Transfer Reinforcement Learning using Output-Gated Working Memory

Arthur S. Williams
Center for Computational Science
Middle Tennessee State University
Murfreesboro, TN, USA
asw3x@mtmail.mtsu.edu

Joshua L. Phillips
Department of Computer Science
Middle Tennessee State University
Murfreesboro, TN, USA
Joshua.Phillips@mtsu.edu

Abstract

Transfer learning allows for knowledge to generalize across tasks, resulting in increased learning speed and/or performance. These tasks must have commonalities that allow for knowledge to be transferred. The main goal of transfer learning in the reinforcement learning domain is to train and learn on one or more source tasks in order to learn a target task that exhibits better performance than if transfer was not used (Taylor and Stone 2009). Furthermore, the use of output-gated neural network models of working memory has been shown to increase generalization for supervised learning tasks (Kriete and Noelle 2011; Kriete et al. 2013). We propose that working memory-based generalization plays a significant role in a model's ability to transfer knowledge successfully across tasks. Thus, we extended the Holographic Working Memory Toolkit (HWMtk) (Dubois and Phillips 2017; Phillips and Noelle 2005) to utilize the generalization benefits of output gating within a working memory system. Finally, the model's utility was tested on a temporally extended, partially observable 5x5 2D grid-world maze task that required the agent to learn 3 tasks over the duration of the training period. The results indicate that the addition of output gating increases the initial learning performance of an agent in target tasks and decreases the learning time required to reach a fixed performance threshold.

Introduction

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).

Copyright © 2020, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

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 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 and output gating, respectively. It has been observed in a generalization task that the addition of an output gate markedly improved generalization (Kriete and Noelle 2011). Also, in that same task it was shown that as the action space size increased, the model's generalization disparity between having an output gate and not having one became more pronounced. This suggests that as we increase task difficulty, the output gate's utility as it pertains to generalization also increases.

The n-task learning algorithm (nTL) (Phillips and Jovanovich 2018) extended temporal difference (TD) learning models (Barto and Sutton 1998; Sutton 1988) by utilizing abstract representations to generate multiple policies allowing for dynamic task switching. While nTL learns multiple, disparate tasks, its metrics do not show transfer across tasks. 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. The updated model's utility was tested on several variations of a temporally extended, partially observable 5×5 2D grid-world maze task which aim to demonstrate generalization (or lack thereof) across tasks. We anticipate the output gating approach to be critical to success.

Background

Working Memory Models

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 2000). 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). 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, Dayan, and Montague 1997). 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, Dayan, and Montague 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.

Our working memory model captures the PFC and dopamine system through the use of the temporal difference (TD) 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 action selection, making it well suited for leaning internal action selection as well. This would consist of the opening or closing of circuits in the PFC for either flushing working memory contents or maintaining them.

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.

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, Khoshgoftaar, and Wang 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.

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 SRN, RBP, and LSTM networks. Even though PBWM shows superior generalization, 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 proven 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.

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 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 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

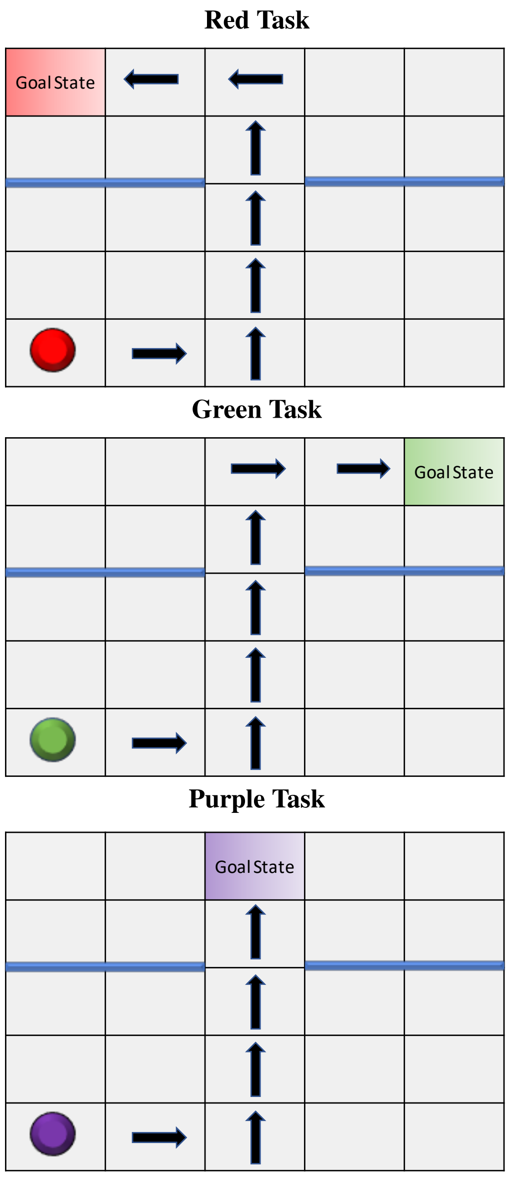


Figure 1: Grid-world environment for the Red, Green, and Purple tasks. The agent is indicated by the colored dot. The dot color is based on the initial flashed cue at beginning of the trial. Based on the initial cue color, the goal state will be in the upper far left corner (red cue) the upper far right 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.

will be located in the upper middle portion of the grid-world maze as illustrated in Figure 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 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 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 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. This action m_{I_t} is selected by choosing the highest Q-value in the input gate network at time t. The formula can be computed as follows:

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

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 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 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

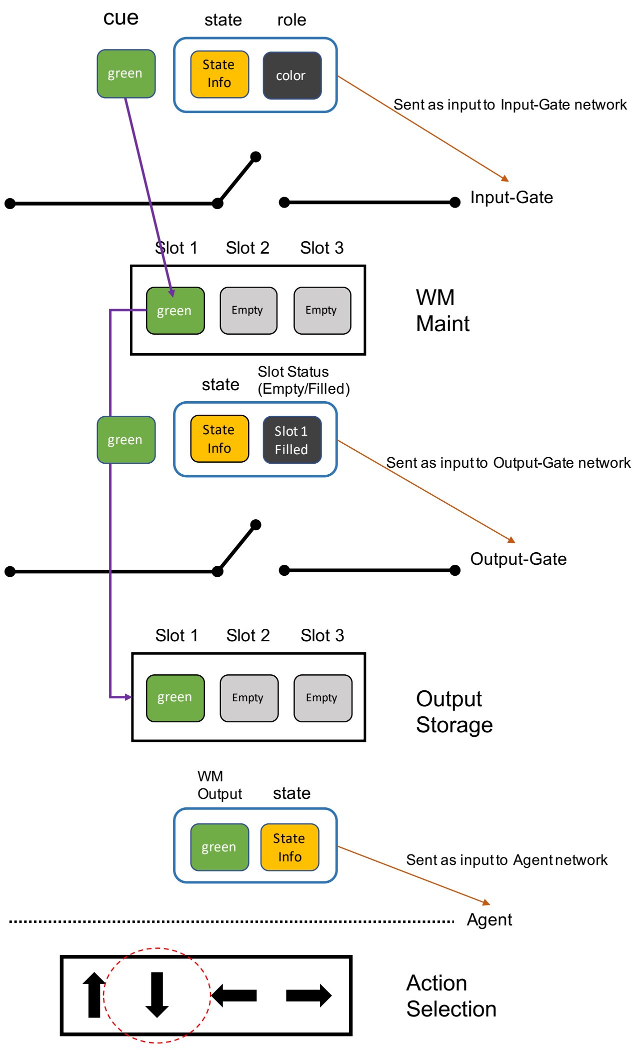


Figure 2: This diagram shows the flow of information within our model. Initially, the agent is presented with a color. The role associated with the cue, along with state information, is sent as input to the input gate's network where a decision to open or close the gates is determined. If the gate is 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's network uses the knowledge of whether or not the working memory slot is empty or filled as input. The output gate's 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.

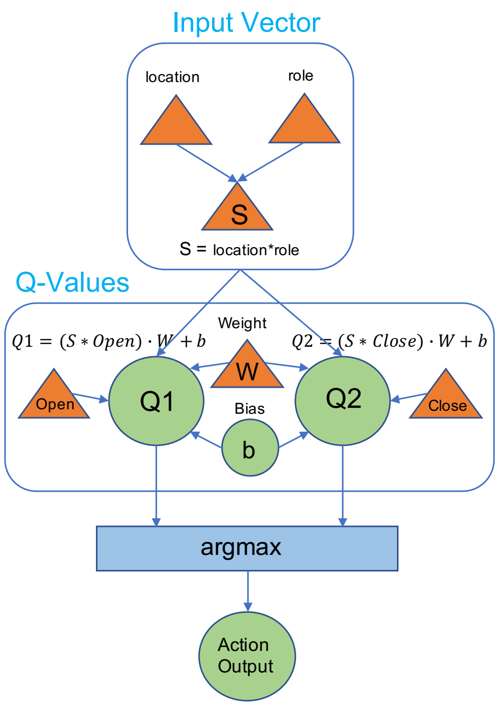


Figure 3: This diagram shows the mechanisms for the single layer neural network used to store the Q-values of the input gate, output gate, and agent network. The orange triangles represent vectors; the green circles represent scalar values; and the blue bar represents a function. The blue arrows show 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`.

this formula:

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

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 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:

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

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:

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

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:

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

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

Table 1: Parameter 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

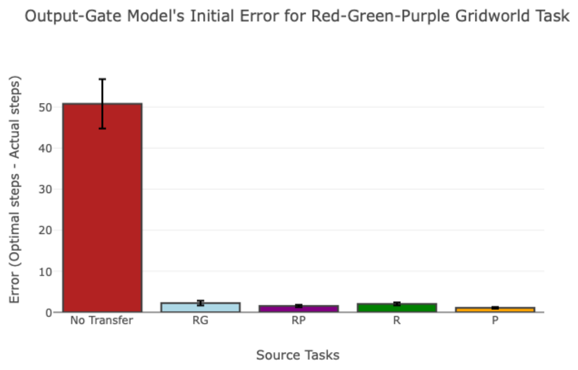
In order for the model to generate a shared representation within the task, some constraints had to be placed on the input gate and the output gate. In the earlier stages of building the model, we allowed both the input gate and output gate to have full control and autonomy over their actions. We provided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

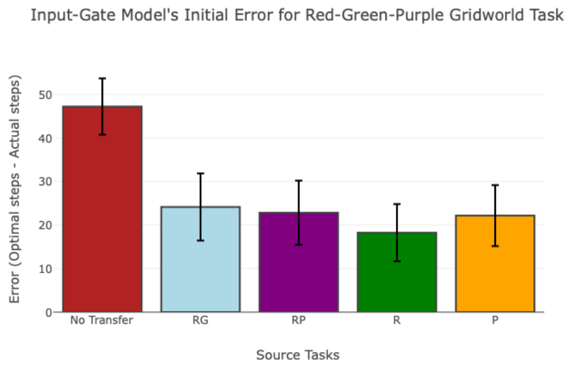
One main feature of our current output gate model is the ability to create a shared representation. The output gate model creates these shared representations by partitioning the state space into 3 parts. The first state space partitioning occurs by having the agent internally monitor whether or not 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

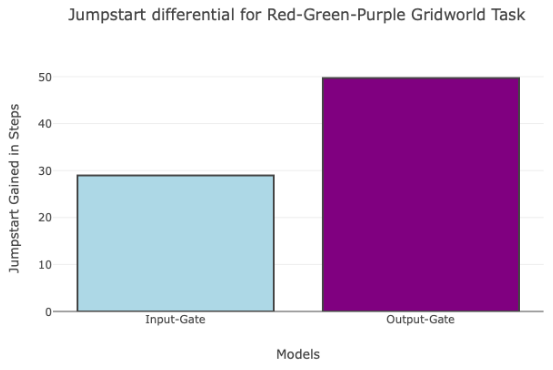
To test our models ability to transfer knowledge across tasks, we observed whether or not there was significant *jumpstart* provided to the target task. The *jumpstart* metric deals with the reduction of initial error from training on the source task relative to the target task. Also, we measured the *time to threshold* metric for the target task and the source task for



(a) Jumpstart metrics for the output gate model

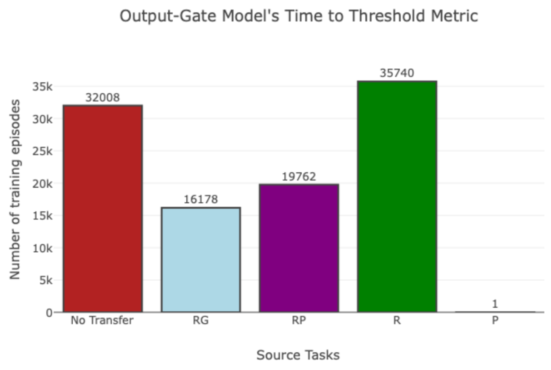


(b) Jumpstart metrics for the input gate model

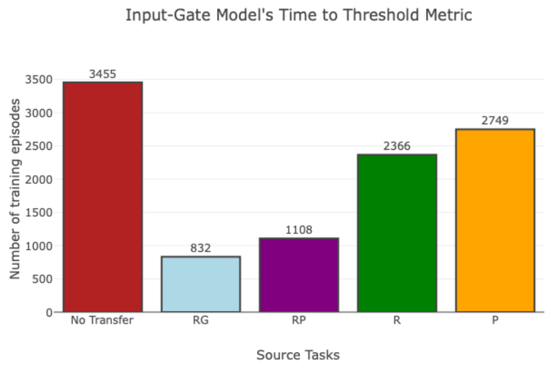


(c) Compares jumpstart between the input gate model and the output gate model

Figure 4: Jumpstart metrics for the output gate model (a) and the input gate model (b) display the initial error for the Red-Green-Purple gridworld 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 5: Time to threshold metrics for the output gate model (a) and the input gate model (b) displays 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.

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.

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 4(a) and Figure 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 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 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.

Table 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

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 5(a) and Figure 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 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 2 and Figure 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 immediatle generalize to new tasks using the Purple source task for transfer.

In the future, we intend to use our model on more tasks to test where transfer is possible. Moreover, we hope to test transfer on known cognitive science tasks such as the AX-CPT task and its variant the 1-2-AX-CPT task. By exploring these tasks, we might also be able to start comparing our transfer data with human and animal performance data.

References

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.

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*.

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.

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.

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

Phillips, J. L., and Jovanovich, M. 2018. n-task Learning : Solving Multiple or Unknown Numbers of Reinforcement Learning Problems. *Proceedings of the 40th Annual Meeting of the Cognitive Science Society, Madison, WI.*

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).

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

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

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.

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.

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