NEURAL NETWORKS FOR HIGHER-ORDER REINFORCEMENT LEARNING AND MULTI-PERSPECTIVE GENERATIVE MODELING

by

Arthur S. Williams

A Dissertation Submitted in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

in

Computational Science

Middle Tennessee State University

June, 2022

Dissertation Committee:

Dr. Joshua L. Phillips, *Chair*

Dr. Salvador Barbosa

Dr. Wandi Ding

Dr. Qiang Wu

To my loving wife, two daughters, my son and my parents.

ACKNOWLEDGMENTS

I would like to thank my advisor Dr. Joshua L. Phillips for taking the time to guide me in the areas of Machine Learning and Computational Cognitive Neuroscience. Your wisdom and patience has been a great asset to me over the years. You are the best adviser and mentor a PhD student could ever have. Additionally, I would like to thank Dr. Medha Sarkar and Dr. Chrisila Pettey for their help as Computer Science department chairs during my time as a Graduate Assistant. My sincere thanks goes to Dr. John Wallin for encouraging me to join the Computational Science PhD program. I truly appreciate all the assistance you have provided me over the years. I would like to thank the rest of my dissertation committee: Dr. Wandi Ding, Dr. Qianq Wu, and Dr. Salvador Barbosa for their astute observations on my dissertation. Finally, I would like to thank my parents, my beautiful and supportive wife Bianca, my two daughters, Aria and Brianne, and my son Kobe. I am grateful for the motivation and support you all have bestowed upon me during this process.

Work performed for Chapter IV was funded by the Stark land grant fund, MTSU grant number 93697.

ABSTRACT

What does it mean for a system or machine to exhibit intelligence? In machine learning, a system is said to exhibit artificial intelligence provided that it improves its performance with experience on a given task. However, some tasks require more cognitive load than others, and higher-order cognitive processes are needed to solve some tasks effectively. We developed a framework inspired by how the human brain utilizes working memory to solve complex tasks, and constructed a model with the ability to solve tasks with multiple context layers. Our results show that working memory plays a vital role in solving cognitive context processing tasks. Furthermore, we extended our model to utilize the generalization benefits of output gating within a working memory system. This modification resulted in successfully transferring knowledge across a temporally extended, partially observable grid-world maze task that required the agent to learn three tasks throughout the training period. Finally, Generative Adversarial Networks (GANs) can be viewed as using higher-order cognitive processes to perform image-to-image translation. More specifically, we are concerned with generating video frames from a missing camera feed, where each feed is viewing the same area from different positions and angles. Results show that our model can produce realistic video frames that resemble the missing video source.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I : INTRODUCTION

The ability to engage in higher-order thinking is critical to developing other forms of reasoning (Richland and Simms, 2015). By utilizing our higher-order thinking skills (Anderson et al., 2001), we are able to learn a rule and generalize to new stimuli with minimal effort. Some tasks require more cognitive load than others, and higher-order cognitive processes are needed to solve these tasks effectively. Furthermore, partially observable and temporally extended problems are difficult for machines and may require cognitive mechanisms in order to be solved. Our goal in this dissertation is to demonstrate novel approaches to solving partially observable reinforcement learning problems by developing a framework inspired by how the human brain utilizes working memory to solve complex tasks (Williams and Phillips, 2018, 2020). Additionally, we show the effectiveness of Generative Adversarial Networks (GANs) (Zhu et al., 2017; Isola et al., 2017) for generating missing video frames in a multi-perspective environment.

In cognitive neuroscience, the working memory system enables the ability to hold multiple pieces of information in one's mind while continuing to process other information. In the human brain this system is supported by the prefrontal cortex and the mesolimbic dopamine system (O'Reilly and Munakata, 2000a). Computational models seek to represent such systems through the use of artificial neural networks. The Working Memory Toolkit (WMtk) aimed at creating a working memory framework that could easily be incorporated in a robotic system (Phillips and Noelle, 2005). The WMtk used the delayed saccade task to demonstrate the effectiveness of the framework. The WMtk framework was improved to utilize Holographic Reduced Representations (HRRs) (Plate, 1995) in order to encode representations of the environment as well as working memory concepts (Dubois and Phillips, 2017). While the WMtk has been applied to various robotic tasks (Kawamura et al., 2008; Busch et al., 2007; Erdemir et al., 2008; Wang et al., 2009), it often fails to solve *multilayer hierarchical problems* commonly explored in the cognitive sciences.

Therefore, we have extended the use of HRRs within the WMtk to simulate multilayer hierarchical context reasoning in a working memory system. We show that our revised model can solve working memory tasks that are hierarchical in nature.

We take for granted our ability to adapt and respond appropriately to novel stimuli while performing a task. Similarly, it is also routine for humans to use knowledge gained from prior tasks in order to speed up the learning for new tasks. For example, imagine trying to throw a baseball for the first time. Even though a person might have never thrown a baseball, they can still use prior knowledge of "throwing" and transfer that information to the current task of throwing a baseball. Furthermore, if a person was unable to transfer knowledge across these two task, they would be relegated to relearning the process of "throwing" due to the novel context presented by the baseball. This example presents a common transfer reinforcement learning problem where a source task (e.g. throwing a football) is used to improve learning performance on a target task (e.g. throwing a baseball). In addition, output-gating is a mechanism for controlling how knowledge retained in upstream neural processes influence downstream cognitive processes in computational cognitive neuroscience models, and has been utilized to improve generalization performance within tasks (Kriete and Noelle, 2011; Kriete et al., 2013). These models are based on established interactions between the prefrontal cortex (PFC) and mesolimbic dopamine system. Each PFC neuronal stripe (assumed anatomical unit of storage) has the implied ability to allow or disallow maintained representations to flow both inside as well as outside of the PFC via input-gating and output-gating, respectively. Here we propose a novel approach to the transfer learning problem that utilizes the generalization benefits of output-gating coupled with the flexibility of the HWMtk (Dubois and Phillips, 2017; Phillips and Noelle, 2005), a working memory framework based on the putative interactions between the PFC and basal ganglia.

Generative Adversarial Networks (GANs) are typically used in computer vision and image processing to translate an image from one domain to another (Zhu et al., 2017; Isola et al., 2017). However, what happens when one wants to translate a video feed from one domain to another? Currently, there is minimal research dedicated to the process of video-to-video translation in the deep learning literature. Moreover, image-to-image translation with GANs is a highly explored topic, with multiple works addressing the issue (Turkoglu et al., 2019; Huang et al., 2018; Odena et al., 2017; Isola et al., 2017; Mirza and Osindero, 2014; Zhu et al., 2017). Video data consists of a sequence of video frames, with each representing a single image. Therefore, we can then use the latest GAN methods and extend them to the video-to-video translation domain by leveraging the current research in image-to-image translation. This work presents a novel modification to the GAN architecture that enables video data generation. More specifically, we are concerned with generating video frames from a missing camera feed, where each feed is viewing the same area from different positions and angles. In addition, we utilize a novel cycle consistency component within our model. As a result, our model can produce realistic video frames that resemble the missing camera feed.

In Chapter II we discuss multilayer context reasoning and detail a model that successfully solves a common hierarchical working memory task in the cognitive sciences. Next in Chapter III, we present a novel reinforcement leaning approach to the transfer learning problem. And in Chapter IV, we showcase a novel modification to the GAN architecture that enables video data generation in a multi-perspective environment. The associated publications are as follows:

> Williams, A. and Phillips, J. (2018). Multilayer Context Reasoning in a Neurobiologically Inspired Working Memory Model for Cognitive Robots. Proceedings of the 40th Annual Meeting of the Cognitive Science Society, pages 2687–2692.

Williams, A. and Phillips, J. (2020). Transfer Reinforcement Learning Using Output-Gated Working Memory. Proceedings of the AAAI Conference on Artificial Intelligence, 34(02):1324–1331.

Williams, A., Phillips, J., Cui, S., and Butler, D. (2022). A Multi-Perspective Generative Model for Video Interpolation. In preparation for the Proceedings of the 34th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2022 (submission deadline of July 31st, 2022).

CHAPTER II : MULTI-LAYER CONTEXT REASONING

In this chapter we describe a working memory based architecture for solving multi-layer hierarchical problems (Williams and Phillips, 2018). We utilize a reinforcement learning method, the temporal difference learning algorithm, to simulate the modulation of dopamine. The modulation of dopamine is important because it regulates the flow of information in and out of working memory stripes within the PFC. Our model was applied to a common cognitive load task known as the 1-2-AX-CPT task. Typical reinforcement learning algorithms are unable to solve temporally extended hierarchical problems. We show that our model can successfully solve these class of reinforcement learning problems.

Background

Working Memory Architecture

Working memory plays an important role in cognitive neuroscience. Working memory allows the ability to hold on to task-relevant information needed for further processing. An example of this can be shown by the task of storing someones phone number into your cell phone. When you are entering their number, you must maintain each number in memory. Once all the numbers have been entered, the phone number is no longer maintained and flushed out of memory. Once out of memory, the phone number is forgotten and is unavailable for recall. This example illustrates how working memory keeps active representations of relevant information while likewise ignoring distracting information (such as a horn blast from a passing car). The neural network keeps these representations active by making concepts available for rapid updating while having them readily accessible for ongoing processing.

Attractor dynamics is the process of achieving a stable activation state from a range of different starting states in a neural network (O'Reilly and Munakata, 2000b). Each stable attractor could be used to actively maintain information over a period of time. In the

prefrontal cortex (PFC), attractors provide a mechanism for robust active maintenance of information while counteracting the interference presented by ongoing processing. This allows the firing rate of the neurons to encode representations that will then be maintained in working memory. In previous models, the neural firing rate encoding of representations was handled by a complex multilayer artificial neural network (ANN). In our model, we represent the encoding of representations in the form of holographically reduced representations (HRRs) (Plate, 1995). The use of HRRs in our model reduces the complexity of the network down to a single layer, having the HRRs provide the encoding instead of using multiple layers within an ANN.

The interaction between the prefrontal cortex and mesolimbic dopamine system is key to the emergent behavior of working memory. Additionally, the use of reinforcement learning is also important as it pertains to working memory. Literature in the cognitive sciences suggest that learning is driven by rewards and punishments in response to the changes in expectation of future events (Schultz et al., 1997). This led researchers to postulate that the mesolimbic dopamine system is a neural substrate responsible for reinforcement learning in the brain. This system consists of the basal ganglia and ventral tegmental area (VTA). Recorded data from the firing of dopamine cells in monkeys show that dopamine cells fire in response to adjustments in expected future reward (Schultz et al., 1997). The basal ganglia is responsible for broadcasting dopamine signals to the PFC. The interaction between the PFC and basal ganglia create a feedback loop where internal memory updates are chosen based on dopamine signals.

Our working memory model captures the PFC and dopamine system through the use of the temporal difference (TD) learning algorithm (Barto and Sutton, 1998). TD learning simulates the modulation of the dopamine system through the use of TD error (δ) signals. A TD δ of zero is given if the environment behaves as expected, positive TD δ in reaction to unexpected reward, and negative TD δ in response to the absence of expected reward

(Smith et al., 2007). These TD δ signals aim to simulate how the basal ganglia computes the firing of dopamine neurons. TD has traditionally been used to learn external action selection, making changing the external environment well suited for leaning internal action selection as well. Changing internal representations of the environment would consist of the opening or closing of circuits in the PFC for either flushing working memory contents or maintaining them.

Working Memory Toolkit

The Working Memory Toolkit (WMtk) is a framework that was created for the purpose of providing a robot control system with the ability to utilize working memory (Phillips and Noelle, 2005). The utility of the WMtk has been tested on several robotic tasks. One task aimed to explore the feasibility of cognitive control systems in humanoid robots (Kawamura et al., 2008). Another task was a spatial memory task that required the robot to learn to associate perceptual information with that of a particular location in order solve the task (Busch et al., 2007). The next task required a robot to explore different internal scenarios before actually executing an external action (Erdemir et al., 2008). And a final task explored automatic scene recognition in regard to robotic navigation (Wang et al., 2009).

The Holographic Working Memory Toolkit (HWMtk) improved upon the Working Memory Toolkit (WMtk) (Phillips and Noelle, 2005) by providing an interface between the distributive encoding of artificial neural networks and symbolic encoding (Dubois and Phillips, 2017). Previously, using the WMtk required the user to manage the conversion between distributive encoding (DE) and symbolic encoding (SE). An example of DE is the use of a vector in an artificial neural network. An example of SE is the use of a word or phrase as a representation of a concept. Through the use of holographically reduced representations (HRRs), the HWMtk was able to automate the SE/DE conversion problem. In order to solve this problem a HRR engine was developed to facilitate the conjunctive encoding and decoding of concepts. Holographic reduced representations (HRRs) are created

using a vector of real values taken from a Normal distribution with mean zero and standard deviation $1/\sqrt{n}$, with n being the length of the vector (Plate, 1995). HRRs can be used to encode a particular concept within a model. Circular convolution allows for complex representations by combining two representations together into a single HRR of the same vector length. The circular convolution operation can be computed using Fast Fourier transforms which takes $O(nlog(n))$ time. Another operation is correlation which uses a HRR as a key in order to retrieve information from complex HRRs containing multiple combined HRR representations.

The HWMtk allows agents to learn to gate-in appropriate cues in order to make appropriate action choices later in time. However, the HWMtk shows selective interference between policies contingent on hierarchically structured cues, suggesting future improvement to the framework may be needed.

Cognitive Tasks

The AX Continuous Performance Test (AX-CPT) is a cognitive task that tests the context processing ability of an individual (Braver et al., 2005). In this task, an individual is presented with a pair of letters. The individual is only rewarded when they see the letter "A" followed by the letter "X" and press the right button. All other letter combinations are seen as distractors. The presentation of a letter sequence followed by an action marks the end of a trial. Performing well on this task requires correctly updating context information after each trial. Our model learns the correct button to press based on the sequence of letters presented to it. The agent was presented the A-X letter sequence 70% of the time and presented the B-X, A-Y, and B-Y sequences at a rate of 10% each. The particular letter sequence was determined randomly before each trial.

The 1-2-AX-CPT task is an extension of the AX-CPT task that adds an extra layer of context that must be maintained by the working memory system (Frank et al., 2001). The target sequence is dependent on a previous context cue and varies depending on which

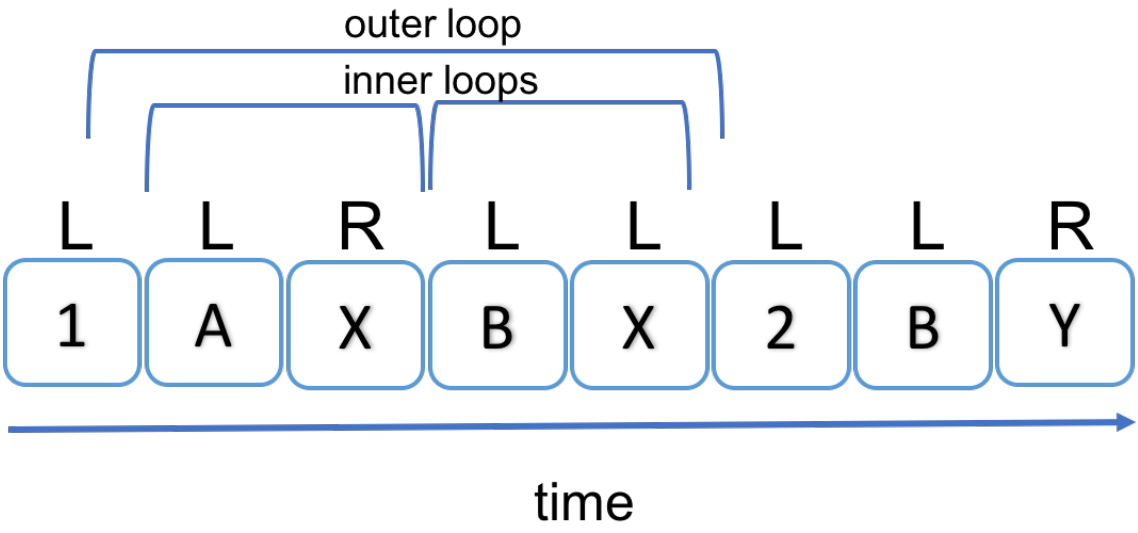


Figure 2.1: The 1-2-AX-CPT task. Shows the letters presented to the agent over time. Also, the action choice of the agent, which is either L (for left button press) or R (for right button press), is determined at each time step.

stimulus was observed by the agent. For example, if the agent saw a 1, then the target sequence will be A-X. Instead, if the agent saw a 2, then the target sequence will be B-Y. In order to successfully learn the 1-2-AX-CPT task, the agent must maintain the outer loop of context, which is the task context cue (1 or 2), and the inner loop of context, which is the sequence of letters as detailed in Figure 2.1. The context cue, which lets the agent know whether the target is A-X or B-Y, was provided to the agent randomly with equal probability at the beginning of each trial.

Methods

We implemented a multilayer hierarchical model of working memory that extends the capability of the WMtk. Our model uses a novel approach to internal working memory updates that relies on the SARSA TD learning algorithm (Barto and Sutton, 1998). This differs slightly from the approach provided by the WMtk, which learns a state value function instead of a state-action value function as a means for action selection and working memory updates. The utility of our model was determined by testing it on a set of learning

tasks. First, we created a 2-layer hierarchical maze task for proof of concept. We then tested our model on the AX-CPT task, which is a common working memory task in the cognitive sciences that tests one's ability to maintain relevant features over time. Finally, to fully observe the multilayer hierarchical learning of context features, we implemented the 1-2-AX-CPT task. The 1-2-AX-CPT extends the AX-CPT task by adding an additional layer of context, thereby requiring a multilayer context to be learned in order to complete the task effectively. We then compared the accuracy of our model to that of the LSTM model (Hochreiter and Urgen Schmidhuber, 1997). LSTM is a common deep learning architecture in sequence learning. The juxtaposition of our model and the LSTM model, while performing the 1-2-AX-CPT task, is key toward displaying our model's overall utility. The source code for the AX-CPT task, 1-2-AX-CPT task, and 2-layer hierarchical maze task is available online at: https://github.com/arthurw125/TD_learning. Next, we will explain the computational models and tasks.

Multilayer Hierarchical Context Model

The WMtk uses the temporal difference (TD) learning algorithm which learns the correct action to take by learning the value function $V(s)$, an estimate of the sum of discounted future rewards, for all states (Sutton, 1988). The SARSA learning algorithm is another reinforcement learning algorithm. SARSA is similar to TD in the sense that they are both temporally extended and estimate a value function. The difference is that SARSA learns the state-action value function called the Q function, $Q(s,a)$, instead of the state-value function (Sutton, 1988). We learn $Q(s,a)$ by making transitions from one state-action pair to the next, thus learning the value of all state-action pairs. Reward is given only to the state-action pair that transitioned to the goal state. We decided to use the Q function approach as opposed to the value function approach used by the WMtk for memory updates.

In order to use SARSA to model working memory, we modified the Q function to be a state-action-working memory triplet. The amount of states, actions, and working memory

slots are determined by the particular task being observed. The value of $Q(s_t, a_t, wm_t)$ can be written as:

$$Q(s_t, a_t, wm_t) = r(s_{t+1}) + \gamma Q(s_{t+1}, a_{t+1}, wm_{t+1}), \tag{II.1}$$

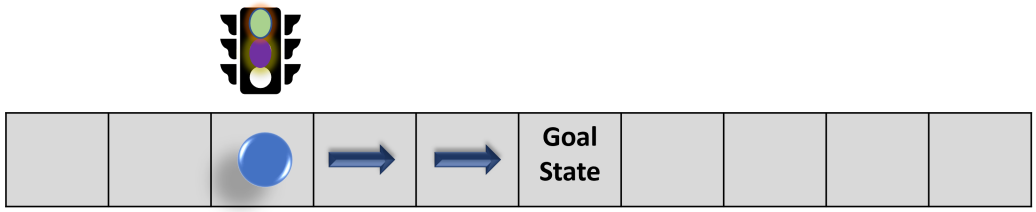
where t is the time, s is the set of observable states, a is the action taken, γ is the discount factor, α is the learning rate and wm is the set of working memory slots. The error is computed by taking the difference between the expected and true value of $Q(s_t, a_t, wm_t)$. The Q function is then updated using the following:

$$\begin{aligned} \delta(s_t, a_t, wm_t) =& [r(s_{t+1}) + \gamma Q(s_{t+1}, a_{t+1}, wm_{t+1})] \\ & - Q(s_t, a_t, wm_t) \end{aligned} \tag{II.2}$$

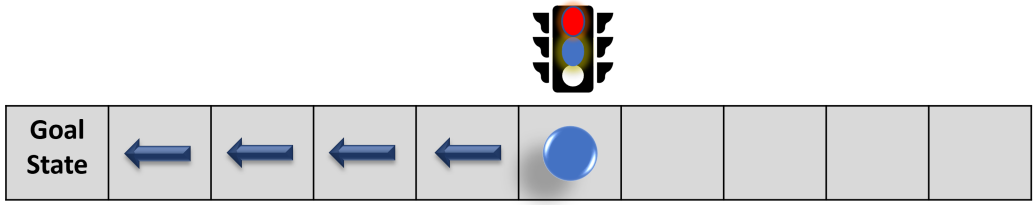
$$Q(s_t, a_t, wm_t) = Q(s_t, a_t, wm_t) + \alpha \delta(s_t, a_t, wm_t) \tag{II.3}$$

where α is a learning rate hyperparameter in the range [0,1] and γ is the discounting hyperparameter for future rewards in the range [0, 1). This process is repeated over several episodes until the Q function is learned for all state-action-working memory triplets. With this new configuration of the Q function, we can now begin to apply it as a means towards updating working memory slots within our model. The exact contents of what is being held in working memory is available to our model and can be recalled if needed. This feature of our model is contrasted with LSTM's mechanism, which stores its memory information in the weights of the recurrent neural network, thus making it more difficult to determine what memory content is being stored and how it is being maintained.

For our model to learn multiple layers of context, we must provide as many working memory slots as there are contextual layers within the task. We encode the external features and their internal representations for a task using HRRs. The HRR size is typically set to be an order of magnitude greater than the number of input permutations (NIP) of the task. This NIP can be computed by multiplying together all the states, actions, and working memory for a given task. Our model then learns the hierarchy of a task by computing the Q function $Q(s, a, wm)$ at each time step to determine whether to gate-in new working



(a) Green/Purple Maze Task



(b) Red/Blue Maze Task

Figure 2.2: Multilayer hierarchical maze task. The goal state when the agent is flashed the color green followed by purple (a) and the goal state when the agent is flashed the color red followed by blue (b). In the actual simulation, the color cues are only present when the agent first enters the maze.

memory representations or maintain old working memory representations. Also, external action selection is determined at this step. This is achieved by computing all the state-action-working memory combinations within the Q function and choosing the combination with the highest learned discounted future reward. A weight vector for the current state is updated after each state transition and provides an input to output mapping of the Q function's single-layer neural network.

2-layer Hierarchical Maze Task

To demonstrate that our model was able to tackle tasks that were hierarchical in nature, we created the multilayer hierarchical maze task. The 2-layer hierarchical maze task requires the agent to learn the location of the goal state within a 1D maze environment as illustrated in Figure 2.2. The agent is dropped in the maze at a random location. The agent is then flashed the color red or green followed by blue or purple at the beginning of the task. The agent then must navigate the maze to find the goal state within the maze. Based on what combination of colors was flashed initially, the goal will be either in the middle or

at the front of the maze. The objective of the agent is to find the goal based on the set of initial cues seen. This task is partially observable because the agent does not have access to cues after initial sightings. This is why the agent must use its working memory system to actively hold on to the color cue at both levels of hierarchy throughout the task. The purpose of the task is to show the agent's ability to maintain viable concepts in its working memory while discarding irrelevant information. Also, this task seeks to show how multilayer context reasoning can have an effect on the working memory system.

Our computational model for the 2-layer hierarchical maze task requires HRR representations for states, actions, color cues, and working memory slots. We then precompute the convolutions of each representation for efficiency and store the output of vectors in an array. The convolutions of each representation can also be computed on the fly if needed. We initialize the weight vector and set the bias b equal to 1 for an optimistic initialization. We then set the number of action and color cues to 2. We randomly place the agent in our maze environment by choosing a random number between 0 and the number of states. This value is then mapped to a location in a grid maze environment. A color signal is then flashed to the agent, representing either red or green. If the color is red followed by blue then the goal is in the far-left corner of the maze environment as seen in Figure 2.2b. Furthermore, if the color is green followed by purple then the goal will be located in the middle of the maze environment as seen in Figure 2.2a. We provided 2 working memory slots to our model so that the agent has the ability to maintain both the inner and outer cues being relayed from the environment.

A NIP of 8100 was computed for this task. This was evaluated by multiplying all the encoded states, actions, and working memory features. We noticed that in order to achieve stability in our model, we needed to provide an HRR size that was an order of magnitude greater than the NIP of the task. This is why we chose the HRR and weight vector length for this task to be $n = 64000$. Epsilon-greedy ε is a hyperparameter that controls the probability

Table 2.1: Hierarchical Maze Task Hyperparameters

Name	Value	Description
n	64000	Size of HRR vectors
ε	0.01	Probability of non-greedy action choice
γ	0.9	Discount factor
α	0.1	Learning rate
$episodes$	10000	Number of training cycles
b	1	Network bias

in which a non-optimal action is taken by the agent. Moreover, ε is important because it forces exploration within a reinforcement learning environment. The model was trained for 10000 episodes. A list of the hyperparameters for the 2-layer hierarchical maze task is available in Table 2.1.

AX-CPT Task

The implementation of the AX-CPT task required that we use the modified SARSA algorithm from equations 1, 2 and 3 for action selection and HRRs for concept encoding. We encoded HRRs for 1 working memory slot, 4 different external cues, and 2 actions. The HRRs were pre-convolved and stored in an array for later use. We created the weight vector from a new HRR, and fixed bias weight of 1. The possible working memory contents that could be used by the agent consisted of seeing nothing or the outer signal. Using SARSA along with a softmax function, the outer working memory is chosen along with an action of 1 or 0 (left or right button). The error is computed and the weight vector is updated. This process is repeated until the Q function is learned for the task.

We computed the NIP of this task to be 54. In order to achieve stable learning, the HRR length for our encoded features had to be an order of magnitude larger than the NIP. Due to this constraint, the HRR and weight vector length for this task was $n = 128$. The model was trained on 1000 trials, where each trial is a different stream of letter combinations presented to the agent. The λ hyperparameter controls the decay factor of eligibility traces. The temperature hyperparameter $temp$ affects the final probabilities of the softmax function. A

Table 2.2: AX-CPT Task Hyperparameters

Name	Value	Description
n	128	Size of HRR vectors
ε	0.01	Probability of non-greedy action choice
γ	0.9	Discount factor
α	0.1	Learning rate
λ	0.8	Trace decay
$temp$	0.1	Adds entropy to the softmax function
$trials$	1000	Number of trials during training
b	1	Network bias

low $temp$ generates softmax probabilities where the distance between probabilities is more pronounced, while a high $temp$ does the opposite. A list of the hyperparameters for the AX-CPT task is available in Table 2.2.

1-2-AX-CPT Task

The implementation of the 1-2-AX-CPT task required that we use the modified SARSA algorithm from equations 1, 2 and 3 for action selection and internal memory updates. We encoded HRRs for 2 different working memory slots, 6 different external cues, and 2 action HRRs. The HRRs were pre-convolved and stored in an array for later use. We created a weight vector from a new HRR, and fixed bias weight of 1. The possible working memory contents that could be used by the agent consist of seeing nothing or the outer signal for the outer context stimulus. For the other working memory slot, the possible working memory contents are nothing or the inner signal. Using the modified SARSA algorithm along with a softmax function, the outer and inner memory slots are chosen, as displayed in Figure 2.3, along with an action of 1 or 0 (left or right button). The TD error is computed and the weight vector is then updated.

A NIP of 486 was computed for this task. As in the previous models, we observed that in order to achieve stability in our model we needed to provide an HRR size that was an order of magnitude greater than the NIP of the task. This is why we chose the HRR and weight vector length for this task to be $n = 1024$. Choosing a learning rate of 0.4

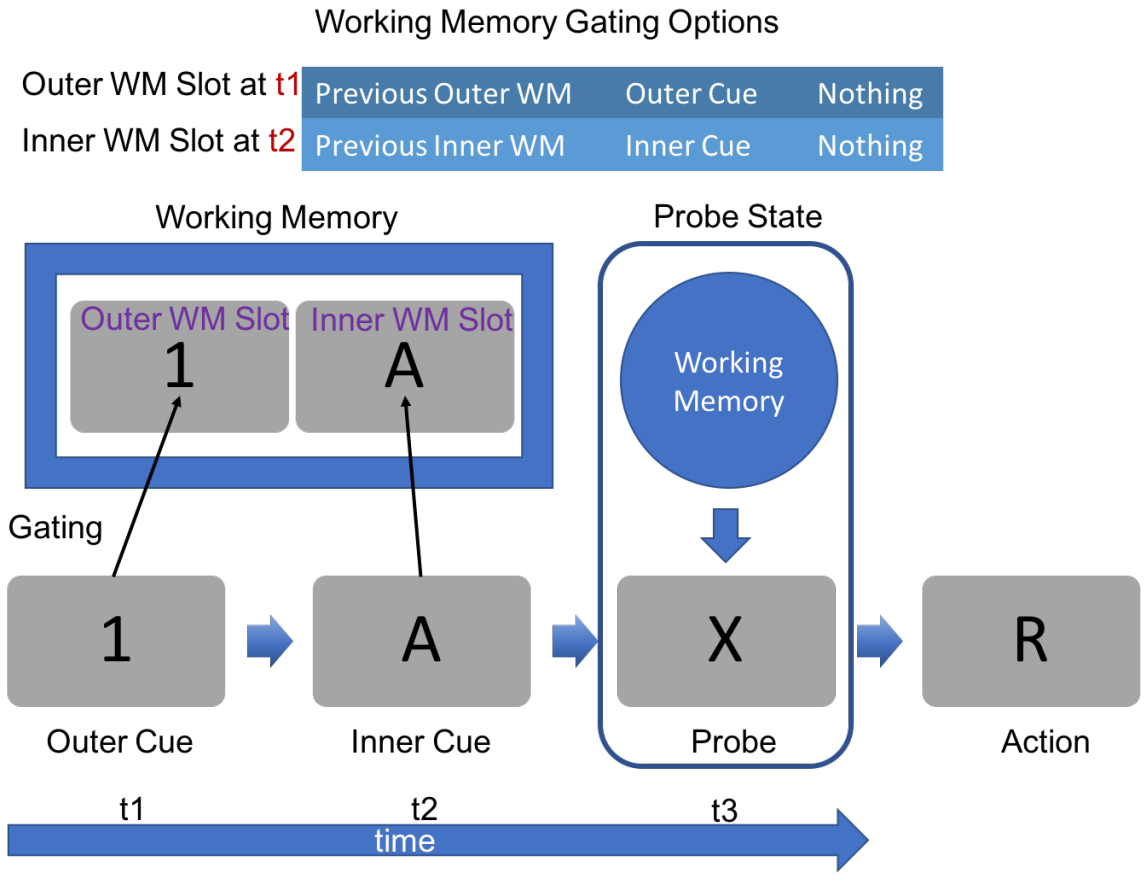


Figure 2.3: Diagram of our working memory model while solving the 1-2-AX-CPT task. The gating policy is learned by our modified Q function and is responsible for making internal working memory updates. At time step t1, the agent is presented with an outer cue of "1". The gating policy then determines what should be gated into the outer working memory slot. The options available to gate-in at time t1 is as follows: gate-in the current outer cue of "1", maintain what was previously in the outer working memory slot, or flush the outer working memory slot. At time step t2, the agent is presented with an inner cue of "A". The gating policy then determines what should be gated into the inner working memory slot. The options available to gate-in at time t2 is as follows: gate-in the current inner cue of "A", maintain what was previously in the inner working memory slot, or flush the inner working memory slot. At time step t3, the Probe State consists of convolving working memory with probe "X". Finally, the action "R" is selected and evaluated for correctness.

allowed our model to learn the task at a faster rate, causing the number of trials to equal only 2000, while still being low enough to allow for convergence. Also, we elected to use an ε-greedy policy set at 0.01 which allowed for exploration of suboptimal states. This

Table 2.3: 1-2-AX-CPT Task Hyperparameters

Name	Value	Description
n	1024	Size of HRR vectors
ε	0.01	Probability of non-greedy action choice
γ	0.9	Discount factor
α	0.4	Learning rate
λ	0.8	Trace decay
$temp$	0.1	Adds entropy to the softmax function
$trials$	2000	Number of trials during training
b	1	Network bias

allowed our model's loss function to avoid getting caught in local minima. A list of the hyperparameters for the 1-2-AX-CPT task is available in Table 2.3.

It is worth noting that without working memory, an agent within a multilayer hierarchical task is unable to learn the outer loop context of a task, resulting in performance that is equivalent to random chance. In Figure 2.4, we see that for the A-X and B-Y presentations the model struggles to perform above random chance. Without working memory our model is still able to learn the B-X and A-Y presentations. This learning occurred because the target sequence for both B-X and A-Y did not change during context switches. In order for our model to achieve adequate performance, it was required that we constrain the possible working memory options made available to the model. The graph in Figure 2.5 displays how learning can become erratic when constraints are taken off of the working memory slots. This meant that we had to limit the options for each working memory slot to only have the ability to gate-in current external stimuli or gate-in nothing. Without these constraints in place the model would become unstable and unlearn a given task over time.

We used Keras with a TensorFlow backend to implement the LSTM version of the 1-2-AX-CPT task for comparison with our own working memory model. We randomly generated test data to be fed into the neural network for learning. The LSTM model contained 3 hidden layers, each with 32 nodes. The output layer used softmax for action selection of either left button press or right button press. The hyperparameters were as fol-

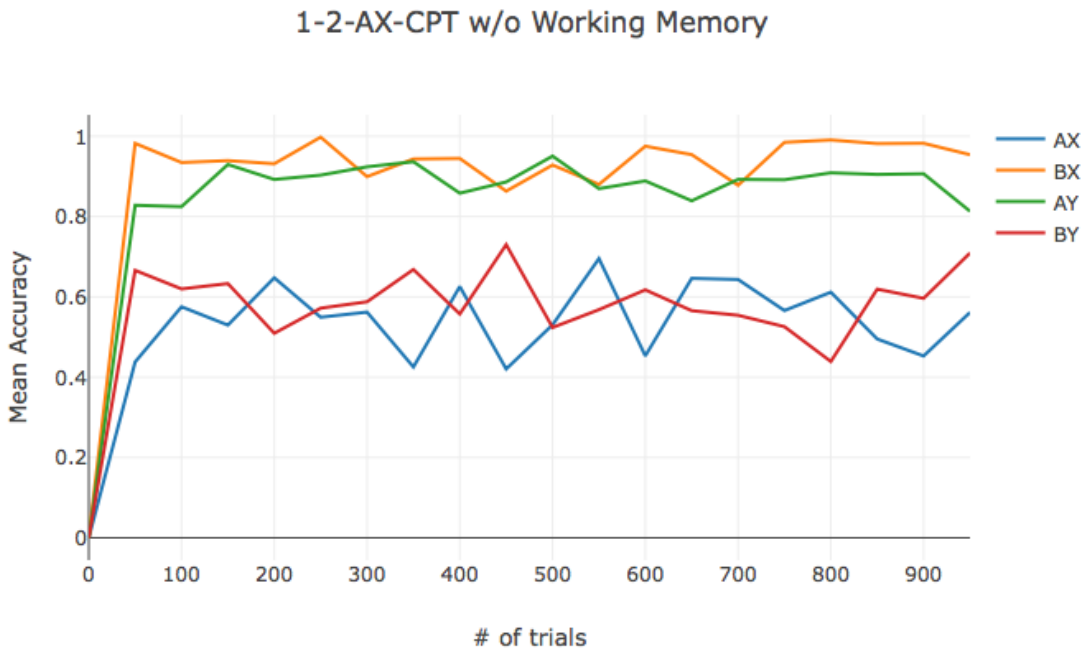


Figure 2.4: The 1-2-AX-CPT task without working memory made available to the system. The graph depicts the mean accuracy of the 1-2-AX-CPT task over a 100 trial window. The graph shows the mean accuracy for all the AX, BX, AY and BY presented during each trial window.

lows: loss=categorical crossentropy; optimizer=rmsprop; and metrics=accuracy. We used the default setting for all other parameters. We then generated 10,000 sequences as training data and 100 sequences as testing data. The model was fit using a batch size=64 and epochs=20.

Results

Adding additional layers of working memory slots allowed our model to retain information in a hierarchical manner without the need for the continuous presence of stimuli. This allowed our model to solve the 2-layer hierarchical maze task in which the cue was only flashed once and the system had to determine whether or not to remember the initial cue. The model was able to learn the correct Q function for the given task despite multiple distractors. Also, the 2-layer hierarchical maze task has a large state space, so we noticed

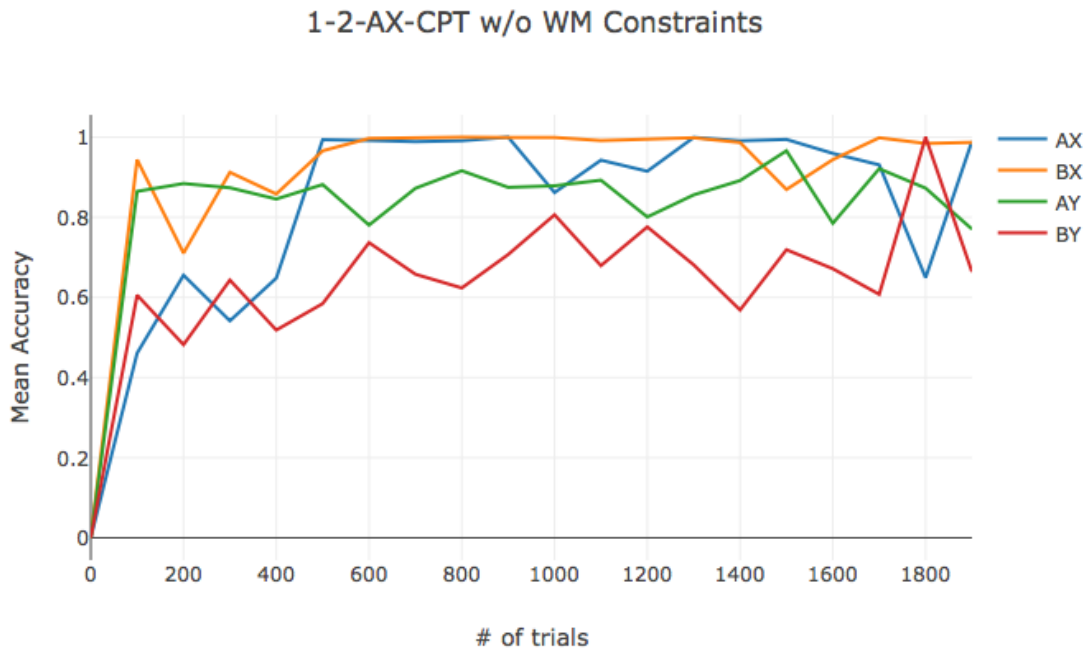
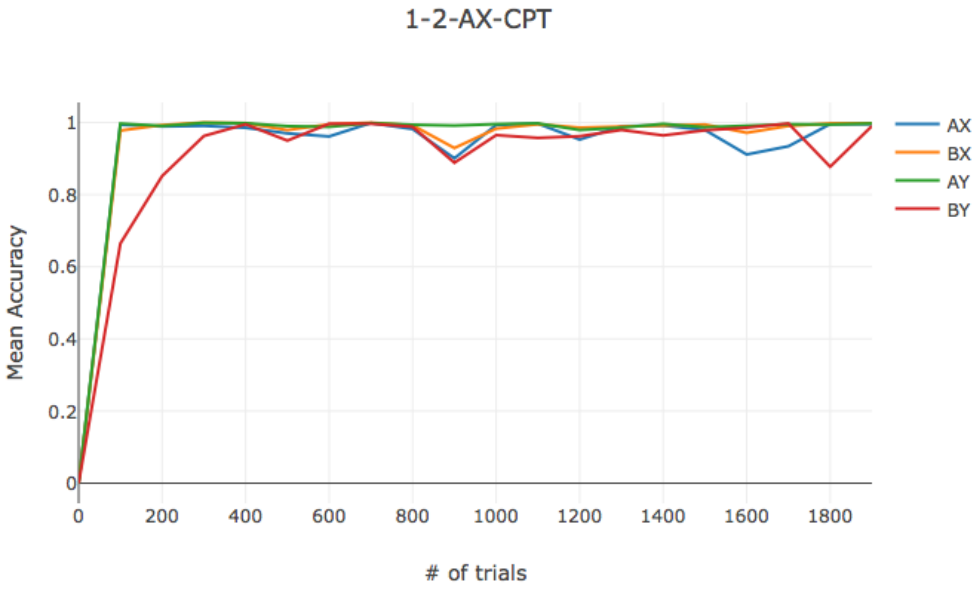


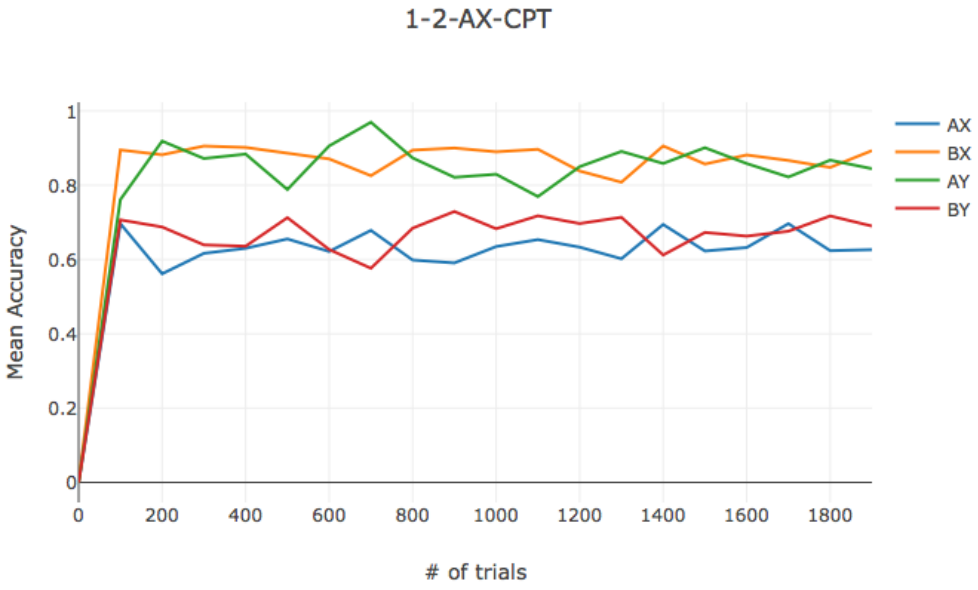
Figure 2.5: 1-2-AX-CPT task without working memory constraints. The graph depicts the mean accuracy of the 1-2-AX-CPT task over a 100 trial window. The graph shows the mean accuracy for all the A-X, B-X, A-Y and B-Y presented during each trial window.

that the working memory encoding required large HRRs in order to observe adequate learning. The HRR size needed to increase from 32000 for a 1-layer hierarchical task to 64000. If the HRR was less than 64000 we observed that the Q function for the task was unable to properly learn the correct values.

The performance metric we used in order to validate our model's ability to learn the 1-2-AX-CPT task was the mean accuracy per 100 trials. The mean accuracy was gathered by calculating the mean percentage of correct responses to the A-X, B-X, A-Y and B-Y letter sequences independently. We then graphed the mean correct action and saw that the percentages of correct actions maintained levels above 95% accuracy and was stable thereafter (see Figure 2.6). Small residual error remains due to the ε-greedy policy employed, indicating our model can successfully solve the 1-2-A-X-CPT task. Comparatively, the simple one layer model generated suboptimal accuracy as conveyed in Figure 2.6. Also,



(a) Multilayer Context Model



(b) One Layer Context Model

Figure 2.6: Mean accuracy of the 1-2-AX-CPT task using the multilayer hierarchical context model (a), and the 1-2- AX-CPT task using a simple one layer context model (b), presented over a 100 trial window. AX, BX, AY and BY trials are plotted independently.

behavioral data showed that human subjects were able to perform the 1-2-A-X-CPT task at a 95% mean accuracy rate (Nee and Brown, 2013). Once we set ε=0 our model was able to exceed human performance and reach 100% accuracy resulting in optimal behavior. We validated our model's ability to solve the A-X-CPT task by logging the mean percentage of correct actions per 100 episodes. Similarly, we logged the agent's response to the A-X, B-X, A-Y, and B-Y presentations. We observed our model was able to achieve performance accuracy of 100% and stabilize for the remaining episodes. Our model's ability to solve these cognitive tasks makes it amenable for human and animal performance data comparison.

The LSTM model was also able to successfully learn the 1-2-A-X-CPT task and obtain 100% accuracy rates while learning the task. This makes sense, being that the LSTM model uses a more informative supervised learning signal, making it easier to learn a given task. In contrast, our model only relies on reinforcement error signals to adjust the weights for learning. Also, our hierarchical model is a more biologically plausible simulation of the interaction between the prefrontal cortex and the mesolimbic dopamine system. Despite the previously stated constraints, our model was still able to obtain 100% accuracy rates while learning the 1-2-A-X-CPT task, just like the LSTM model, once we set ε=0.

Discussion

The ability to model the brain gives us the ability to run simulations of certain cognitive phenomenon in which we lack clear understanding. Our working memory models are based on the interactions between the prefrontal cortex and the mesolimbic dopamine system. Understanding the role that the dopamine system plays in relation to cognition is one aspect that our model focuses on. To model dopamine function, we use learning algorithms based on temporal differences in estimated reward. In addition, we used HRRs which allowed for us to encode concepts dynamically, as compared to previous models within the WMtk. Our model extended the WMtk by allowing for multilayer of hierarchical context to be learned.

Adding multiple layers of working memory slots gives our model the ability to handle more complex problems and tasks. Our model's capacity to learn both the A-X-CPT and 1-2-A-X-CPT validates our model's ability to learn partially observable tasks with hierarchical context reasoning. Unlike the LSTM model, our working memory model allows for the transparent viewing of the explicit storage of working memory contents in the PFC.

CHAPTER III : TRANSFER REINFORCEMENT LEARNING

The previous chapter introduced a working memory model with the ability to solve temporally extended hierarchical tasks in the reinforcement learning domain. In this chapter, we explore designing a model that has the ability to generalize not just within a task but across tasks that share similarity. This is known as the transfer learning problem. We detail a novel approach that utilizes the generalization benefits of output-gating, combined with the effectiveness of our working memory architecture for learning multiple policies (Williams and Phillips, 2020). In addition, we show that these mechanisms aid in increasing transfer across tasks as indicated by our results. We tested the model's utility on several variations of a temporally extended, partially observable 5×5 2D grid-world maze which aimed to demonstrate generalization (or lack thereof) across tasks.

Background

Transfer Learning

Transfer learning requires generalization to occur not just within tasks but across tasks (Taylor and Stone, 2009). Therefore, the goal of transfer learning is to train on a source task and then exploit that knowledge within the target task. The source task you choose can positively or negatively affect performance on the target task. If the source and target tasks are not adequately related then performance on the target task may not show improvement. This is known as negative transfer (Weiss et al., 2016; Taylor and Stone, 2009). Common metrics used to evaluate a transfer learning method's efficacy are as follows: *jumpstart*, *asymptotic performance*, *total reward*, *transfer ratio*, and *time to threshold* (Taylor and Stone, 2009). First, the metric *jumpstart* relates to increasing the initial performance on the target task. Second, *asymptotic performance* is concerned with testing the improvements of the final accuracy of the agent. Third, the metric for *total reward* compares the total reward accumulated by transfer as opposed to without transfer. Fourth, *transfer ratio* is a

ratio of the total reward acquired with and without transfer. Finally, *time to threshold* deals with how fast the agent can reach a predefined performance threshold (Taylor and Stone, 2009). We decided to use the *jumpstart* and *time to threshold* metrics to show transfer performance because we were interested in showing our models ability to reduce training time on a target task. Furthermore, *asymptotic performance* is difficult to obtain in practice and reward based metrics are specific to reinforcement learning, making them difficult to use when comparing with other forms of learning.

The PFC and basal ganglia (BG) interact with one another in order to flexibly and dynamically update information in a working memory system. The PBWM model (O'Reilly and Frank, 2006) demonstrates this interaction by utilizing a dynamic gating mechanism that sits in between the PFC and BG. This gate is modulated by the dopaminergic system through the use of reinforcement learning. The PBWM model was proven to reach performance criteria faster and have the lowest error when compared to a simple recurrent network (SRN), a recurrent backpropagation learning network (RBP), and a LSTM network. Even though PBWM shows superior generalization on single tasks, it was not tested for generalization across tasks, which is a key feature of transfer learning. It was then observed that with the addition of a PFC stripe based output-gating mechanism, the model became markedly better as simulated in a vocabulary task (Kriete and Noelle, 2011). Output-gating was even demonstrated to assist in tasks that are hierarchical in nature (Unger et al., 2016). Unlike the tasks used on the previously discussed models, some domains may contain multiple competing optimal policies and temporally-delayed feedback. Moreover, this increases the complexity of solving the task. We hypothesize that the addition of output-gating mechanisms to the HWMtk can overcome these difficulties.

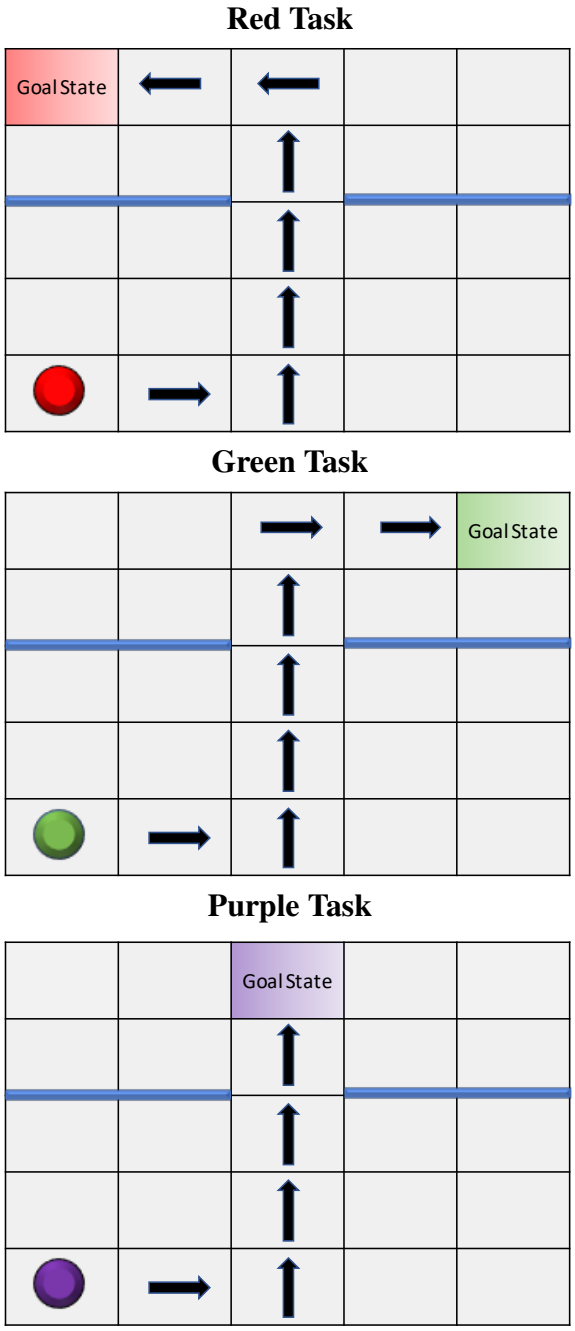


Figure 3.1: Grid-world environment for the Red, Green, and Purple tasks. The agent is indicated by the colored dot. The dot color is based on the initial flashed cue at beginning of the trial. Based on the initial cue color, the goal state will be in the upper far left corner (red cue) the upper far right corner (green cue), or the upper middle (purple cue). The grid is divided into 25 states. The thick blue lines represent barriers that restrict how the agent can transition within the grid world. The agent's goal is to reach the colored goal state in an optimal manner. The arrows show an optimal path that can be taken by the agent to reach the goal. All tasks share the same optimal policy for the lower half of the grid.

Methods

Source and Target Tasks

In order to show our model's ability to transfer knowledge, we utilize a 5×5 2-D grid-world maze task containing 25 states as our source task (see Figure 3.1). In this task the agent starts off by being randomly placed in the lower half of the maze which is located below the barriers. Once spawned, the agent is flashed a color and then released to explore the grid-world maze on its own. The available colors that the agent can experience during the initial spawn consist of red or green. These colors are cues that signal the location of the goal state. As displayed in Figure 3.1, if the agent is flashed the color red then the goal state will be located in the upper left corner of the grid-world. Also, if the agent is flashed the color green, then the goal state is positioned in the upper right corner. Finally, located 2 rows from the top of the grid-world is a barrier with a single opening. We intend to use this single point of entry onto the other side of the grid-world as the basis for forming a shared representation between our source task and target task since both tasks share an identical optimal policy for the lower half of the maze.

For knowledge transfer to take place, there must be some shared characteristics between the source task and target task. In our target task, we will maintain the same 5×5 2-D grid-world maze as that of the source task. The agent will spawn in a random position on the lower half of the maze below the barriers. Now instead of 2 colors (red and green) available at the initial state, another color (purple) is added to the list of possible cues. The addition of the color purple means that there is another task that can be present during a trial. Additionally, if purple is flashed then the goal state will be located in the upper middle portion of the grid-world maze as illustrated in Figure 3.1. Both the source task and target task share the lower half of the grid-world. We utilize this shared representation to demonstrate transfer learning between our source task and target task.

Transfer Learning Model

A detailed diagram of our model is provided in Figure 3.2. Presented in that diagram, we see that the input gate controls the flow of information into `WM Maint`. The slots in `WM Maint` represent stripes in the PFC. The input gate, output gate and the agent's motor output are each controlled by a separate neural network that store the Q values. We modulate the opening and closing of the gates using the temporal difference learning algorithm (Barto and Sutton, 1998). Holographic Reduced Representations (HRRs) (Plate, 1995) are used to encode the actions (both internal and external) and the states. HRRs key utility is that it allows the conjunction and/or disjunction of features without an increase in dimensionality. Also each representational vector is approximately orthogonal, which enables the use of an efficient single layer network. Furthermore, each slot in `WM Maint` is mapped to its own input gate and output gate. This means that an input gate designated for one slot cannot interfere with the updating of another slot's contents. Finally, the output gate for that slot would not be able to gate-out the contents of another slot.

The input gate governs whether or not information will be ignored or stored for maintenance by the agent. The input gate's neural network receives the state information of the environment as input. It also receives as input the role associated with the cue it is presented with. This role information serves as an abstract representation of the cue. For example, in Figure 3.2 we see that the cue `green` is bound with the role `color`. Furthermore, if different cues are presented, the role will still be that of `color`. This feature allows the model to effectively generalize to new cue representations. Next, the input gate's neural network determines whether the gate should open or close as detailed in Figure 3.3. This action m_{I_t} is selected by choosing the highest Q-value in the input gate network at time t as follows:

$$m_{I_t} = \underset{\mathbf{a}_I \in \mathbf{A}_I}{\operatorname{argmax}}((\mathbf{s}_I * \mathbf{a}_I) \cdot \mathbf{w}_I + b) \tag{III.1}$$

where m_{I_t} represents the highest Q-value for the input gate at time t, $\mathbf{s}_I$ is the state information for the input gate, $\mathbf{a}_I$ is an action vector for the input gate, $\mathbf{A}_I$ is the set of all possible

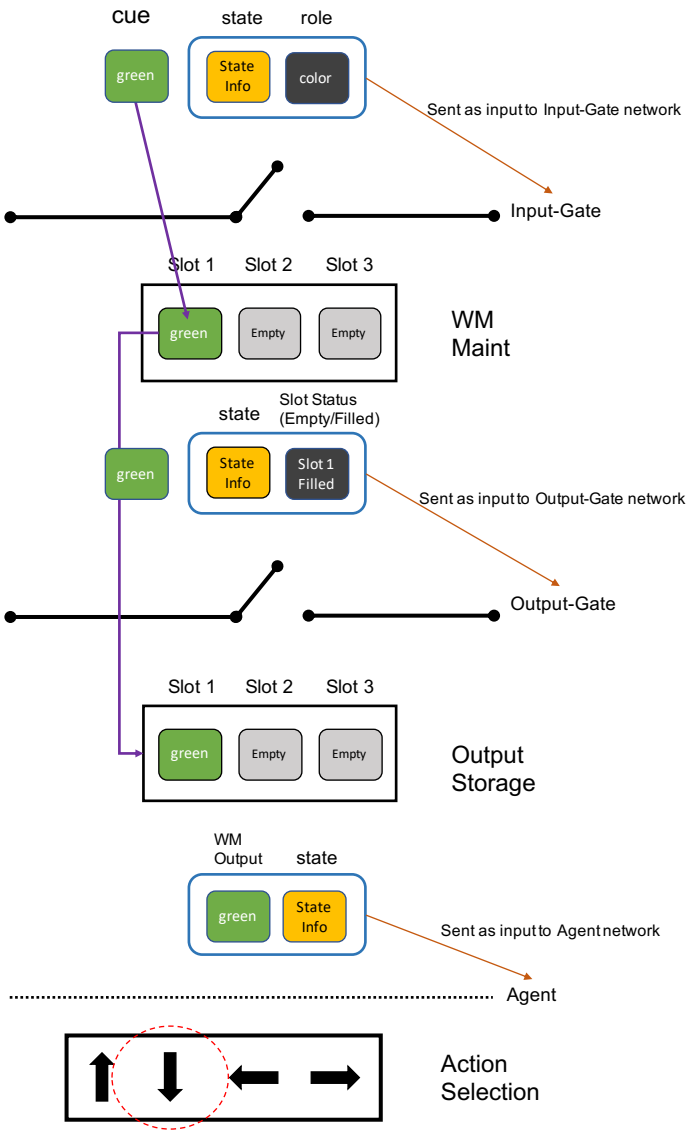


Figure 3.2: This diagram shows the flow of information within our model. Initially, the agent is presented with a color. The role associated with the cue, along with state information, is sent as input to the input gate's network where a decision to open or close the gates is determined. If the gate is open, the color is then stored in working memory and maintained, otherwise the slot remains empty. Each working memory slot has its own dedicated input gate and associated output gate. The output gate network uses the knowledge of whether or not the working memory slot is empty or filled as input. The output gate network then decides to either open the output gate or have the output gate stay closed. If opened, the contents of the working memory slot associated with the gates are then propagated as output. The output and state information is then sent to the agent network where an action is determined.

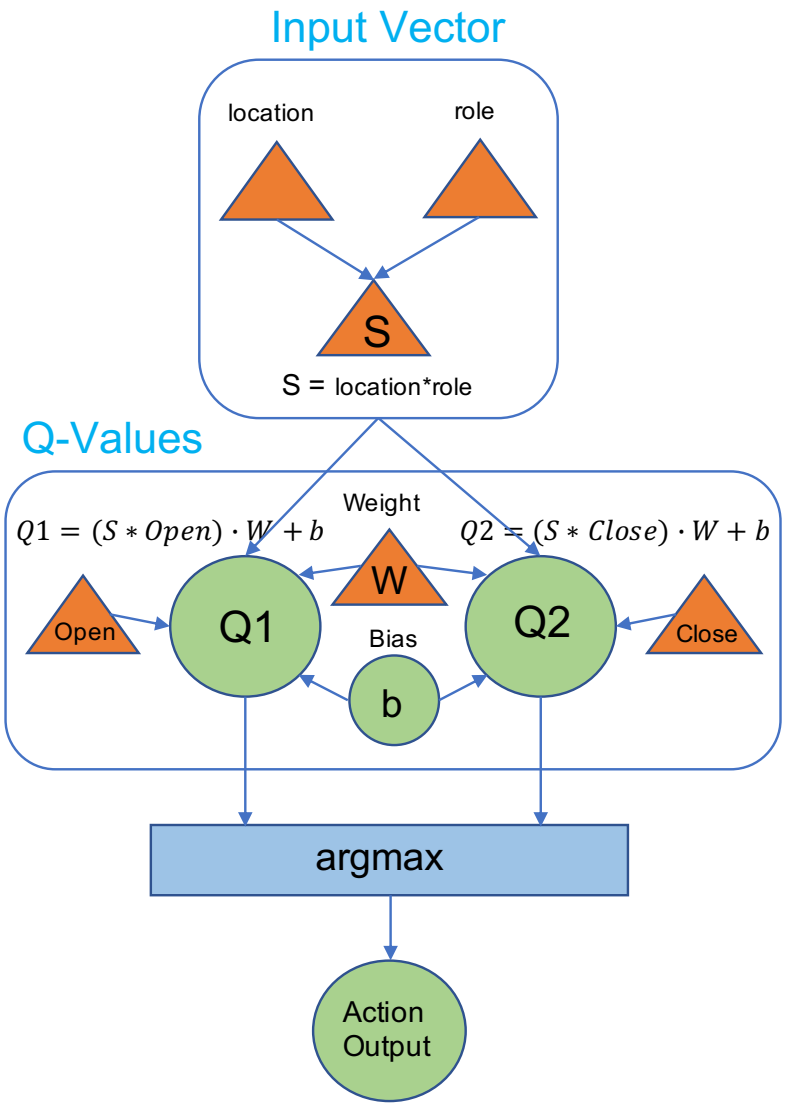


Figure 3.3: This diagram shows the mechanisms for the single layer neural network used to store the Q-values of the input gate, output gate, and agent network. The orange triangles represent vectors; the green circles represent scalar values; and the blue bar represents a function. The blue arrows show the direction in which data is flowing within the network. The input vector is created when `location` is convolved with `role`, creating the vector `S`. Next, the Q-values are computed by convolving the `S` vector with all combinations of actions (*Open* and *Close*). The dot product of the previous result and the weight vector `W` is computed and the bias `b` is added thus creating the Q-values of `Q1` and `Q2`. Finally, the Q-values are passed to the `argmax` function and the highest Q-value is selected resulting in an action selection (e.g. `Open` or `Close`) signified by `Action Output`.

action vectors for the input gate, $\mathbf{w}_I$ is the weight vector for the input gate network and b is the bias for the input gate. If opened, the cue is stored and maintained in `WM Maint` as

depicted with the purple arrow in Figure 3.2. Otherwise, the cue information is ignored and the `WM Maint` slot does not get updated.

Downstream processing is influenced by whether or not the contents in `WM Maint` are allowed to pass the output gate. The neural network for the output gate takes as input the state information of the environment and a representation that signifies whether the `WM Maint` slot is empty or filled. The output gate then decides to open or close using this formula:

$$m_{O_t} = \underset{\mathbf{a}_O \in \mathbf{A}_O}{\operatorname{argmax}}((\mathbf{s}_O * \mathbf{a}_O) \cdot \mathbf{w}_O + b) \tag{III.2}$$

where m_{O_t} represents the highest Q-value for the output gate at time t, $\mathbf{s}_O$ is the state information for the output gate, $\mathbf{a}_O$ is an action vector for the output gate, $\mathbf{A}_O$ is the set of all possible action vectors for the output gate, $\mathbf{w}_O$ is the weight vector for the output gate network and b is the bias for the output gate.

If the gate is open, then the contents of `WM Maint` is allowed to affect the downstream processing of input at the motor output layer or action selection stage. In Figure 3.2 we see the purple arrow illustrating the transition of slot contents from the maintenance layer to the output layer. Finally if the gate is selected to be closed, then the slot will be empty and the working memory contents will have no affect on the agent‘s motor action. The agent then makes an action to move in a particular direction. This action is governed by the formula:

$$m_{A_t} = \underset{\mathbf{a}_A \in \mathbf{A}_A}{\operatorname{argmax}}((\mathbf{s}_A * \mathbf{a}_A) \cdot \mathbf{w}_A + b) \tag{III.3}$$

where m_{A_t} represents the highest Q-value for the agent at time t, $\mathbf{s}_A$ is the state information for the agent, $\mathbf{a}_A$ is an action vector for the agent, $\mathbf{A}_A$ is the set of all possible actions for the agent, $\mathbf{w}_A$ is the weight vector for the agent network and b is the bias for the agent. Once the agent has completed an action, we calculate δ for each network using the following

formulas:

$$\begin{aligned} \delta_A &= (r + \gamma m_{A_t}) - m_{A_{t-1}} \\ \delta_O &= (r + \gamma m_{A_t}) - m_{O_{t-1}} \\ \delta_I &= (r + \gamma m_{A_t}) - m_{I_{t-1}} \end{aligned} \tag{III.4}$$

where r is the reward given for the current state and γ is the discount factor. We use the agent's action as the target and force the input gate and output gate to adjust their error accordingly. This helps to correlate the agent's action to the gates' actions. We then update the input gate network, output gate network, and agent network as follows:

$$\begin{aligned} \mathbf{w}_A &= \mathbf{w}_A + \alpha \delta_A \mathbf{x}_{A_t} \\ \mathbf{w}_O &= \mathbf{w}_O + \alpha \delta_O \mathbf{x}_{O_t} \\ \mathbf{w}_I &= \mathbf{w}_I + \alpha \delta_A \mathbf{x}_{I_t} \end{aligned} \tag{III.5}$$

with $\mathbf{x}_{A_t}$, $\mathbf{x}_{O_t}$ and $\mathbf{x}_{I_t}$ being the value of the input vector for each of the networks at time t.

Table 3.1 details the hyperparameters used along with the values for the input gate, output gate, and agent networks. Also, the source code is available online at: https://github.com/arthurw125/AAAI20_Transfer.

Table 3.1: Hyperparameter Descriptions and Values

Name	Value	Description
n	1024	Size of HRR vectors
ε	0.1	Probability of non-greedy action choice
γ	0.9	Discount factor
α	0.1	Learning rate
λ	0.9	Trace decay
b	1	Network bias

Model Constraints and Features

In order for the model to generate a shared representation within the task, some constraints had to be placed on the input gate and the output gate. In the earlier stages of building the model, we allowed both the input gate and output gate to have full control

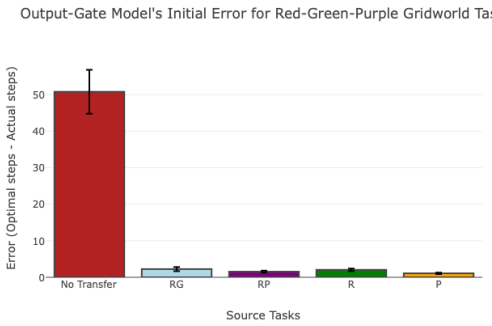
and autonomy over their actions. We provided both gates with the ability to open or close regardless of the environment encountered during each time step. We observed that the model was unable to learn on tasks that were more than 10 states large. The input gate's ability to overwrite working memory contents played a major part in causing the model to fail. Using this knowledge, we constrained the input gate such that it only has the option to open if the environment of the agent presented features available for storage. This constraint allowed our model to solve the 5×5 2-D grid-world maze task presented in Figure 3.1 which contains 25 states. Even though the model was able to solve the task, it did not exhibit the behavior that we had expected. Our desired policy was one in which the output gate would stay closed on the shared side of the maze and only open once the agent passed through the opening in the barrier. In actuality, the output gate learned a policy that allowed it to toggle actions (open or close) from one state to the next. This prevented the model from learning a shared representation, which is a feature we needed to exploit for transfer learning given it's necessity for the model of Kriete and Noelle to perform generalization.

In our previous output gate model, the working memory contents were not affected by the actions of the output gate. But due to the toggling of actions between state transitions, we changed our approach of the output gate's functionality as well. In our current model, if the output gate chooses to open then the contents of working memory is utilized only once and is then flushed and is no longer available for use by the agent. The previous iteration of our output gate model allowed for multiple uses of working memory contents for downstream processing. By constraining memory usage, the agent is forced to learn when to use it's memory contents. If the output gate stays closed then the working memory contents are maintained. Once the output gate opens, working memory is used by the agent for action selection and then flushed. The goal is to learn the state in which to gate-out what is maintained in working memory so that it yields the most reward.

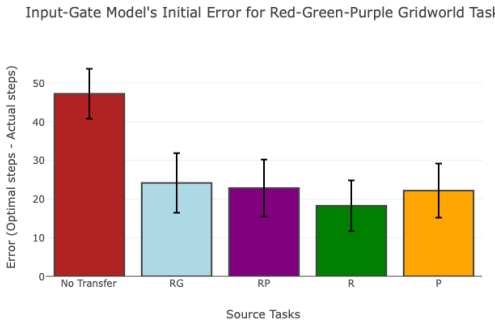
One main feature of our current output gate model is the ability to create a shared representation. The output gate model creates these shared representations by partitioning the state space into 3 parts. The first state space partitioning occurs by having the agent internally monitor whether or not there is anything loaded into the working memory slot. An internal query happens on each time step and the agent is presented with a HRR vector for `remembering` if the working memory slot is full or `not_remembering` if the working memory slot is empty. This representation is presented to the agent until the output gate is opened or the trial ends. If the agent is maintaining something in memory and gates it out, the agent is then presented with a HRR vector `memory_used`. The item stored in working memory is also presented to the agent and then flushed out of working memory, unable to be used by the agent in future time steps. Finally the `memory_used` representation persists with the agent there on after until the trial ends. This mechanism is a weak form of episodic memory which helps the agent understand the context of its own working memory decisions.

Results

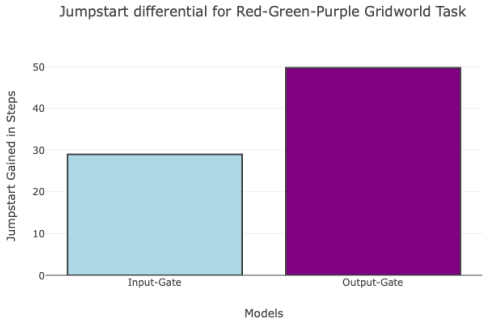
To test our model's ability to transfer knowledge across tasks, we observed whether or not there was significant *jumpstart* provided to the target task. The *jumpstart* metric deals with the reduction of initial error from training on the source task relative to the target task. Also, we measured the *time to threshold* metric for the target task and the source task for comparison on both the input gate model and output gate model. The *time to threshold* metric calculates how long it takes the error to reach a fixed error threshold for the system. For both metrics the error was determined by calculating the optimal number of steps to get to the goal minus the actual steps taken by the agent. We then took the mean of this step difference across 100 sample runs of learning. Our goal is to determine whether or not output-gating can provide significant transfer across similar task sets.



(a) Jumpstart metrics for the output gate model

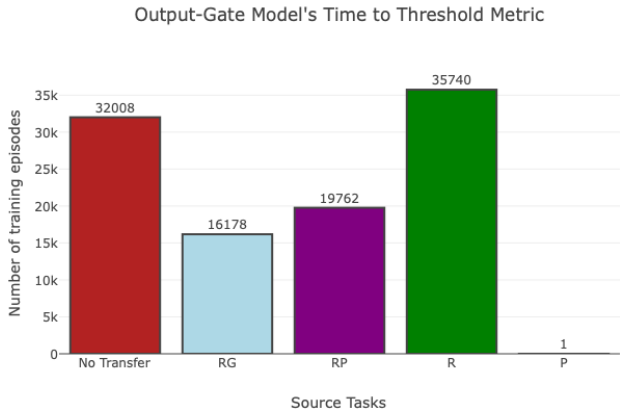


(b) Jumpstart metrics for the input gate model

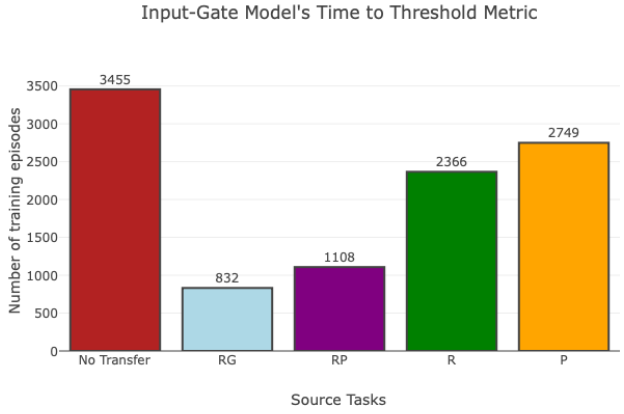


(c) Compares jumpstart between the input gate model and the output gate model

Figure 3.4: Jumpstart metrics for the output gate model (a) and the input gate model (b) display the initial error for the Red-Green-Purple grid-world task. Each model trained on a set of source tasks where `RG` is the Red-Green source task; `RP` is the Red-Purple source task; `R` is the red source task; and `P` is the purple source task. `No Transfer` means that no source task was used to train the model. Furthermore, these values come from taking the mean of 100 training sample runs. Next, jumpstart is compared between models (c) by taking the difference between the `No Transfer` training run and the lowest training run in which a source task was used for transfer.



(a) Time to threshold metrics for the output gate model



(b) Time to threshold metrics for the input gate model

Figure 3.5: Time to threshold metrics for the output gate model (a) and the input gate model (b) display the training time required to reach an error threshold for the Red-Green-Purple grid-world task. Each model trained on a set of source tasks where `RG` is the Red-Green source task; `RP` is the Red-Purple source task; `R` is the red source task; and `P` is the purple source task. `No Transfer` means that no source task was used to train the model. Furthermore, these values come from taking the mean of 100 training sample runs.

The first metric we will discuss is the *jumpstart* criterion. In our experiment, we used 4 different source tasks respectively to test the utility of our output gate model. These source tasks consist of the Red-Green task, the Red-Purple task, the Red task, and the Purple task. Looking at Figure 3.4(a) and Figure 3.4(b), we see that both the output gate model and the input gate model both achieved a noticeable level of *jumpstart* as compared to the task

Table 3.2: Time to Threshold Statistics

	Input Gate	Output Gate
No Transfer	3.46k	32k
Red-Green	0.83k	16.18k
Red-Purple	1.11k	19.76k
Red	2.37k	35.74k
Purple	2.75k	1

with no source task training. Also when we compare the *jumpstart* criterion between both the input gate model and output gate model, we see that the *jumpstart* differential is larger for the output gate (see Figure 3.4(c)). *Jumpstart* differential is measured by taking the difference between the `NO Transfer` training run and the lowest training run utilizing a source task. This is computed for each model respectively. Our findings show that the output gate model is more robust to transferring knowledge that aids in improving initial learning performance.

The *time to threshold* metric measures the time it takes to learn to a specified threshold level. This metric aims to see a decreased training time relative to the `NO Transfer` task, which is trained without source task knowledge. Also, we chose an error threshold of 1 which relates to the agent being 1 step off of being optimal within a 25 state grid-world maze. In Figure 3.5(a) and Figure 3.5(b) we see the *time to threshold* metrics for both the output gate model and the input gate model. These metrics were computed by logging the error of each episode of a simulation run. In Figure 3.5(a) we see that the we were able to achieve transfer that resulted in one-shot learning when the purple task was used as a source task with the output gate model. This is made evident by the output gate model's Purple source task only requiring 1 episode to reach the error threshold as indicated in Table 3.2 and Figure 3.5(a). Conversely, the input gate model is unable to achieve this level of performance, even though it requires significantly less training episodes to reach the error threshold. This shows that an output gate model is capable of robustly transferring knowledge resulting in immediate generalization to novel tasks.

Discussion

We utilize transfer learning throughout our everyday life without thinking about it. The mechanisms that allow the transfer of knowledge from source task to target task are pivotal as it relates to cognition. Output gate models have been shown to provide added levels of generalization as observed in several tasks (Kriete and Noelle, 2011; Kriete et al., 2013). But the question is whether such generalization is a key component of transfer? Furthermore, does generalization as it pertains to output-gating provide noticeable utility in transfer learning? We propose that generalization and specifically, the ability to control downstream processing provided through output-gating mechanisms, are integral to solving the transfer learning problem.

Our claims were validated when our model exhibited transfer learning by illustrating a marked performance increase in 2 transfer metrics: *time to threshold* and *jumpstart*. We noticed that our model was able to achieve *jumpstart* by a large margin. Additionally, our model dramatically outperformed the input gate model for that same criterion. Next, we observed that our model displayed positive transfer as it relates to the *time to threshold* criterion and was able to immediately generalize to new tasks using the Purple source task for transfer.

CHAPTER IV : MULTI-PERSPECTIVE GENERATIVE MODEL

In Chapters II and III, we discussed techniques related to reinforcement learning and working memory. In this chapter, we explore solving the multi-perspective problem with GANs. The multi-perspective problem consists of multiple cameras viewing the same location within an environment except from different perspectives. This problem is usually used to showcase object tracking and detection across multiple video feeds (Zhang and Izquierdo, 2019a; Nguyen et al., 2022; Strbac et al., 2020; Shim et al., 2021; He et al., 2018; Han et al., 2021). For the purpose of our experiment, we are concerned with generating video data as opposed to object tracking. However, research is limited on the utilization of GANs for video-to-video translation in a multi-perspective environment (Mahmud et al., 2018). Furthermore, these models fail to address the scenario in which a total camera feed must be reconstructed.

Generally, GANs are used for image-to-image translation. We know that video data is composed of a sequence of image frames. Therefore, we can transform the video-to-video translation problem into an image-to-image translation problem, thus allowing us to leverage GANs generative capabilities. With the addition of a novel loss formulation we refer to as Multi-Perspective Cycle Consistency Loss, results show that our GAN model can generate realistic video frames.

Background

Generative Adversarial Networks

GANs are the state-of-the-art in image generation and translation. The idea behind a GAN is that there are two networks, a generator and a discriminator. The purpose of the generator is to create a real enough image whereby it is able to spoof the discriminator. The discriminator's goal is to adequately classify which images are real and which are generated. The generator learns to produce fake images drawn from a distribution similar

to that of the real images it is trained on (Goodfellow et al., 2014). Given a generator G and discriminator D, both models are trained using adversarial loss which is quantified as:

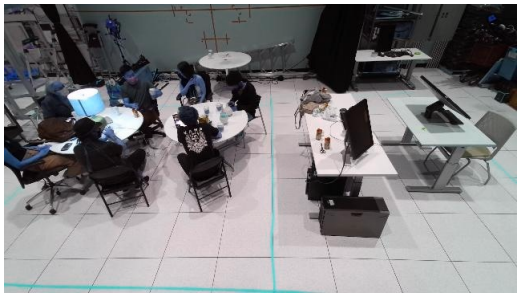
$$\begin{aligned} loss_G &= \big(D(G(z)) - 1\big)^2 \\ loss_D &= \big(D(y) - 1\big)^2 + \big(D(G(z)\big)^2 \end{aligned} \tag{IV.1}$$

where y is a real image drawn from domain Y such that $y \in Y$ and z is random input vector to introduce noise into the system.

Why would we choose GANs over other generative models such as (Kingma and Welling, 2014; Vincent et al., 2010; Sohn et al., 2015; Salakhutdinov and Hinton, 2009) for image generation? GANs are known for producing sharper images compared to other generative models. Furthermore, since a GAN is an unsupervised learning method, we do not have to concern ourselves with the labeling of data for training.

Currently, there are several GANs: Radford et al. (2016); Odena et al. (2017); Almahairi et al. (2018); Zhu et al. (2017); Isola et al. (2017); Goodfellow et al. (2014); Yoon et al. (2018); Turkoglu et al. (2019). Our model is built off the Cycle GAN (Zhu et al., 2017). The Cycle GAN is typically used for image-to-image translation where the goal is to translate an image from a source domain to a target domain. For example, given source images of horses and target images of zebras, a mapping is learned that translates an image of a horse into that of a zebra. The model is comprised of two mapping functions $G : X \rightarrow Y$ and $F : Y \rightarrow X$ where X and Y are different image domains with associated discriminators D_Y and D_X. Also, the Cycle GAN introduces cycle consistency loss which aims to regularize the mappings with the concept that if an image is translated from domain X to domain Y then we should be able to arrive back to domain X where we began. Forward cycle consistency loss and backward cycle consistency loss can be seen as:

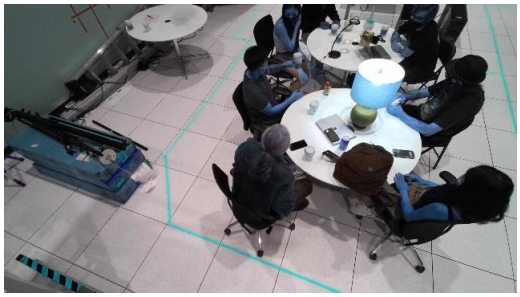
$$\begin{aligned} x \rightarrow G(x) \rightarrow F(G(x)) \approx x \\ y \rightarrow F(y) \rightarrow G(F(y)) \approx y \end{aligned} \tag{IV.2}$$



(a) Camera 1

(b) Camera 2

(c) Camera 3



(d) Camera 4

Figure 4.1: Sample Video frames from the cafe scene environment within the MMPTRACK dataset for camera 1 perspective (a), camera 2 perspective (b), camera 3 perspective (c), and camera 4 perspective (d).

where $x \in X$, $y \in Y$, function G maps to the Y domain, and function F maps to the X domain.

The Multi-Perspective Video Generation Problem

The experiment consists of four video feeds, with all the feeds viewing the same location within a room. Additionally, each feed views this location from different positions in the room and different angles. However, what if we removed one of the video feeds? Is there enough information within the remaining three video sources to produce the missing fourth video source? To further explore this question, we will need to find a dataset that possesses the aforementioned qualities.

The MMPTRACK dataset is a collection of five simulated environments with around five hours of video for training and one and a half hours of video for validation (Han et al., 2021). The environments are as follows: retail, lobby, industry, cafe, and office. Also, 28

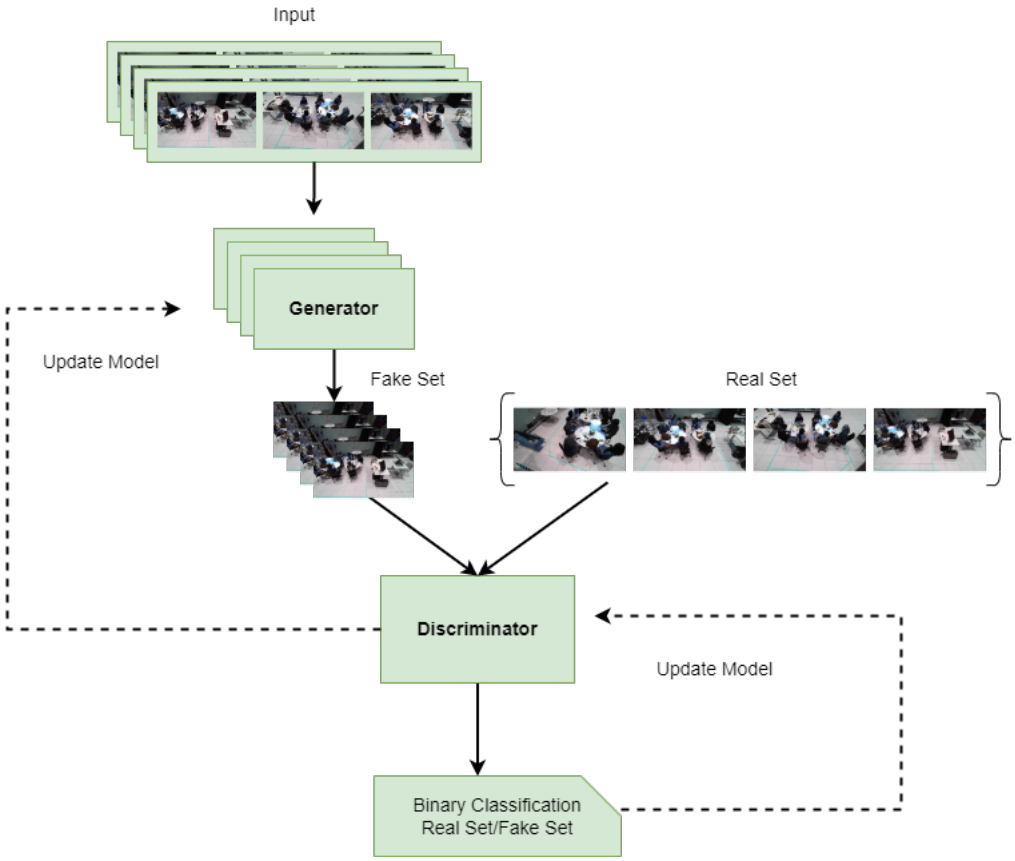


Figure 4.2: A high level overview of the Multi-Perspective GAN model conditioned on three video frames from different perspectives to generate the fourth perspective. We see four generators that takes 3 video frame perspectives as input. The outputs from the generators are used as input into the discriminator along with the real set of video frames. The discriminator classifies each set as either real or fake, and the discriminator along with the four generators are updated.

people participated in the video recording, with 14 in training, 7 in validation, and 7 in testing. The cameras were placed at different angles, with each camera having a field of view overlap with at least one of the other cameras. Video frames are synchronized and extracted at 15 frames per second at 640×320 resolution. We utilized the cafe scene for our experiment, which contained four camera feeds viewing seven individuals sitting at two tables as seen in Figure 4.1.

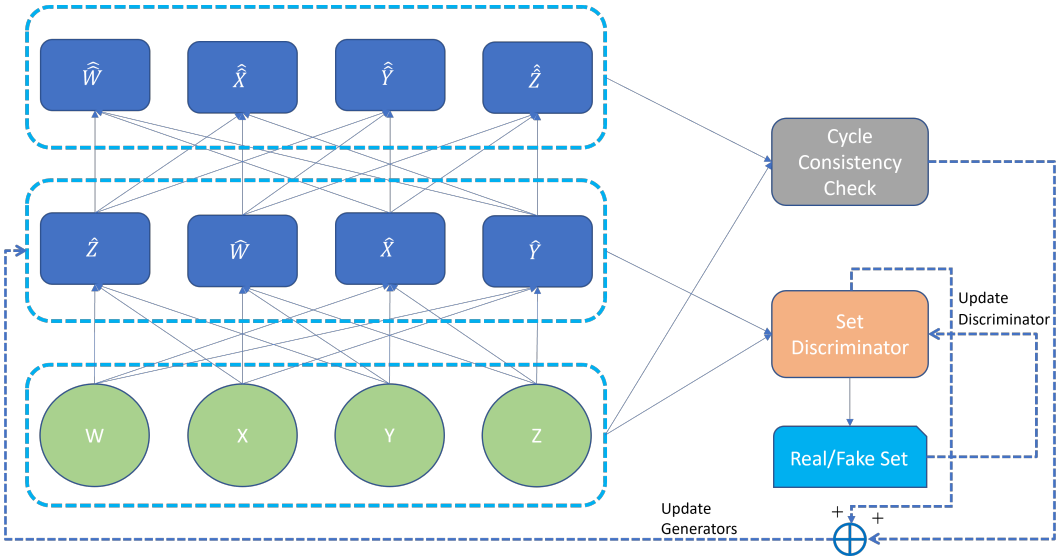


Figure 4.3: The computation of cycle consistency loss within the Multi-Perspective GAN model that we refer to as Multi-Perspective Cycle Consistency Loss. The green circles represent different video frame perspectives, marked by W, X, Y, and Z. The blue squares are four different Generators, each designated to produce a video frame perspective specified by their label. Solid blue lines represent the flow of data, while dashed blue lines represent the flow of computed loss.

Methods

Model Design

The Multi-Perspective GAN model in Figure 4.2 consists of four generators and one discriminator. Each generator is responsible for producing a different camera perspective. For example, camera perspectives 1, 2, 3, and 4 would each have a generator tasked with producing their respective video frame. Also, each generator takes three video frame perspectives as input. Therefore, given the camera perspectives 1, 2, 3, and 4, which we will denote as W, X, Y, and Z, respectively, we formulate the generators as:

$$\begin{aligned} G_W(x_i, y_i, z_i) &\rightarrow \hat{w}_i \\ G_X(w_i, y_i, z_i) &\rightarrow \hat{x}_i \\ G_Y(w_i, x_i, z_i) &\rightarrow \hat{y}_i \\ G_Z(w_i, x_i, y_i) &\rightarrow \hat{z}_i \end{aligned} \tag{IV.3}$$

where w_i, x_i, y_i, and z_i are the images at frame i for camera perspectives 1, 2, 3, and 4 respectively and $w_i \in W$, $x_i \in X$, $y_i \in Y$, and $z_i \in Z$. Next, the discriminator takes as input the set of generated frames and the set of real frames and can be defined as:

$$D_{set}\big(\{w_i,x_i,y_i,z_i\},\{\hat{w}_i,\hat{x}_i,\hat{y}_i,\hat{z}_i\}\big) \rightarrow \begin{cases} 0 & \text{if Fake} \\ 1 & \text{if Real} \end{cases} \tag{IV.4}$$

where $\{w_i,x_i,y_i,z_i\}$ is the real set of images and $\{\hat{w}_i,\hat{x}_i,\hat{y}_i,\hat{z}_i\}$ is the generated set of images at frame i. Also, D_{set} outputs a 0 if it predicts that the set is fake/generated and outputs a 1 if it predicts that the set is real.

The generator and discriminator are trained using adversarial loss. The generator is updated by minimizing the sum squared difference between the predicted and expected values as though the generated image set is real. The loss for G_W, G_X, G_Y, and G_Z is:

$$loss_{G_W,G_X,G_Y,G_Z} = \big(D_{set}(\{\hat{w}_i,\hat{x}_i,\hat{y}_i,\hat{z}_i\}) - 1\big)^2 \tag{IV.5}$$

The discriminator is updated by minimizing the sum squared difference between predicted and expected values for real and fake image sets. The loss for D_{set} is:

$$\begin{aligned} loss_{D_{set}} = &\big(D_{set}(\{w_i,x_i,y_i,z_i\}) - 1\big)^2 \\ &+\big(D_{set}(\{\hat{w}_i,\hat{x}_i,\hat{y}_i,\hat{z}_i\})\big)^2 \end{aligned} \tag{IV.6}$$

Cycle consistency for the Multi-Perspective GAN requires a two stage approach because the mappings are not a one-to-one translation as illustrated in Figure 4.3. Each generator is conditioned with 3 frame perspectives as input. We utilized the outputs $\hat{w},\hat{x},\hat{y}$, and $\hat{z}$ from the generators and feed them as input back into the same generators to produce a set of image frames $\hat{\hat{w}},\hat{\hat{x}},\hat{\hat{y}}$, and $\hat{\hat{z}}$. The forward cycle is represented as:

$$\begin{aligned} G_W(\hat{x},\hat{y},\hat{z}) &\rightarrow \hat{\hat{w}} \\ G_X(\hat{w},\hat{y},\hat{z}) &\rightarrow \hat{\hat{x}} \\ G_Y(\hat{w},\hat{x},\hat{z}) &\rightarrow \hat{\hat{y}} \\ G_Z(\hat{w},\hat{x},\hat{y}) &\rightarrow \hat{\hat{z}} \end{aligned} \tag{IV.7}$$

The real $w, x, y,$ and z are then compared with $\hat{\hat{w}}, \hat{\hat{x}}, \hat{\hat{y}},$ and $\hat{\hat{z}}$ to produce the cycle consistency loss as follows:

$$\begin{aligned} \underset{cyc}{loss_W} &= \left(w_i - \hat{\hat{w}}_i\right)^2 \\ \underset{cyc}{loss_X} &= \left(x_i - \hat{\hat{x}}_i\right)^2 \\ \underset{cyc}{loss_Y} &= \left(y_i - \hat{\hat{y}}_i\right)^2 \\ \underset{cyc}{loss_Z} &= \left(z_i - \hat{\hat{z}}_i\right)^2 \end{aligned} \tag{IV.8}$$

$$\underset{cyc}{total\_loss} = \underset{cyc}{loss_W} + \underset{cyc}{loss_X} + \underset{cyc}{loss_Y} + \underset{cyc}{loss_Z} \tag{IV.9}$$

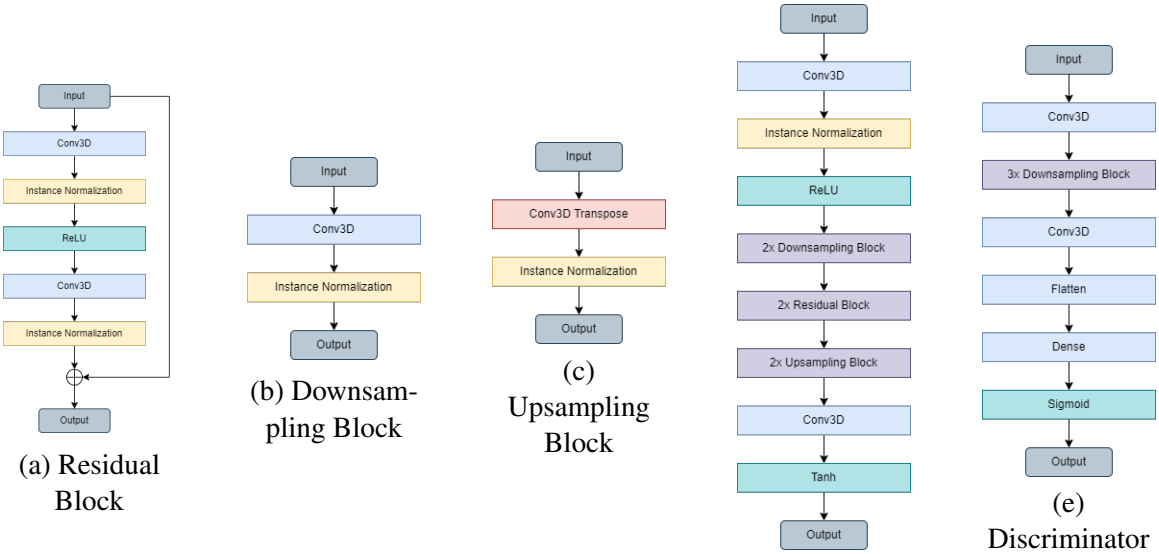
where w_i, x_i, y_i, and z_i are the real images at frame i and $\hat{\hat{w}}_i, \hat{\hat{x}}_i, \hat{\hat{y}}_i$, and $\hat{\hat{z}}_i$ are the generated images at frame i after the outputs in Equation IV.3 are used as inputs for generators G_W, G_X, G_Y, and G_Z as seen in Figure 4.2. Therefore, the total loss used to update the generator models is defined as:

$$loss_{total} = loss_{G_W, G_X, G_Y, G_Z} + \lambda \underset{cyc}{total\_loss} \tag{IV.10}$$

where $loss_{G_W, G_X, G_Y, G_Z}$ is the loss as defined in Equation IV.5, λ is a hyperparameter that controls the importance of the cycle consistency loss, and $\underset{cyc}{total\_loss}$ is the loss as defined in Equation IV.9.

Implementation

Our Multi-Perspective GAN model consists of four generator networks. All the generator networks have the same architecture as illustrated in Figure 4.4d. The network architecture for the generator is composed of a 3d-convolution layer with 64 filters and a 7x7x7 kernal size. The next layer is instance normalization followed by a rectified linear activation unit (ReLU). Also, the generator contains two downsampling blocks from Figure 4.4b, two residual blocks (He et al., 2016) from Figure 4.4a, and two upsampling blocks from Figure 4.4c. Finally, the output from the previous upsampling block proceeds to a 3d-



(a) Residual Block

(b) Downsampling Block

(c) Upsampling Block

(d) Generator

(e) Discriminator

Figure 4.4: The neural networks for the Multi-Perspective GAN model. The Residual Block (a), Downsampling Block (b), and Upsampling Block (c) are smaller networks, when put together aid in the building of larger network architectures such as the Generator model (d) and Discriminator model (e).

convolution layer that has three filters and a kernal size of 7x7x7 with a hyperbolic tangent (Tanh) activation function.

We utilize one discriminator network to classify the set of generated and real frames as either real or fake. A network overview can be seen in Figure IV.4. The network consists of a 3d-convolution layer containing 64 filters with a kernal size of 4x4x4 and a 2x2x2 stride. Also, the discriminator network has three downsampling blocks, and a 3d-convolution layer with one filter, 4x4x4 kernal size, and a 1x1x1 stride. Next, the output from the previous layer is flattened and sent to a fully connected layer with a sigmoid activation function. All models were implemented in Keras/Tensorflow and the source code can be found at https://github.com/arthurw125/GAN_Research.

Model Training

The model was trained for 50 epochs with at batch size of 16. The image frames were scaled down from 640×320 to 100×56 before being sent as input into the model.

(a) Real Frame Set from the cafe scene environment

(b) Generated Frame Set with Multi-Perspective Cycle Consistency Loss

(c) Generated Frame Set without Multi-Perspective Cycle Consistency Loss

Figure 4.5: The real and generated set of video frames across the four different camera perspectives from the cafe scene environment within the MMPTRACK dataset. The image frames were scaled down from the original size of 640×320 to 100×56. The model was trained for 50 epochs.

Table 4.1: Training Hyperparameters

Name	Value	Description
α	$2e{-}4$	Learning rate for Adam optimizer
β_1	0.5	1^{st} moment exp. decay for Adam optimizer
β_2	0.999	2^{st} moment exp. decay for Adam optimizer
ε	$1e{-}7$	Numerical stability constant
λ	10	Cycle consistency loss importance level

The image frames were scaled down to reduce the network size and processing time during training. The optimization method used during training was Adam (Kingma and Ba, 2015), which is the combination of AdaGrad (Duchi et al., 2011) and RMSProp and was applied to the four generator networks and the discriminator network. The Adam hyperparameter of the learning rate α was set to $2e{-}4$ and the exponential decay β_1 was set to 0.5 and λ set to 10. Also, the Adam hyper-parameter ε was set to $1e-7$. We trained the model on

Table 4.2: MSE for Generators

Generator	MSE w/ Cycle Loss	MSE w/o Cycle Loss
Cam1	0.020171724	0.39746407
Cam2	0.0850371	0.23035194
Cam3	0.042203967	0.22101375
Cam4	0.06065816	0.24351284

1000 image frames from the cafe scene within the MMPTRACK dataset and withheld 200 image frames for testing. A list of the training hyperparameters is available in Table 4.1.

Results

We performed an ablation study on the cycle consistency loss function of our Multi-Perspective GAN model. Also, we provided qualitative and quantitative metrics to evaluate the efficacy of our model. We used Mean Squared Error (MSE) to evaluate how far off the generated image frames are from the real image frames. In Table 4.2, we show the MSE results with and without our Multi-Perspective Cycle Consistency Loss for all four camera perspectives. Furthermore, Figure 4.5 shows the real image frame set (see Figure 4.5a) along with a generated image frame set utilizing Multi-Perspective Cycle Consistency Loss (see Figure 4.5b) and without utilizing it (see Figure 4.5c). These results were gathered from a 200 image frame test set from the cafe scene within the MMPTRACK dataset. The camera perspectives for the generators in Table 4.2 are mapped to Figures 4.5b and 4.5c where the top left image corresponds to Cam1, the top right image corresponds to Cam2, the bottom left image corresponds to Cam3, and the bottom right image corresponds to Cam4.

Discussion

There is limited research dedicated to applying GANs for video generation. Furthermore, the multi-perspective video generation problem is an area that GAN research has not fully explored. Typically, multi-perspective problems are in the area of object tracking

within a multi-camera environment (Zhang and Izquierdo, 2019b; Bredereck et al., 2012; Khan et al., 2021; Liu et al., 2017; Ristani et al., 2016). Our goal is to show that additional mechanisms must be added to GANs in order to generate realistic video frames in a multi-perspective environment. To do so, we introduced a novel cycle consistency loss that we refer to as Multi-Perspective Cycle Consistency Loss.

Qualitative results in Figure 4.5b show that our model is able to produce realistic video frames when compared to the true image frames for each camera perspective in 4.5a. We are able to achieve these quality generated video frames because of the addition of Multi-Perspective Cycle Consistency Loss. We are able to make this claim because when Multi-Perspective Cycle Consistency Loss is removed, we notice a significant degradation in video frame quality as seen in Figure 4.5c. We also observe an order of magnitude increase in MSE when Multi-Perspective Cycle Consistency Loss is ablated from our Multi-Perspective GAN model.

Even though our model is able to produce realistic video frames, it still suffers from some mode collapse in scenes that have minimal change. Future work will consists of exploring additional mechanisms that can be added that will better capture small changes that may occur over a long temporal horizon. This may include utilizing transformers for the generator and discriminator model (Lee et al., 2019; Jiang et al., 2021). In addition, relational networks (Santoro et al., 2017, 2018) could be useful because of their capacity to learn relationships between objects within an image and across a set of images.

CHAPTER V : CONCLUSION

Machine learning algorithms are powerful tools that can solve a multitude of problems and tasks. But that does not mean they can solve every problem. Some problems are too difficult for vanilla machine learning algorithms and additional mechanisms are needed. To effectively tackle these problems, we draw inspiration from the human brain and its ability to utilize working memory to solve complex tasks. Understanding that some tasks require more cognitive load than others allows us to implement higher-order thinking skills within the machine learning domain, and progresses us closer to creating an agent that exhibits higher-order cognition. Current machine learning research is mainly concerned with beating specific benchmarks in accuracy and error. Furthermore, minimal focus is given on internal cognitive actions within an artificial agent and how that can aid in creating a more robust artificial intelligence system. This work demonstrates that by augmenting machine learning algorithms with a cognitive inspired approach, we can achieve better performance results on certain tasks while furthering our understanding of the potential neural substrates of the human mind.

Reinforcement learning algorithms lack the mechanisms needed to effectively solve temporally extended hierarchical problems. Designing a system with the ability to solve *multilayer hierarchical problems* required that we provide additional mechanisms to the reinforcement learning framework. By providing a working memory system based on the interaction between the mesolimbic dopamine system and the prefrontal cortex, we were able to successfully solve a common hierarchical working memory load task in the cognitive science known as the 1-2-AX-CPT task. Furthermore, by augmenting our working model with an output-gating mechanism, we were able to generalize not just within task but across tasks within similar domains. These changes allowed our model to exhibit transfer learning in a partially observable reinforcement learning domain as tested by our grid-world maze task.

GANs are great for image-to-image translation. Adversarial loss coupled with cycle consistency allows for a realistic image generation. But what about video-to-video translation? Currently, there is minimal research in the area of using GANs for video-to-video translation. Moreover, GANs are mostly used for translating an image from one domain to another. In comparison, we are concerned with taking frames from three different video sources, and recreating the missing fourth video frame. These frame are sourced from cameras viewing the same scene but from different perspectives. To achieve this, we designed a novel cycle consistency loss we refer to as Multi-Perspective Cycle Consistency Loss. Results show that the addition of utilizing our Multi-Perspective Cycle Consistency Loss allows for a sharper and more realistic video frame generation in this domain.

BIBLIOGRAPHY

Almahairi, A., Rajeswar, S., Sordoni, A., Bachman, P., and Courville, A. (2018). Augmented CycleGAN: Learning Many-to-Many Mappings from Unpaired Data. *35th International Conference on Machine Learning, ICML 2018*, 1:300–309.

Anderson, L. W., Krathwohl Peter W Airasian, D. R., Cruikshank, K. A., Mayer, R. E., Pintrich, P. R., Raths, J., and Wittrock, M. C. (2001). *A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives*. Pearson, 1st edition.

Barto, A. G. and Sutton, R. S. (1998). Reinforcement learning: An introduction. *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, 9(5):1054.

Braver, T. S., Rush, B. K., Satpute, A. B., Racine, C. A., and Barch, D. M. (2005). Context processing and context maintenance in healthy aging and early stage dementia of the Alzheimer's type. *Psychology and Aging*, 20(1):33–46.

Bredereck, M., Jiang, X., Körner, M., and Denzler, J. (2012). Data association for multi-object tracking-by-detection in multi-camera networks. In *2012 Sixth International Conference on Distributed Smart Cameras (ICDSC)*, pages 1–6.

Busch, M. A., Skubic, M., Keller, J. M., and Stone, K. E. (2007). A robot in a water maze: Learning a spatial memory task. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 1727–1732.

Dubois, G. M. and Phillips, J. L. (2017). Working Memory Concept Encoding Using Holographic Reduced Representations. In *Proceedings of the 28th Modern Artificial Intelligence and Cognitive Science Conference*.

Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(61):2121–2159.

Erdemir, E., Frankel, C. B., Kawamura, K., Gordon, S. M., Thornton, S., and Ulutas, B. (2008). Towards a cognitive robot that uses internal rehearsal to learn affordance relations. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2016–2021.

Frank, M. J., Loughry, B., and O'Reilly, R. C. (2001). Interactions between frontal cortex and basal ganglia in working memory: A computational model. *Cognitive, Affective, & Behavioral Neuroscience*, 1(2):137–160.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N., and Weinberger, K., editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc.

Han, X., You, Q., Wang, C., Zhang, Z., Chu, P., Hu, H., Wang, J., and Liu, Z. (2021). Mmptrack: Large-scale densely annotated multi-camera multiple people tracking benchmark.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778.

He, W., Tao, W., Sun, K., Xu, L., Fu, Q., and Zhao, H. (2018). Multi-camera object tracking via deep metric learning. In Yu, Y., Zuo, C., and Qian, K., editors, *Sixth International Conference on Optical and Photonic Engineering (icOPEN 2018)*, volume 10827, pages 331 – 339. International Society for Optics and Photonics, SPIE.

Hochreiter, S. and Urgen Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780.

Huang, X., Liu, M. Y., Belongie, S., and Kautz, J. (2018). Multimodal Unsupervised Image-to-Image Translation. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 11207 LNCS:179–196.

Isola, P., Zhu, J. Y., Zhou, T., and Efros, A. A. (2017). Image-to-image translation with conditional adversarial networks. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017-Janua:5967–5976.

Jiang, Y., Chang, S., and Wang, Z. (2021). Transgan: Two pure transformers can make one strong gan, and that can scale up. In Ranzato, M., Beygelzimer, A., Dauphin, Y., Liang, P., and Vaughan, J. W., editors, *Advances in Neural Information Processing Systems*, volume 34, pages 14745–14758. Curran Associates, Inc.

Kawamura, K., Gordon, S. M., Ratanaswasd, P., Erdemir, E., and Hall, J. F. (2008). Implementation of cognitive control for a humanoid robot. *International Journal of Humanoid Robotics*, 05(04):547–586.

Khan, M. N., Al Hasan, M., and Anwar, S. (2021). Improving the robustness of object detection through a multi-camera–based fusion algorithm using fuzzy logic. *Frontiers in Artificial Intelligence*, 4.

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In Bengio, Y. and LeCun, Y., editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Kingma, D. P. and Welling, M. (2014). Auto-Encoding Variational Bayes. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*.

Kriete, T. and Noelle, D. C. (2011). Generalisation benefits of output gating in a model of prefrontal cortex. *Connection Science*, 23(2):119–129.

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

Lee, J., Lee, Y., Kim, J., Kosiorek, A., Choi, S., and Teh, Y. W. (2019). Set transformer: A framework for attention-based permutation-invariant neural networks. In Chaudhuri, K. and Salakhutdinov, R., editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 3744–3753. PMLR.

Liu, W., Camps, O., and Sznaier, M. (2017). Multi-camera multi-object tracking.

Mahmud, T., Billah, M., and Roy-Chowdhury, A. K. (2018). Multi-view frame reconstruction with conditional gan. In *2018 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, pages 1164–1168.

Mirza, M. and Osindero, S. (2014). Conditional generative adversarial nets.

Nee, D. E. and Brown, J. W. (2013). Dissociable frontal-striatal and frontal-parietal networks involved in updating hierarchical contexts in working memory. *Cerebral Cortex*, 23(9):2146–2158.

Nguyen, P., Quach, K. G., Duong, C. N., Le, N., Nguyen, X.-B., and Luu, K. (2022). Multi-camera multiple 3d object tracking on the move for autonomous vehicles.

Odena, A., Olah, C., and Shlens, J. (2017). Conditional image synthesis with auxiliary classifier gans. *34th International Conference on Machine Learning, ICML 2017*, 6:4043–4055.

O'Reilly, R. and Munakata, Y. (2000a). *Computational Explorations in Cognitive Neuroscience*. A Bradford Book.

O'Reilly, R. and Munakata, Y. (2000b). *Computational Explorations in Cognitive Neuroscience*. MIT Press.

O'Reilly, R. C. and Frank, M. J. (2006). Making Working Memory Work: A Computational Model of Learning in the Prefrontal Cortex and Basal Ganglia. *Neural Computation*, 18(2):283–328.

Phillips, J. L. and Noelle, D. C. (2005). A Biologically Inspired Working Memory Framework for Robots *. *In Proceedings of the 27th Annual Meeting of the Cognitive Science Society*.

Plate, T. A. (1995). Holographic Reduced Representations. *IEEE TRANSACTIONS ON NEURAL NETWORKS*, 6(3).

Radford, A., Metz, L., and Chintala, S. (2016). Unsupervised representation learning with deep convolutional generative adversarial networks. *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, pages 1–16.

Richland, L. E. and Simms, N. (2015). Analogy, higher order thinking, and education. *Wiley Interdisciplinary Reviews: Cognitive Science*, 6(2):177–192.

Ristani, E., Solera, F., Zou, R. S., Cucchiara, R., and Tomasi, C. (2016). Performance measures and a data set for multi-target, multi-camera tracking.

Salakhutdinov, R. and Hinton, G. (2009). Deep boltzmann machines. In van Dyk, D. and Welling, M., editors, *Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics*, volume 5 of *Proceedings of Machine Learning Research*, pages 448–455, Hilton Clearwater Beach Resort, Clearwater Beach, Florida USA. PMLR.

Santoro, A., Hill, F., Barrett, D., Morcos, A., and Lillicrap, T. (2018). Measuring abstract reasoning in neural networks. In *International Conference on Machine Learning*, pages 4477–4486.

Santoro, A., Raposo, D., Barrett, D. G., Malinowski, M., Pascanu, R., Battaglia, P., and Lillicrap, T. (2017). A simple neural network module for relational reasoning. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Schultz, W., Dayan, P., and Montague, P. R. (1997). A neural substrate of prediction and reward. *Science*, 275(June 1994):1593–1599.

Shim, K., Yoon, S., Ko, K., and Kim, C. (2021). Multi-target multi-camera vehicle tracking for city-scale traffic management. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 4188–4195.

Smith, A. J., Li, M., Becker, S., and Kapur, S. (2007). Linking animal models of psychosis to computational models of dopamine function. *Neuropsychopharmacology : official publication of the American College of Neuropsychopharmacology*, 32(1):54–66.

Sohn, K., Lee, H., and Yan, X. (2015). Learning structured output representation using deep conditional generative models. In Cortes, C., Lawrence, N., Lee, D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc.

Strbac, B., Gostovic, M., Lukac, Z., and Samardzija, D. (2020). Yolo multi-camera object detection and distance estimation. In *2020 Zooming Innovation in Consumer Technologies Conference (ZINC)*, pages 26–30.

Sutton, R. S. (1988). Learning to Predict by the Methods of Temporal Differences. *Machine Learning*, 3:9–44.

Taylor, M. E. and Stone, P. (2009). Transfer Learning for Reinforcement Learning Domains: A Survey. *Journal of Machine Learning Research*, 10:1633–1685.

Turkoglu, M. O., Thong, W., Spreeuwers, L., and Kicanaoglu, B. (2019). A layer-based sequential framework for scene generation with gans. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):8901–8908.

Unger, K., Ackerman, L., Chatham, C. H., Amso, D., and Badre, D. (2016). Working memory gating mechanisms explain developmental change in rule-guided behavior. *Cognition*, 155:8–22.

Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., and Manzagol, P.-A. (2010). Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, 11(110):3371–3408.

Wang, X., Tugcu, M., Hunter, J. E., and Wilkes, D. M. (2009). Exploration of configural representation in landmark learning using working memory toolkit. *Pattern Recognition Letters*, 30(1):66 – 79.

Weiss, K., Khoshgoftaar, T. M., and Wang, D. (2016). A survey of transfer learning. *Journal of Big Data*, 3(1):9.

Williams, A. and Phillips, J. (2018). Multilayer Context Reasoning in a Neurobiologically Inspired Working Memory Model for Cognitive Robots. *Proceedings of the 40th Annual Meeting of the Cognitive Science Society*, pages 2687–2692.

Williams, A. and Phillips, J. (2020). Transfer Reinforcement Learning Using Output-Gated Working Memory. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(02):1324–1331.

Yoon, J., Jordon, J., and Van Der Schaar, M. (2018). GAIN: Missing data imputation using generative adversarial nets. *35th International Conference on Machine Learning, ICML 2018*, 13:9042–9051.

Zhang, X. and Izquierdo, E. (2019a). Real-time multi-target multi-camera tracking with spatial-temporal information. In *2019 IEEE Visual Communications and Image Processing (VCIP)*, pages 1–4.

Zhang, X. and Izquierdo, E. (2019b). Real-time multi-target multi-camera tracking with spatial-temporal information. In *2019 IEEE Visual Communications and Image Processing (VCIP)*, pages 1–4.

Zhu, J. Y., Park, T., Isola, P., and Efros, A. A. (2017). Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks. *Proceedings of the IEEE International Conference on Computer Vision*, 2017-October:2242–2251.