2024

Bayesian Inference in reinforcement learning neural networks during a Markov decision processes?

Katherine Graham

Bayesian inference in reinforcement learning neural networks in a Markov decision process?

Thesis presented by

Katherine Graham

Presented to

Keck Science Department of Claremont Mckenna, Scripps, and Pitzer Colleges
and
The Cognitive Science and Linguistics Department of Pomona College

In Partial fulfillment of

The Degree of Bachelor of Arts

Senior Thesis in Physics and Cognitive Science

April 22, 2024

Contents

Abstract

The predictive mind theory proposes that brains work in a way that makes predictions about future stimuli to process information efficiently and accurately. Bayesian brain theory suggests that the brain utilizes Bayesian probability models to make predictions, while the free-energy minimization hypothesis proposes that these predictions are made to minimize energy or uncertainty, ensuring accurate perceptions. Vertechi et al. (2020) explored animal participants' utilization of stimulus-bound strategy versus inference-based strategy to solve a Markov decision process with a 2-state environment, one of which is always active. These sites have a certain probability of switching to a different site and the inverse probability of staying in the same site for the next guess or iteration. This setup served as the basis for my experiment, where I employed three types of model-free artificial neural networks in the 2-state MDP environment: Deep Q-learning, Proximal Policy Optimization, and Recurrent PPO with long-short term memory architecture. Each agent was tested in three environments with varying probabilities of active site switching and reward allocation.

The data showed that all but one ANN in the medium environment failed to learn with an accuracy above the expected rate limited to 1-back memory. In the medium and difficult environments, the DQN was the best performer, followed closely by the RPPO. Across past studies, the DQN was outperformed by the PPO agents, which is inconsistent with our findings. However, our findings are consistent with Vertechi et al.'s (2020) prediction that a model-free and stimulus-bound agent would get worse at learning the environment depending on the frequency at which the rewards were given. These findings also show that animals must have at least a mixture of model-free and model-based processing involved when problem solving and doing other cognitive tasks.

Bayesian inference in reinforcement learning neural networks in Markov decision processes?

1. INTRODUCTION

Example: One rainy day in east LA county, I ran into a crawfish on my way to my apartment. I was offput and confused. At first glance, it looks like a crayfish, but it would make more sense, given my environment, that it is a strange type of scorpion. Upon further investigation and to my surprise, it was indeed a crayfish.

I was faced with a situation that did not match my internal representation of the world; I did not factor in the possibility of seeing a live water creature outside my apartment in a desert climate, probably because I have never seen anything like this before. This situation was also deeply surprising to me because I had not lived in an environment with scorpions for very long, so I also didn't have a trustful idea of what a scorpion experience would be like.

The predictive mind theory suggests that the brain makes top-down predictions about current and future stimuli based on some model that correlates real world stimuli to the experiences of them. These top-down predictions allow us to respond to stimuli quickly and to draw closer attention to unexpected, important information from the environment. This also allows us to process a large amount of information with limited resources: neurons, neural connections, energy, and time to name a few.

For these predictions to be accurate, they need to be made by appropriate models. One promising type of model would be a statistical one. In physics, a group of particles through a period of time, depending on the conditions of the environment (pressure, number of particles, etc.), "behave" in a Bayesian way. This means that without outside forces, particles structure themselves naturally in a way that follows a normal distribution of probable states; there is a particular state that minimizes the free energy of a system, and therefore is the state that is most

probable in the set conditions. However, there are many other states that also relatively minimize free energy, but not as much, and therefore it would be less likely to find these states randomly.

If brains had a Bayesian model for prediction, they would need to be able to establish statistical conditions of the environment and of experiences of the environment. By doing so, they would be able to know what action would most likely lead to finding a food source, or what action would most likely boost their mood. However, it is difficult to determine if this model, or any model, is being utilized to any extent by the brain.

There is evidence that suggests that the brain is scanning for new information before the information is present, which would support the claim that the brain makes predictions in some way before bottom-up processing starts. There is also evidence that shows that animals make inferences (predictions) in novel environments in order to maximize reward. Despite these findings, it is still unclear how bottom-up and top-down processing interact to form memory, knowledge, and experiences consistent with what exists in the world.

The purpose of my study is to better understand to what extent agents perform at a Bayesian performance level. The first section will focus on a literature review of both the predictive mind theories and research, as well as the background for the artificial neural networks that act as the agents in the experiment. The second section will describe the procedure and materials of the experiment where we compared the performance of artificial neural network (ANN) agents, measured by their average correct accuracy rate, to the optimal performance of a 1-back memory Bayesian agent. To conclude, the calculation of the optimal performance ranges will be discussed along with the agents' performances.

While this study doesn't provide evidence for Bayesian inference in model-free neural networks, it does show support for the idea that animal brains utilize both model-based processes along with model-free processes to perform their complex and novel tasks.

2. LITERATURE REVIEW

2.1 The Predictive Mind

The traditional view of the brain is that it is a passive blob of tissue and neural circuits that awaits stimulus to trigger an action potential, and ultimately a perceptual experience. This is a natural view to instinctually believe when looking at the mechanical systems by which we gather information about the world external to our sack of cells. The most typical demonstration is sight; photons hit the retina, which are reflected through a lens and then hit receptive cells on the back of the eye. These cells send signals that are carried down the optic nerve, resulting in the excitation of neurons in the occipital lobe, primarily the primary and secondary visual cortices. Once this process is completed, the view claims, visual experience occurs. And this explanation purportedly applies to all modes of perception, as well as cognition in general and even to agency (Hohwy 2013, 258). Once presented with stimuli, the brain decides how to handle the new information. This is known as the stimulus-bound strategy, where the agent is limited to make actions based on this presently available information (Wilson et al. 2014).

However, some cognitive issues arise when you take this view. How is the brain able to processes so much detail at once? How do dreams and imagination come about? And how do we have detailed representations of external objects? For example, we are able to convert the 3-D world through our 2-D visual system into a 3-D picture that is representational of the world; and we would be able to do this without our other modes of perception. In response, Helmholtz introduced the idea of ‘unconscious inferences’, which explained how the 2-D initial perception is transformed into a mental object that is 3-D, and how the representation could be imagined further as a different color or shape (Patton 2023).

Unconscious inferences occur for all perceptual experiences, and perhaps even many other cognitive tasks, including metacognition. However, the brain does not seem to get

overwhelmed by the conflicting data and inferences, and also seems to be able to weigh the value of the perceptual information to determine which one is better to "listen" to (Yon and Frith 2021, 1). This process is often referred to as an inference-based strategy of navigating an unknown environment (Wilson et al, 2014). In this model, the brain has the capabilities to somehow combine both bottom-up stimuli with top-down inferences, which has led people to research the role of the Orbitofrontal Cortex and the mechanisms used in inference making (Wilson et al. 2014, Vertechi et al. 2020).

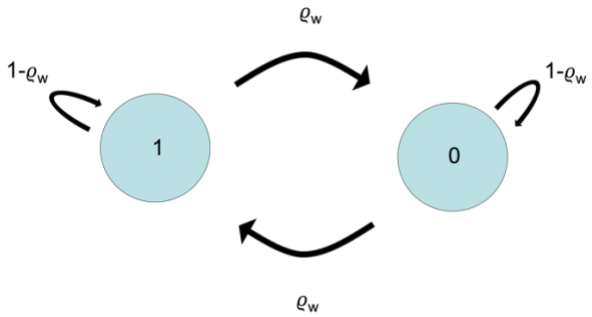
It is hypothesized that the brain, or a region of the brain, follows the rules of Bayesian probability to make inferences, known as *Bayesian inferences*. These inferences influence the brain's selection of information that is more reliable, or what results in the *minimization of uncertainty*. Based on the Bayesian brain theory, hypotheses are made based on the inference that will minimize uncertainty, which suggests that perception is a process starting with top-down processing– which does the hypothesis calculating and selecting– and is then met with bottom-up data from the stimulus and combines the two processes to create a perception (Yon and Frith 2021, 2). The process of formulating and choosing a hypothesis is referred to as *predictive processing* which is a key signature of active inference; the example of the light patterns hitting the retina would be passive inference, in comparison (Smith et al, 2021). By this process, top-down feedback (predictions), or information from previous experiences, meet the signal from external stimuli to form an accurate (and sometimes inaccurate) representation of the world. One concern with the Bayesian brain theory is that to truly achieve minimum uncertainty, the brain would have to have access to unlimited resources, which is not possible. However, it does suggest that predictive processing could be, or include, an approximation of the Bayesian brain.

There have been a number of approaches to understanding how the brain would be able to do this through the notion of rationality, which claims that people act in a way "[that maximizes] their expected utility, reason based on the laws of logic, and handle uncertainty according to probability theory" (Lieder and Griffiths 2020, 2). However, it has been proven that most people act in a way that deviates from these rules, which led to the development of the 'resource-rational analysis' (Lieder and Griffiths 2020, 4). Resource-rational analysis accounts for cognitive constraints and limited time to analyze based on resource availability, such as energy, neurons, and memory. Therefore, it is argued that animals must be able to limit the information processed and stored in order to feasibly make a rational analysis. From Merleau-Ponty's idea that perception is always directed towards something, in addition to the resource-rationality analysis that states that resources are limited, it is reasonable to believe that our brain sorts through or keeps a running average of stimuli information most important or applicable for survival (De Ridder et al. 2014, 5).

If one buys into the notion of resource rationality, then it is accepted that there are a limited number of fibers and pulses per second for a neural pathway, which puts a limit on the application of Bayes' theorem. So, the next step is to understand how the vast amount of information that humans are confronted with is managed by the brain given its limited resources; how is a neuron or neural pathway able to minimize redundancy in order to minimize resource expenditure, but also relay a sensory message without losing its information (Barlow 1961, 223)? A way to understand maximizing efficiency while maintaining information is through the idea of minimizing energy expenditure. The idea of *free-energy minimization* also came from Helmholtz in the realm of physics but was applied to this predictive mind framework by Karl Friston (Friston et al. 2007).

Friston claims that each neuron has the ability to process information, and all these neurons, or nodes, are interconnected such that the information that they relay affects how connected nodes respond to the information. The effect that one node has on another is often referred to as its weights. By the free-energy minimization theory, there is a state for that neuron to use the least amount of energy or do the least amount of action (Friston et al. 2012, 2). Another way that free-energy minimization is talked about in relation to the brain is by surprise, or the measure of improbability based on the Bayesian model (Friston et al. 2012, 1). Furthermore, because of the web of connections between all neurons, there is a state for which the entire system uses the least amount of energy needed to effectively process information. And, to maintain this type of equilibrium, the weights between nodes must constantly be updated as the predicting system learns how to better minimize free energy when processing information. This is often modeled by a Markov Model, which depicts the states as circles and arrows between the circles with valued weights that show the probability of the neighboring neuron being activated in the next state (Visser et al. 2000, 5).

Figure 1
Markov Model



Markov models can also depict a probability distribution of states in an environment, or a probability model of an environment. For example, in Figure 1, the ‘environment’ has two

possible active states that have different probabilities for staying in the same state in the next output, or for switching to a different state. This is useful in applications for understanding when and how an individual might be navigating a task using a sort of Bayesian influence, often referred to as a Markov Decision Process (MDP). MDPs are used to model decision making in stochastic, discrete, and sequential environments (Littman, 2001). By calculating an optimal accuracy rate, given a certain amount of memory, an individual's performance can be compared to the optimal rate given Bayesian probabilities. Additionally, if an individual were to excel in a task beyond the optimal rate, it would suggest that it is implementing a form of prediction that is optimally efficient in resource dependent environments.

2.2 Implementation

When learning something new, it is essential that the agent have a way of knowing when they are right or wrong, often referred to as feedback. Feedback can be given in two forms: through rewarding correct behavior or through error signaling when it is incorrect behavior.

An example that demonstrates how feedback is a constant necessity for completing any task is someone catching a ball (Clark 2023, 81). The brain is taking in data about where the initial location of the ball is, the size of the ball, and how far away you initially are from the ball. However, we don't idly sit back and see where the ball could go. We have learned from watching people throw balls that the direction that they're facing and the direction their arms move can be used to narrow down the possible places that the ball could end up. Additionally, we consider who the person is that is throwing the ball; do they play a sport related to ball-throwing, and do they have a strong arm? All these factors are being processed pre-ball throw by the brain so that it is able to lessen the uncertainty of where the ball will end up.

Now, humans also don't sit idly by once the ball is thrown and assume it is going to go exactly where they initially predicted it would. The predictor must also be updated as new information is processed and stored so that the predictor maintains minimized uncertainty. One is constantly reevaluating where the ball is in space and is calculating how they need to move their body in order to catch it. Finally, once the ball has reached them, they reflect on whether they were successful at catching the ball, and what they could have done differently to make their next encounter with this experience smoother, or more predictable. And, as one experiences more balls thrown in various ways, their predictive abilities will improve, thus improving their ability to catch thrown balls.

In this scenario, feedback is mostly error signals updating the predictor on where the ball is located in space and how it is moving; at first you predict it will be at one point in space, but then the ball's velocity changes and the next position that you predicted it would be at is false. Thus, in order to actually catch the ball, your predictor would need to take new stimuli information into account when predicting how to catch the ball. As further evidence for this updating process, when there is a mismatch between the prediction and reality, brain activity increases, resulting in structural changes in the neural system (Wessel 2017, 3). The weights within the prediction machine are recalculated based on the error. That is to also say that weights strengthen with the repetition of positive interactions between top-down and bottom-up data or signals.

This process is referred to as error processing. In order to form a predictive model, one must monitor one's performance in order to correct their model for the next prediction given. In predictive mind work, there are currently two theories for error processing: maladaptive and adaptive error processing (Wessel 2017, 3). Maladaptive error processing poses the idea that

errors momentarily impair cognitive processing, resulting in increased rates of error on trials post-initial error. By this theory, it is understood that errors are infrequent and unexpected, which results in an orienting response where "a cascade of autonomic and central nervous system activity occurs" (Wessel 2017, 3). However, an opposite effect occurs when errors become more frequent in a trial, suggesting that this orienting response occurs when a response's validity matches what is expected; even if it is more likely for a response to be incorrect, having a correct response would trigger delayed responses in following trials. This suggests that it is an implicit process where the individual is not in control of the evaluation of the response's accuracy.

On the other hand, adaptive post-error processing, proposed by R. J. Laming (1968), suggests that individuals begin sampling information before imperative stimuli from the task are even presented, consistent with predictive processing (Wessel 2017, 2). This is supported by an experiment done by King et al. (2010), who found that BOLD activity in the early sensory region significantly increased after errors compared to accurate responses. A result of this research was that there was post-error slowing following an incorrect response, which was suggested by the authors to be explained by motor-system inhibition and thus is an explicit process where the individual is conscious of the error analysis.

While it is evident that error-processing plays a role in the learning process, it is also evident that rewarded behavior aids in the learning of an agent, commonly understood via the ways in which the mesolimbic dopamine pathway is involved in learning. Additionally, the famous example of Pavlov's dogs where Pavlov showed that organisms can be conditioned to associate a signal with a behavior (Berridge 2000, 264) is an example of reward learning. Reward learning is also utilized in reinforcement learning (RL) for training ANNs. Therefore, depending on which form of feedback is implemented, an agent with an internal Bayesian model

of their environment would utilize this feedback to model the probabilities of states of the environment.

We now have predictive processing as a theoretical framework that is supported by computational processing and studies done on neural activity (Barlow 1961, Friston 2006, Wessel 2017, Lieder and Griffiths 2020, Yon and Frith 2021), and it is established that the brain possibly approximates predictions by a Bayesian model. However, it is still a question whether there is evidence that prediction is occurring. Rao and Ballard (1999) searched for evidence by noticing patterns in research on monkeys, where there were visual neurons that responded optimally to line segments of a particular length and that they had an interesting property, called endstopping (or end-inhibition). It was found that most neural responses in the classical receptive field (RF) were suppressed when the stimuli in the peripheral areas (or extra-classical RF) matched the stimuli in the classical RF, whether it be orientation, velocity, or direction of motion. By suppressing stimuli that are unnecessary or that are being double-counted, free-energy is minimized. Additionally, it would limit the amount of prediction-stimuli matching that has to occur. This finding also implies a hierarchical predictive strategy for encoding natural images (p. 79).

To demonstrate the hierarchical structure of this specific visual perception, Rao and Ballard carried out a study based on the principle of Kalman filtering, which used linear models to give the computer the ability to make predictions of what information to fill in when provided a new image. Imagining each retinal cell input as a pixel, it is understood that closely connected pixels have correlating intensities and therefore the most-center image or pixel can often be predicted by surrounding values of other pixels. The information is given to the computer via basis vectors, which allow the computer to make all possible images from those vectors. Then, in

the trial, the computer was given images that it was not exposed to in the training. The results showed that the computer was much more accurate when the trial image more closely resembled training images, and accuracy decreased as the similarities decreased (Rao and Ballard 1999, 81).

Furthermore, it was found that when the computer didn't predict the image accurately, it removed feedback from the trained images to the trial images, caused by endstopped neurons, so that it was able to continue to respond to the trial image as it gathered more information about the length of the bar in the image (p. 82). These extra-classical effects "are interpreted as error-detecting neurons that signal the difference between an input and its prediction from a higher visual area" (p. 84). This suggests that as predictive errors occur, there is an increased reliance and emphasis on stimuli information as top-down processes gather enough information to predict more accurately what matches reality.

This transition of strategies can also be described as a transition from a model-based strategy where the individual had an internal representation of the environment and its conditions, to a model-free strategy where the agent is only making decisions based on the present stimuli and information. Model-based learning is useful when there is an accurate model of the environment's condition. However, when an agent is not succeeding based on that model, a model-free strategy allows an individual to put greater emphasis on the value of the most recent or present information until they are able to succeed in the environment again. Humans are believed to use both methods of learning in parallel, which aids in their ability to apply previously learned information to new tasks, as well as to learn new rules of an environment quickly (Haith and Krakauer (2013), Doody et al (2022)).

One may raise the puzzle of why the brain wouldn't seek a protected and sealed place with no new information in order to minimize surprise and energy expenditure. Friston's

response to “The Dark Room Problem” is that the brain is structured in a way to expect new stimuli– it searches for more information to process, and thus seeks learning (Friston et al. 2012, 2). Because of this, when the brain receives no new stimulus, its predictions are majorly incorrect, and it will want to seek an environment that is more like its predictive model, such as a space with trees, blue skies, and the smell of grass as to minimize its future action. There is evidence that the brain is predicting the state of retinal ganglion cells, and specifically as it relates to motion (Palmer et al. 2015, 6911), which suggests that being in a space with little to no movement would invoke surprise in an individual, thus resulting in a heightened predictive uncertainty situation.

It is valuable to establish a deeper understanding of the role prediction plays in average learning, and appreciation for why humans learn instead of isolate. Nagai (2019) notes that neonates do not have the inherent ability to control their body, but with more experience and learning they are able to produce accurate and purposeful actions. Additionally, the idea of dynamical change, where “new behaviors are thought to emerge as a result of many decentralized and local interactions between infants and their environment” suggests that infants and children learn when the new information is closely related, but not identical, to previous information they know (Nagai 2019, 1). This suggests that learning occurs in the space where the child’s predictor is able to make predictions based on previous experiences, but also experiences a prediction error that allows for the predictor to be updated and better calibrated. The predictor has also been modeled on random generator ANNs that make quite horrible predictions at first, but improve as they continue to make more predictions and get feedback on their accuracy (Clark 2023, 28).

Finally, much of the research focuses on understanding the distinction between implicit knowledge versus explicit knowledge in the context of forming predictions and responding to the environment. Explicit knowledge is that of which we are aware of and can consciously think of and draw from. For example, completing an exam for a class relies heavily on explicit knowledge of the topic on the exam, specific formulae, or definitions. Implicit knowledge is that which is ingrained in us– we don't necessarily know that we know it. For example, walking and riding a bike are activities that we are able to do without thinking about it– without intentionally trying to remember the exact steps necessary to accomplish the goal.

Therefore, scientists are studying whether prediction is heavily dependent on humans effortfully remembering the previous stimulus and consciously (to some extent) calculating which one is more likely to appear next, or whether it is more dependent on unconscious processes that have an algorithm that implements these previous experiences (Visser et al 2007,1502). One way to study implicitly versus explicitly learned information is to present participants with a set of information and ask them to reproduce that information using their own symbols (Reber 1967).

However, instead of depending on participants reciting what they had learned, Nissen and Bullemer (1987) asked participants to predict what stimulus was next in the sequence given the knowledge they acquired from the previous part of the sequence. This type of task is a generative task, which can be helpful for analyzing the encoding and recall of information learned. However, one limitation with generative tasks is dealing with feedback. In order to evaluate explicit sequence knowledge, the generation task requires feedback. However, feedback can provide extra time for learning to occur during the task, which can skew the measure of knowledge learned during the RT phase. Additionally, feedback can interfere with subjects'

memory when trying to predict imminent stimuli (Visser et al. 2007, 1503). Despite these limitations and the importance of noting the role of them in this research, feedback is a crucial component of learning and important for tracking an individual's progress.

2.3 The Neural Network

In order to better understand how an agent might implicitly apply Bayesian inference to accomplish a task, it would be beneficial to track the learning process of the agent as a MDP in a new environment. If an agent learns how to most likely accomplish a task or best understand its new environment, the learning process of the agent during this process can be compared to an optimal model of Bayesian prediction probabilities to see if the agent seems to be performing at a similar level after learning.

Vertechi et al. (2020) created an environment to test mice and human subjects' inferences where the reward was located at one of two locations, and the location had uncertainty associated with it to reflect "non-sensory uncertainty" (p.173). In the environment, the door that had the reward would only give the reward a certain percentage of the time, leading the subject to having to decide whether it was worth staying at the same door or if they should switch doors. The reward also had a set probability of being released by the active door. Researchers in this study found that early on "the number of consecutive failures was positively correlated with the propensity to leave" and that "subjects were sensitive to quantitative changes in the statistics of the foraging site" (p. 173), which was evidence for stimulus-bound strategy. However, with time, they saw a transition in both mice and humans from a stimulus-bound strategy to an inference-based strategy where the amount of feedback didn't affect their performance.

Their research showed that both humans and mice use inference-based foraging to maximize reward and success in an environment. However, one key finding that these researchers found

was that humans nearly instantly completed the foraging task using inference-based processing while it took the mice several sessions to eventually get to the same point (p. 174). Similar studies have been run on recurrent neural networks (RNNs) who have been able to succeed in Markov designed environments by following Markov Decision processes (MDP). RNNs are a category of neural networks that use sequential data to learn how to perform a task in an environment by assuming that there is a relationship between input and output, or that "its output is dependent on prior elements within the sequence" (IBM, *What are Recurrent Networks?).*

Another specific architecture for neural networks is a Deep Q-Network (DQN). DQNs introduce hidden layers to the feedforward learning process between the input and output layers. DQNs try to create a formula that consists of hyperparameters that enable the agent to calculate a Q-score for the next possible moves, also known as Q-learning (Mnih et al, 2013). Hyperparameters consist of the learning rate, exploration rate, and the discount factor. Q-scores, or Q values, are evaluated to determine which output, or action, will most likely return the greatest reward (Equation 1).

Equation 1

Q-learning equation that is updated to estimate accurate Q-values.

$$Q^*(s,a) = \mathbb{E}_{s' \sim \mathcal{E}}\left[r + \gamma \max_{a'} Q^*(s',a') \middle| s, a \right]$$

Both the DQN and RNN agents use model-free reinforcement learning and can do so in two ways: through action-based learning of maximizing the function value between two points, Q-learning, and through a policy optimization method, such as Proximal Policy Optimization (PPO). PPO is a gradient descent method, or algorithm, that optimizes its policy by performing multiple gradient descent to update the network. The goal of this method is to minimize the

change in policy necessary between iterations, which maintains more stability in the agent. The most common policy-optimizer, implemented in control systems and power grids, is an Actor-Critic (Khandelwal 2023). In this architecture, the critic is responsible for evaluating the actor's optimization of the policy.

When the PPO algorithm is implemented in a RNN, long-short term memory (LSTM) is implemented. LSTM keeps a running average of the reward possible for each possible output. By implementing this form of memory, the network is able to learn quickly in simple environments by using an efficient memory system. LSTM has been commonly implemented in Markov Decision processes to navigate image selection and maze navigation (Wang et al. 2018).

Additionally, both DQNs and RNNs can use a Multilayer Perceptron (MLP) architecture, a form of reinforcement learning that implements an actor-critic algorithm. MLP architecture consist in an input, which is data representing the state, and a certain number of hidden layers of nodes that represent neurons. Based on the connections between the input and these layers, an output layer collects Q-scores that determine the agent's output, or action. A key feature of MLP is that it is able to differentiate data that is nonlinear, which is helpful in situations where two factors in the environment are being adjusted and there is a nonlinear relationship between how they affect the conditions of the environment. MLP's do this via forward propagation where data is propagated forward from the input to output, where the error is then calculated and minimized during the learning process. By calculating the minimum error, the DQN is then able to calculate Q-values and choose its actions from that (Banoula, 2023).

2.4 Prediction and learning in neural networks: My next steps and hypotheses

By creating an environment with a set probability of switching states and a set probability of rewarding the agent, such as the one used by Vertechi et al. (2020), the behavior and learning of

an agent in this new environment can be tracked to determine if the agent is learning or operating in a way that is to some degree tracking the next probability of the best next guess. This experiment tests the agent's ability to learn in a MDP. If an agent were to be acting in this way, there would be strong evidence to support that the agent uses Bayesian inference to better understand the environment it is in and predict the next active state that the environment will be in.

I predict that all three agents will perform above the optimal 1-back memory accuracy range because they all have a memory that keeps track of more than 1 trial at a time. The DQN has room for thousands of experiences to be remembered in "memory" that it can randomly draw on for establishing the Q-function. Therefore, if it has a Bayesian model for learning, I predict that it would be able to establish a more accurate model of the environment than if an agent only had 1-back memory.

3. METHODS

3.1 General Remarks

We will use a python environment designed by Vertechi et al (2020) and the Stable Baselines3 agents—DQN, PPO, and Recurrent PPO—to observe the agent's learning trend and compare it to the optimal Bayesian decisions or predictions given knowledge of the previous state (1-back memory) (Section 4.1.2). The agent will be untrained and will run 10,000 trials before it is terminated. In each trial, the state of the environment, the action or guess of the agent, and the reward given to the agent will be printed and recorded.

3.2 Participants

For this experiment, 75 ANN agents were each run in three different environments: 25 were DQNs, 25 were PPOs, and 25 were Recurrent PPOs. A 5x5 parameter spread of agents were run to determine the best learning agent in the category given the type of environment it was in.

3.3 Materials

3.3.1 The Environment

The environment is designed allocentrically[1], which means that the state of the environment (self.state) is either "left" or "right." Another way to imagine the environment is as though the environment has two sites, a left and right, and one of which is always active at a given time. Three hidden Markov Model (HMM) environments were implemented. The first environment was the "easy environment" (Environment C) because it rewarded the agent almost every single time and had a high switching probability so that the agent could easily follow a

[1] In an allocentric environment, the agent moves according to the environment, such as choosing the left option or bottom option in the environment. In contrast, an egocentric environment is one in which the agent moves according to its own location in the environment. For example, the agent chooses to move up or stay where it is.

pattern once it learned what state it was in. The "medium" environment (Environment B) rewarded the agent just as often but had a lower probability of switching active states. The hard environment (Environment A) had both a low probability of switching and a low probability of rewarding the agent, making it an uncertain environment and hard to learn in (Table 1). By lowering the probability of the state switching (ϱ_w), it requires the agent to work harder to determine when they need to switch from the port they are at. When the probability of reward is low, it adds to the uncertainty of how many errors the agent will take before it switches active states.

Table 1
Three Environments Implemented

	Difficulty	p (probability of switching)	q (probability of reward)
Environment A	Hard	0.3	0.3
Environment B	Medium	0.3	0.9
Environment C	Easy	0.9	0.9

In the original study done by Vertechi et al. (2020), the mice and humans have an extra factor of energy trade-off associated with an action. This is not accounted for in the artificial neural network agents. Therefore, the environment cannot switch states unless the agent has chosen the correct active site. This is built in so that the agent isn't able to remain at the same site for the entire run after getting rewarded once.

In addition, the environment handles the reward given to the agent. At all times, one site is active; however, the agent will not get rewarded every time it chooses the correct site. This is in order to implement an additional layer of uncertainty to the environment and to replicate more accurately the complex uncertainty of the environment animals operate in. Therefore, if the agent chooses the correct port, it has an ϱ_r chance of getting rewarded and 1-ϱ_r chance of not getting rewarded.

Figure 2

Environment Python Code with descriptions of important lines.

```
# this version encodes the state in terms of the port
class SwitchingTaskAllo(gym.Env):
    metadata = {'render.modes': ['console']}

    def __init__(self, p=0.9, q=.9):
        super(SwitchingTaskAllo, self).__init__()
        self.action_space = spaces.Discrete(2)
        self.observation_space = spaces.Discrete(4)
        self.p = p #probability of switching
        self.q = q #probability of emitting reward
        self.state = np.random.choice([0, 1]) #state starts in random port 1 or 0
        self.printed_text = []

    def step(self, action):
        reward = int(np.random.rand() < self.q and self.state == action) #number less than .8 and action equals state (guess correctly) you get reward
        print(f"state:{self.state} action:{action} reward:{reward} p:{self.p} q:{self.q}") #data collection for researcher
        self.printed_text.append([self.state, action, reward, self.p, self.q])  # Append as list of state, action, reward
        if np.random.rand() < self.p and self.state == action:
            self.state = 1 - self.state  #if agent persists at non-active gate the environment won't change states until the agent guesses correctly
        done = False #keep learning after this trial
        return (action*2 + reward), reward, done, {} #first space is observation space

    def export_to_excel(self, filename):
        df = pd.DataFrame(self.printed_text, columns=["State", "Action", "Reward", "p", "q"])  # Creating DataFrame with separate columns for excel use
        df.to_excel(filename, index=False)
        print(f"Excel file '{filename}' generated successfully.")

    def reset(self): #assumes first action is
        reward = int(np.random.rand() < self.q and self.state == 0)
        if np.random.rand() < self.p and self.state == 0:
            self.state = 1 - self.state
        return reward

    def render(self, mode='console'):
        if mode != 'console':
            raise NotImplementedError()
        print(f'Current state: {self.state}')
```

3.3.2 The Agents

The first agent is the stable baselines3 DQN (Figure 3) which is a model-free neural network that uses an MLP policy (Section 2.3). This agent was run in the allocentric environment for a given q (ϱ_r) and p (ϱ_w) value, and the key parameters for this DQN are the learning rate and hidden node layers, which are tested for parameterization that resulted in the best accuracy rate.

Figure 3

DQN Python Code

```
#DQN
env = SwitchingTaskAllo(q=.3)
model = DQN("MlpPolicy", env, #network architecture (MLP = multilayer perceptron)
                      policy_kwargs=dict(net_arch=[]), #hidden layers ([] = none)
                      exploration_initial_eps=.05, #initial fraction of actions chosen randomly
                      exploration_final_eps=0.00, #final fraction of steps chosen randomly
                      exploration_fraction=0.5, #fraction of training at which final_eps is reached
                      learning_rate=.001, #How quickly to learn (often < 1)
                      buffer_size=1000,#The number of most recent experiences preserved for replay
                      target_update_interval=1,#How often to update the target Q-Network
                      train_freq = (1, "step"),#How often to train the model
                      learning_starts=0, #After how many actions to start learning
                      verbose=0) #whether to output statistics re: model training
model.learn(total_timesteps=10000)
```

The second agent is the stable baselines3 PPO (Figure 4) which utilizes an optimizing policy predictor, which makes it a slightly more complex network than the DQN. This agent was also run in the allocentric environment and utilizes a MLP Policy. Verbose was set to 0 and the learning rate and net architecture, or layers of hidden nodes, were adjusted to optimum performance.

Figure 4
PPO Python Code

```
[ ] #PPO
    env = SwitchingTaskAllo(q=.3)
    model = PPO('MlpPolicy', env, verbose=0, learning_rate=.01, policy_kwargs={'net_arch': [{'pi': [1], 'vf': [1]}]})
    model.learn(total_timesteps=10000)
```

The third agent is the stable baselines3 Recurrent PPO (Figure 5) which implements LSTM for recurrent learning. By implementing LSTM, this agent is able to keep the best running memory of past trials of all the agent types in this experiment. It also utilizes a MLP policy, and the learning rate and network architecture were adjusted to improve performance.

Figure 5
Recurrent PPO (RPPO) Python Code

```
[ ] #RPPO
    from sb3_contrib import RecurrentPPO

    env = SwitchingTaskAllo(q=.3)
    model = RecurrentPPO("MlpLstmPolicy",env, verbose=0,  learning_rate=.001, policy_kwargs={'net_arch': [{'pi': [16], 'vf': [16]}]})
    model.learn(total_timesteps=10000)
```

The agents were not trained, and instead the agent's first 10,000 timesteps in the environment were recorded to observe its progress of accuracy in a novel environment. This replicates how mice and humans in Vertechi et al.'s (2020) experiment were trained on the task. In order to determine the most efficient learner in each category for each environment, a 5x5 array of agents with varying learning rates (.0001, 0.001, 0.01, 0.1, 1) and varying hidden nodes

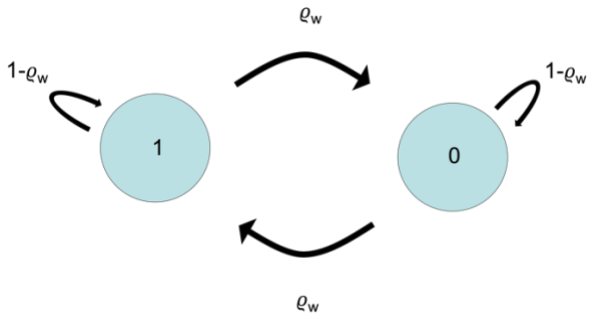
(0, 1, 2, 4, 16) were run. For each category in each environment a gradient array was run to determine the highest accuracy percentage in the last 1,000 trials run (4.1.3-5).

3.4 Procedure

Three environments were implemented (Table 1). The environment was based on a hidden Markov model (Figure 6), where the active state is determined by an underlying model that is not visible to the participant. The environment would start in state 1 or 0 randomly. The agent then randomly guesses what state the environment is in, and then the environment tells the agent the reward it received for its guess and what state it chose. The state of the environment and the state that the agent chose are printed as '1' or '0'; however, it can help to imagine them as 'left' and 'right.' Once the environment was implemented, 25 agents per each RL network category were run in the environment. Then the optimal agent was chosen based on the highest accuracy rate in the last 1,000 trials for further analyzation.

Figure 6

Hidden Markov Model determining the underlying the active states of the environment.



4. DISCUSSION

4.1 Results

In order to determine the optimal parameters for each agent in the different environments, a 5x5 array of agents were run in each environment (Section 3.3). The agents from each array with the highest accuracy, identified by the lightest shaded square, were selected and their individual learning progress was analyzed. In order to have a more comprehensive understanding of when the agent reached its threshold accuracy rate, each agent was run for 10,000 timesteps in each environment.

4.1.1 1-back Memory Comparison

An expected performance based on Bayesian 1-back memory utilization was calculated (Section 4.1.2). The probability of active state switching (ϱ_w) determined which conditions to use as the expected accuracy rates. If the probability was greater than 0.5, such as in Environment C, then conditions Alpha and Delta were set as the upper and lower bounds for expected performance. When ϱ_w was less than 0.5, such as in Environments A and B, conditions Beta and Gamma were set as the upper and lower bounds for expected performance. It would be expected that a memoryless agent would perform at about this standard (Section 4.1.2).

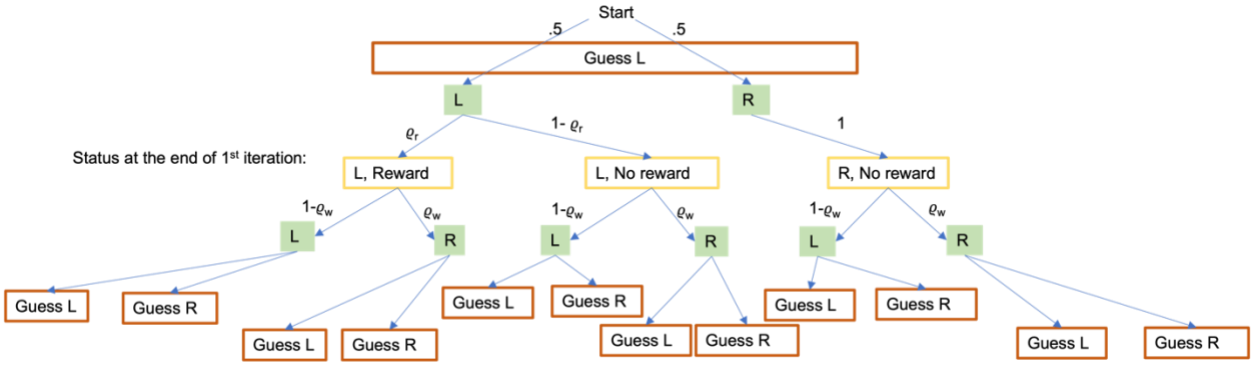
4.1.2 Calculating Optimal 1-back Memory Performance

Figure 7 is a tree diagram of the possible outcome paths available for the agent, taking probability of getting rewarded and probability of the environment's active state switching into account. The environment is determined by a hidden Markov model that has been run long enough so that it can be assumed there is a 50% chance of the environment starting with the left site active and 50% chance of it starting with the right state active (Section 3.2.4). These are represented by the first arrows labeled with '.5'. Given that the agent has a 50% chance of being

correct if they guess *left* first and a 50% chance of being correct if they guess *right* first, the preceding probabilities do not depend on their initial guess. For ease, we assume the first guess is *left*.

Figure 7: Probability Tree for Average Correct Guesses based on 1-back Memory

Orange boxes indicate the agent's guess. The green boxes indicate the active state of the environment. The yellow boxes indicate the status of the environment and the feedback the agent received.



If you guess *left*, and the active state is the left site, then you have a probability of ϱ_r of getting rewarded and a 1-ϱ_r probability of getting no reward. If you guess *left* and the active state is the right site, then you are guaranteed to get no reward.

After the status update at the end of iteration one, where the agent either got a reward or did not get a reward, the agent can either guess *left* again, or they can choose to switch sites and guess *right*. The environment has 1-ϱ_w probability of staying and keeping the same active site, and a ϱ_w probability of switching active sites. The guess of the agent is independent of what happens to the environment in the second iteration. Therefore, by multiplying the three steps of probabilities leading up to the second iteration, one can calculate the probability of the environment being active in the left state or active in the right state. From there, one can

calculate the probability of the agent guessing the correct active state of the environment in the second iteration.

If ϱ_w is less than 0.5, then use Option Alpha to calculate the upper bound of the optimal accuracy expected by an agent with Bayesian 1-back memory. This indicates that if the agent gets rewarded the first iteration, the best choice is for the agent is to stick with the same guess for the second iteration. For the lower bound, use Option Delta to calculate the optimal accuracy. This indicates that if the agent does not get rewarded the first iteration, your best choice is to switch your guess to the other active site. This is because with a ϱ_w of less than 0.5, it is most likely that the environment will remain in the same active state for the second iteration.

Option Alpha: *If you get rewarded the first iteration, what is the probability you will get rewarded if you guess Left again?*

There are two possible states of the environment where you get rewarded iteration one, and then guess *left*: one where the active state stays at the left site and one where the active state switches to the right site. Out of these possible states, there is only one where you would be guessing correctly. Therefore, the probability of guessing correctly if you got rewarded the first iteration and guessed left again would be:

$$\frac{0.5 * \rho_r * (1 - \rho_w)}{0.5 * \rho_r * (1 - \rho_w) + 0.5 * \rho_r * \rho_w}$$

Simplifies to...

$$1 - \rho_w$$

Option Delta: *If you didn't get rewarded the first iteration, what is the probability you will get rewarded if you guess Left again?*

There are four possible states of the environment that are possible with the conditions that the agent doesn't get rewarded iteration one, and then guesses *left* again: one where the left state was active for iteration one (it just didn't reward the agent) and then remains active for iteration two, one where the left state was active for iteration one but switches to the right site for iteration two, one where the initial active state of the environment was the right site and it switches to be active

at the left site, and then one where the initial active state of the environment was the right site and it stays active at the right site for iteration two. Out of these possible states, there are only two states in which the agent would be correct by guessing *left* being active for iteration two. Therefore, by choosing the left site again for iteration two, the agent has the probability below of being correct:

$$\frac{0.5 * (1 - \rho_r) * (1 - \rho_w) + 0.5 * 1 * \rho_w}{0.5 * (1 - \rho_r) * (1 - \rho_w) + 0.5 * (1 - \rho_r) * \rho_w + 0.5 * 1 * \rho_w + 0.5 * 1 * (1 - \rho_w)}$$

Simplifies to...

$$\frac{1 - \rho_r + \rho_r \rho_w}{2 - \rho_r}$$

If ϱ_w is greater than 0.5 then use Option Beta to calculate the upper bound of the optimal accuracy based on 1-back memory. This indicates that if the agent gets rewarded the first iteration, their best guess is to switch to a different active site because the environment is most likely to switch active sites in the second iteration. If the agent does not get rewarded, use Option Gamma use to solve for the lower optimal bound. This indicates that the agent should stay at the same port for their second guess if they did not get rewarded the first time, because it is most likely that the environment will switch into the active site the agent is already at.

Option Beta: *If you get rewarded the first iteration, what is the probability you will get rewarded if you guess Right next time?*

There are two possible states of the environment where you get rewarded iteration one, and then guess *right*: one where the active state stays at the left site and one where the active state switches to the right site. Out of these possible states, there is only one where you would be guessing correctly. Therefore, the probability of guessing correctly if you got rewarded the first iteration and guessed *right* for the second iteration would be:

$$\frac{0.5 * \rho_r * \rho_w}{0.5 * \rho_r * (1 - \rho_w) + 0.5 * \rho_r * \rho_w}$$

Simplifies to...

$$\rho_w$$

Option Gamma: *If you didn't get rewarded the first iteration, what is the probability you will get rewarded if you guess Right next time?*

There are four possible states of the environment that are possible with the conditions that the agent doesn't get rewarded iteration one, and then guesses *right* for iteration two: one where the left state was active for iteration one (it just didn't reward the agent) and then remains active for iteration two, one where the left state was active for iteration one but switches to the right site for iteration two, one where the initial active state of the environment was the right site and it switches to be active at the left site, and then one where the initial active state of the environment was the right site and it stays active at the right site for iteration two. Out of these possible states, there are only 2 states in which the agent would be correct by guessing *right* as active for iteration two. Therefore, by choosing *right*, the agent has the probability below of being correct:

$$\frac{0.5 * (1 - \rho_r) * \rho_w + 0.5 * 1 * (1 - \rho_w)}{0.5 * (1 - \rho_r) * (1 - \rho_w) + 0.5 * (1 - \rho_r) * \rho_w + 0.5 * 1 * \rho_w + 0.5 * 1 * (1 - \rho_w)}$$

Simplifies to...

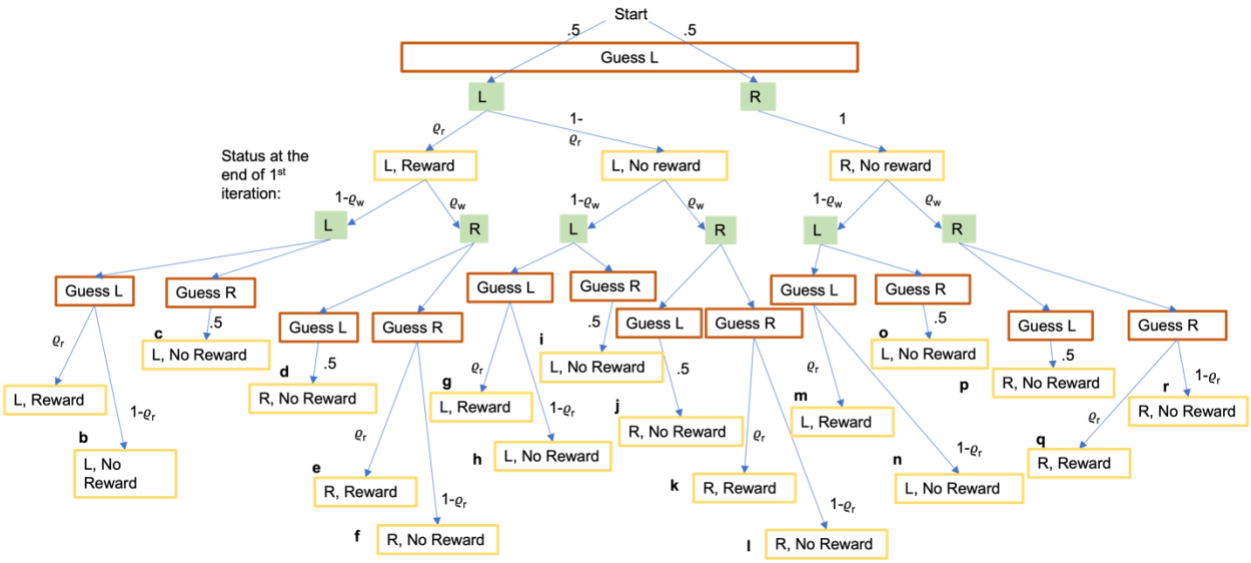
$$\frac{1 - \rho_r \rho_w}{2 - \rho_r}$$

For environment A, the highest optimal bound was 70%. The lowest optimal bound was 52.53%. For environment B, the highest optimal bound was 70%. The lowest optimal bound was 66.36%. For environment C, the highest optimal bound was 90%. The lowest optimal bound was 82.72%.

For calculating the optimal accuracy rate based on Bayesian probabilities of the events happening, the only steps necessary are the ones up to the second guess. For these optimal probabilities, it does not matter if the agent got a reward or not; it matters if it guessed correctly. However, one could continue down the tree to calculate how often the agent should optimally be getting rewarded on the second guess based on 1-back memory (Figure 8).

Figure 8

Probability Tree for 1-back memory calculations through the status at the end of the 2nd iteration. This tree is needed to calculate the optimal rate of being rewarded based on 1-back memory. Orange boxes indicate the agent's guess. The green boxes indicate the active state of the environment. The yellow boxes indicate the status of the environment and the feedback the agent received. The bolded letters indicate the final step of each possible pathway, which are used to indicate the possibility of it happening calculated in Figures 1-3 in Index.



4.1.3 Environment A

The data shows that all agents had the lowest accuracy rate in this environment, which is consistent with it being the most difficult environment. The best DQN had a learning rate of 0.001 and no hidden layers (Figure 9). Its threshold performance was an accuracy rate of 61.03%. This agent performed in the middle of the optimal range, suggesting that a DQN MLP agent has memory equivalent to an optimal 1-back Bayesian memory model (Figure 10). It learned in about 380 timesteps, and its progress improved significantly over timesteps 260- 380. For the remainder of the timesteps, its accuracy hovered between 55% and 75%, reaching its maximum accuracy rate of 80% once at timestep 2,220.

Figure 9

DQN array in Environment A. Refer to Table 1 (Section 3.2.1) for the specifics of the environments. Hidden node layers were adjusted to either 0, 1, 2, 4, and 16. The learning rates were set to either 1, 0.1, 0.01, 0.001, and 0.0001. The bar to the right indicates the average correct guesses in the last 1000 timesteps for each agent. These arrays were used to determine the best parameters for the type of agent given the environment conditions.

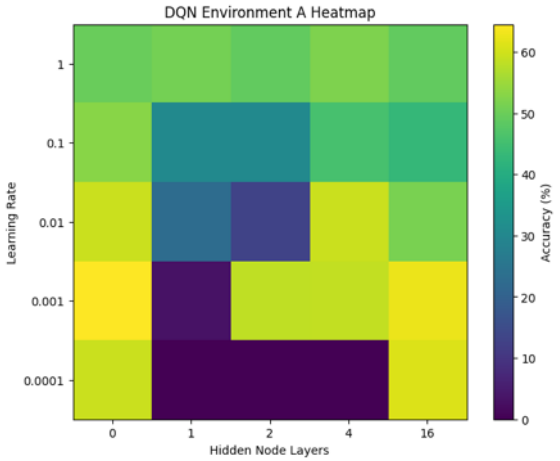
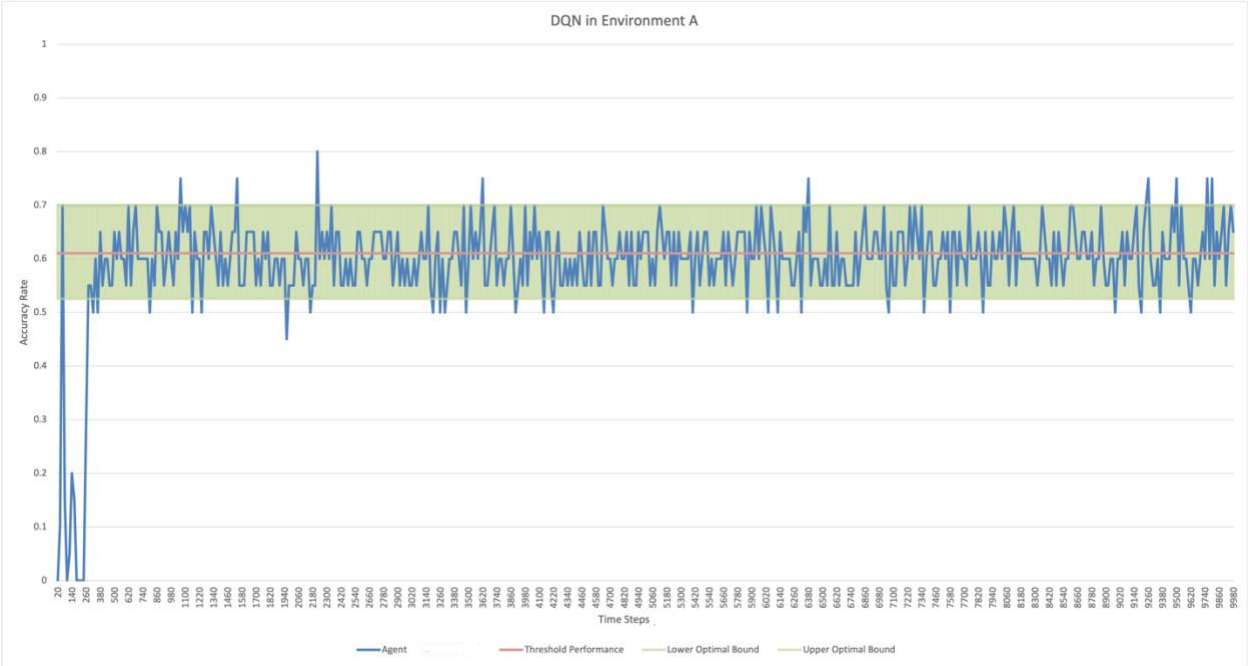


Figure 10

DQN's progress in the run in Environment A measured by average correct guesses every 20 timesteps. The green region is bounded by the lower optimal 1-back memory performance and the upper optimal performance. The orange line indicates the average of the agent over the last 2,000 timesteps, indicating an average threshold performance.



The best PPO had a learning rate of 0.01 and 1 hidden node (Figure 11). Its threshold was an accuracy rate of 56.12% (Figure 12). It took about 7,300 timesteps to learn which was much

slower than the DQN. Additionally, the data for this agent was very noisy which made it unclear how confident the agent was in this environment, despite showing a trend of learning.

Figure 11

PPO array in Environment A. Refer to Table 1 (Section 3.2.1) for the specifics of the environments. Hidden node layers were adjusted to either 0, 1, 2, 4, and 16. The learning rates were set to either 1, 0.1, 0.01, 0.001, and 0.0001. The bar to the right indicates the average correct guesses in the last 1000 timesteps for each agent. These arrays were used to determine the best parameters for the type of agent given the environment conditions.

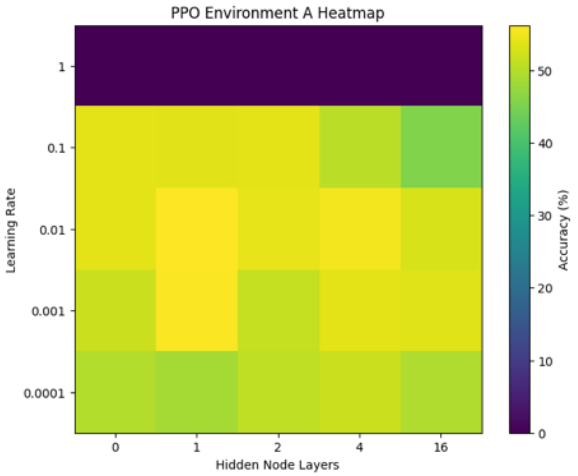
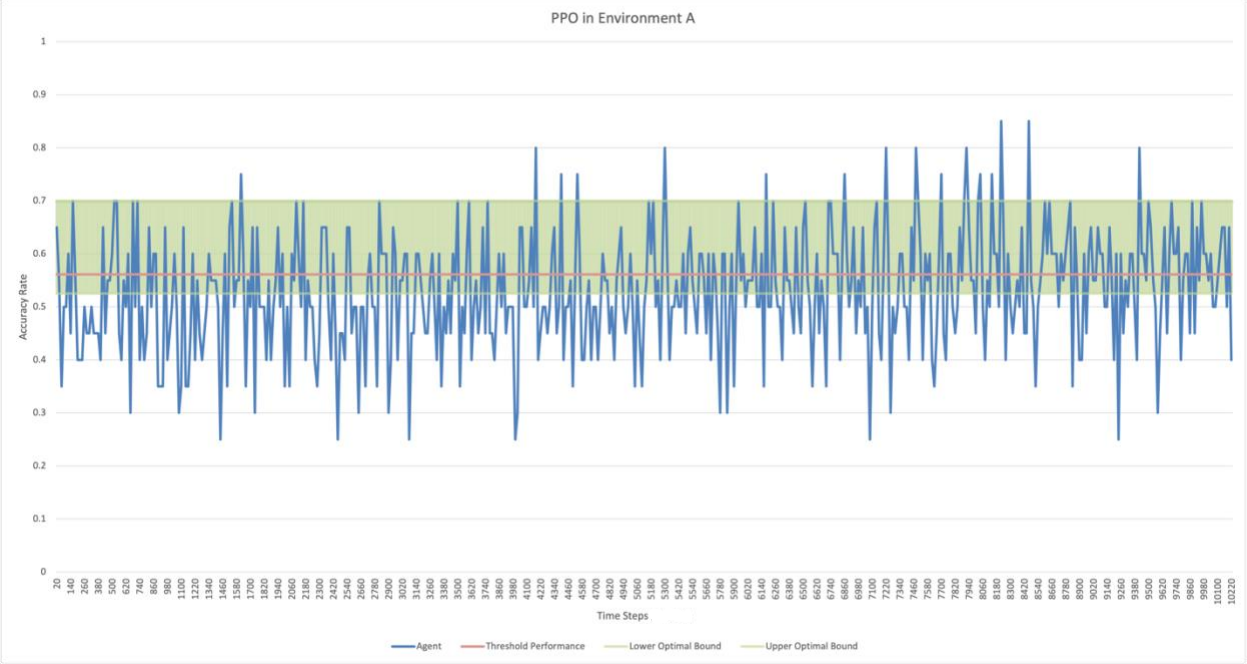


Figure 12

PPO's progress in the run in Environment A measured by average correct guesses every 20 timesteps. The green region is bounded by the lower optimal 1-back memory performance and the upper optimal performance. The orange line indicates the average of the agent over the last 2,000 timesteps, indicating an average threshold



The best RPPO had a learning rate of 0.001 and 16 hidden nodes (Figure 13). Its threshold was an accuracy rate of 60.87% (Figure 14). When run for 15,000 timesteps the average

accuracy continued to be about 60%. It took about 5,500 timesteps to reach the threshold average. However, it learned nearly as well as the DQN and significantly better than the PPO.

Figure 13

RPPO array in Environment A. Refer to Table 1 (Section 3.2.1) for the specifics of the environments. Hidden node layers were adjusted to either 0, 1, 2, 4, and 16. The learning rates were set to either 1, 0.1, 0.01, 0.001, and 0.0001. The bar to the right indicates the average correct guesses in the last 1000 timesteps for each agent. These arrays were used to determine the best parameters for the type of agent given the environment conditions.

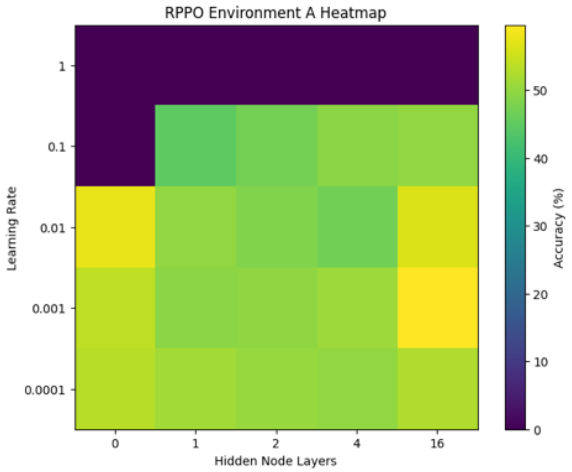
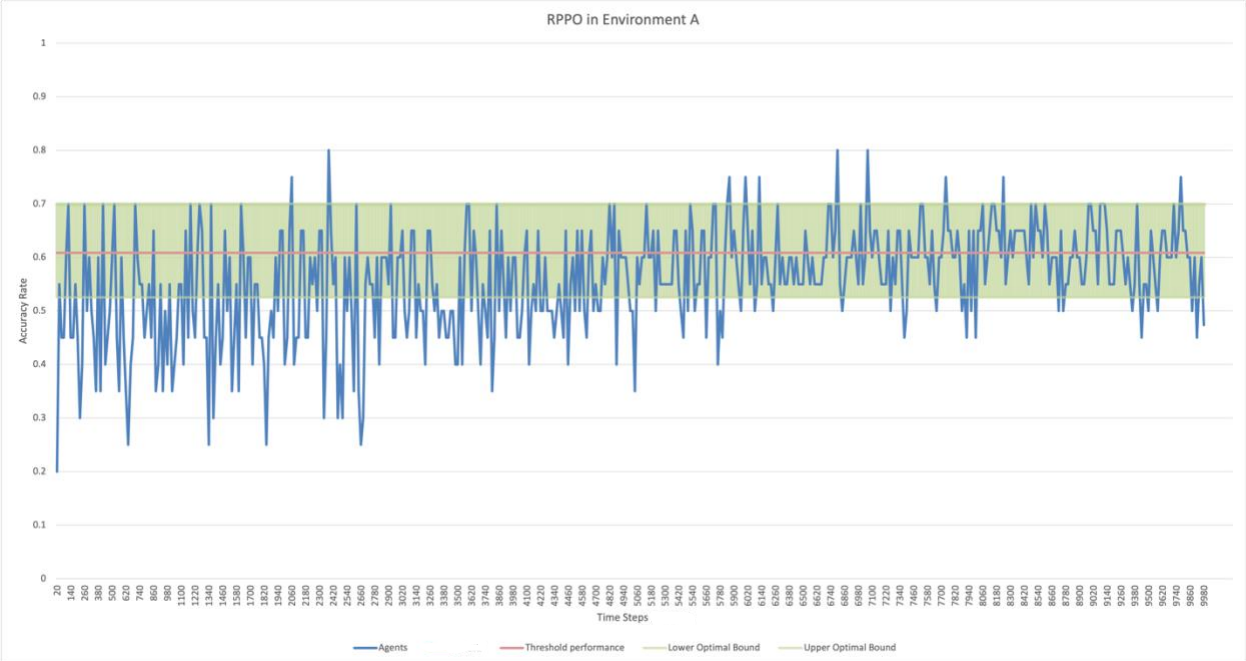


Figure 14

RPPO's progress in the run in Environment A measured by average correct guesses every 20 timesteps. The green region is bounded by the lower optimal 1-back memory performance and the upper optimal performance. The orange line indicates the average of the agent over the last 2,000 timesteps, indicating an average threshold performance.



In the difficult environment, the DQN and RPPO did significantly better than the PPO and also learned significantly faster than the PPO agent. The PPO agent's data also had significantly more fluctuation in accuracy throughout the entire run, compared with the RPPO and DQN that stabilized to stay within a rough 0.2-differene window.

4.1.4 Environment B

In the medium-difficult environment, the best DQN had a learning rate of 0.0001 and no hidden layers (Figure 15). Its threshold was an accuracy rate of 75.46% (Figure 16). It did significantly better than the optimal bound for 1-back memory, which suggests that this model could possibly have a better Bayesian memory than 1-back, although more research is needed to determine to what extent. This agent learned in about 7,300 timesteps, with a significant jump in accuracy at 7,220 timesteps, evidence that the agent suddenly learned a Q-function that was accurate at performing in this environment.

Figure 15

DQN array in Environment B. Refer to Table 1 (Section 3.2.1) for the specifics of the environments. Hidden node layers were adjusted to either 0, 1, 2, 4, and 16. The learning rates were set to either 1, 0.1, 0.01, 0.001, and 0.0001. The bar to the right indicates the average correct guesses in the last 1000 timesteps for each agent. These arrays were used to determine the best parameters for the type of agent given the environment conditions.

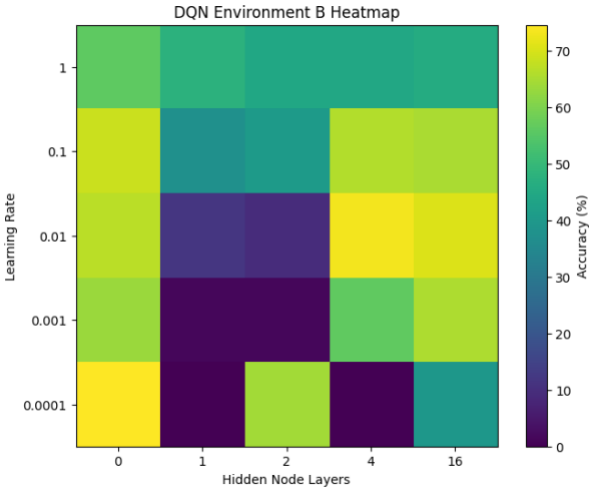
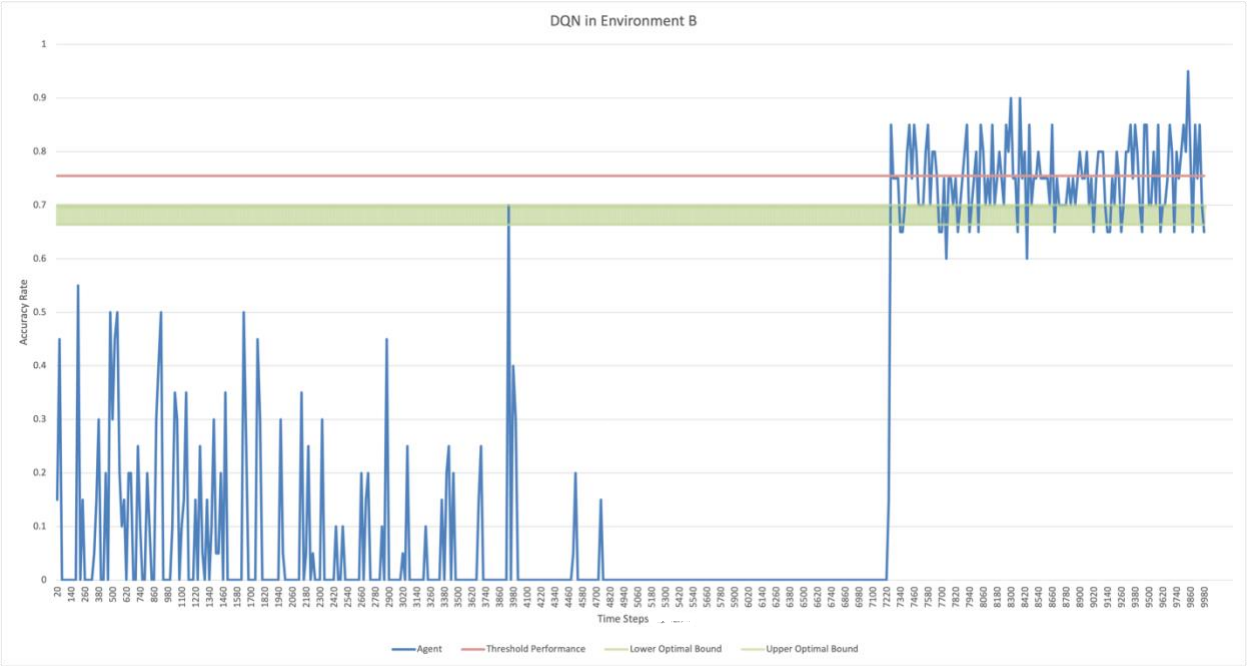


Figure 16

DQN's progress in the run in Environment B measured by average correct guesses every 20 timesteps. The green region is bounded by the lower optimal 1-back memory performance and the upper optimal performance. The orange line indicates the average of the agent over the last 2,000 timesteps, indicating an average threshold performance.



The best PPO had a learning rate of 0.01 and one hidden node (Figure 17). Its threshold was an accuracy rate of 63.97% (Figure 18). This agent performed below the optimal bound given 1-back memory and learned after 6,500 timesteps. This was also the only agent to perform below the 'optimal' bound.

Figure 17

PPO array in Environment B. Refer to Table 1 (Section 3.2.1) for the specifics of the environments. Hidden node layers were adjusted to either 0, 1, 2, 4, and 16. The learning rates were set to either 1, 0.1, 0.01, 0.001, and 0.0001. The bar to the right indicates the average correct guesses in the last 1000 timesteps for each agent. These arrays were used to determine the best parameters for the type of agent given the environment conditions.

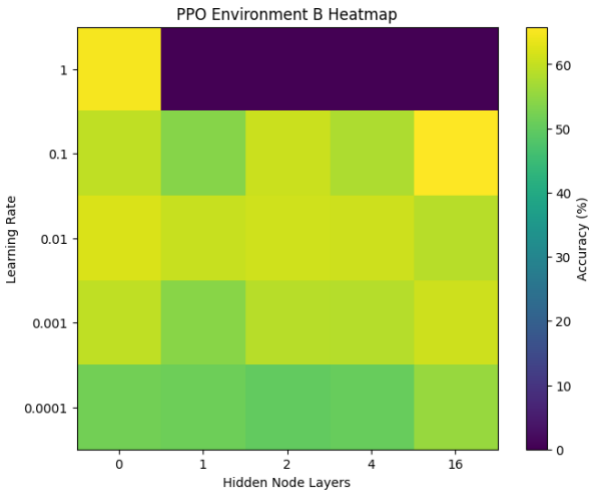
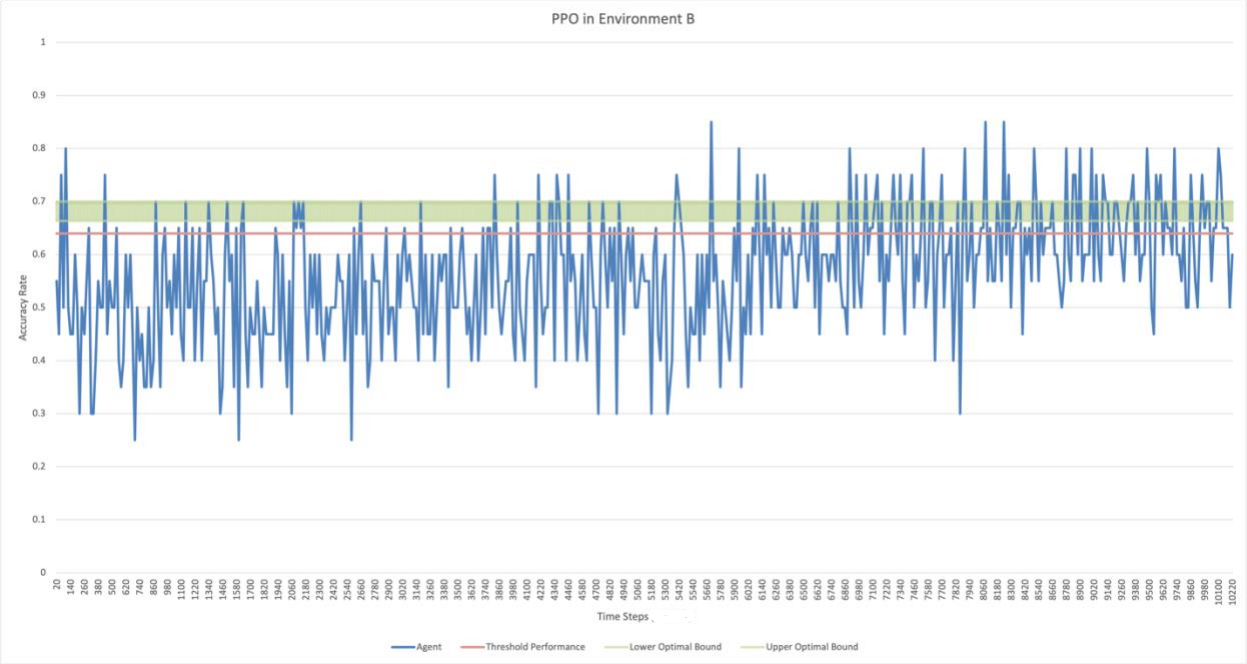


Figure 18

PPO's progress in the run in Environment B measured by average correct guesses every 20 timesteps. The green region is bounded by the lower optimal 1-back memory performance and the upper optimal performance. The orange line indicates the average of the agent over the last 2,000 timesteps, indicating an average threshold performance.



The best RPPO had a learning rate of 0.001 and 16 hidden nodes (Figure 19). Its threshold was an accuracy rate of 67.12% which was within the optimal range based on 1-back memory (Figure 20). Additionally, this agent appeared to learn in about 1,220 timesteps, which was the fastest of all agents in this environment. There was also a period between 2,300 timesteps and 5,060 timesteps where the agent was doing better than it did in the remaining 5,000 timesteps.

Figure 19

RPPO array in Environment B. Refer to Table 1 (Section 3.2.1) for the specifics of the environments. Hidden node layers were adjusted to either 0, 1, 2, 4, and 16. The learning rates were set to either 1, 0.1, 0.01, 0.001, and 0.0001. The bar to the right indicates the average correct guesses in the last 1000 timesteps for each agent. These arrays were used to determine the best parameters for the type of agent given the environment conditions.

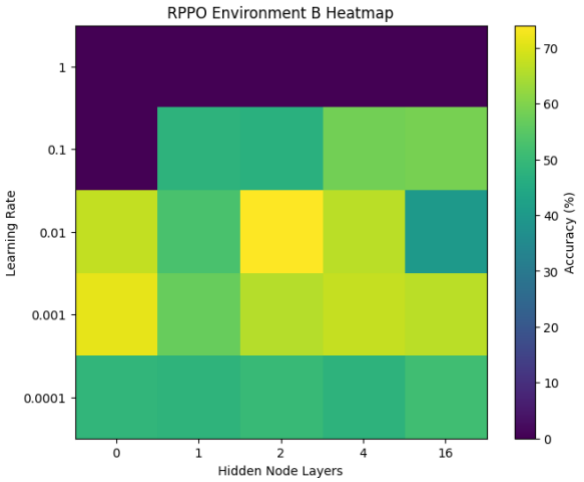
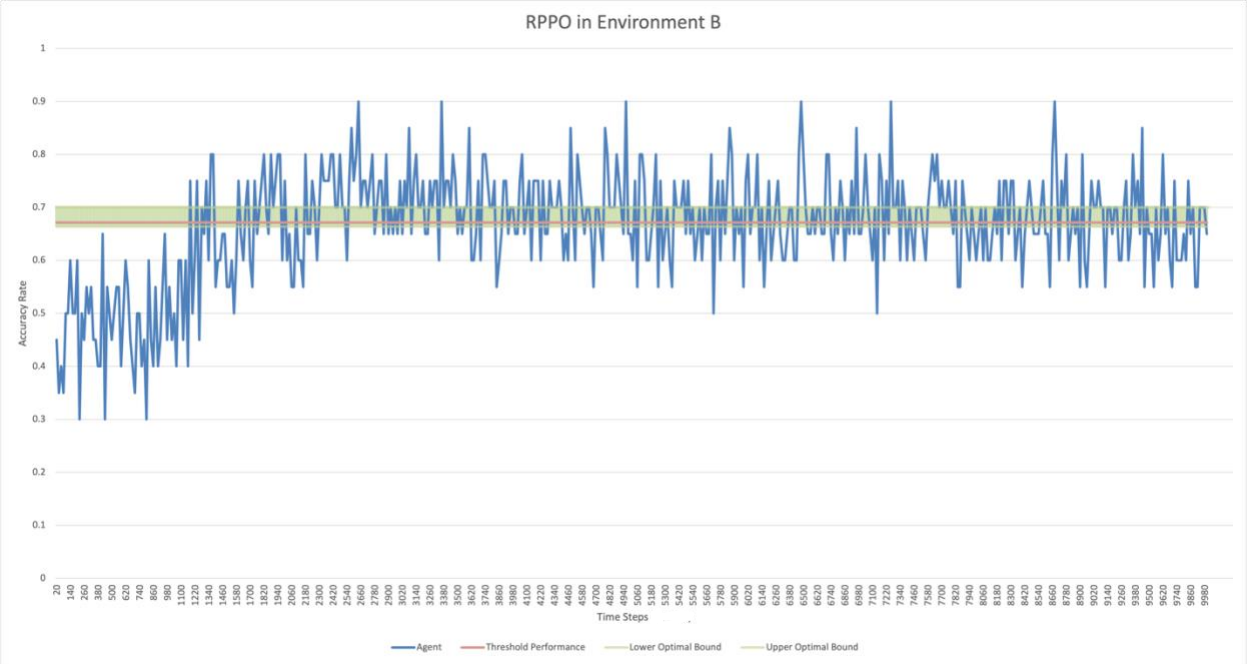


Figure 20

RPPO's progress in the run in Environment B measured by average correct guesses every 20 timesteps. The green region is bounded by the lower optimal 1-back memory performance and the upper optimal performance. The orange line indicates the average of the agent over the last 2,000 timesteps, indicating an average threshold performance.



The DQN did significantly better in the medium environment, although it learned the slowest of the three agents. One noteworthy observation is that while the PPO did improve its performance in Environment B from Environment A, it did worse in Environment B compared to the 1-back memory expected accuracy rate. This is surprising and requires further investigation.

4.1.5 Environment B

The best DQN had a learning rate of 0.1 and 4 hidden layers (Figure 21). Its threshold was an accuracy rate of 89.01% (Figure 22). This agent performed at about the same rate as the upper optimal bound. Although it started guessing at this rate at around 750 timesteps, it leveled out and appeared to learn after about 1,480 timesteps.

Figure 21

DQN array in Environment C. Refer to Table 1 (Section 3.2.1) for the specifics of the environments. Hidden node layers were adjusted to either 0, 1, 2, 4, and 16. The learning rates were set to either 1, 0.1, 0.01, 0.001, and 0.0001. The bar to the right indicates the average correct guesses in the last 1000 timesteps for each agent. These arrays were used to determine the best parameters for the type of agent given the environment conditions.

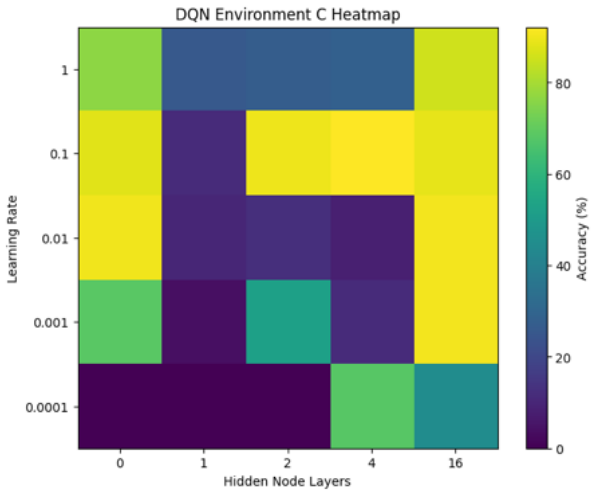
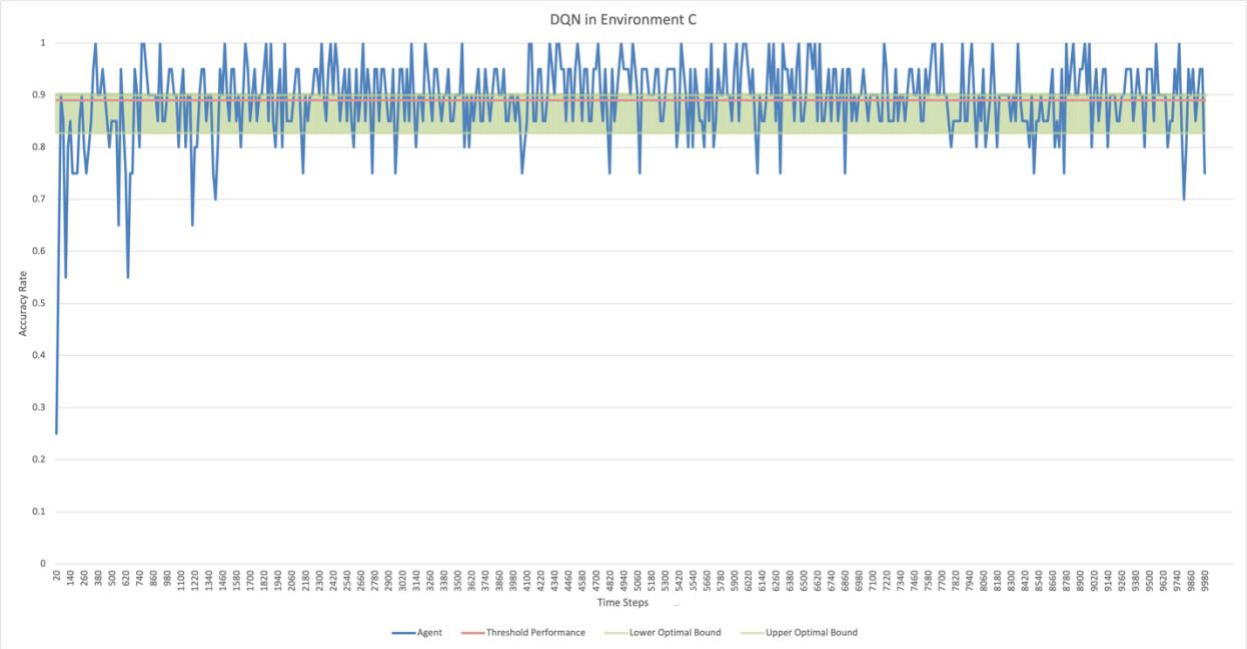


Figure 22

DQN's progress in the run in Environment C measured by average correct guesses every 20 timesteps. The green region is bounded by the lower optimal 1-back memory performance and the upper optimal performance. The orange line indicates the average of the agent over the last 2,000 timesteps, indicating an average threshold performance.



The best PPO had a learning rate of 1 and no hidden nodes (Figure 23). Its threshold was an accuracy rate of 90.40% (Figure 24). This agent's accuracy plateaued at the upper optimal bound. It learned in about 2,180 timesteps and made considerable improvement at timestep 2,060-2,180.

Figure 23

PPO array in Environment C. Refer to Table 1 (Section 3.2.1) for the specifics of the environments. Hidden node layers were adjusted to either 0, 1, 2, 4, and 16. The learning rates were set to either 1, 0.1, 0.01, 0.001, and 0.0001. The bar to the right indicates the average correct guesses in the last 1000 timesteps for each agent. These arrays were used to determine the best parameters for the type of agent given the environment conditions.

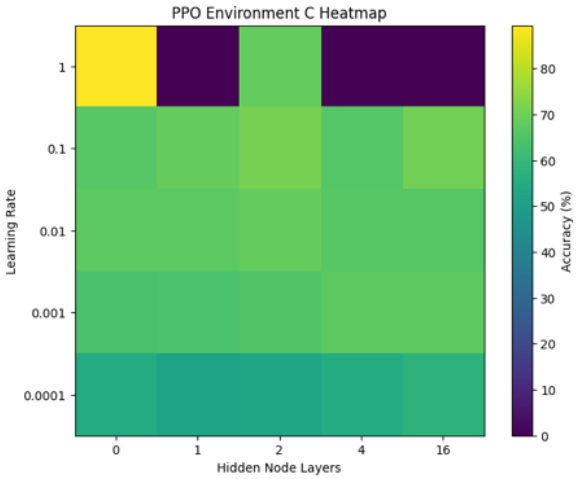
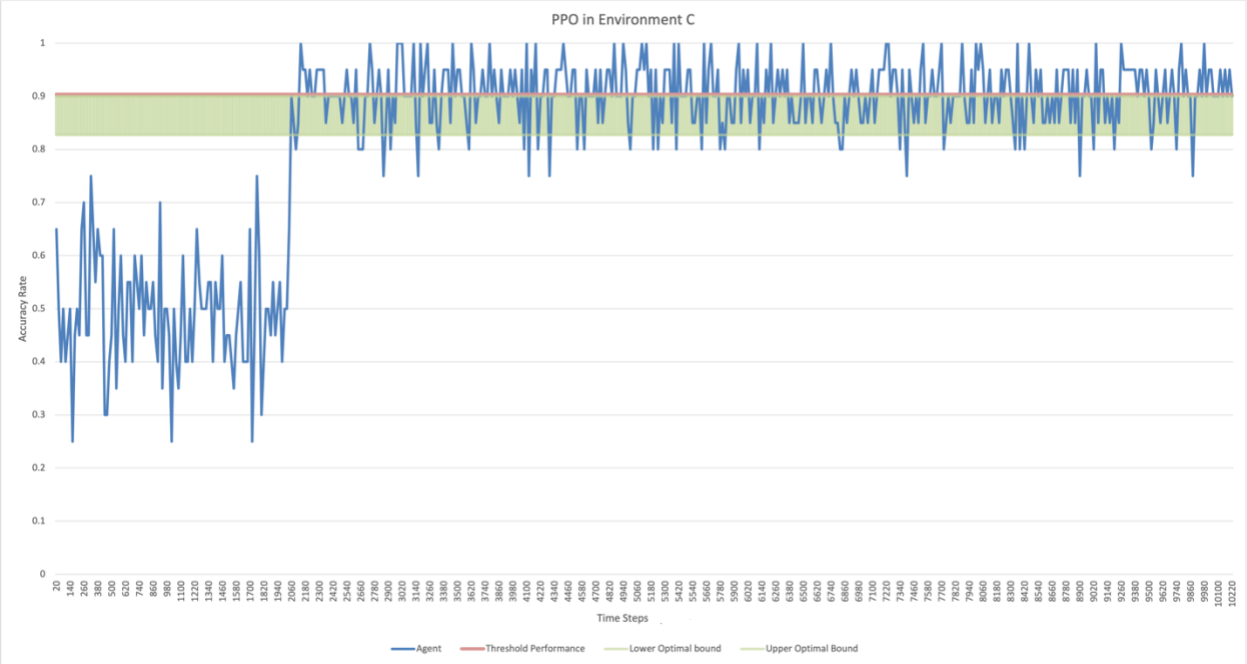


Figure 24

PPO's progress in the run in Environment C measured by average correct guesses every 20 timesteps. The green region is bounded by the lower optimal 1-back memory performance and the upper optimal performance. The orange line indicates the average of the agent over the last 2,000 timesteps, indicating an average threshold performance.



The best RPPO had a learning rate of 0.01 and 16 hidden nodes (Figure 25). Its threshold was an accuracy rate of about 90.40% which was just at the upper optimal bound (Figure 26). It appeared to have learned after about 2,420 timesteps and hovered between an accuracy rate of 85% and 100%.

Figure 25
RPPO array in Environment C. Refer to Table 1 (Section 3.2.1) for the specifics of the environments. Hidden node layers were adjusted to either 0, 1, 2, 4, and 16. The learning rates were set to either 1, 0.1, 0.01, 0.001, and 0.0001. The bar to the right indicates the average correct guesses in the last 1000 timesteps for each agent. These arrays were used to determine the best parameters for the type of agent given the environment conditions.

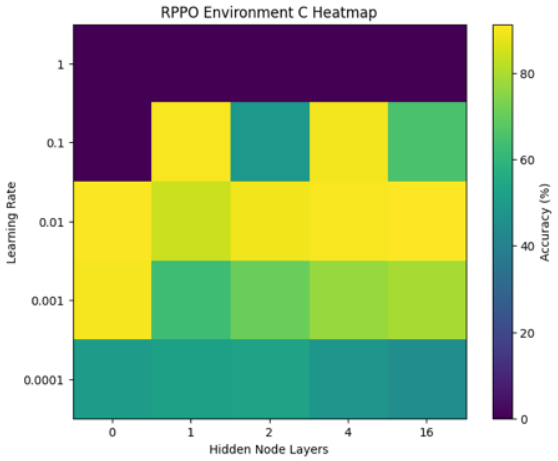
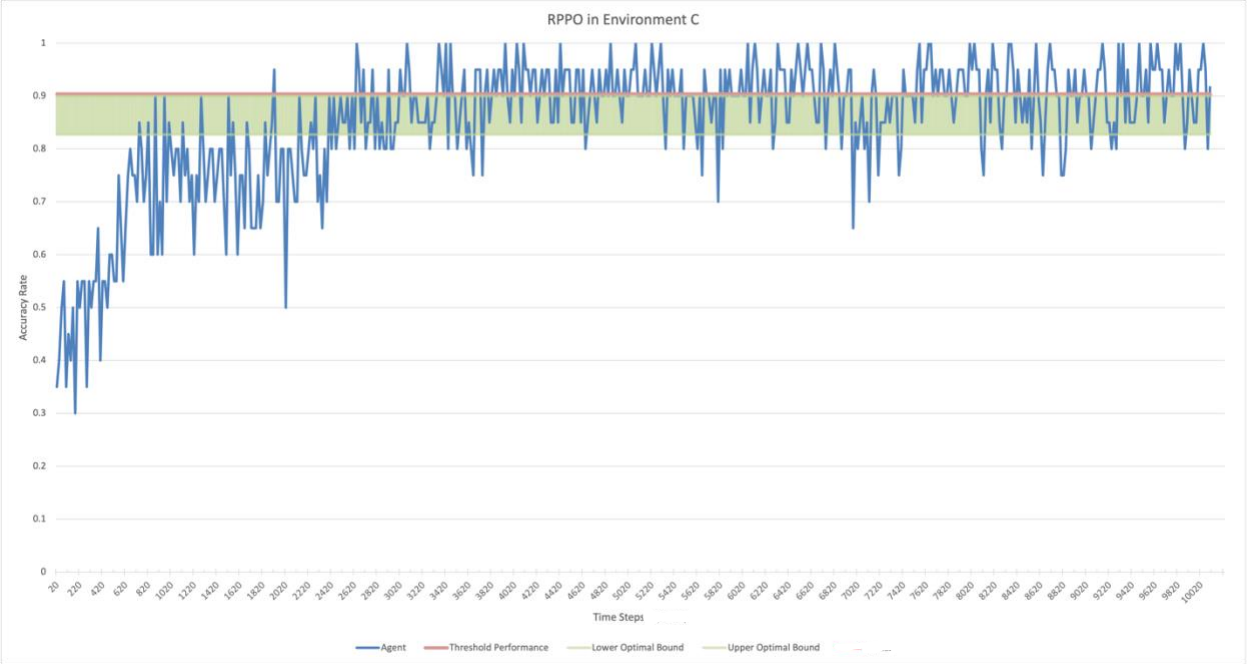


Figure 26
RPPO's progress in the run in Environment C measured by average correct guesses every 20 timesteps. The green region is bounded by the lower optimal 1-back memory performance and the upper optimal performance. The orange line indicates the average of the agent over the last 2,000 timesteps, indicating an average threshold performance.



Again, we see the RPPO had a higher accuracy rate in the middle of the run, and then drops slightly for the remainder of the timesteps. This is slightly surprising, given that it would be expected that the accuracy rate would increase in the last 5,000 timesteps. Additionally, while the RPPO plateaued in performance in the upper optimal bound, it guessed multiple 20-step

periods perfectly, which is noteworthy.

4.2 General Take Aways

In this experiment, an untrained DQN, PPO, and Recurrent PPO were run in three environments of varying difficulty, which was varied by changing the probability of the environment switching active states and by changing the probability of the environment providing a reward to the agent. In the easy environment, both PPO agents performed better than the DQN agent. However, as difficulty of the environment increased, the DQN outperformed the PPO agent significantly, and performed comparable to the RPPO agent.

The first main observation was that all agents did not perform as well as initially predicted. However, the data does support most of the research regarding model-free neural networks. The agents' performance is consistent with their inability to create an internal model of the environment, and thus were very limited in the ways they could learn the environment and did not perform exceedingly well. Notably, given their inability to represent the environment through an internal model, these ANNs performed optimally well compared to an agent with optimal Bayesian 1-back memory.

Second, across all three environments, the DQN appeared to have a moment where it figured out the environment, and within a few timesteps had learned the environment. By looking at the progress of all three DQN agents (Figures 10, 16, 22), there is a clear jump from a guessing accuracy rate to an optimal accuracy rate. This is a signature of DQNs, as it takes them time to explore different pathways or options as it develops an algorithm to calculate the Q-scores, and once they find a pathway that works, the algorithm for calculating the Q-scores becomes quite accurate. On the other hand, both PPO agents started off performing at a rate higher than the DQN and slowly increased in accuracy to the threshold.

The third observation was that their performance decreased with the change in reward frequency, a signature of stimulus-bound strategy being utilized. This was described and predicted by Vertechi et al. (2020). Additionally, the difference in accuracy in the last 2000 timesteps for each agent is much greater in Environment A than it was in Environment B, and even more so than Environment C. This is another signal that these agents have no internal model of the environment.

Finally, I found it most surprising that the DQN performed better than the RPPO in the medium-difficult environment. Given that the DQN had a limited memory, and the RPPO had an LSTM architecture that should have been able to utilize nearly all of the timesteps the agent experienced, I would expect that it would be easier for the RPPO to learn in a more difficult environment.

4.3 Possible Explanations for the Data and Deeper Dives

One explanation for why the PPO agents did worse is that they were not trained. PPO updates its policy after every epoch; however, in this experiment the PPO did not update its policy until the end of the timesteps recorded. To explore this further, one could add a parameter n_steps that identifies how many timesteps are measured in an update for each environment (n_env). Additionally, one could add a parameter that defines what timestep the agent should start learning on. We did explore adding an original timestep parameter and include 2 and 10,000 n_steps, and the agents did considerably worse with these parameters added. However, one could run a different array with these parameters to see if the PPO agent could learn better.

It was also interesting that none of the agents exceeded the 1-back (essentially memoryless) memory expected performance, besides the DQN one time, especially given that the DQN and Recurrent PPO both have a memory capability. The DQN had a buffer size set to 1,000, which

meant that these DQN agents had 1,000 experiences to pull from when calculating the Q-values, and the RPPO utilized LSTM. This could partly also be explained by the minimal parameters implemented in both PPO agents. Regardless, further investigation is necessary to determine the discrepancy.

This behavior is also consistent with a stimulus-bound strategy described by Vertechi et al. (2020). The agents seem to only be affected by the current stimulus or information present to it, instead of it drawing on a prediction based on an internal model of the environment's structure. Additionally, all three types of model-free agents' performance decreases by at least 7% across all types of agents between the medium state to the hard state where only the frequency of rewards (q) changes. Thus, these agents never seemed to learn at the same level as the humans and mice in Vertechi et al. (2020), and instead persist to perform in dependence with the frequency of rewards. This is also expected by the architecture of model-free agents who only act based on rewards associated with actions (Section 2.3).

It would be of use to do a deeper analysis of the agents' data. One could analyze how many no-reward steps it took before the agent switched sites, and if that number changed based on the probability of rewards being given (Vertechi et al. 2020, 170-171). Given that the probability of reward affected how well these agents did in the environment, even with the same environmental conditions, I would predict that the length of time spent at a port would increase for these agents as the probability of reward decreased. Additionally, in future experiments it would be interesting to run a trained Recurrent PPO in this environment to see how it learns. By training the RPPO, it could provide a better understanding of the agent's strategy; would it use a more inference-based strategy and perform better, or is the model-free architecture is a limitation regardless of training and memory of agent?

Another observation was that the DQN learned significantly quicker than both agents utilizing PPO—by nearly 1,000 timesteps. One factor that would be worth exploring further is the effect that exploration rate had on the speed of the agent's learning. The exploration rate was set to 0.05, so it was relatively low, which could have helped the agent quickly choose the maximum Q-scores, or one reliable one. This would be supported by the minimal noise evident in the learning progress of the DQN agents. Once the agent reaches its threshold accuracy, it hovers within +/- 0.1 of that average rate. By increasing the exploration rate, I would expect to see an increase in time it takes the agent to learn the environment but could also improve the agent's ability to perform the task.

Both of these last findings regarding how well the DQN performed are surprising. In research done by Kozlica et al. (2023), they found that the PPO outperformed the DQN in a sorting task across several evaluation metrics. When looking at the accuracy rate of their DQN and PPO compared to ours, the performances seem opposite of our findings. One difference between the research methods is that the sorting task involved an action space of much higher dimensions, which could explain why the DQN outperformed the PPOs in our study. Given the low state and space dimensions of our task, the benefits of policy optimizations might not have been helpful.

4.5 Proposals

In future work, this procedure could be done with an additional element for tracking reaction times to see how predictive processing and updating could be affecting performance. A proposed experiment would be to create a computer game task, similar to the one used by Vertechi et al. (2020), but with a much simpler structure. Participants would be given a computer game where they are asked to guess 0's and 1's as accurately as possible, instead of guiding a

figure to different active ports. They will receive feedback after each guess and will be asked to guess as quickly as they can while being accurate, and to continue this process until the game is over.

The computer would act as the environment and the game would use either a preselected pattern or a hidden Markov model, similar to the one run on the neural networks, that generates numbers as the game continues. These numbers will represent an active state of the game's environment and would follow the same probabilistic constraints. The computer will also track the reaction time between it giving the participant feedback and their next guess.

By tracking reaction time, one could better understand when an individual might be feeling more confident in their understanding of the environment, and how a surprising input might slow down the computation or decision-making process for the next trial or guess. If reaction times were roughly the same throughout a series of correct guesses, and never shortened with familiarity or lengthen with confusion, then there would be more evidence for stimulus-bounded decision strategy. In this case, it might suggest that the participants are only taking the current state into account when making a guess. On the other hand, if reaction times were significantly different, this could suggest that individuals are developing a policy for decision making that develops a perceptual expectation and is surprised and delayed in the next state when confronted with unexpected information. This would be consistent with studies done by King et al. (2010) on increased classical RF neural activity when prediction errors occur, and studies by Palmer (2015) that show that there is an increase in neural activity when preceding information is new to the retinal ganglion cells.

Both the executed and proposed experiments have tasks that are simpler than those done by Vertechi et al (2020). On one hand, this has its downsides because it is not able to directly

track the minimization of energy, proposed by Friston (Section 2.1) and supported by Vertechi et al. (2020). On the other hand, these experiments would be a good comparison to show the effects of energy trade-off in decision making and learning processing.

Additionally, if one wanted to make the environment for the neural networks resemble Vertechi et al.'s (2020) experiments closer, one could implement an egocentric environment. This way, instead of the agent choosing left or right site being active, they would choose to either stay at the same port or switch ports. This would simulate a similar choice that the mice and humans were having to make, and would better match the selection of easy, medium, and hard environment conditions. Furthermore, it would be interesting to see if the egocentric environment would be better suited for the PPO agents compared to the allocentric environment.

An alternative direction is to implement a DQN that utilizes an LSTM architecture to see how it would learn in this environment. Because the DQN and Recurrent PPO outperformed the regular PPO, it would be interesting to see if combining the recurrent learning with the DQN MLP policy would improve the performance of the neural network on this task.

Finally, as previously noted, it appears to take the PPOs much longer to learn the environment than DQNs. However, it still performs at a level nowhere close to a human. An explanation for humans' abilities to learn tasks so speedily is that they utilize both model-free and model-based learning in parallel. By doing so, they are able to learn new information about a novel environment by learning associations between actions and rewards (thus implementing a model-free function) while also building on what they have experienced and learned in other tasks (implementing a model-based function).

A next step could be to implement the model-based DQN that Gou and Liu (2019) proposed. This would combine both model-free and model-based approaches that could better

replicate how a human uses the two in parallel (Chapter 2.2). I predict that this DQN would learn faster, and perhaps perform better in a difficult environment when the rewards are scarce.

4.6 Conclusion

This study showed that exclusively model-free neural networks are limited in the inferences they are able to make, which makes it difficult, and perhaps impossible, for them to learn novel environments anywhere close to the level that animals can. From this, there are at least two positive takeaways.

For one, these results suggest that more focus in this field of research should go towards designing agents with both model-free and model-based components. This research is difficult because just like it is difficult to know how top-down and bottom-up processing interact in animals, it is very difficult to figure out the optimal combination of model-free and model-based processes. However, by improving these types of agents, I would predict that the field would be able to create more dynamic ANNs that have varying capabilities that resemble the abilities of animals and humans.

The second positive takeaway is that this provides insight into the complexity of the brain's prediction model, if there is one. In order to both learn through a reward system, and then to go beyond that learning strategy to not be dependent on the rewards, but to pick up on the underlying statistical structures of novel environments is an incredibly complex ability that animals have. The fact that it takes humans a fraction of the amount of time to learn these systems suggests that there might be an extra process for efficiency that would be worth exploring deeper.

Acknowledgements

Thank you to Dr. Brian Keeley at Pitzer College for introducing me to predictive mind theory and for providing help and discussion throughout this project despite the bumps in the road. Thank you to Dr. Adam Landsberg at W.M. Keck Science Department for stepping in and tirelessly helping me understand a topic neither of us knew well, regardless of the somewhat drastic pivots along the way; I enjoyed getting to learn with you. A special thank you to Dr. Gautam Agarwal at Keck who spent hours helping me figure out the ANNs and code so that I not only had data, but also understood what I was doing; I truly could not have finished this without your help. Although we did not get to finish the project together, thank you Dr. Sarah Marzen for providing a tremendous amount of guidance through the crucial foundational part of this work. Finally and with deep appreciation, thank you to Dr. Jody Graham and Dr. Michael Watkins at Auburn University for editing my papers for the past 22 years (yes, from birth), and to Justin Walker for helping me design my poster in a way that was bearable for others to look at and for bearing through the worst of me during the years.

References

Banoula, M. (2023, August 10). *An overview on multilayer perceptron (MLP) [updated]*. Simplilearn.com. https://www.simplilearn.com/tutorials/deep-learning-tutorial/multilayer-perceptron

Barlow, Horace. (1961). Possible Principles Underlying the Transformations of Sensory Messages. Sensory Communication. 1. 10.7551/mitpress/9780262518420.003.0013.

Berridge, K. C. (2000). Reward learning: Reinforcement, incentives, and expectations. *Psychology of Learning and Motivation*, 223–278. https://doi.org/10.1016/s0079-7421(00)80022-5

Cheng, C., Ying, V., & Laird, D. (n.d.). Deep Q-learning with recurrent neural networks. https://cs229.stanford.edu/proj2016/report/ChenYingLaird-DeepQLearningWithRecurrentNeuralNetwords-report.pdf

Clark, Andy. (2023). *Experience machine: How our minds predict and shape reality*. Penguin Books.

De Ridder, D., Vanneste, S., & Freeman, W. (2014). The Bayesian Brain: Phantom percepts resolve sensory uncertainty. *Neuroscience & Biobehavioral Reviews*, *44*, 4–15. https://doi.org/10.1016/j.neubiorev.2012.04.001

Doody, M., Van Swieten, M.M.H. & Manohar, S.G. Model-based learning retrospectively updates model-free values. *Sci Rep* **12**, 2358 (2022). https://doi.org/10.1038/s41598-022-05567-3

DQN. DQN - Stable Baselines3 2.3.0 documentation. (n.d.). https://stable-baselines3.readthedocs.io/en/master/modules/dqn.html

Friston, K., Kilner, J., & Harrison, L. (2006). A free energy principle for the brain. *Journal of Physiology-Paris*, *100*(1–3), 70–87. https://doi.org/10.1016/j.jphysparis.2006.10.001

Friston, K., Thornton, C., & Clark, A. (2012). Free-energy minimization and the dark-room problem. *Frontiers in Psychology*, *3*. https://doi.org/10.3389/fpsyg.2012.00130

Gou, S. Z., & Liu, Y. (2019, March 22). *DQN with model-based exploration: Efficient learning on environments with sparse rewards*. arXiv.org. https://doi.org/10.48550/arXiv.1903.09295

Haith, A. M., & Krakauer, J. W. (2013). Model-based and model-free mechanisms of human motor learning. *Advances in experimental medicine and biology*, *782*, 1–21. https://doi.org/10.1007/978-1-4614-5465-6_1

Hohwy, J. (2013). *The Predictive Mind: Vol. First edition*. OUP Oxford.

Klapp, S. T. (2010). Comments on the classic Henry and rogers (1960) paper on its 50th anniversary. *Research Quarterly for Exercise and Sport*, *81*(1), 108–112. https://doi.org/10.1080/02701367.2010.10599634

Koehn, J. D., Dickinson, J., & Goodman, D. (2008). Cognitive demands of error processing. *Psychological Reports*, *102*(2), 532–538. https://doi.org/10.2466/pr0.102.2.532-538

Kozlica, R., Wegenkittl, S., & Hiränder, S. (2023). Deep Q-learning versus proximal policy optimization: Performance comparison in a material sorting task. *2023 IEEE 32nd International Symposium on Industrial Electronics (ISIE)*. https://doi.org/10.1109/isie51358.2023.10228056

Magazine, S. (2014, April 1). *Does thinking fast mean you're thinking smarter?* Smithsonian.com.https://www.smithsonianmag.com/science-nature/does-thinking-fast-mean-youre-thinking-smarter-180950180/?no-ist

Markov decision process. Markov Decision Process - an overview | ScienceDirect Topics. (n.d.). https://www.sciencedirect.com/topics/computer-science/markov-decision-process

Nagai, Y. (2019). Predictive learning: Its key role in early cognitive development. *Philosophical Transactions of the Royal Society B: Biological Sciences*, *374*(1771), 20180030. https://doi.org/10.1098/rstb.2018.0030

Palmer, S. E., Marre, O., Berry, M. J., & Bialek, W. (2015). Predictive information in a sensory population. *Proceedings of the National Academy of Sciences*, *112*(22), 6908–6913. https://doi.org/10.1073/pnas.1506855112

Patton, L. (2023, May 27). *Hermann von Helmholtz*. Stanford Encyclopedia of Philosophy. https://plato.stanford.edu/entries/hermann-helmholtz/#TheoPerc

PPO. PPO - Stable Baselines3 2.3.0 documentation. (n.d.). https://stable-baselines3.readthedocs.io/en/master/modules/ppo.html

Rao, R., Ballard, D. Predictive coding in the visual cortex: a functional interpretation of some extra-classical receptive-field effects. *Nat Neurosci* **2**, 79–87 (1999). https://doi.org/10.1038/4580

"Recurrent PPO." *Recurrent PPO - Stable Baselines3 - Contrib 2.3.0 Documentation*, sb3-contrib.readthedocs.io/en/master/modules/ppo_recurrent.html. Accessed 16 Apr. 2024.

Vertechi, P., Lottem, E., Sarra, D., Godinho, B., Treves, I., Quendera, T., Oude Lohuis, M. N., & Mainen, Z. F. (2020). Inference-Based Decisions in a Hidden State Foraging Task: Differential Contributions of Prefrontal Cortical Areas. *Neuron*, *106*(1), 166–176.e6. https://doi.org/10.1016/j.neuron.2020.01.017

Visser, I., Raijkmakers, M. E, & Molenaar, P. C. (2000). Reaction Times and Predictions in Sequence Learning: A Comparison. *Proceedings of the Annual Meeting of the Cognitive Science Society*, 22. Retrieved from https://escholarship.org/uc/item/31w8h46p

Visser, I., Raijmakers, M.E.J. & Molenaar, P.C.M. Characterizing sequence knowledge using online measures and hidden Markov models. *Memory & Cognition* **35**, 1502–1517 (2007). https://doi.org/10.3758/BF03193619

Wang, J.X., Kurth-Nelson, Z., D Tirumala, Soyer, H., Leibo, J.Z., Munos, R., Blundell, C., Kumaran, D., Botvinick, M. *Learning to reinforcement learn.* Arxiv.

Wessel, J. R. (2017). An adaptive orienting theory of error processing. *Psychophysiology*, *55*(3). https://doi.org/10.1111/psyp.13041

What are recurrent neural networks?. IBM. (n.d.). https://www.ibm.com/topics/recurrent-neural-networks

Yon, D., & Frith, C. D. (2021). Precision and the bayesian brain. *Current Biology*, *31*(17). https://doi.org/10.1016/j.cub.2021.07.044

Appendix

A. Link to the workbook of python code used for the entire methods section and for the 5x5 agent arrays: https://colab.research.google.com/drive/1Wl-7EkicWSlp9u-a2CP9RKTtjS_3MvvX?usp=sharing

B. Figures B1-B3: Probability of each 2nd guess outcome. Useful for calculating how often an agent should be getting rewarded for optimal 1-back memory.

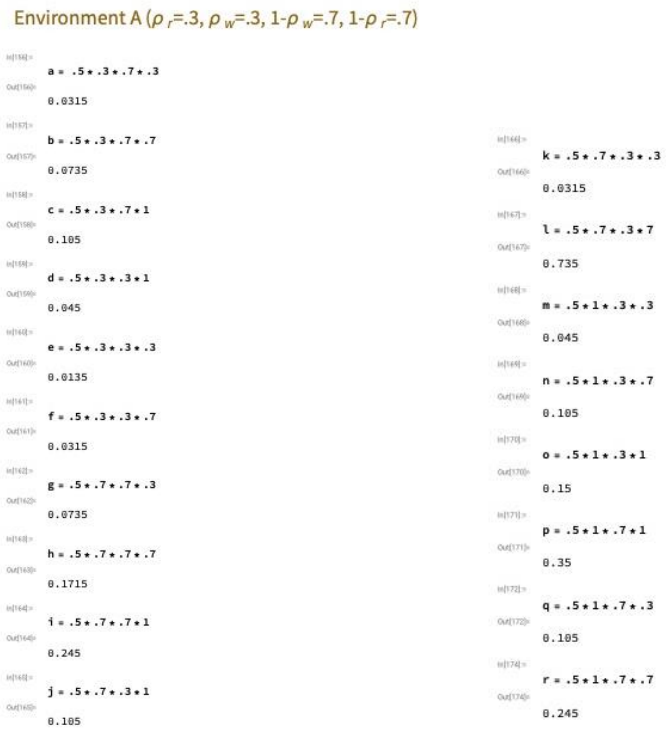


Figure B1: Environment A probabilities of outcomes. The letters refer to the ending step of each possible outcome labeled in Figure 8.

Environment B (ρ_r=.9, ρ_w=.3, 1-ρ_w=.1, 1-ρ_r=.7)

```
In[165]:
a = .5*.9*.7*.9
Out[165]:
0.2835

In[166]:
b = .5*.9*.7*.1
Out[166]:
0.0315

In[167]:
c = .5*.9*.7*1
Out[167]:
0.315

In[168]:
d = .5*.9*.3*1
Out[168]:
0.135

In[190]:
e = .5*.9*.3*.9
Out[190]:
0.1215

In[191]:
f = .5*.9*.3*.1
Out[191]:
0.0135

In[192]:
g = .5*.1*.7*.9
Out[192]:
0.0315

In[193]:
h = .5*.1*.7*.1
Out[193]:
0.0035

In[194]:
i = .5*.1*.7*1
Out[194]:
0.035

In[195]:
j = .5*.1*.3*1
Out[195]:
0.015
```

```
In[196]:
k = .5*.1*.3*.9
Out[196]:
0.0135

In[197]:
l = .5*.1*.3*.1
Out[197]:
0.0015

In[ ]:
m = .5*1*.3*.9
Out[ ]:
0.135

In[206]:
n = .5*1*.3*.1
Out[206]:
0.015

In[204]:
o = .5*1*.3*1
Out[204]:
0.15

In[203]:
p = .5*1*.7*1
Out[203]:
0.35

In[202]:
q = .5*1*.7*.9
Out[202]:
0.315

In[201]:
r = .5*1*.7*.1
Out[201]:
0.035
```

Figure B2: Environment B probabilities of outcomes. The letters refer to the ending step of each possible outcome labeled in Figure 8.

Environment C (ρ_r=.9, ρ_w=.9, 1-ρ_w=.1, 1-ρ_r=.1)

```
In[204]:
a = .5*.9*.1*.9
Out[204]:
0.0405

In[207]:
b = .5*.9*.1*.1
Out[207]:
0.0045

In[208]:
c = .5*.9*.1*1
Out[208]:
0.045

In[209]:
d = .5*.9*.9*1
Out[209]:
0.405

In[210]:
e = .5*.9*.9*.9
Out[210]:
0.3645

In[211]:
f = .5*.9*.9*.1
Out[211]:
0.0405

In[212]:
g = .5*.1*.1*.9
Out[212]:
0.0045

In[213]:
h = .5*.1*.1*.1
Out[213]:
0.0005

In[214]:
i = .5*.1*.1*1
Out[214]:
0.005

In[215]:
j = .5*.1*.9*1
Out[215]:
0.045
```

```
In[216]:
k = .5*.1*.9*.9
Out[216]:
0.0405

In[217]:
l = .5*.1*.9*.1
Out[217]:
0.0045

In[218]:
m = .5*1*.9*.9
Out[218]:
0.405

In[133]:
n = .5*1*.9*.1
Out[133]:
0.045

In[134]:
o = .5*1*.9*1
Out[134]:
0.45

In[219]:
p = .5*1*.1*1
Out[219]:
0.05

In[220]:
q = .5*1*.1*.9
Out[220]:
0.045

In[221]:
r = .5*1*.1*.1
Out[221]:
0.005
```

Figure B3: Environment C probabilities of outcomes. The letters refer to the ending step of each possible outcome labeled in Figure 8.