Working Memory Concept Encoding Using Holographic Reduced Representations

Grayson M. DuBois and Joshua L. Phillips
Department of Computer Science
Middle Tennessee State University
Murfreesboro, TN 37132

Abstract

Artificial neural networks (ANNs) utilize the biological principles of neural computation to solve engineering problems, and ANNs also serve as formal, testable hypotheses of brain function and learning in the cognitive sciences. However, ANNs are often underutilized in applications where symbolic encoding (SE) of conceptual information is preferred (robotics, games, theorem proving, etc.) since ANN models often employ distributed encoding (DE). The Working Memory Toolkit (WMtk) was developed to aid the integration of an ANN-based cognitive neuroscience model of working memory into symbolic systems by mitigating the details of ANN design and providing a simple DE interface. However, DE/SE conversion is still managed by the user and tuned specifically to each learning task. Here we utilize Holographic Reduced Representation (HRR) to overcome this limitation where HRRs provide a framework for manipulating concepts using a hybrid DE/SE formalism that is compatible with ANNs. We validate the performance of the new Holographic Working Memory Toolkit (HWMtk) using two simple partially observable reinforcement learning problems, and show how the HWMtk automates the process of DE/SE conversion for the user while seamlessly providing additional cognitive capabilities such as context switching, cross-task generalization and concept chunking.

The field of artificial intelligence (AI) is synergistic with a wide range of disciplines but artificial neural networks (ANNs) is perhaps the most prolific subfield. Not only are biological principles of neural computation and neuroanatomy adapted to solve engineering problems, but ANNs also serve as formal, testable hypotheses of brain function and learning in the cognitive sciences. Still, since ANN models often employ distributed encoding (DE), most have limited application in other areas of AI where symbolic encoding (SE) is the norm (e.g. planning, reasoning, robotics).

There is extensive evidence that the brain contains a working memory (WM) system that actively maintains a small amount of task-essential information and supports learning in myriad ways: focusing attention on the most task-relevant features, transfering learning across tasks, limiting the search space for perceptual systems, providing a means to avoid the out-of-sight/out-of-mind problem, and providing robust behavior in the face of irrelevant events (Baddeley 1986; Waugh and Norman 1965). The prefrontal cortex and mesolimbic dopamine system have been implicated as the functional components of WM in humans and animals, and biologically-based ANNs for WM have been developed based on electrophysiological, neuroimaging, and neuropsychological studies (O'Reilly et al. 2002; Kriete et al. 2013). A software library, the working memory toolkit (WMtk), was developed to aid the integration of ANN-based WM into robotic systems by mitigating the details of ANN design and providing a simple DE interface (Phillips and Noelle 2005). The WMtk has been tested in several task domains in the area of cognitive robotics (Tugcu et al. 2007; Erdemir et al. 2008; Busch et al. 2007; Gordon, Kawamura, and Wilkes 2010).

Despite the fact that the WMtk can solve common tests of working memory performance, the DE/SE distinction is still problematic for the WMtk. DE/SE conversion still needs to be programmed directly by the user and tuned specifically to each learning task. A technique called holographic reduced representation (HRR) may provide the technical assistance needed to overcome this limitation (Plate 1995). HRRs provide a framework for creating and combining symbolic concepts using a distributed formalism that is compatible with ANNs. Our aim is to create a software engine for encoding and manipulating concept representations using HRRs and integrate it into the WMtk. The newly integrated HRR Engine (HRRE) would greatly simplify the user interface by automating DE/SE conversion. We assess the performance of the new Holographic Working Memory Toolkit (HWMtk) on two main criteria: 1) significant improvement in the ease of use of the toolkit with automated DE/SE conversion, and 2) robust performance on working memory tasks while using HRRs in place of user-defined distributed representations.

Background

An example of the capabilities of the WMtk can be seen in a robotic simulation written using the toolkit based on the delayed saccade task (DST) (Phillips and Noelle 2005). In the DST, the robot is required to focus attention on a crosshair in the center of the screen. After a variable time delay, a target object will appear in the periphery of the screen, but the robot must continue to focus on the crosshair in the face of this distraction. After some time, the target object disappears and the robot must continue to focus on the crosshair. Finally, the crosshair disappears and the robot must then look

at (or saccade to) the location where the target object appeared during the task. Rather than programming the robot to solve the DST, the WMtk allows the robot to learn how to solve the DST by repeatedly attempting the task as a series of episodes. The robot's WM learns to both override automatic behaviors (such as immediate saccades) and store task-relevant information (such as target locations) in order to guide future actions. Importantly, the robot is given feedback (positive reward) only at the end of correctly performed episodes. Even under these conditions, the WMtk learned to correctly manage items in WM and attain proficiency on the DST within merely hundreds of episodes.

Even though the toolkit mitigates many challenges to the integration of a well-established model of WM into learning systems, the toolkit does not aid the user in developing reasonable representations of the environment or working memory concepts themselves. Each of these components needs to be encoded using a sparse, distributed formalism that support learning by the underlying neural network. Such representations are challenging to develop and implement for even expert users, and are limited in applicability to the specific learning task in question. For example, a simple binary encoding of two distinct concepts for WM would require the user to write a function which takes a two-element vector and a string concept. The function would be used to set the appropriate element of the vector to one based on which concept the WMtk is trying to encode. This function would need to be rewritten every time additional concepts need to be encoded if they were not anticipated from the start. Also, this framework leaves the user open to difficult debugginc concerns that might otherwise be avoided if the encoding process was handled automatically. Additional functions for encoding information about the current state of the task and calculating reward based on this state information were also needed by the original WMtk. A more flexible encoding scheme is needed to make the toolkit more accessible to end-users.

HRRs may provide all of the necessary capabilities to solve the SE/DE conversion problem in an automated fashion. In the HRR formalism, distinct concepts, each represented by unique, distributed vectors of real numbers, can be combined and reduced to a single vector that represents the combined knowledge of its constituent concepts. Importantly, the length of the HRR vector encodings for all concepts (both constituent and combined) remains fixed since concepts are compressed using a mathematical operation known as circular convolution. Nevertheless, combined concepts still retain information about each the constituents which is a key property of holographic storage methods. Additionally, HRRs are DE representations which are compatible with neural network architectures and other machine learning approaches. However, since all concepts are encoded into unique vector representations, each HRR can be tied to a complementary SE representation (eg. the concept name) more commonly used in other symbolic or logic-based learning systems (Plate 1995). By replacing the DE interface of the WMtk with an HRR interface, DE/SE conversion would be automated and concepts learned from one task would naturally carry over to new tasks. Additional cognitive phenomena (e.g. chunking, cross-task generalization) may be investigated as well. Here we aim to develop and test a holographic reduced representation engine (HRRE) for accomplishing automated DE/SE conversion, and integrate the HRRE into the Working Memory Toolkit.

Methods

Work on this project was performed in 2 phases, both separated into three parts. During the first phase we created an engine to generate and manipulate HRRs. This HRR Engine provided us with a means of encoding, storing, and manipulating representations of concepts used in the WMtk. During the second phase, we rebuilt the WMtk around the HRRE, replacing the original DE interface with a simple SE (string-based) interface, and automating the encoding and manipulation of concepts using HRRs. Testing of new holographic working memory toolkit (HWMtk) was then performed using a basic memory retrieval task, and a partially observable maze exploration task.

Phase 1: Creating an HRR Engine for Concept Encoding

Our 3-part process for creating the HRRE consisted of (a) researching holographic reduced representation, (b) developing a conjunctive encoding engine, and (c) developing a conjunctive decoding engine. These phases were implemented as follows:

Holographic Reduced Representation HRR is a robust method of representing symbolic concepts in a distributed formalism. Simple concepts can be combined to make holographic representations for complex concepts containing information from each of the constituents. Additionally, it is possible to use HRRs in place of corresponding symbolic concepts for interfacing with ANNs. HRRs are constructed as a vector of real values typically drawn from a Normal/Gaussian distribution with zero mean ($\mu = 0$), and standard deviation, $\sigma = 1/\sqrt{(n)}$ where n is the length of the vectors. Longer vectors allow the encoding of more unique concepts as well as storage of more constituents within combined representations. Therefore, n is often adjusted for adequate performance on requisite tasks. Additional constraints can be placed on the vectors as well, such as in the case of "unitary" vectors, which exhibit some additional, useful mathematical properties, and were the types of HRRs employed here (Plate 1995). A newly generated HRR can then be assigned to a unique symbol for a single concept. Here we use strings to represent symbols, where the string value is simple the name of the concept we wanted the HRR to represent. For example, to generated an HRR for the concept red, we generated a unitary HRR, and assigned it to the string value "red".

The base data structure for our HRRE is a dictionary, where the string name for the concept is the key and the HRR representing that concept is the value. This dictionary serves as the engines long-term concept memory. From this point forward, the term *concept* will be used to refer to the entity composed of a string value and its associated HRR. These concepts are the key-value pairs stored in concept memory.

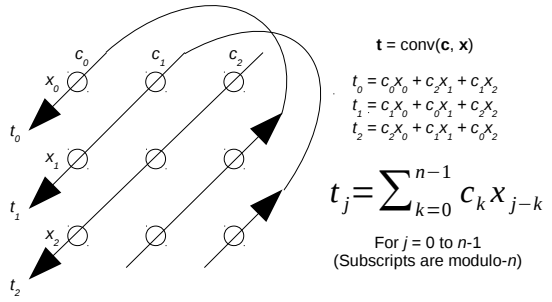

Figure 1: Example of how to perform circular convolution on two vectors (c and x), both of length n=3.

The term *representation* may be used interchangeably with HRR, since HRRs are the structure we use to form the distributed representation of the concepts we will use.

Developing a Conjunctive Encoding Engine After setting up the base dictionary for the engines concept memory and building the functionality for HRR generation, we needed to add a conjunctive encoding function to the engine for combining the HRRs to form complex concepts. We combine the representations for the concepts using the circular convolution operation. This operation is what makes holographic representations reduced, as it combines information from two HRRs into a single HRR of the same size. Circular convolution consists of constructing the outer product of the two HRRs and summing the elements along the trans-diagonals, as indicated in Figure 1. This $O(n^2)$ operation results in an HRR of size n with information from both constituent HRRs. This operation can be performed in $O(n\ log(n))$ time, however, using fast-fourier transforms (FFT). Therefore, we use FFTs in the HRRE for convolution operations.

Circular convolution is the key operation used in the main part of the HRREs conjunctive encoding functionality: the *construct* function. This function takes a list of concept names, or a string containing the concatenation of concept names, delimited by an asterisk ("*"), and combines all individual concepts into a single complex concept. The construct function would first make sure that there is a representation in concept memory for each given concept name, reorder each concept by lexicographical order, and then convolve each together to form the final representation. In this way, the construction of the concept made from "big", "red", and "ball" would result in the complex concept, "ball*big*red". What makes the construct function so powerful, however, is that in addition to constructing the final combination of all concepts, it also constructs representations for every combination of the constituent concepts. For example, constructing a concept using the values "big", "red", and "ball", would not only create the full combination above, but also the combinations "ball*big", "ball*red", and "big*red". These additional operations speed up memory retrieval by minimizing the number of circular convolutions that the system may need to perform in the future.

We ensured that the HRRE always sorts concepts into lexicographical order before working with them in order toeliminate duplicate representations made for the same concept. For example, we do not want the engine to create a new representation for "big*red*ball", if the analogous concept, "ball*big*red" exists. Generating a new representation for "big*red*ball" would be redundant. There is an additional safeguard built into the engine that protects against generating redundant constituent subconcepts this as well (eg. "ball*red" and "red*ball"). Beyond these constraints, whenever a concept is requested from the HRRE that it does not currently have in memory, it constructs the new representations. Constituent concepts are split apart by name, and combinations are constructed using the process described above. In this way, the user can pass in "ball*big*red", "red*ball*big", or any other permutation of these concepts, and the HRRE will always construct or perceive it as "ball*big*red".

Developing a Conjunctive Decoding Engine The final piece added to the HRRE was the conjunctive decoding function. Whereas conjunctive encoding is the combination of two representations through circular convolution, conjunctive decoding is the extraction of a constituent concept from a complex concept using circular involution (i.e. the inverse operation of circular convolution). Using circular involution, we can take the representation for the concept, "red*ball" and probe it using the representation for the concept, "ball". The resulting HRR would be the representation of the alternative constituent concept, "red". Identification of "red" is performed by calculating the dot product between the resulting HRR from the involution operation and all other concepts currently stored in the HRRE. The concept with the highest dot product similarity will be "red" in this case. Therefore, the HRRE can infer that the HRR for "red*ball" consists of both the probe concept "ball", and the response concept, "red".

Similar to the encoding part of the engines construct function, the decoding part of the engine has an *unpack* function that deconstructs a complex concept into all combinations of its constituent parts. Whereas the construct function is merely encoding each combination to ensure that they are all recognizable concepts for the HRRE, the unpack function serves to find all combinations and return them as a list of concepts. This is useful to the WMtk, as it will be the means by which a list of concepts will be constructed as candidates for WM contents.

By combining the encoding and decoding function, the HRRE is capable of 1) generating HRRs for new concepts, 2) storing concepts as key-value pairs of names and representations in the concept memory dictionary, 3) combining concepts through circular convolution, 4) extracting concepts through circular involution, 5) constructing and encoding all combinations of a list of concepts, and 6) unpacking all combinations of constituent concepts from a complex concept, and returning the resulting list to the user.

Phase 2: Building the Holographic Working Memory Toolkit

Our three-part process for building the HWMtk comprised of (a) researching the specifications of the original WMtk and making a development plan for the augmented toolkit, (b) rebuilding the WMtk around the HRRE, and (c) testing the augmented toolkit to ensure that it still learns using the new HRR interface.

Original WMtk Architecture The Working Memory toolkit implements a working memory model inspired by the interactions between the human pre-frontal cortex (PFC) and mesolimbic dopamine system (MDS). In this working memory model, the PFC maintains task-relevant information, and the DMS decides when such information should be gated into or out of PFC storage. The WMtk implements the PFC as a fixed set of *slots* for storing task-relevant concept representations. However, the MDS is implemented as a single-layer neural network, or *critic network*, which evaluates the utility of storing concepts in the slots. The network is passed an HRR representation consisting of representations (potentially) stored in working memory slots convolved with the HRR representation of the current task state. The value produced by the critic network determines how valuable that particular set of working memory contents is under the current task state. All states and combinations of WM contents are equally valuable at first, but the critic network employs temporal difference (TD) learning (Sutton and Barto 1998; O'Reilly et al. 2007) to learn the correct value of each WM-state combination by experiencing repeated episodes of the learning task. In this way, the working memory learns what information is the most valuable to retain for future processing given what it is currently experiencing. At this point, the user designs their learning task in such a way that the agent decides to make an action according to what is currently held in working memory.

We decided to start with a minimal design for our augmented toolkit since our aim was to improve ease of use for researchers using the toolkit. Therefore, here we utilize the HRRE to provide a simple interface for automating the concept encoding process. Additional functionality will be added to the toolkit in future work.

Rebuilding the WMtk Around the HRRE We determined that the two main components of the toolkit would be the Working Memory (WM) and the Critic Network (CN). We created both components under the following specifications:

Working Memory The WM component is the workhorse of the toolkit. It houses the HRRE, which serves to store all basic concepts, as well as the processor for the representations of all combinations of combined concepts. First, WM receives a string representation of the current state. The state is set up as a string containing a concatenation of the concepts describing the state, delimited by the addition symbol ("+"). An example of a state containing a cross in the center of the environment and a target in the north position of the environment could be denoted "center*cross+north*target". WM parses the state string for the list of concepts it contains, splitting each by the addition delimiter. These concepts are then passed to the HRRE, which returns a list of all the unpacked combinations of concepts. Following the cross-target example, the list of candidate chunks would be "center", "center*cross", "cross", "north", "north*target", and "target". This list of concepts becomes our list of candidate chunks: all are candidates for retention in WM. It is important to note that the *previous* contents of WM are also included in the list of candidates for retention. This process allows WM to potentially store task-relevant information for long-term. WM then goes through every combination of all candidate chunks that can fit in its WM slots. The value of WM candidates is determined by convolving them with the representation of the state, and feeding each combination into the critic network. Element-wise addition is used to combine vectors separated by the addition operator within the state representation before convolution. When passed into the critic, the set of WM contents that returns the highest value in the given state is chosen for retention, and the control is returned to the user until WM is passed a new state on the next step of the task.

Critic Network The CN component is the neural network that drives learning in the WMtk. It is passed representations from WM in order to compute the value function. The value function for the CN is a dot product calculation of the WM-state combination with a weight vector that is retained for the duration of the simulation. The weight vector is initialized with small random values. Therefore, initial values for each representation will be quite low. However, the CN employs TD-learning over many repeated task episodes, which will update the values in the weight vector. The value function eventually converges to the correct values for each WM-state combination, according to their effectiveness at determining task outcomes.

TD learning is implemented through 3 functions in the toolkit: *Initialize Episode*, *Step*, and *Absorb Reward*. Each function is passed the string representation of the state and the reward for that state. These functions are implemented and called through the WM object, but are closely tied to the CN for TD calculations. Initialize episode resets all episodic variables, clears and chooses the initial contents for WM, and stores reward and value information about the initial state for later use. Step chooses the current contents of WM, calculates reward and value information for the current state, and uses those values along with those stored from the previous state to update the weight vector using the CNs TD learning functions. Step then stores the current states value and reward for use in the next step of the episode. Step is called on each time step of the simulation to update working memory and drive learning. Finally, Absorb Reward is called at the end of the episode, which takes the state string for the final state, and computes the TD update for the previous state as well as the final state. Typically, a scalar reward of zero is provided throughout all steps of the task. On the final step, a reward value of 1 is provided if the agent successfully completes the task and zero for task failure. However, other reward schedules are permissible. When a new episode begins, these functions are called again, in the same order:

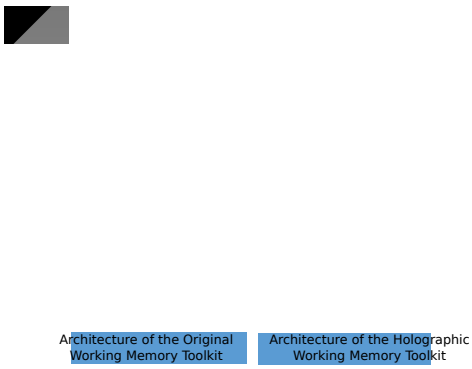

Figure 2: Comparison of the original WMtk architecture (left) to the architecture of the HWMtk (right).

Initialize Episode, a sequence of calls to Step, and finishing the episode with Absorb Reward. We do also use eligibility traces in our TD calculations, and an ϵ-soft policy is implemented by generating random WM contents, ϵ percent of the time.

A visual comparison between the architectures of the original and augmented toolkit is shown in Figure 2. The main difference between the two architectures is in the amount of code the user needs to provide in the form of functions/methods. Many of these user-defined functions are now completely performed within the HWMtk. Sensory information can now be provided in a symbolic, English-like syntax. Symbols are automatically converted to appropriate vectors by the HRRE for presentation to the CN. The selection of task-relevant concepts for storage in working memory is learned over time, enabling the agent to override prepotent responses with task-relevant behaviors. Also, while the function calculating reward information still needs to be specified by the user, the augmented toolkit does not need to call this function directly. This simplifies the user's implementation since it no longer needs to be concerned with the inner-workings of the toolkit to perform reward calculations.

Testing the Toolkit to Ensure Learning Capabilities We developed two tasks for the HWMtk to determine if the user interface is indeed easier for developing new tasks compared to the original toolkit. Additionally, the tasks test the basic components of working memory function: learning to store task-relevant information and ignore task-irrelevant information (distractors). For the first task, the agent is shown 7 colors in random order, and is rewarded if it remembers the color "red" at the end of the simulation. This task would be equivalent to shuffling 7 cards of different colors, and showing them all to the agent, one at a time, and asking at the end which color we were thinking of. The task is simple, but not trivial, as the toolkit can choose to remember nothing or any of the other colors as well. Also, the presentation order is randomized, so the agent cannot anticipate when the relevant color is being presented. The agent must decide to hold onto the color "red" and retain this concept in working memory even while other colors (distractors) are being presented to the agent until the end of the episode is reached. We repeat this process many times (each repetition being a single episode). The agent must learn that it is only rewarded upon remembering "red", regardless of presentation order or the number of distractors encountered. This ability to retain task-relevant information in the face of competing distractions is one of the core mechanism of focused attention needed to perform all working memory tasks.

Learning parameters for the task were set to similar values as the defaults for the standard WMtk: CN learning rate parameter, α=0.1; future reward discounting factor, γ=0.9; past action eligibility factor, λ=0.1; ϵ-soft random working memory selection probability, ϵ=0.01; number of working memory slots, s=1; and HRR vector length, n=64. The HRR vector length (n) is the only new parameter on this list, and must be set to a value large enough that the dot products between base HRR concept vectors remain close to zero. A value of 64 was the minimum size needed to run 100 successful trials (described below), but larger values did not show any noticable difference in learning behavior.

A second task was developed which provided a more rigorous test of working memory function. We simply reward the agent for remembering a specific item in the first task. However, in the second task, the agent must also learn to utilize stored WM representations to make decisions which impact future reward. Contextual information is provided early in this task that is then needed for proper action selection later in the task. We created a simple reinforcement learning agent which utilized temporal difference learning to solve a 1D maze task consisting of 20 independent states. Each state has two immediately accessible neighbors to the left and right. For example state 4 is located to the left of state 5 and state 6 is located to the right of state 5. The maze is periodic such that state 20 is the left neighbor of state 1, and 1 is the right neighbor of state 20. A reward value of 1 is provided to the agent upon reaching the goal, and each each episode consists of starting in a random state and proceeding for no more than 100 steps (early termination if the goal was not reached).

To make the above task *partially observable*, and therefore a good test of WM performance, we also provided a "signal" (S1 or S2) on *only the first step* of each episode which was correlated with the location of the goal for that episode. For example, a presentation of S1 indicated that the goal would be at state 1, but for S2 the goal would be at state 10. This information would be provided to the toolkit on the first step of an episode, but never again for the remainder of the episode. Therefore, only retention of this contextual signaling cue would allow for optimal performance since failure to remember the context signal would make selecting the optimal action an ambiguous decision. The task type (S1 or S2) was chosen at random at the start of each episode, so that there were no predictable performance correlations between tasks.

One HRR vector (n=4096) was generated for each of the 20 states, the 2 signals (S1 and S2), and the two possible actions (left and right). A single-layer critic network was constructed to evaluate the value of combining any working memory representations remembered (or not if WM did not

store the necessary signal cue), and the current state via circular convolution. Convolving this representation with either of the two action representations would allow critic network to select the appropriate action using the same TD-learning formalisms described above.

Learning parameters for the task were set to the following values: CN learning rate parameter, α=0.05; future reward discounting factor, γ=0.9; past action eligibility factor, λ=0.5; epsilon-soft random working memory selection probability *and* action selection probability, ϵ=0.001; number of working memory slots, s=1; and HRR vector length, n=4096.

Results

When testing the HWMtk with the colors task, we were looking to see if it held up to the two main criteria for success mentioned in the introduction: 1) ease of use in setting up a learning task using the new string-passing SE interface, and 2) successful learning using HRRs in place of the old distributed encodings.

Ease of Use

Setting up the colors and signal learning tasks above proved to be more straightforward compared to setting up tasks using the original toolkit. Had we been using the original WMtk, we would have had to write a function to create distributed representations of each color as a chunk of information usable to WM, as well as a similar function for encoding the state, and a reward function to check to provide a reward value according to the agents performance. We would have had to write each of these before writing the logic for the task itself, but using the augmented toolkit, none of this preparation was necessary. We simply set up an array of n color strings, shuffled them at the beginning of each episode, initialized episode with the first color, called the Step function with each subsequent color less than n, and called the Absorb Reward function with the nth color string. The only logic for the reward was written in line with the rest of the task, and it entailed a check to see if red was stored in the contents of WM. If it was, Absorb Reward was provided a reward value of 1.0 for success, else a 0.0 for failure. Considering the simplicity and ease of setting up the task, the HWMtk meets our first and most important criterion for success: simplification of interface and ease of use for the developer.

The C++ code for the HWMtk and colors task agent is open source, distributed under the GPLv3, and available online at: `https://github.com/jlphillipsphd/wmtk/`.

Effective Learning Using HRRs

The colors taks test was run for 100 learning trials. We gathered information over every trial, keeping track of the number of episodes the agent successfully completed the task and recording the number of successes per every 1000 episodes. We considered a 98 percent success rate per thousand episodes an indication that the agent had effectively learned the task. Over the 100 trials, we found that the agent

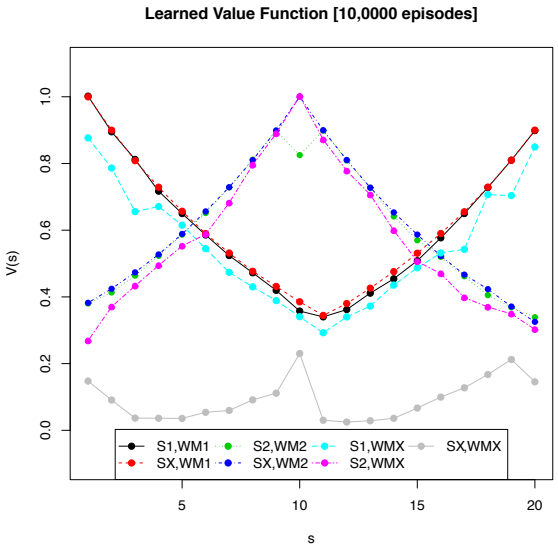

Figure 3: Learned value function of the partially observable 1D maze task. The learned value, V(s), is shown for each state (s). S1 indicates states for when the first task signal is present, S2 indicates states for when the second task signal is present, and SX indicates states when *no signal* was present. These options are combined with possible WM contents: WM1 means the first task signal was stored, WM2 means the second task signal was stored, and WMX means that *no signal* was stored.

learned the task to a 100 percent success rate within an average of 8000 episodes. Therefore, the HWMtk meets the requirement of being capable of learning using holographic reduced representations instead of distributed representations for concepts.

The 1D maze-signal task showed successful learning within 10000 episodes, and the resulting value function across the 20 states for the different possible WM contents is shown in Figure 3. The toolkit was able to learn that the signal information provided (S1,WM1 or S2,WM2) was relevant for proper task discrimination. This is indicated by the expected rise in the value function for each of the respective subtasks as the agent approaches the goal for that subtask. Conditions where signals were retained in WM are shown by SX,WM1 and SX,WM2 for retention of the S1 or S2 signals, respectively. Figure 3 also shows conditions for when the signal was observed, but not yet retained in WM (prior to the call to Step). These are shown in conditions S1,WMX and S2,WMX, respectively. All six of the conditions described above fall along the optimal path to the goal(s) and the CN has learned the correct discounted reward values in all cases. Thus, the value function for each task was learned independently along with learning of the proper contextual cues needed to perform this separation of learning concerns. The performance of the agent when signal information was

not retained in WM memory was also learned. This subproblem (SX,WMX) occurs when the ϵ-soft policy decides to remove signal representations currently stored in working memory, or early in learning when the value of this option looks preferable to storing signal information in WM. This subproblem is equivalent to creating a learning agent without WM capabilities, and the average discounted reward for this subproblem is significantly reduced (0.15 on average) compared to the optimal values. Therefore, WM is critical for learning this task, and indicates that the HWMtk provides the necessary capabilities for learning systems to deal with the out-of-sight, out-of-mind problem.

Discussion

The HWMtk has several advantages over the WMtk by using HRRs for SE/DE representation.

- HRRs are much more robust than the task specific, manually encoded representations used in the original toolkit. New, complex concepts can be encoded automatically without having to alter the topology of the CN since such concepts are constructed via new HRRs or convolved representations of equivalent length. Thus, complex concepts fit into the same WM slots as simple ones, allowing slots to encode increasingly more complex concepts.
- Tasks that were previously beyond the capabilities of the previous toolkit are now more realizable. For example, since new concepts can be formed when needed, learning performance on a simple task might transfer to a more complex task. More complex tasks might be more learned in far fewer episodes by leveraging such previous knowledge rather than learning the task from scratch. Also, since HRRs provide a natural method for encoding hierachical structure, tasks which require paying attention to heirachical signals will be easier to program, and possibly easier to learn.
- The HWMtk antiquates the need for user-specified concept encoding mechanisms. This greatly reduces both the time and knowledge of ANNs needed to adequately set up encoding functions when developing learning agents. Specifically, the user no longer needs background knowledge on how to construct sparse, distributed, conjunctive codes. The user also does not need to rewrite encoding functions when *new* concepts need to be proposed to WM or encoded into the state descriptions. We hope that this alone will increase the interest in the HWMtk, and will make it a better resource for other researchers wishing to test WM-related tasks.
- The HWMtk provides a unique cognitive framework for solving partially observable reinforcement learning problems, by learning to store contextual information that proves useful for later action selection by other learning systems and illustrates the utility of WM as an attention focusing mechanism.

The development of the HWMtk has opened up several new avenues for future work. First, we plan to utilize the HWMtk to create a new version of the delayed saccade task. This task is no more complicated, in practice, than the colors task presented earlier, but it would provide a more intuitive comparison of how the distributed encoding process is simplified by the HRRE component of the HWMtk as shown in Figure 4. Second, the ability to rehearse and group items using convolution might be added to tackle tasks which require memorizing long sequences of information quickly. Such functionality might be used to study how limits on cognitive faculties arise from a small set of WM slots. Additionally, the TD learning element of the toolkit is currently being used to learn internal actions (selecting working memory contents), but has traditionally been used to learn external actions. It seems likely that the toolkit could be provided with a list of symbolic actions to choose from and the TD learning element could then learn to select appropriate actions given the current state and working memory concepts. This avenue would further reduce the programming burden placed on the user, but would also complicate the learning process by needing to learn both internal actions and external actions simultaneously. However, our 1D maze task results indicate that this hurdle was easily overcome by utilizing the same HRR approach to encode each of the possible actions and provides a clear path forward in this regard. More complex tasks involving hierarchical structure and role-filler generalization will also be explored in the future, but may require additional changes to the current generic "slot" approach used by the HWMtk since slots currently all have equal precedence. This assumption may not hold for such tasks.

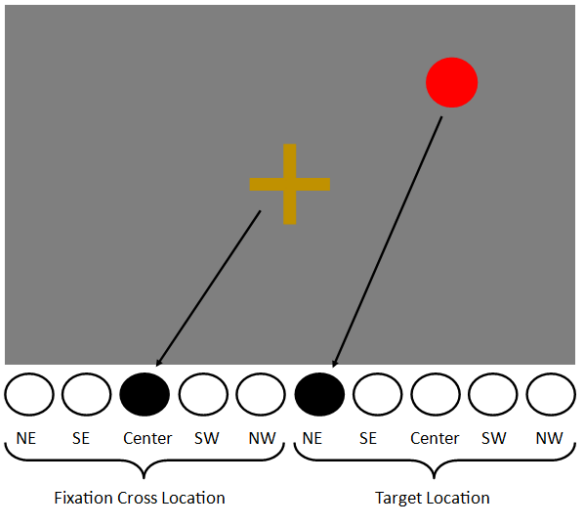

Figure 4: Example of how task-specific, sparse, distributed encoding was performed in the original WMtk. In the HWMtk, an appropriate distributed HRR representation can be built automatically without the users aid from a symbolic description of the environment: "center*cross+northeast*target".

Acknowledgments The authors would like to thank Dr. David C. Noelle (University of California, Merced) for help-

ful discussions and two anonymous reviewers for their supportive feedback.

References

[Baddeley 1986] Baddeley, A. 1986. Working memory. In *Oxford Psychological Series*, volume 11. Oxford: Clerendon Press.

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

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

[Gordon, Kawamura, and Wilkes 2010] Gordon, S. M.; Kawamura, K.; and Wilkes, D. M. 2010. Neuromorphically inspired appraisal-based decision making in a cognitive robot. *IEEE Transactions on Autonomous Mental Development* 2(1):17–39.

[Kriete et al. 2013] 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 et al. 2002] O'Reilly, R. C.; Noelle, D. C.; Braver, T. S.; and Cohen, J. D. 2002. Prefrontal cortex and dynamic categorization tasks: representational organization and neuromodulatory control. *Cerebral Cortex* 12(3):246–257.

[O'Reilly et al. 2007] O'Reilly, R. C.; Frank, M. J.; Hazy, T. E.; and Watz, B. 2007. PVLV: The primary value and learned value Pavlovian learning algorithm. *Behavioral Neuroscience* 121(1):31–49.

[Phillips and Noelle 2005] Phillips, J. L., and Noelle, D. C. 2005. A biologically inspired working memory framework for robots. In *IEEE International Workshop on Robot and Human Interactive Communication*, 599–604. Nashville, TN: IEEE.

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

[Sutton and Barto 1998] Sutton, R., and Barto, A. 1998. *Reinforcement Learning*. Cambridge, MA: MIT Press.

[Tugcu et al. 2007] Tugcu, M.; Wang, X.; Hunter, J. E.; Phillips, J. L.; Noelle, D. C.; and Wilkes, D. M. 2007. A Computational Neuroscience Model of Working Memory with Application to Robot Perceptual Learning. In *Proceedings of the 3rd International Conference on Computational Intelligence*.

[Waugh and Norman 1965] Waugh, N., and Norman, D. 1965. Primary memory. *Psychological Review* 72:89–104.