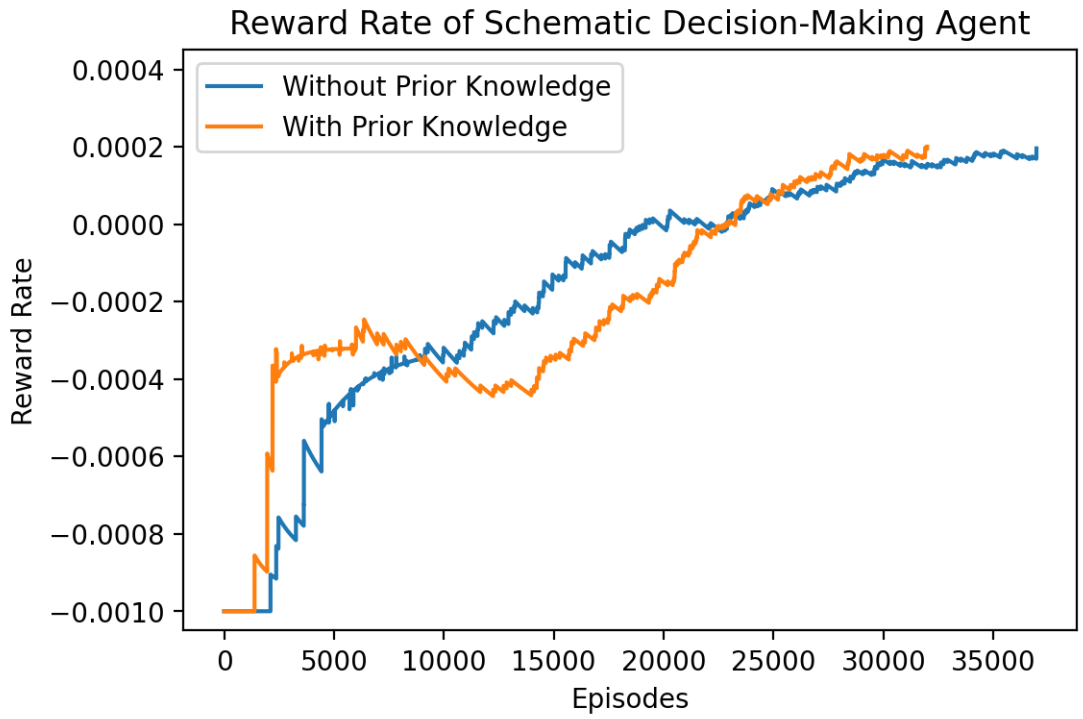

Figure 25: Reward rate vs steps for schematic decision-making agent.

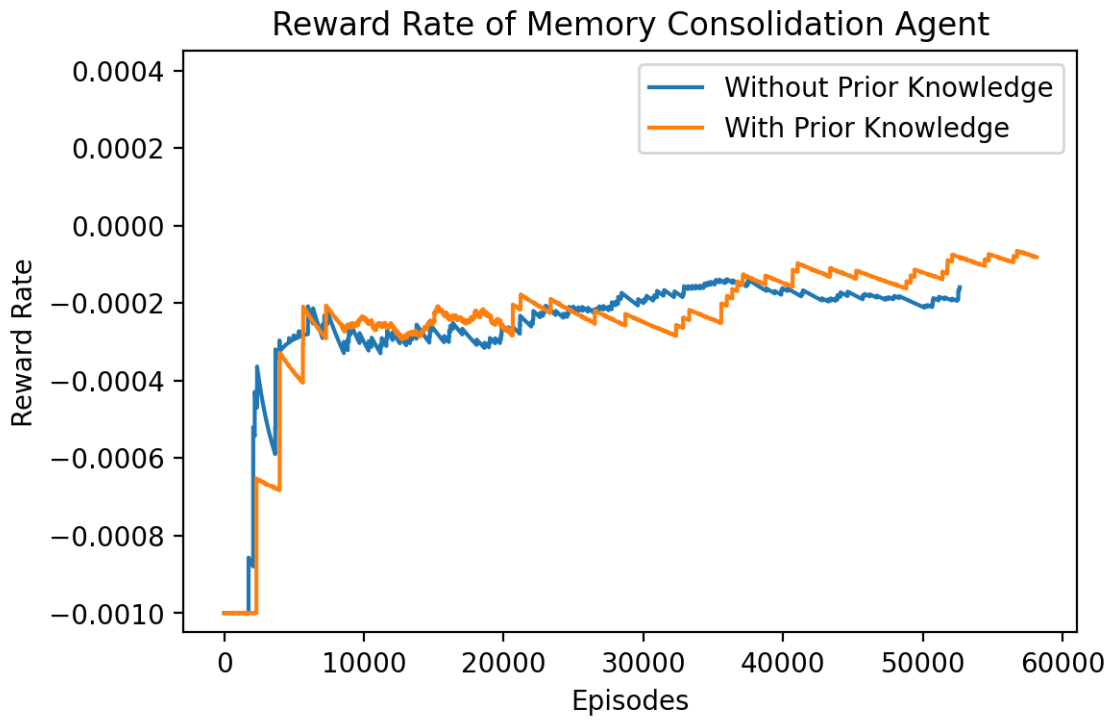

Figure 26: Reward rate vs steps for memory consolidation agent.

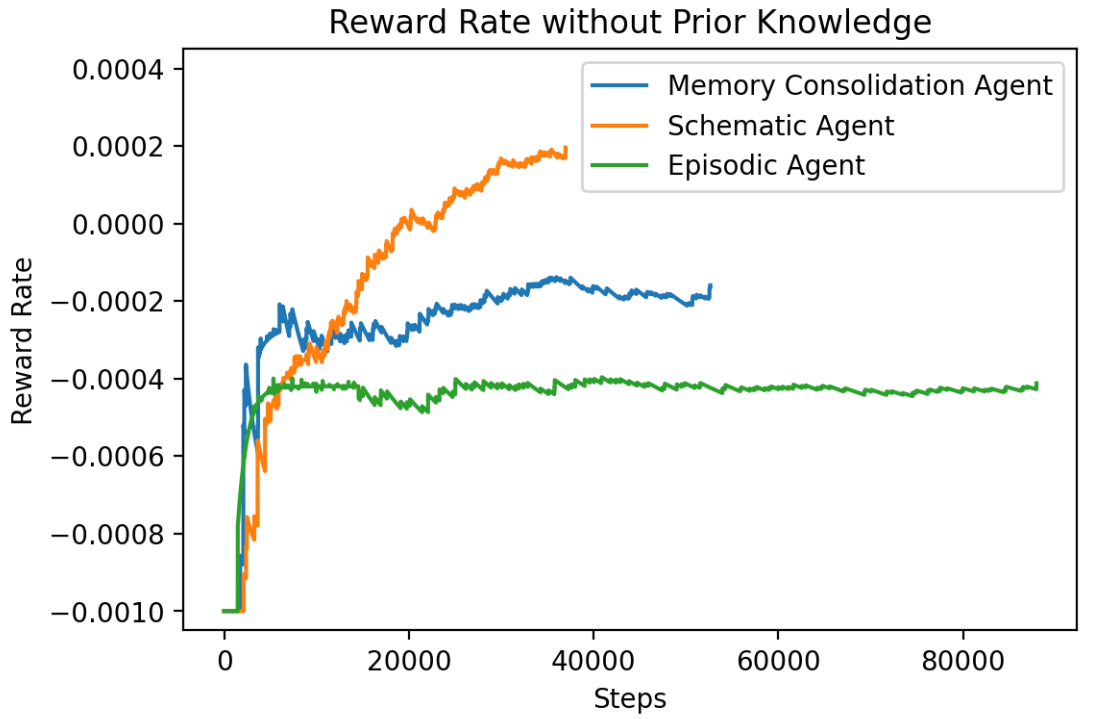

Figure 27: Reward rates of all agents vs steps when prior knowledge was not provided.

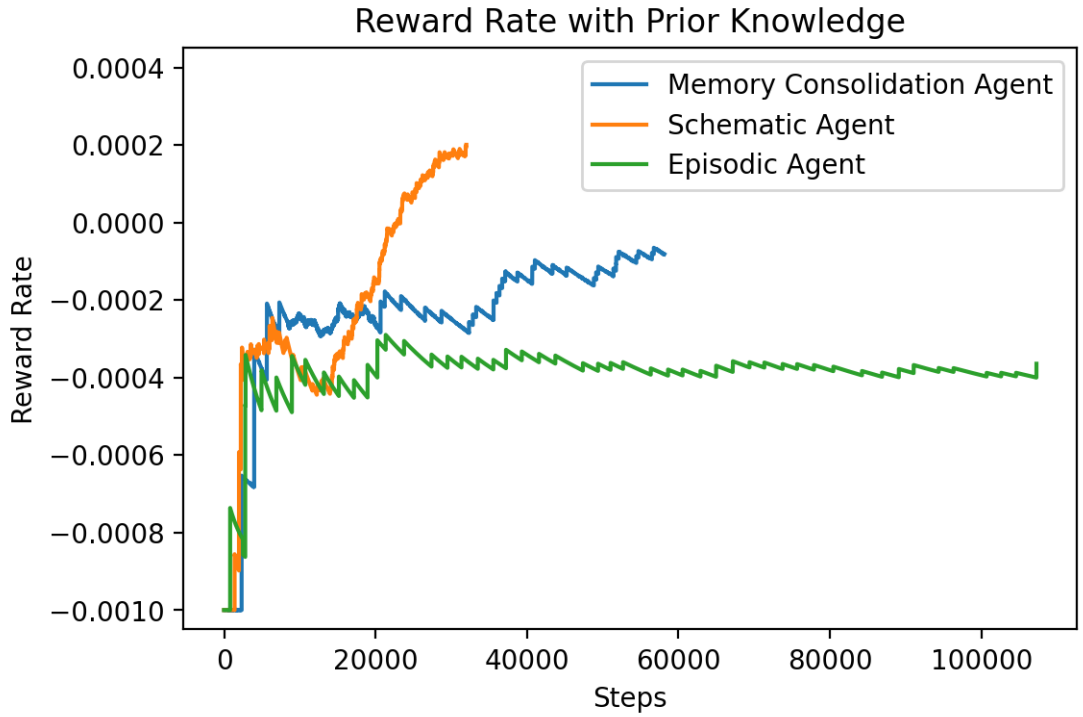

Figure 28: Reward rates of all agents vs steps when prior knowledge was provided.