Combined Model for Partially-Observable and Non-Observable Task Switching: Solving Hierarchical Reinforcement Learning Problems

Nibraas A. Khan and Joshua L. Phillips
Department of Computer Science
Middle Tennessee State University
Murfreesboro, TN 37132, USA
(nak2z@mtmail.mtsu.edu), (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 and the Basal Ganglia to achieve this focus, which is known as working memory. The working memory toolkit (WMtk) was developed based on a computational neuroscience model of this phenomenon with the use of temporal difference learning for autonomous systems. Recent adaptations of the toolkit either utilize abstract task representations to solve non-observable tasks or storage of past input features to solve partially-observable tasks, but not both. We propose a new model, which combines both approaches to solve complex tasks with both Partially-Observable (PO) and Non-Observable (NO) components called PONOWMtk. The model learns when to store relevant cues in working memory as well as when to switch from one task representation to another based on external feedback. The results of our experiments show that PONOWMtk performs effectively for tasks that exhibit PO properties or NO properties or both.

In the pursuit of truly autonomous systems that mimic living beings, certain fundamental abilities are required. For a system to be truly 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. Perceptual learning is the ability to form representations of the sensory information based on their statistical information at the perceptual level (Olshausen and Field 1996) Goldstone 1998).

Many autonomous systems don’t perceive the world in the same way that humans do. Therefore, there is no way to guarantee that autonomous systems can work well in environments where objects are defined by human design. Hence, for an autonomous system to work well in realistic environments, the systems must have the ability to form internal representations of their own (Tugcu et al. 2007).

In a realistic environment, autonomous systems will be presented with a large number of sensory stimuli and many of these might be irrelevant for the completion of a particular task. A prominent way of achieving this focus is through the WMtk, in which focusing on relevant percepts for the completion of a task results in a reward (Jovanovich and Phillips 2018) [Phillips and Noelle 2005] [DuBois and Phillips 2017]. The WMtk is an adaptation of Reinforcement Learning (RL) where working memory is used to solve tasks based on a reward system. This is a notable use of modified versions of RL that have been used to focus attention on relevant percepts, thus modeling pre-frontal cortex working memory (OReilly et al. 2002) Collins and Frank 2012).

The open-source WMtk was developed with the use of RL techniques to achieve the ability to focus attention on relevant percepts. The biologically inspired toolkit was created for easy integration of an artificial neural network-based working memory model within autonomous systems by mitigating complex, internal details. With the use of working memory which is implemented within the toolkit, the autonomous systems are able to focus on details relevant to the current task, limit the search space through reward-based learning and behave robustly when faced with large amounts of stimuli (Baddeley 1992).

RL algorithms, specifically temporal difference learning algorithms, work well when the Markov property is met (Kunz 2000). The WMtk is successful when the Markov property is not met, specifically for partially-observable (PO) tasks (OReilly et al. 2002) Phillips and Noelle 2005 DuBois and Phillips 2017). The WMtk essentially turns a Non-Markovian (NM) task into a Markovian (M) task by using working memory. However, it struggles in situations when the environment provides no relevant information at any time such as when several sequential, conflicting tasks need to be learned. Such tasks are both NM and non-observable (NO) in nature, and the WMtk is ineffective and hinders the performance on such tasks.

The solution to this NM-NO problem is the n-task learning algorithm (nTL), which serves as an extension to temporal difference learning algorithms (Jovanovich and Phillips 2018). The algorithm works by forming abstract task representations (ATRs) based on reward feedback as opposed to perceptual features. The model uses ATRs analogously to lenses with which to look at the environment, directing attention across different subsets of features within a common state space.

Thus, there are two distinct approaches to focusing attention on relevant percepts: working memory based on gat-
ing in relevant information and abstract task representations based on different understandings of the same environment. The two models address the distinct problems in which the Markov property is not met, PO and NO, respectively. In this work, we propose a new model, which combines both approaches to solve complex tasks with both PO and NO components called the PONOWMtk.

Background

Working Memory and the Working Memory Toolkit

The term working memory (WM) is used by cognitive psychologists to refer to a type of memory that is active and relevant for a short period of time. The purpose of WM is to aid in the completion of tasks by holding relevant information for use while ignoring irrelevant information.

Computational neuroscience defines working memory in terms of the interactions between the Pre-Frontal Cortex (PFC) and the Mesolimbic Dopamine System (MDS) as observed in primates. The model for the MDS is Temporal Difference (TD) learning, where the learning of relevant information about stimuli or actions is based on the rewards and punishments associated with them (Sutton and Barto 2018). In a single-layer neural network implementation, the value function (sum of discounted future rewards) is approximately a simple dot product between a stimulus vector \( \vec{u} \) and a weight vector \( \vec{w} \):

\[
\nu_t = \vec{w}^T \vec{u}_t
\]

A learning rule is used to update the weight vector, but first, the error, \( \delta \), must be calculated:

\[
\delta_t = (r_t + \gamma \nu_{t+1} - \nu_t)
\]

where \( \nu_t \) is the predicted sum of future rewards, \( \nu_{t+1} \) is the observed sum of future rewards, and \( \gamma \) is the reward discount factor.

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

\[
\vec{w}_{t+1} \leftarrow \vec{w}_t + \alpha \delta_t \vec{u}_t
\]

Through experience, the value function converges to the actual sum of discounted rewards.

For working memory, 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 working memory, 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 and Noelle 2005). The original toolkit consists of a set of classes and methods that allow for the construction of a working memory 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).

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 the WM concepts. Since the toolkit uses a neural network for learning, these representations and concepts need to be encoded using a sparse, distributed formalism. It is difficult to develop and implement good representations, even for experts. For a simple binary encoding of two WM concepts, the user must define a function which utilizes a two-element vector (DuBois and Phillips 2017). By forcing the user to manually create functions such as these, the WM model is prone to errors that can be mitigated by an automatic encoding process and cannot adapt to varying working memory 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 HRRE formalism, independent representations are defined by a distributed vector of real numbers. The engine is able to 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 mathematical operation 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. Additionally, HRRs form a sparse, distributed formalism so they are compatible with the WMtk’s underlying neural network architecture allowing the same neural network to process increasingly complex concepts without modification to the architecture. Since each representation is tied to a unique vector representation, DE, 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/Gaussian distribution with zero mean (\( \mu = 0 \)) and standard deviation, \( \sigma = \frac{1}{\sqrt{n}} \), where \( n \) is the length of the vectors (Plate 1995). Orthogonality, near-zero dot product, between all HRRs and all convolutions of HRRs allows for robust learning of the function, \( v \).

The main functionality of the HRRE is to encode, store, and manipulate the representations used in the WMtk. This was done by creating a conjunctive encoding engine and a conjunctive decoding engine. Additional details about the holographic working memory toolkit (HWMtk) is outside the scope for this paper and can be found in (DuBois and Phillips 2017).

Abstract Task Representations

The HWMtk is contingent on the presence of a reward predicting stimulus at some time during the PO task but often at the beginning without loss of generality. Policy changes that are driven by NO hidden information can lead to the model failing when learning several conflicting tasks sequentially. A solution to this problem is found in the n-task learning algorithm (nTL). nTL allows for any member of the TD
learning family of algorithms to better handle scenarios in which the agent is required to switch between several tasks with different optimal policies. nTL uses abstract task representations to identify and separate tasks by only using the feedback from the critic, in particular, the TD error, $\delta$. ARTs are essentially a filter through which the agent perceives its environment where each filter is mapped to a unique task.

nTL shares aspects with other models in machine learning as well as the HWMtk, but it has the unique ability to self-monitor and react only with reward and feedback. nTL addresses the problem in which the contextual cues offer no information that can be used to determine an appropriate selection policy (Jovanovich and Phillips 2018).

The nTL can be used as an extension to any TD learning algorithm, but to show the effectiveness of the extension, SARSA will be used. The action selection equation with the nTL extension becomes:

$$m = \arg\max_c ((\vec{s} \land \vec{c} \land a\vec{r}) * w_q + b)$$

(4)

where $\land$ is circular convolution, $m$ is the move 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:

$$\Delta w_i = \alpha_q [\text{sgn}(\delta) * \log(|\delta| + 1) * (s \land m \land atr)_i]$$

(5)

where $w_i$ is the value of the weight vector at index $i$, $\alpha_q$ is the learning rate, $\delta$ is the error, and $(s \land m \land atr)_i$ 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 for the ATR value function. In the equation below, $A$ is the function determining the ATR values, $\alpha_a$ is the learning rate for the ARTs, and $\delta$ is $r - A(atr)$:

$$A(atr) \leftarrow A(atr) + \alpha_a [\text{sgn}(\delta) * \log(|\delta| + 1)]$$

(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 $\alpha_t$ is the learning rate:

$$t \leftarrow t - \alpha_t [\text{sgn}(\delta) * \log(|\delta| + 1)]$$

(7)

The $t$ value is not penalized for a task change external to the agent. The nTL is made to work for both preset static number of tasks and dynamic number of tasks. See (Jovanovich and Phillips 2018) for details which are out of scope for this paper.

HWMtk solves the problem of PO tasks by utilizing the storage of past input representations and nTL solves the problem of NO tasks with the use of ATRs. These models can solve problems effectively within their respective task domains, but they are not able to solve tasks that contain both PO and NO features. That means that these models will not be able to solve real-life problems, due to the fact that these problems will most likely will contain both PO and NO features. We hypothesize that a synthesis of these two models would provide a framework which is capable of solving such tasks. 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 $\delta$, but PO problems sometimes exhibit large, unanticipated positive $\delta$ values as well. If not carefully considered, the wrong ATR may be rewarded with a positive $\delta$ and result in learning instabilities. Therefore, we also anticipate that a mechanism for handling large, unanticipated positive $\delta$s will be needed.

Methods

Model Description

Here we describe a new model, PONOWMtk for solving both PO and NO tasks. At every time step, the agent needs to take into account the state, signal, working memory, abstract task representation, and reward. To accommodate all the details our agent needs to know, the following representation is used:

$$\vec{u} = (\vec{s} \land \vec{p} \land \vec{wm} \land a\vec{r} \land \vec{r})$$

(8)

where $\vec{u}$ is the HRR representation of all the relevant information: $\vec{s}$ is the representation of the state, $\vec{p}$ is the relevant signal vector, $\vec{wm}$ is the internal memory vector, $a\vec{r}$ is the of the abstract task vector, and $\vec{r}$ is the reward. The signal is a PO feature that is only available for the first time step and is just an identity HRR for the rest of the time steps. The identity HRR represents the absence of some information and using it in place of an HRR is formally equivalent to not performing the convolution operation (for example, $\vec{a} \land \vec{I} = \vec{a}$, where $\vec{a}$ is an HRR and $\vec{I}$ is the identity HRR). The possible values for the working memory are the internal representation of the signal, previous working memory, or the identity HRR. There will always be a vector present for the abstract task representation so that the agent is never left without context. 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 the appropriate decisions. The value is maintained and calculated by 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:

$$v(\vec{u}) = (\vec{u} \ast \vec{w}) + b$$

(9)

where $\vec{u}$ is the input, $\ast$ is the dot product, and $\vec{w}$ is the weights of the network.

To update the weights, temporal difference error needs to be calculated. The error, $\delta$ at time $t$ can be calculated using:

$$\delta_t = (r_t - \gamma \ast v(\vec{u}_{t+1})) - v(\vec{u}_t)$$

(10)

where $r_t$ is the scalar reward value.
The final formula needed to update the weights is the eligibility trace. The eligibility trace allows for a backward view of the steps as opposed to the usual forward view. On each time step, the trace is scaled using \( \lambda \) for all previous states. The accumulation of states using the eligibility trace allows for more effective, stable learning. The eligibility trace in terms of time, \( t \), is defined as:

\[
e_t = \lambda \cdot e_{t-1} + \bar{u}_t
\]

(11)

where \( e_{t-1} \) is the accumulation of all other previous states with a discount factor, \( \lambda \), and \( \bar{u}_t \) is the current state.

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

\[
\bar{w}_t = \bar{w}_{t-1} + \alpha \cdot \logmod(\delta_t) \cdot e_t
\]

(12)

In the above equation, \( \bar{w}_{t-1} \) is the weight vector at the previous time step, \( \alpha \) is the learning rate, \( \logmod \) is a logarithmic modulus transform (to stabilize learning by scaling error), and \( e_t \) is the eligibility trace at the current time step.

As the weights change over time, the agent is able to maneuver through the environment with more and more confidence. 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 working memory can be calculated using:

\[
m, c = \arg\max_{\bar{s} \in S, \bar{w}m \in WM} (v(\bar{s} \land \bar{p} \land \bar{w}m \land a\bar{r}t \land \bar{r})
\]

(13)

The above equation defines a simple \( \arg\max \) function where the agent enumerates through all possible states in set \( S \), which are all possible states the agent can enter at the next time step using available actions and enumerates through all possible working memory in the set \( WM \), which could be the internal representation of the signal (if present), the current \( \bar{w}m \), or the identity HRR. At any time step \( t \), the agent can use the above equation to decide the move, \( m \) (external decision) and the working memory, \( c \) (internal decision).

Values that yield high reward are learned as the agent progresses through the task. However, it could serve useful to make random decisions, rather than just relying on learned values once in while in case the agent is stuck in a local minimum. This is known as the idea of exploration versus exploitation. To accommodate for exploration, we implement an epsilon soft policy, \( \epsilon \), which allows for the agent to make random decisions when a random value less than \( \epsilon \) is drawn. During the random move, \( \epsilon \) and the \( atr \) are not affected, only \( m \) is.

Along with the decision of the external and internal move, the agent must also decide which context to use. This decision is independent of the decision above. The context switch is triggered by the error rather than maximum value estimates. When \( \delta_t \) crosses a certain threshold, the context switch is triggered. The threshold, \( t \), in the model is a static hyper-parameter (unlike the dynamic \( t \) defined in nTL).

When the agent receives a large negative error that crosses \(-t\), the agent uses the TD error to interpret that the wrong \( atr \) was used. For this case, the agent chooses the next \( atr \) sequentially rather than choosing the highest value because the correct \( atr \) cannot be determined by value for the case of large negative errors.

When the positive \( t \) is crossed with a large positive TD error, a simple \( \arg\max \) function is used to determine the next \( atr \) to use. The agent enumerates through all possible \( atrs \), and the \( atr \) with the highest value is chosen. The function can be defined as:

\[
\text{atrs} \leftarrow \arg\max_{\text{atrs}} (\bar{s} \land \bar{p} \land \bar{w}m \land a\text{atrs} \land \bar{r})
\]

(14)

When the agent switches \( atrs \) (for either large positive or large negative errors), the \( e_t \) is cleared out so that the agent doesn’t learn under the wrong context.

| Parameters      | PO   | NO   | PONPWMtk |
|-----------------|------|------|----------|
| Episodios       | 50000| 50000| 50000    |
| HRR Length (\( n \)) | 6144 | 6144 | 15360    |
| Max Steps Per Episode | 100  | 100  | 300      |
| Discount (\( \gamma \)) | 0.9  | 0.7  | 0.9      |
| Alpha For Neural Network (\( \alpha \)) | 0.1  | 0.1  | 0.1      |
| Alpha For Testing (\( \alpha \)) | 0.01 | 0.01 | 0.01     |
| Epsilon Soft (\( \epsilon \)) | 0.0001 | 0.0001 | 0.00001 |
| Threshold (\( t \)) | -    | \( \pm 0.3 \) | \( \pm 0.3 \) |
| Discount For The Trace (\( \lambda \)) | 0.01 | 0.01 | 0.01     |
| Signals (\( \bar{p} \)) | R, G, B | -    | R, G     |
| Goals | 5, 10, 15, 10, 15 | 0, 5, 10, 15 |
| NO Task Switch Rate | -    | 500  | 1000     |
| Size of Maze    | 20   | 20   | 20       |

Table 1: Major parameters for all the models. All parameters can be found on the github page.

Equations 8 through 14 describe the combined model, but the constituents are directly present within the combined model. Either of the constituents can be used independently by presenting only PO or NO tasks.
Test Protocols
We provide three tasks that test the effectiveness of our model. The three tasks test the PO, NO, and PONO features of the PONOWMtk.

Partially-Observable To isolate the PO constituent of our model, a maze task using a one-dimensional array with three signals corresponding to three goals was constructed. The three signals R, G, and B correspond to the three goals at locations 5, 10, and 15. There is only one context in the task since the signals always refer to the same goals. By setting the goals and signals in this manner, the PONOWMtk acts exactly like its PO constituent. For a visual representation, consider another maze as shown in Figure 1. Each row acts, independently, as a PO task because of the signal.

At the first time step, the agent is randomly dropped into the maze with the PO feature (signal vector) present in the environment. The agent must use its working memory feature which essentially turns this NM task into a M one. At each time step, the agent must decide what move, left or right, to make using equation 13 along with all other necessary equations. As the agent progresses through the task, it is presented with scalar rewards. At states that are not the goal, the agent receives a reward of −1, and at 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. The agent uses this identification to form the internal representation of the goal. During the training phase, the agent can also use ε-soft to find the global minimum.

To test the effectiveness of the combined model on a task with only PO features, a simple error checking system was used for the last 10% of the episodes. The agent calculates the difference between the total steps it had taken to complete the task and the optimal steps it should have taken. To make sure that the testing was stable, α was set to 0.01 and ε was set to 0. Additional experimental parameters are shown in Table 1.

Non-Observable For the isolated NO constituent of our model, a maze task with three goals and no signals was constructed. The three goals were set to locations 5, 10, and 15 with three available atrs. The combined model, with these settings, behaves the same way that the NO constituent behaves because no information in the environment directly relates to the goals. By setting the goals and signals in this manner, the PONOWMtk acts exactly like its NO constituent. For a visual representation, consider another maze as shown in Figure 1. The two rows act as NO tasks, if the signal is removed from the figure.

The agent is randomly dropped into the state array just as with the above task, but there is no signal present. The agent must learn to map the three atrs to the goals. The decision making is similar for left and right moves as the task above, but there are no wm decisions. During the learning and testing phase of the NO task, the context switches after a set number of episodes have passed. The agent must decide which atr to use using equation 14 and all other relevant equations. The reward system is the same as described above in the PO section.

The testing was done almost the same way as above, but when the agent switches atrs due to error, the steps count and optimal steps were reset. Therefore, the agent would not be penalized for the switch. Additional experimental parameters are shown in Table 1.

Combined 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. Under one context, signals R and G correspond to goals 0 and 5. Under the other, signals R and G correspond to goals 10 and 15. The context automatically switches every 1000 episodes. For a visual representation, consider another maze as shown in Figure 1.

At time step one, the agent is dropped into the array at a random spot with a PO signal vector present in the environment. Unlike the PO task, the signal might have different meanings depending on the context. Due to this, the agent must learn to use wm using equation 13 (and all other related equations) to solve the PO part of the task and use atrs using equation 14 (and all other related equations) to solve the NO part of the task. The reward system is the same as described above in the PO section.

Just as with the other two tasks, the agent was tested for the last 10% of the episodes with ε set to 0 and α set to 0.01. The optimal steps are calculated in the same way as the NO task along with the step reset. Details about the parameters and hyper-parameters are listed in Table 1.

Results
The results are based on the tasks described above in the test protocol section. Figure 2 shows the results of testing the model on a task with only PO features while Figure 3 shows the results of testing the model with only NO features, as described above. Finally, Figure 4 shows the results of the combined model from the task described above. The parameters for all the runs are shown in Table 1. The x-axis of the graphs are the different states in the one-dimensional maze described above, and the y-axis shows the values of the states calculated using the neural network.

Figure 2 shows the agent’s ability to learn to associate the different signals to their goals. The top-left plot in Figure 2 shows that the agent is able to use the internal representation, RIm, of the signal R to solve the problem. With RIm in working memory, the agent is able to traverse the maze using the red line shown in the first plot to find the goal at state 5. The next top-right and bottom-left plots show the internal values of GIm and BIm, which are the respective internal representations of the signals G and B. The agent is able to use these values to solve the maze by finding the appropriate goal, based on the initial signal. The bottom-right plot shows the value at each state on the first time step when the signal is present. By using the information present in the figure above, the agent is able to solve this task with perfect accuracy.

There is a dip at the goal state because the agent doesn’t use the same internal representation at the goal, versus the rest of the states. At the goal state, the agent receives a re-
ward token which is convolved with the current goal state representation. The values for the goal state with the token are now shown, but they just have values close to one, due to the optimistic critic.

The agent completed this task with a 100% accuracy using the parameters listed in Table 1. The perfect accuracy means that the agent was able to utilize working memory by using the internal representation of the signal at time step one to reach the goal with optimal steps.

Figure 3 shows the values of the model when testing on a task with only NO features. The three lines in the figure represent the three atrs the model has available. The agent is able to use one atr to solve each of the tasks. When the task is switched on the agent, it is able to cycle through its available atrs and find the right one to solve the task. With the task switching mechanism, the agent is able to solve the task with perfect accuracy.

The agent completed this task with a 100% accuracy using the parameters listed in Table 1. The 100% accuracy means that the agent was able to switch atrs based on internal feedback and reach the goals with optimal steps.

Figure 3: Values of the model when run on a purely NO task with goals 5, 10, and 15.

Figure 3 shows the values of the model when testing on a task with only NO features. The three lines in the figure represent the three atrs the model has available. The agent is able to use one atr to solve each of the tasks. When the task is switched on the agent, it is able to cycle through its available atrs and find the right one to solve the task. With the task switching mechanism, the agent is able to solve the task with perfect accuracy.

The agent completed this task with a 100% accuracy using the parameters listed in Table 1. The 100% accuracy means that the agent was able to switch atrs based on internal feedback and reach the goals with optimal steps.

Figure 4: Values of the model when run on a purely NO task with goals 5, 10, and 15.

Figure 4 shows the values of the model when testing on a task with only NO features. The three lines in the figure represent the three atrs the model has available. The agent is able to use one atr to solve each of the tasks. When the task is switched on the agent, it is able to cycle through its available atrs and find the right one to solve the task. With the task switching mechanism, the agent is able to solve the task with perfect accuracy.

The agent completed this task with a 100% accuracy using the parameters listed in Table 1. The 100% accuracy means that the agent was able to switch atrs based on internal feedback and reach the goals with optimal steps.

Figure 4: Values of the model when run on a purely NO task with goals 5, 10, and 15.

Figure 4 shows the values of the model when testing on a task with only NO features. The three lines in the figure represent the three atrs the model has available. The agent is able to use one atr to solve each of the tasks. When the task is switched on the agent, it is able to cycle through its available atrs and find the right one to solve the task. With the task switching mechanism, the agent is able to solve the task with perfect accuracy.

The agent completed this task with a 100% accuracy using the parameters listed in Table 1. The 100% accuracy means that the agent was able to switch atrs based on internal feedback and reach the goals with optimal steps.

Figure 4: Values of the model when run on a purely NO task with goals 5, 10, and 15.

Figure 4 shows the values of the model when testing on a task with only NO features. The three lines in the figure represent the three atrs the model has available. The agent is able to use one atr to solve each of the tasks. When the task is switched on the agent, it is able to cycle through its available atrs and find the right one to solve the task. With the task switching mechanism, the agent is able to solve the task with perfect accuracy.

The agent completed this task with a 100% accuracy using the parameters listed in Table 1. The 100% accuracy means that the agent was able to switch atrs based on internal feedback and reach the goals with optimal steps.

Figure 4: Values of the model when run on a purely NO task with goals 5, 10, and 15.
ically. As a result, the agent will try to step towards the goal with the wrong atr in mind and fail to reach the correct goal, ultimately receiving an error. Using this error, the agent can switch the atr and continue to solve the problem.

The agent was able to achieve a 100% during the testing phase, which includes the switching of abstract tasks and working memory. The 100% accuracy means that when the context is switched, the agent is able to use the switching mechanisms to choose the appropriate atr to solve the task along with the appropriate wmn.

| ATR Switching Mechanism                  | Maximum Accuracy |
|-------------------------------------------|------------------|
| Only Positive Error Switch                | 24.98%           |
| Only Negative Error Switch                | 99.01%           |
| Both Positive and Negative Error Switch   | 100%             |
| No Error Switching                        | 24.52%           |

Table 2: Accuracy for different combinations of task switching based on error for the combined model. All other features and parameters are kept static and match the runs above.

For perfect accuracy, the agent needs to use both positive and negative error switching. Table 2 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 perfect accuracy. The lack of a positive switch in this particular run had a slight but important impact; on other runs, the agent was observed to perform the same as at chance.

For tasks with only PO or NO features, the agent was able to achieve perfect accuracy for all runs we tested on. When the task has both features, the agent doesn’t always solve it with perfect accuracy. On most runs, the agent is able to achieve 98% – 100% accuracy. The results can be reproduced by using the code found on github (https://github.com/nibraaska/Working-Memory-Temporal-Difference).

Discussion
Since autonomous systems don’t perceive the world in the same way that humans do, there needs to be a toolkit that allows the agent to form its own percepts. The agent must also be able to disregard irrelevant information from the environment and use relevant information to solve this task. To give autonomous systems the ability to do this, the HWMTk was created (DuBois and Phillips 2017). The HWMTk allows for problems, specifically PO tasks, where the Markov property is not met to be solved with the use of Holographic Reduced Representations, but it is not useful for tasks where the environment doesn’t have all the relevant information. To overcome this hurdle, the nTL was created (Jovanovich and Phillips 2018).

The PONOWMTk was created by taking inspiration from the HWMTk (DuBois and Phillips 2017), with most of its important features to make the PO part of our model work. However, we didn’t implement all of the major features of nTL (Jovanovich and Phillips 2018). A large difference in our model is the type of algorithm used; the nTL uses SARSA, which learns the state-value function (Sutton 1988), while we used value function learning. Another large difference concerns the dynamic threshold. Equation 9 shows the dynamic threshold for the nTL with which any number of tasks can be learned. Our model doesn’t have a dynamic threshold; rather, it has a static one. With the static threshold, our model is able to achieve perfect accuracy on a predetermined number of tasks, but it is not able to learn tasks dynamically.

These two distinct models described above can either utilize abstract task representations to solve NO tasks or storage of past input features to solve PO tasks, but cannot do both. However, in the real world, tasks are not simple. They contain both PO and NO features, so there needs to be a model that can effectively solve these complex tasks. In this paper, we presented a model, PONOWMTk, that can utilize both methods to solve tasks with both PO and NO features. From the results above, it is evident that the PONOWMTk is able to solve complex tasks that have both PO and NO features.

In the future, we would like to extend the model with a dynamic threshold to adapt to a changing number of NO tasks. Additionally, we would like to explore ATR reuse or duplication for similar tasks to encourage generalizations. The PONOWMTk presents new and promising avenues for further exploration.

Acknowledgement
The authors would like to thank the Undergraduate Research Center at Middle Tennessee State University for funding the project.

References
[Baddeley 1992] Baddeley, A. 1992. Working memory. Science 255(5044):556–559.
[Collins and Frank 2012] Collins, A., and Frank, M. 2012. How much of reinforcement learning is working memory, not reinforcement learning? A behavioral, computational, and neurogenetic analysis. The European journal of neuroscience 35:1024–35.
[DuBois and Phillips 2017] 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, 137–144.
[Fukuda et al. 2001] Fukuda, T.; Michelini, R.; Potkonjak, V.; Tzafestas, S.; Valavanis, K.; and Vukobratovic, M. 2001. How far away is “artificial man”. IEEE Robotics & Automation Magazine 8(1):66–73.
[Goldstone 1998] Goldstone, R. L. 1998. Perceptual learning. Annual review of psychology 49(1):585–612.
[Jovanovich and Phillips 2018] Jovanovich, M., and Phillips, J. L. 2018. n-task learning: Solving multiple or unknown numbers of reinforcement learning problems. In Chuck Kalish, Martina Rau, J. Z., and Rogers, T., eds., CogSci 2018, 584–589.
[Kunz 2000] Kunz, F. 2000. An introduction to temporal difference learning. In Seminar on Autonomous Learning Systems.

[Olshausen and Field 1996] Olshausen, B. A., and Field, D. J. 1996. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. Nature 381(6583):607.

[OReilly et al. 2002] OReilly, R.; Noelle, D.; S Braver, T.; and D Cohen, J. 2002. Prefrontal cortex and dynamic categorization tasks: Representational organization and neuromodulatory control. Cerebral cortex (New York, N.Y. : 1991) 12:246–57.

[Phillips and Noelle 2005] Phillips, J. L., and 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., 599–604.

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

[Sutton and Barto 2018] Sutton, R. S., and Barto, A. G. 2018. Reinforcement Learning: An Introduction. The MIT Press, second edition.

[Sutton 1988] Sutton, R. S. 1988. Learning to predict by the methods of temporal differences. Machine learning 3(1):9–44.

[Tugcu et al. 2007] Tugcu, M.; Wang, X.; Hunter, J. E.; Phillips, J.; Noelle, D.; and 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).