Combined Model for Sensory-Based and Feedback-Based Task Switching: Solving Hierarchical Reinforcement Learning Problems Statically and Dynamically with Transfer Learning

1st Nibraas Khan
Department of Computer Science
Middle Tennessee State University
Murfreesboro, TN
nak2z@mtmail.mtsu.edu

2nd Joshua Phillips
Department of Computer Science
Middle Tennessee State University
Murfreesboro, TN
Joshua.Phillips@mtsu.edu

***Abstract*—An integral function of fully autonomous robots and humans is the ability to focus attention on a few relevant percepts to reach a certain goal while disregarding irrelevant percepts. Humans and animals rely on the interactions between the Pre-Frontal Cortex (PFC) and the Basal Ganglia (BG) to achieve this focus called Working Memory (WM). The Working Memory Toolkit (WMtk) was developed based on a computational neuroscience model of this phenomenon with Temporal Difference (TD) Learning for autonomous systems. Recent adaptations of the toolkit either utilize Abstract Task Representations (ATRs) to solve Feedback-Based (FB) tasks or storage of past input features to solve Sensory-Based (SB) tasks, but not both. We propose a new model, SBFBWMtk, which combines both approaches, ATRs and input storage, with a static or dynamic number of ATRs. The results of our experiments show that SBFBWMtk performs effectively for tasks that exhibit SB, FB, or both properties.**

***Index Terms*—Working Memory, Cognitive Neuroscience, Temporal Difference Learning, Reinforcement Learning, Sensory-Based, Feedback**

I. Introduction

In the pursuit of autonomous systems that mimic living beings, certain fundamental abilities are required. For a system to be autonomous, sensing, perception, cognition, planning, control, and actuation are integral (Fukuda et al., 2001). As a result, there have been many attempts to address the problem of perceptual learning (the ability to form representations of sensory information based on statistical information at the perceptual level).

Many autonomous systems do not perceive the world in the same way that humans do. Thus, there is no way to guarantee that autonomous systems can work well in environments where objects are defined by human design. So the systems must have the ability to form internal representations on their own (Tugcu et al., 2007).

In a realistic environment, systems will be presented with large numbers of sensory stimuli, some irrelevant. A noted way of achieving focus is through the Working Memory Toolkit (WMtk), in which focusing on relevant percepts for the completion of a task results in a reward (Jovanovich & Phillips, 2018; Phillips & Noelle, 2005; DuBois & Phillips, 2017) - thus modeling Pre-Frontal Cortex (PFC) - Basal Ganglia (BG) interactions for Working Memory (WM) with a reward system (O'Reilly, Noelle, Braver, & Cohen, 2002; Collins & Frank, 2012).

The WMtk was created for easy integration of a neural network-based WM model within autonomous systems by mitigating complex, internal details. With the use of the toolkit, autonomous systems can exhibit several key WM functions: focus on relevant task details, limit the search space through reward-based learning, and behave robustly (Baddeley, 1992).

Reinforcement Learning (RL) algorithms, specifically Temporal Difference (TD) Learning algorithms, work well when the Markov property is met (Kunz, 2000). The original WMtk is successful when the Markov property is not met, specifically for Sensory-Based (SB) tasks (O'Reilly et al., 2002; Phillips & Noelle, 2005; DuBois & Phillips, 2017) such as a game of poker. The WMtk essentially turns a Non-Markovian (NM) task into a Markovian task by using WM. However, it struggles in situations when the environment provides no relevant information at any time (Feedback-Based) such as the Wisconsin Card Sorting Test. The toolkit is ineffective when several sequential, conflicting tasks need to be learned which are NM and Feedback-Based (FB).

The solution to this NM-FB problem is an adaptation of the WMtk known as the n-task learning algorithm (nTL), which serves as an extension to the TD Learning framework (Jovanovich & Phillips, 2018). The algorithm works by forming ATRs based on reward feedback as opposed to perceptual features. The model uses ATRs, which are analogous to lenses that can observe the environment, directing attention across different subsets of features within a common state space.

There are two distinct approaches to focusing attention on relevant percepts: WM based on gating in relevant information and ATRs based on different understandings of the same environment. The two models address distinct problems in which the Markov property is not met, SB and FB, respectively. In

2375-0197/20/$31.00 ©2020 IEEE
DOI 10.1109/ICTAI50040.2020.00055

Authorized licensed use limited to: Middle Tennessee State University. Downloaded on April 14,2021 at 20:45:06 UTC from IEEE Xplore. Restrictions apply.

this work, we propose a new model, which combines both approaches to solve complex tasks with both SB and FB components called the SBFBWMtk to solve tasks such as chess with different strategies.

II. Background

A. Working Memory and the Working Memory Toolkit

Computational neuroscience defines WM in terms of the interactions between the PFC and the BG as observed in primates. A model for the interactions is TD learning, where the learning of relevant information about stimuli or actions is based on the rewards and punishments associated with them (Sutton & Barto, 2018). In a single-layer neural network implementation, the value function (sum of discounted future rewards) is calculated using a simple dot product between a stimulus vector $\vec{u}$ and a weight vector $\vec{w}$:

$$v_t = \vec{w}^T \vec{u}_t \tag{1}$$

A learning rule is used to update the weight vector, but first, the error, δ, must be calculated:

$$\delta_t = (r_t + \gamma \cdot v_{t+1}) - v_t \tag{2}$$

where v_t is the predicted sum of future rewards at the current time step, v_{t+1} is the estimated sum of future rewards at the next time step $t+1$, and γ is the reward discount factor.

With the error, the weights can be updated using a Rescorla-Wagner-like rule defined as:

$$\vec{w}_{t+1} \leftarrow \vec{w}_t + \alpha \cdot \delta_t \cdot \vec{u}_t \tag{3}$$

where α is the learning rate. As the model learns, the value function converges to the actual sum of discounted rewards.

For WM, the RL framework is modified so that $\vec{u}$ consists of a conjunction of both the current perceptual features and potential features for storage in neural circuits analogous to the PFC. Due to the importance of WM, Noelle and Phillips created the original set of software tools for developing working memory systems that can be easily integrated into robotic control mechanisms known as the WMtk (Phillips & Noelle, 2005). The toolkit consists of a set of classes and methods that allows for the construction of a WM system that uses TD learning to choose working memory content. The original toolkit works through the aid of a neural network for decisions about memory management, configurable parameters, user-defined reward functions, and user-defined release of useless WM (Tugcu et al., 2007).

B. Holographic Reduced Representations

The original toolkit mitigates many challenges of integrating WM into a learning system but fails to provide aid to the user for the development of reasonable representations of the environment and WM concepts. The toolkit uses a neural network for learning, so these representations and concepts need to be encoded using a sparse, distributed formalism. It is difficult, even for experts, to develop and implement good representations. Even for a simple binary encoding of two WM concepts, the user must define a function that produces a two-element vector encoding (DuBois & Phillips, 2017). By forcing the user to manually create functions such as these, the toolkit is prone to errors that can be mitigated by an automatic encoding process, and the toolkit cannot adapt to varying WM demands.

To solve the problem of automatic encoding, the toolkit was integrated with a Holographic Reduced Representation Engine (HRRE). The purpose of the engine is to provide all the necessary capabilities to solve the automatic symbolic encoding (SE) to distributed encoding (DE) conversion. In the HRR formalism, independent representations are defined by a distributed vector of real numbers (Plate, 1995). The engine can generate a DE based on a SE represented by a string.

Individual representations can be combined and reduced to a single vector that represents the combined knowledge of its constituents through a well known mathematical operation in signal processing known as circular convolution. The combined representation retains the knowledge of both its constituents while the length of the combined vector and the constituents remains the same (Plate, 1995). In our model, circular convolution allows us to combine multiple complex representations of HRRs such as WM, ATRs, signal, and goal representation into one. Consider that the model receives a signal ”Red” (HRR which is a DE of ”Red”), has WM of ”RedIn” (HRR which is a DE of ”RedIn”), and ATR of 0 (HRR which is a DE of context 0), and is not at the goal (identity HRR which means not at goal). All of these HRRs are of n length and contain different but key pieces of information. However, it is difficult to use all of this information separately in our model. Instead of using the information piece by piece, we can use circular convolution to combine all of this information into a single HRR that contains all the information while the size of the combined HRR remains of size n. The implementation and design details of circular convolution are out of scope for this paper.

Additionally, HRRs form a sparse, distributed formalism for compatibility with the WMtk's underlying neural network architecture, allowing the same neural network to process increasingly complex concepts without modification to the architecture. Each representation is tied to a unique vector representation, DE, so each HRR can be tied to a complementary SE representation.

At its core, HRRs are vectors of real numbers that are typically drawn from a Normal distribution with zero mean ($\mu = 0$), and standard deviation, $\sigma = \frac{1}{\sqrt{n}}$ where n is the length of the vectors. Orthogonality, near-zero dot product, between all HRRs and all convolutions of HRRs allows for robust learning of the function, v.

C. Abstract Task Representations

The HWMtk is contingent on the presence of a reward predicting stimulus at some time during the SB task. However, Policy changes that are driven by FB information can lead to the model failing when learning several conflicting tasks sequentially. A solution to this problem is found in nTL. nTL allows for any member of the TD learning family of algorithms

Authorized licensed use limited to: Middle Tennessee State University. Downloaded on April 14,2021 at 20:45:06 UTC from IEEE Xplore. Restrictions apply.

to better handle scenarios in which the agent is required to switch between several tasks with different optimal policies. nTL uses ATRs to identify and separate tasks by using the feedback from the critic - TD error, δ, in particular.

nTL can be used as an extension to any TD learning algorithm, but we arbitrarily use SARSA below. The action selection equation with the nTL extension becomes:

$$m = \underset{c \,\in\, C}{\text{argmax}}((\vec{s} \wedge \vec{c} \wedge \vec{atr}) \cdot \vec{w}_q + b) \tag{4}$$

where $\wedge$ is circular convolution, m is the action chosen, s is the current state representation, C is the set of all candidate action choices for the current trial, atr is the current representation in memory, w_q is the weight vector for the Q function neural network, and b is the scalar bias term.

The weight update becomes:

$$\Delta w_i = \alpha_q[\text{sgn}(\delta) \cdot \log(|\delta| + 1) \cdot (\vec{s} \wedge \vec{m} \wedge \vec{atr})_i] \tag{5}$$

where w_i is the value of the weight vector at index i, α_q is the learning rate, δ is the error, and $(\vec{s} \wedge \vec{m} \wedge \vec{atr})_i$ is the HRR input vector at index i.

Each ATR is associated with an independent value function as well, which is updated with the TD error. In the equation below, A is the function determining the ATR values, α_a is the learning rate for the ATRs, and δ is $r - A(\vec{atr})$:

$$A(atr) \leftarrow A(\vec{atr}) + \alpha_a[\text{sgn}(\delta) \cdot \log(|\delta| + 1)] \tag{6}$$

When the TD error crosses a threshold, t, the model substitutes the next ATR in sequential order. The t value is first set to negative one times the reward for the goal state, and is then updated at each time step using the TD error from the Q function, where α_{at} is the learning rate:

$$t \leftarrow t - \alpha_{at}[\text{sgn}(\delta) \cdot \log(|\delta| + 1)] \tag{7}$$

The t value is not updated for a task change external to the agent. nTL is made to work for both a preset static number of tasks and a dynamic number of tasks (Jovanovich & Phillips, 2018).

We hypothesize that a synthesis of the HWMtk and nTL would provide a theoretical framework that is capable of solving real world problems with SB and FB features. However, it is not clear whether additional mechanisms must be developed to form this synthesis. In particular, nTL only triggers ATR swaps for large, unanticipated negative values of δ, but SB problems sometimes exhibit large, unanticipated positive δ values as well. If not carefully considered, the wrong ATR may be rewarded with a positive δ and result in learning instabilities. Therefore, we also anticipate that a mechanism for handling large, unanticipated positive δs will be needed.

III. Methods

A. Model Description

At every time step, the agent needs to take into account the state, signal, WM, ATR, and reward with:

$$\vec{u} = (\vec{s} \wedge \vec{p} \wedge \vec{wm} \wedge \vec{atr} \wedge \vec{r}) \tag{8}$$

where $\vec{u}$ is the HRR representation of all the relevant information (all vectors are HRRs): $\vec{s}$ is the state, $\vec{p}$ is the signal, $\vec{wm}$ is the WM, $\vec{atr}$ is the ATR, and $\vec{r}$ is the reward.

The possible values for WM are the internal representation of the signal, previous WM, or the identity HRR. There will always be a vector present for the ATR so that the agent is never left without context. The signal is a SB feature that is only available for the first time step and is just an identity HRR for the rest of the time steps. The reward is only present when there is a goal at the state the agent is in, otherwise, it is just an identity HRR.

With the representation, $\vec{u}$, its value needs to be calculated for the agent to make appropriate decisions with a simple one-layer neural network where the weights are initialized as a HRR vector and the bias (b) is set to one (optimistic critic). The value, v, is defined as:

$$v(\vec{u}) = (\vec{u} \cdot \vec{w}) + b \tag{9}$$

where $\cdot$ is the dot product and $\vec{w}$ is the weights of the network.

To update the weights, TD error, δ, needs to be calculated using:

$$\delta_t = (r_t - \gamma \cdot \text{v}(\vec{u}_{t+1})) - \text{v}(\vec{u}_t) \tag{10}$$

where r_t is the scalar reward value.

An eligibility trace allows for a backward view of the steps as opposed to the usual forward view for more stable learning. On each time step, the trace is scaled using λ for all previous states. The eligibility trace in terms of time, t, is defined as:

$$\vec{e}_t = \lambda \cdot \vec{e}_{t-1} + \vec{u}_t \tag{11}$$

where $\vec{e}_{t-1}$ is the discounted accumulation of all other previous states, and $\vec{u}_t$ is the current state.

The weight update at time t for the neural network can now be defined as:

$$\vec{w}_t = \vec{w}_{t-1} + \alpha \cdot \text{logmod}(\delta_t) \cdot \vec{e}_t \tag{12}$$

In the above equation, $\vec{w}_{t-1}$, is the weight vector at the previous time step, α is the learning rate, *logmod* is a log-modulus transform (to stabilize learning by scaling error), and $\vec{e}_t$ is the eligibility trace at the current time step.

At every time step, the agent must decide what move to make. The move is based on the value of the potential states the agent can step into. The maximum value of the next state and WM can be calculated using:

$$m, c = \underset{\vec{s} \,\in\, S,\ \vec{wm} \,\in\, WM}{argmax} (\text{v}(\vec{s} \wedge \vec{p} \wedge \vec{wm} \wedge \vec{atr} \wedge \vec{r})) \tag{13}$$

The above equation uses the *argmax* function where the agent enumerates through states, $\vec{s}$, of all possible states S and all working memory content, $\vec{wm}$, of all possible working memory contents WM. At any time step t, the agent can use the above equation to decide the move, m (external decision) and WM, c (internal decision).

In case the agent is stuck in a local minimum, we implement an epsilon soft policy, ε, which allows for the agent to make

Authorized licensed use limited to: Middle Tennessee State University. Downloaded on April 14,2021 at 20:45:06 UTC from IEEE Xplore. Restrictions apply.

Parameters	Static SB	Dynamic SB	Static FB	Dynamic FB	Static SBFB	Dynamic SBFB
Hrr Length (n)	10240	15360	6144	7168	25600	25600
Discount (γ)	0.9	0.9	0.7	0.7	0.7	0.7
Alpha for Training (α_{train})	0.3	0.3	0.1	0.1	0.1	0.1
Alpha for Testing (α_{test})	0.01	0.01	0.01	0.01	0.01	0.01
Epsilon Soft (ε_{test})	0.00001	0.00001	0.00001	0.00001	0.00001	0.00001
Threshold (t)	0.3	1	0.3	1	0.3	1
Discount for the trace (λ)	0.0	0.0	0.0	0.0	0.01	0.01
ATR Alpha (α_a)	-	-	0.00065	0.00063	0.00011	0.00011
ATR Threshold (α_t)	-	-	-0.5	-0.5	-0.5	-0.5
Signals ($\vec{p}$)	R, G, B	R, G, B	-	-	R, G	R, G
Goals	3, 10, 14	3, 10, 14	0, 4, 7, 10, 13	0, 4, 7, 10, 13	2, 5; 8, 13	2, 5; 8, 13
Episodes	100000	100000	100000	100000	100000	100000
Max Steps Per Episode	100	100	100	100	100	100
FB Task Switch Rate	500	500	500	500	1000	1000

TABLE I: Major parameters for all the models are shown in this table, but all parameters can be found on the GitHub page. These parameters are the same for both the reset and transfer method.

Parameters	Static SBFB	Dynamic SBFB
FB Task Switch Rate	2000	2000
ATR Alpha (α_a)	-	0.000105
Discount for the trace (λ)	-	0.05
Accuracy Without Tuning	38.33%	33.57%
Accuracy With Tuning	96.88%	96.81%

TABLE II: These are the tuned parameters which lead to drastically better performance.

ATR Switching Mechanism	Maximum Accuracy
Only Positive Error Switch	62.56%
Only Negative Error Switch	48.76%
Both Positive and Negative Error Switch	96.88%
No Error Switching	69.45%

TABLE III: This table shows the accuracy of tuned Static SBFB from Table II with various task switching mechanisms.

exploratory decisions some small fraction of time steps, ε. During the random move, c and the $\vec{atr}$ are not affected, only m is.

The agent must also decide which context to use independent of the decision above. The context switch is triggered by the error rather than maximum value estimates. When δ_t crosses a certain threshold, the context switch is triggered. The threshold, t, in the model can be either a static or dynamic hyper-parameter. With the dynamic threshold, a threshold alpha, α_t, is used to update t at every time step (where t initially set to one):

$$t \mathrel{+}= \alpha_t \cdot \text{logmod}(\delta) \tag{14}$$

When a large negative error occurs that crosses $-t$, the agent interprets this error to mean that the wrong atr was used (i.e. the task demands must have changed and requires a different ATR). For this case, the agent chooses the next atr sequentially rather than choosing the highest value because the correct atr cannot be determined directly for the case of large negative errors.

When the positive, t, is crossed with a large positive TD error, a $argmax$ function is used to determine the next atr:

$$atr \leftarrow \underset{\vec{atr}\,\in\,ATR}{argmax}\,(\vec{s} \wedge \vec{p} \wedge \vec{wm} \wedge \vec{atr} \wedge \vec{r}) \tag{15}$$

When the agent switches $atrs$ (for either large positive or large negative errors), e_t is cleared out so that the agent does not learn under the wrong context.

SBFBWMtk can be used for an arbitrary number of tasks, n, so the model needs to have the ability to grow along with the task. Without growth, the HRRs and the neural network might not be able keep up with the required orthogonality. To facilitate growth, at every time step where the threshold is crossed, both positive and negative, and the mean of the ATR values crosses α_t, the ATR values are reset, the threshold for task switching is reset (for dynamic threshold to one), the HRR size is increased linearly, the weights are reset, and the eligibility trace is reset.

With this method, which we call the Reset Method (RM), all of the HRRs including the WMs, ATRs, and all internal representations will be lost, and the learned value function is also lost. However, and alternate method, called Transfer Method (TM), allows the model to transfer the values from the old, smaller neural network to the larger version. When the model grows, new HRRs are created for the internal representations, the pseudo-inverse of the new HRRs is calculated, and the new weights are calculated using matrix multiplication between the inverses and the values of the old HRRs.

The HRR length, n, is updated using:

$$n \leftarrow \frac{atr_c \cdot n}{atr_c - 1} \tag{16}$$

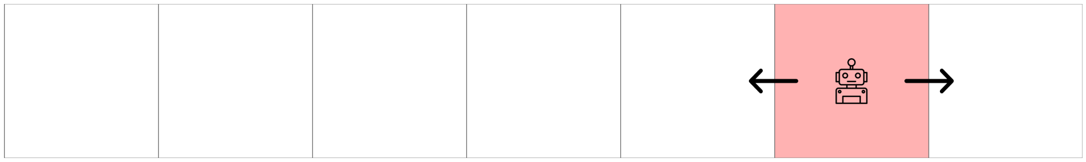
where atr_c is the current number of $atrs$.

The new weights, $\vec{w}$, are set with:

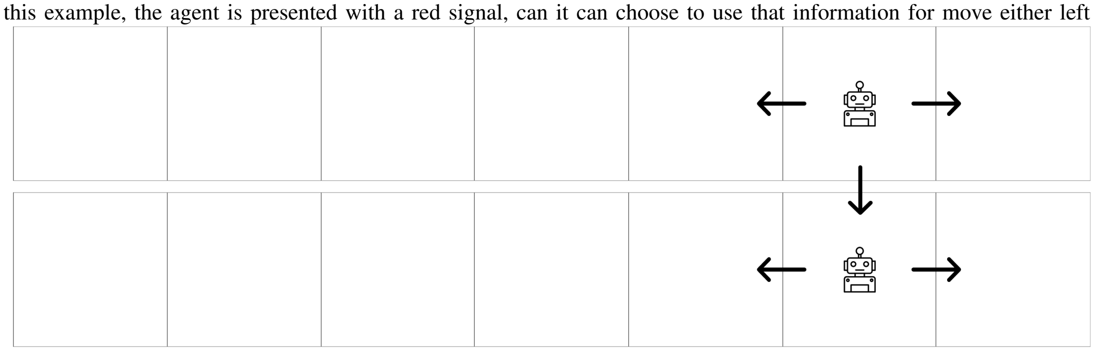
$$\vec{w} \leftarrow [\underset{\vec{u}_{new}\,\in\,U_{new}}{\vec{u}_{new}}]^{-1} * [\underset{\vec{u}_{old}\,\in\,U_{old}}{\mathrm{v}}(\vec{u}_{old})] \tag{17}$$

where $[\underset{\vec{u}_{new}\,\in\,U_{new}}{\vec{u}_{new}}]^{-1}$ is the inverse of the new, empty stacked HRRs, $[\underset{\vec{u}_{old}\,\in\,U_{old}}{\mathrm{v}}(\vec{u}_{old})]$ is the vector of the previous HRR values, and $*$ is the matrix multiplication.

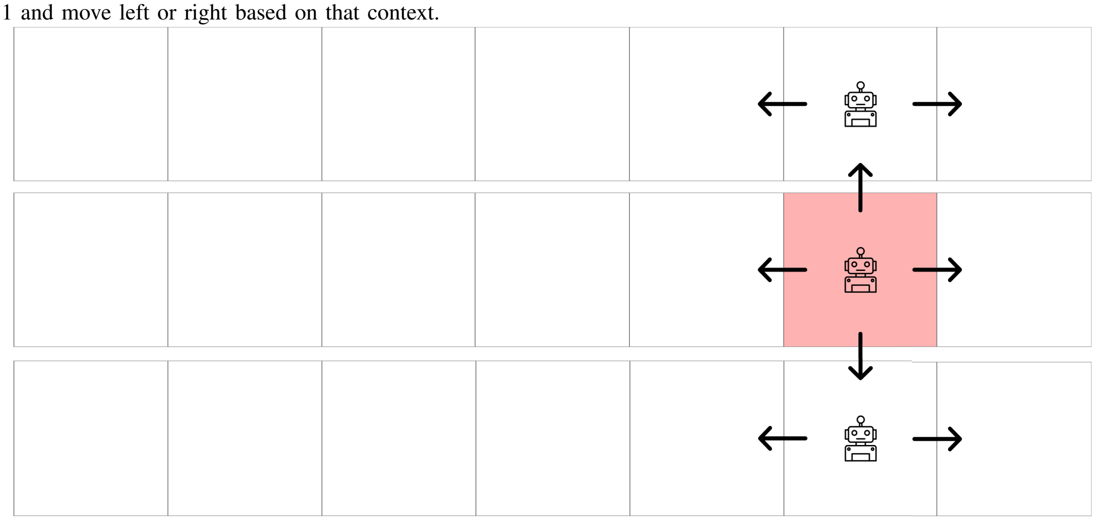
Authorized licensed use limited to: Middle Tennessee State University. Downloaded on April 14,2021 at 20:45:06 UTC from IEEE Xplore. Restrictions apply.



(a) Example of a One-Dimensional maze task with SB feature isolated. The agent is presented with a signal and based on that information, it forms an internal representation of its current state. The agent can make the decision to move left of right based on where it thinks the goal is. In this example, the agent is presented with a red signal, can it can choose to use that information for move either left or right.



(b) Example of a One-Dimensional maze task with FB feature isolated. The agent has no information from the environment, but it forms an internal representation of its state. Then the agent can choose to move left or right. It can also choose to switch context using ATRs based on feedback it gets from the environment. In this example, the agent con choose to move left or right in context 0 or it can choose to switch to context 1 and move left or right based on that context.



(c) Example of a One-Dimensional maze task with combined SBFB. The agent is presented with a signal, and based on that signal the agent forms an internal representation of its current state. Based on this information, the agent can choose to move left or right in the current context, or it can switch contexts with the use of ATRs and then move left or right. In this example, the agent is in context 1 and is presented with a red signal. Based on this information, the agent can choose to stay in its current context and move left or right. It can also choose to switch to context 0 or 2 and then move left or right with the use of ATRs.

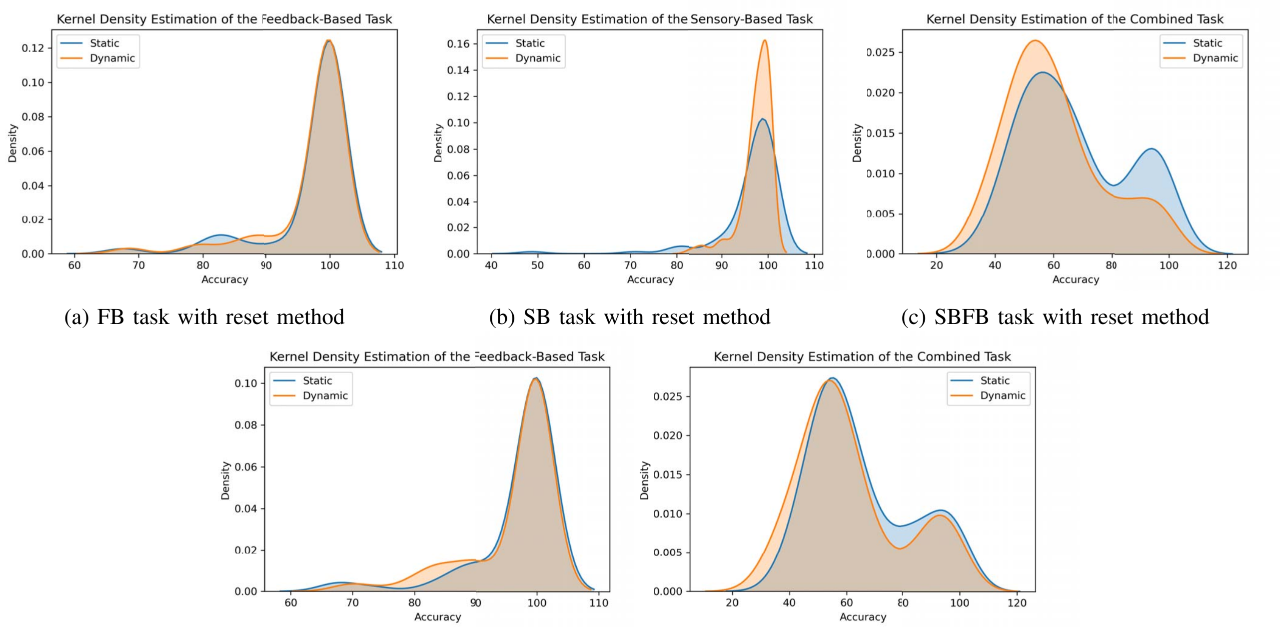
Fig. 1: These figures highlight the three kinds of tasks that the agent can solve: SB, FB, and SBFB. It specially shows how to agent can react on time step one based on the kind of task it is and the information presented to it.

The TM is computationally efficient but becomes increasingly expensive with more tasks. At some point, using the RM might become more computationally tractable but this is beyond the scope of our current work.

The model can keep old knowledge (the partially learned value function), but also have the ability to learn new information without losing orthogonality. The new HRRs have the same values as the old HRRs and all the mathematical and constituent properties are preserved.

The implementation of SBFBWMtk can be found on GitHub (`https://github.com/nibraaska/Working-Memory-Temporal-Difference`). Furthermore, a visual implementation of the model can be found on GitHub (`https://github.com/nibraaska/Working-Memory-Tasks`). The vi-

Authorized licensed use limited to: Middle Tennessee State University. Downloaded on April 14,2021 at 20:45:06 UTC from IEEE Xplore. Restrictions apply.



(a) FB task with reset method (b) SB task with reset method (c) SBFB task with reset method

(d) FB task with transfer method (e) SBFB task with transfer method

Fig. 2: The three plots on the first row show the density of the accuracy of the three maze tasks with the reset method, and the bottom show the two relevant tasks for the transfer method.

sual implementation contains SB, FB, and SBFB tasks that can be seen in real time. Each game in the visual implementation shows off the features of the model and the exact Working Memory and Abstract Task Representation in mind.

B. Test Protocols

We provide three tasks that test the effectiveness of our model. The three tasks test the SB, FB, and SBFB features of the SBFBWMtk. Within these tasks, we compare the effectiveness of the RM and TM.

For all three tasks, the accuracy (whether optimal number of steps were taken including ATR switching) was calculated for the last 10% of the episodes. To make sure that the testing was stable, α_{test} was set to 0.01 and ε_{text} was set to 0. Additional experimental parameters are shown in Table I.

Each of the three tasks, SB, FB, and SBFB, were tested using both static and dynamic thresholds with a hundred seeds for both methods using GNU Parallel (Tange, 2011).

1) Sensory-Based: To isolate the SB constituent of our model, a maze task using a one-dimensional array with three signals corresponding to three goals was constructed. Since there is only one context, there is no growth in the model, and neither the RM or TM will be utilized. An example of an SB task can be seen in 1.

At the first time step, the agent is randomly dropped into the maze with the SB feature (signal vector) present in the environment. The agent must learn to use its WM feature to essentially convert this NM task into a M task. As the agent progresses through the task, it is presented with scalar rewards. At states that are not the goal, the agent receives -1, and at the goal state, the agent receives 0. Along with the 0 reward at the goal state, the agent also receives a goal token, which is just an HRR to identify that a state is the goal. During the training phase, the agent can also use ε-soft to explore and escape local minimum.

2) Feedback-Based: To isolate the FB constituent of our model, a maze task with three goals and no signals was constructed. Each of the goals is distributed to one of the three task contexts. An example of an FB task can be seen in 1.

The agent is randomly dropped into the maze, but there is no signal present. The agent must learn to map the three *atrs* to the goals. The decision making is similar to the task above, but there are no *wm* decisions. During the learning and testing phase of the FB task, the context switches after a set number of episodes have passed. The reward system is the same as described above in the SB section.

3) Sensory-Based and Feedback-Based: The maze task created for the combined model is more complex than either of the tasks listed above. In this task, there are two abstract tasks with two signals which have different meanings under different contexts. An example of an SBFB task can be seen in 1.

At time step one, the agent is dropped into the array at a random spot with a SB signal vector present in the environment. Unlike the SB task, the signal might have different meanings depending on the context. The reward system is the same as described above in the SB section.

Authorized licensed use limited to: Middle Tennessee State University. Downloaded on April 14,2021 at 20:45:06 UTC from IEEE Xplore. Restrictions apply.

IV. Results

The graphs in Figure 2 show the results of the model of all relevant tasks in the form of Kernel Density Estimation. A kernel Density Estimation is a non-parametric way of estimating the density of a random variable. In this case, the variable is the accuracy of the model. Each graph contains the accuracy information for 100 seeds (random initialization such as random goals, start states, etc) for static and dynamic threshold.

The effectiveness of the model is clear for the SB and FB tasks, but the SBFB task shows somewhat lower performance due to the lack of tuning the hyper-parameters.

From Figure 2a, it is clear the model is able to solve the FB task using the RM with a high rate of success for both static and dynamic thresholds. The dynamic threshold method allows the model to find the optimal threshold, and it is able to perform just as well in the FB task as the static Threshold Method. However, there is still a tail with $60\% - 90\%$ accuracy, and this is due to the length of the HRRs and the noise in the hyper-parameters.

The model performs extremely well on SB tasks especially with the dynamic threshold method, as shown in Figure 2b. As with the FB task, there is still a tail, and the way to improve the performance is the same as with the FB task.

With the SBFB task, the model struggles when the testing method does not allow for the tuning of hyper-parameters. Without tuning, Figure 2c shows the model to be ineffective with performance as low as 33%. After tuning the model, it performs significantly better. Table II shows that the static SBFB model is able to improve its performance dramatically by changing the FB task switch rate from 1000 to 2000, and the dynamic SBFB model is also able to perform significantly better after tuning.

There are several ways to tune the hyper-parameters of a model, but we found Bayesian-Optimization (assumes noise is in the hyper-parameters) to be successful. Bayesian-Optimization works by forming a posterior distribution of functions and improving it as observations grow (Brochu, Cora, & De Freitas, 2010).

With the TM, as shown in Figure 2d and Figure 2e, the model performs well with the retention of information. This means that the model is able to transfer learning from the old values, and continue to take advantage of its previous experiences. This method was not available in the nTL (Jovanovich & Phillips, 2018), but our model can utilize it with success dramatically reducing overall training time.

It appears in some cases that the static threshold performs better than the dynamic threshold, but the static threshold cannot learn an arbitrary number of tasks, n. Therefore, the dynamic threshold can be seen as more general.

For near perfect accuracy, the agent needs to use both positive and negative error switching. Table III shows the accuracy of the combined model run on the same parameters as tested above but with certain task switching mechanisms removed. As the table shows, it is important for the agent to use both kinds of task switching to achieve near perfect accuracy.

V. Discussion

Since autonomous systems do not perceive the world in the same way as humans, they need to form their own internal representations. The agent must be able to disregard irrelevant information from the environment and use relevant information to solve the current task. To give autonomous systems the ability to do this, the HWMtk was created (DuBois & Phillips, 2017) inspired by the putative interactions of the PFC and BG for WM and higher-order cognition. For tasks where the environment does not have all relevant information, nTL was created (Jovanovich & Phillips, 2018).

These two distinct models described can either utilize ATRs or storage of past input features but not both. However, in the real world, tasks are not always so simple: they may contain both SB and FB features. In this paper, we presented a model, SBFBWMtk, that can utilize both methods to solve tasks with both SB and FB features. Furthermore, the model can solve both a static and a dynamic number of tasks with the use of a dynamic threshold, and the model can use the TM to retain the value function while also growing.

In the future, we would like to continue to explore TM and more complex neural networks. The model also implicates the PFC and BG as key constitutes to higher-order cognition involved in the solution of SB, FB, and SBFB tasks.

VI. Acknowledgements

This work was supported by an Middle Tennessee State University (MTSU) Undergraduate Research and Creative Activity (URECA) grant sponsored by the MTSU Office of Research.

References

Baddeley, A. (1992). Working memory. *Science*, *255*(5044), 556–559.

Brochu, E., Cora, V. M., & De Freitas, N. (2010). A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599*.

Collins, A. G., & Frank, M. J. (2012). How much of reinforcement learning is working memory, not reinforcement learning? a behavioral, computational, and neurogenetic analysis. *European Journal of Neuroscience*, *35*(7), 1024–1035.

DuBois, G. M., & Phillips, J. L. (2017). Working memory concept encoding using holographic reduced representations. In *Maics* (pp. 137–144).

Fukuda, T., Michelini, R., Potkonjak, V., Tzafestas, S., Valavanis, K., & Vukobratovic, M. (2001, March). How far away is "artificial man". *IEEE Robotics Automation Magazine*, *8*(1), 66-73. doi: 10.1109/100.924367

Jovanovich, M., & Phillips, J. (2018). n-task learning: Solving multiple or unknown numbers of reinforcement learning problems. In *Cogsci* (pp. 584–589).

Authorized licensed use limited to: Middle Tennessee State University. Downloaded on April 14,2021 at 20:45:06 UTC from IEEE Xplore. Restrictions apply.

Kunz, F. (2000). An introduction to temporal difference learning. In *Seminar on autonomous learning systems.*

O'Reilly, R. C., Noelle, D. C., Braver, T. S., & Cohen, J. D. (2002). Prefrontal cortex and dynamic categorization tasks: representational organization and neuromodulatory control. *Cerebral cortex*, *12*(3), 246–257.

Phillips, J. L., & Noelle, D. C. (2005). A biologically inspired working memory framework for robots. In *Roman 2005. ieee international workshop on robot and human interactive communication, 2005.* (pp. 599–604).

Plate, T. A. (1995). Holographic reduced representations. *IEEE Transactions on Neural networks*, *6*(3), 623–641.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Tange, O. (2011, Feb). Gnu parallel - the command-line power tool. *;login: The USENIX Magazine*, *36*(1), 42-47. Retrieved from `http://www.gnu.org/s/parallel` doi: http://dx.doi.org/10.5281/zenodo.16303

Tugcu, M., Wang, X., Hunter, J. E., Phillips, J., Noelle, D., & Wilkes, D. M. (2007). A computational neuroscience model of working memory with application to robot perceptual learning. In *Third iasted international conference on computational intelligence (ci).*

Authorized licensed use limited to: Middle Tennessee State University. Downloaded on April 14,2021 at 20:45:06 UTC from IEEE Xplore. Restrictions apply.