Holographic Reduced Representations for Working Memory Concept Encoding

by

Grayson McKenzie Dubois

A thesis presented to the Honors College of Middle Tennessee State University in partial

fulfillment of the requirements for graduation from the University Honors College

Fall 2016

Holographic Reduced Representations for Working Memory Concept Encoding

by

Grayson McKenzie Dubois

APPROVED:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Dr. Joshua L. Phillips

Computer Science

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Dr. Chrisila Pettey

Computer Science

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Name of Honors Council Representative

Department of HC Representative

Honors Council Representative

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Dr. John Vile

Dean, University Honors College

**OR**

Dr. Philip E. Phillips, Associate Dean

University Honors College

Copyright © 2016 Grayson M. Dubois & Joshua L. Phillips. All rights reserved.

Department of Computer Science

Middle Tennessee State University; Murfreesboro, Tennessee, USA.

The software described in this work is free software. You can redistribute it and/or modify it under the terms of the GNU General Public License version 3 or later as published by the Free Software Foundation. It is distributed in the hope that it will be useful but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTIBILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

GNU General Public License:

https://www.gnu.org/licenses/gpl-3.0.en.html

The software is posted on GitHub under the following repositories:

Holographic Reduced Representation Engine:

https://github.com/G-Dubois/Holographic-Reduced-Representation-Engine

Working Memory Toolkit:

https://github.com/jlphillipsphd/wmtk

For God,

through whom all work depicted herein is made possible.

For my family,

who helped me to become the person I am today.

For Tressie,

my best friend and unwavering pillar of support.

For all of my brothers and sisters in Christ, especially those suffering from mental illnesses and the afflictions of the brain.

# Acknowledgements

I would like to thank David C. Noelle, Ph.D. (Associate Professor, School of Social Sciences, Humanities, and Arts, University of California, Merced) for introducing my mentor, Joshua L. Phillips, to Tony Plate's work on HRRs. Without his contribution, my research would not have been possible.

I am very grateful for the MTSU Honors College, for providing myself and other students a friendly and welcoming community of scholars and friends. Their opportunities help students like myself to grow in diversity, academia, and scholarship. As a student pursuing a STEM major, I especially owe the Honors College for giving me a true appreciation of the liberal arts. The influence and instruction of faculty and staff at the Honors College has taught me to love and pursue wisdom, and because of them I am determined to be a philosopher first, and a computer scientist second. After all, what use is any level of technical feat if one lacks the ability to think critically and apply those feats to some practical use for the betterment of mankind?

I owe a lot to the MTSU department of Computer Science. I could not be the computer scientist I am today without the instruction and guidance of the faculty and the support of my fellow CS classmates. I especially thank those faculty members who had the greatest influence on my growth during my time under their instruction.

To Dr. Chrisila Pettey, had I not taken your Computer Science I class, I may very well still be studying Mechanical Engineering Technology. Had you not convinced me to change my major, my life would have been very different indeed. Not only do I owe you for my last three years of schooling, but the rest of my career as well. The work you do as department chair to make MTSU’s CS program an exceptional learning experience for my fellow students and me does not go unnoticed. Thank you for everything you do for us.

To Dr. Hyrum Carroll, you challenged me to be the very best software developer I can be, to pay attention to detail, to present myself professionally and with confidence, and perhaps most importantly, not to be afraid of approaching my betters with questions, concerns, and even friendly conversation. Thank you for helping me to grow professionally and showing me that you genuinely care about my fellow students and me, both academically and personally.

To Dr. Joshua Phillips, my advisor and mentor. Since the day I approached you asking about the possibility of research, you have freely given me your full support. You have provided me much more than just guidance and direction in my research; you have given me personal and professional advice and have been patient and supportive through every step of this research project. You have shown that you are not only interested in my success as a computer scientists, but also in my growth as an individual as well. Thank you for your mentorship throughout this research project. I truly believe that you have helped me to become a better person, and the toolkit we built together is an accomplishment I shall always be proud of.

I extend my thanks to all computer science faculty not named above. Every one of them contributes to the success of MTSU’s CS program, and thus every one contributes to the success of the students in it. I speak for all of us when I say thank you for all that you do.

# Abstract

Artificial neural networks (ANNs) utilize the biological principles of neural computation to solve many engineering problems while also serve as formal, testable hypotheses of brain function and learning. However, since ANNs often employ distributed encoding (DE) methods they are underutilized in applications where symbolic encoding (SE) is preferred. The Working Memory Toolkit 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 task. Here we utilize holographic reduced representation (HRR) to overcome this limitation since 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 toolkit and show how it automates the process of DE/SE conversion while providing additional cognitive capabilities.

Table of Contents

[Acknowledgements iv](#_Toc463829317)

[Abstract vi](#_Toc463829318)

[Introduction 1](#_Toc463829319)

[The Prefrontal Cortex and Working Memory 4](#_Toc463829320)

[Symbolic and Distributed Encoding Methods 8](#_Toc463829321)

[Holographic Reduced Representations 9](#_Toc463829322)

[Building the Holographic Reduced Representation Engine 13](#_Toc463829323)

[Building the Holographic Working Memory Toolkit 17](#_Toc463829324)

[Testing the Holographic Working Memory Toolkit 21](#_Toc463829325)

[Results and Discussion 24](#_Toc463829326)

[Bibliography 30](#_Toc463829327)

[Table of Figures 32](#_Toc463829328)

[Appendix 33](#_Toc463829329)

# Introduction

The field of artificial intelligence (AI) is synergistic with a widerange of disciplines but artificial neural networks is perhapsthe most prolific subfield. Not only are biological principles ofneural computation and neuroanatomy adapted to solve engineeringproblems, but ANNs also serve as formal, testable hypotheses of brainfunction and learning in the cognitive sciences. Still, since ANNmodels often employ distributed encoding, most have limitedapplication in other areas of AI where symbolic encoding is thenorm (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 that focuses attention on the most task-relevant features, supports learning that transfers across tasks, limits the search space for perceptual systems, provides a means to avoid the out-of-sight/out-of-mind problem and more robust behavior in the face of irrelevant events (Baddeley, 1986; Waugh and Norman) 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, 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).

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

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 (Phillips). 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 very 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 just hundreds of episodes.

Even though the toolkit mitigates many challenges to the integration of a well-established model of working memory function into other systems, such as development of the neural network architecture and performing working memory updates, the toolkit does not aid the user in developing reasonable representations of the environment or working memory concepts themselves. Each component needs to be encoded using a sparse, distributed formalism that is useful for the neural network to learn, but difficult to program and limited to a single specific task. A more flexible encoding scheme is needed to make the toolkit more accessible to end-users and potentially allow for more generalizable task knowledge and working memory performance.

HRRs may provide the necessary tools to solve the SE/DE conversion problem. The name HRR summarizes how many different concepts, each represented by separate, unique vectors, can be combined and reduced to a single vector that represents the combined knowledge of the concepts while still retaining information about each constituent concept which is closely related to the concept of holographic storage. HRRs utilize a mathematical framework which is compatible with the distributed representations expected by neural network architectures, but are also complementary to symbolic representations used in other systems (Plate). By replacing the DE interface of the WMtk with an HRR interface, DE/SE conversion would be automated, concepts learned from one task would naturally carry over to new tasks, and additional cognitive phenomena (e.g. chunking) may be investigated. Therefore, our specific aim was to develop and test a holographic reduced representation engine, and integrate it with the Working Memory Toolkit.

# The Prefrontal Cortex and Working Memory

Remember a time when someone asked you to remember a number as they quickly spouted it off. “1-1-2-3-5-8-1-3-2-1,” they say, and you struggle to remember it long enough for them to pick up the phone and to dial it. Were you to attempt to remember it now, you would probably be hard pressed to do so without some method of breaking it up. After all, ten random digits is a lot of numbers to remember! If I asked you to remember this number right now, you would probably separate it into smaller groups of digits that are easier to remember. This would look something like “112 – 358 – 1321.” Splitting it up this way makes it easier for us to remember so that we can hold it in our mind long enough to dial it, and then we forget it forever… or until we have to dial it again.

WMChunks_PhoneNumberWhy does splitting up these numbers make it easier for us to remember them? This is because of a system in our brains called working memory whose sole purpose it is to retain a few tidbits of information that are immediately useful for whatever task we are focusing on at any given moment. There are three things that are important for you to know about working memory. First, working memory can only hold small tidbits of information long enough for you to use them. Once they have been used, your brain discards them. Second, these tidbits are often grouped together in chunks of information that is similar in nature. Finally, there are only a few slots in WM within which task-relevant information can be stored. This number varies for each individual but it is now most commonly believed to be four slots for most people (Cowan). Keeping these three things in mind, then, it makes sense that when trying to remember a ten digit phone number, our working memory system splits it into three “chunks” of digits (figure 1). The working memory system holds onto this information long enough to dial the number, and then forgets it.

Figure : An example of how working memory stores task-relevant information.

There is evidence to show that the PFC maintains the representations of the chunks of information we are holding in our working memory (Goldman-Rakic). In addition to this, the PFC also regulates the release of dopamine (Shultz et al). Dopamine is the neurotransmitter that acts as a sort of reward chemical which the PFC releases when exposed to rewarding stimuli. For example, a child being potty-trained who successfully uses the toilet receives an M&M for doing a good job. This is called reinforcement learning, since positive actions are “reinforced” with a reward. When the child eats the M&M, their PFC releases dopamine, which tells the rest of the brain that whatever the child just did was good, and elicits reward for imitating the same behavior in the future. Eventually, the PFC learns to release dopamine immediately after a “good” action rather than after the reward, because it expects the reward in the future. Moreover, when the child gets older and no longer receives M&Ms for using the toilet, the dopamine is released because a reward is expected, but dopamine levels drop when no reward is given (Shultz et al). This tells the brain that what we expected was a good action, no longer elicits reward. Thankfully, most children do not go back to messing their pants, despite this neurochemical punishment!

The facts that the PFC regulates reinforcement learning through dopamine release, as well as maintains representations for task-relevant information in working memory, provide a solid foundation for a computational working memory model. The Working Memory toolkit was developed as a data structure with a few memory slots which held representations of WM chunks that utilized a technique called temporal difference (TD) learning to perform reinforcement learning (Phillips). The Critic and Adaptive networks in the original toolkit would then perform the TD algorithm while the WM data structure was fed information throughout the course of a cognitive task. The TD algorithm would then mimic the dopamine release in the PFC by rewarding the WM system when a task was completed correctly. After enough successes, the working memory system has enough information to know the most valuable chunks to retain in working memory relative to the information that is given to it.

To use humans as an example, imagine a toddler in a high chair, frustrated and hungry. The father is teasing the child – a little girl - with a spoonful of her favorite apple sauce with sprinkles, repeating the word “dada” in baby-talk enough times to drive the mother out of the room for sanity’s sake. The toddler reaches for the spoon, tries to escape the high chair, and makes a fuss before trying a different tactic, but the father only feeds her until she echoes his plead of “dada.” Let’s take an inside look at this toddler’s working memory. The child is experiencing all sorts of input from the environment. Spoon with delicious food, high chair prison, dad’s weird faces, and the word “dada” are all things that we know the child is gathering from her senses. These are known as percepts. The child’s PFC now has to determine which of these percepts are important to hold on to. Let us imagine that the child has a WM capacity of one. Initially, she will be randomly selecting one of these percepts based on what her PFC calculates is the most valuable. If she was allowed to eat with a spoon prior to this encounter, this will most likely be the concept “spoon.” After several trials reaching for the spoon and still not finding success, her PFC tries something else, such as “high chair prison,” upon which she will start fussing and trying to escape. But one time she thinks “dada,” and some sounds resembling the word stumbles out of her. To this, she is rewarded with a spoonful of delicious apple sauce. After a variation of these steps occurs enough times, the poor girl’s PFC has gathered enough information to know that when dad is holding a spoonful of food hostage in front of her, the most valuable piece of information to hold in working memory is the word “dada.” When this happens, she will be much more successful at getting her food every time. At least until the lesson turns from Names to Manners. Then her PFC will have to retrain itself from the old code word “dada” to the new “please.”

# Symbolic and Distributed Encoding Methods

{Definitions and explanations of symbolic versus distributed encoding, and how each is currently used in the toolkit.}

# Holographic Reduced Representations

Holographic reduced representation is a robust method of representing symbolic concepts in a distributed form that can be combined to make holographic representations for complex concepts containing the information for each of the constituent concepts. With HRRs, it is possible to use symbolic concepts with ANNs. HRRs are mathematical structures composed of vectors of Guassian values that, when specific operations are performed on them, can very effectively be combined to form complex data structures from many HRRs that are reduced into a single vector. What makes HRRs so powerful is that new vectors formed from combinations of vectors are of the same size as the originals, and yet still contain information from each of the constituent vectors. This means that you can hold multiple layers of information inside a single vector. This is what makes it holographic.

An HRR is formed by generating a vector of real values typically drawn from a Normal distribution with mean zero, and standard deviation where n is the length of the vectors (figure 2). This isn’t what makes HRRs so versatile, as the vectors are rather ordinary in and of themselves. The power of HRRs comes from the operations you can perform on them. In the paper Tony Plate published which first proposed the concept of Holographic Reduced Representations, he described many incredible, powerful, and complex operations that can be done with HRRs. For the purpose of my project, I focused on the two most basic yet robust operations: circular convolution and circular correlation (also sometimes called circular involution).



Figure : An example HRR of length 6.

Circular convolution is the operation used to combine two vectors into a single vector of the same length. It is achieved by first calculating the matrix representing the outer product of the two vectors. Figure 3 shows an example of an outer product matrix formed from two vectors containing the values [2, -1, 1] and [1, 0, 2]. In this and other examples, I may show integer values as the contents of the vectors for simplification and ease of conceptualization. It is important to note that in real applications the values will be small real values drawn from a Gaussian distribution, as described above. Once an outer product has been formed, the circular convolution is calculated by summing each value across the matrix’s trans-diagonals. This is best illustrated in figure 4.



Figure : Summing across the trans-diagonals to calculate the circular convolution.



Figure : Forming the outer product matrix of two vectors of length 3.

Using circular convolution in this way, we can combine the vectors [2, -1, 1] and [1, 0, 2] to form the vector [0, 1, 5], a vector of the same size which contains information from both of the original vectors. To understand this conceptually, we can say that the first vector represents the concept “red” while the second represents the concept “ball,” and the resulting vector from the circular convolution represents the complex concept “red ball” (figure 5). Note that the new vector does not appear to have anything in common with the two constituent vectors. This is to our advantage, because a complex concept that is represented by two vectors is in fact, once combined, a new and unique concept in and of itself. Take for example the concept of the color red and the concept of ball. Individually, these concepts have nothing in common, but each has one element in common with the complex concept of a red ball. Nonetheless, a red ball is a unique concept in its own right. This is reflected in HRRs as well as the three vectors given above. This is another reason HRRs are so well suited for concept representation, because they mirror the distinctive nature of concepts, even when combined.



Figure : Convolving the concepts for "red" and "ball" yields the complex concept "red ball".

One may ask the question: if the HRR of complex concepts do not appear to have anything in common with their constituent HRRs, then how can they contain the information of those original vectors? The answer is in circular correlation. Circular correlation is the inverse operation of circular convolution. While convolving two vectors combines them into a single vector, correlating a complex vector with one of its constituent vectors will yield the other constituent vector. For example, if we have a vector representing the complex concept “red ball” and we correlate it with the vector representing the concept “red,” we get back the vector representing the concept “ball” (figure 6).



Figure : Correlating the complex concept "red ball" with constituent concept "red" yields the other constituent concept "ball".

The circular correlation operation is actually very easy to perform once you understand circular convolution. All that is required to achieve circular correlation is to convolve the vector representing the complex concept “red ball” with the *inverse* of the vector representing one of its constituent concepts, say, “red”. The approximate inverse of a vector is gained by reversing the order of each of its elements after the first. For example, the approximate inverse of the vector [1, 2, 3, 4] is [1, 4, 3, 2].

Unfortunately, if we were to try to extract the one of the constituent vectors from our convolution example from earlier using circular correlation, we would not get a vector that looked quite like our original. This is because using the convolution and correlation operations as described above with the approximate inverses of vectors will yield slightly distorted or noisy results. The math gets answers that are generally very close, but inexact. This problem decreases the larger your vector size. Had I used vectors of size 128 for my examples, my answers would have been much more precise, however you would likely have skipped the dreadful exercise. Precise answers can also be found by performing the operations in fourier space using fast fourier transforms, but that is an advanced method that Plate covers in his paper and is beyond the scope of this project. For our purposes, using large vectors with the operations as described above will suffice.

Another reason to use large vectors for the HRRs is related to the fact that the less number of elements you have, the more likely you are to randomly generate similar vectors for orthogonal concepts. In this case, the two vectors may appear to have something mathematically in common, when this is not the case. Using large vectors is a way to reduce the probability of this happening.

# Building the Holographic Reduced Representation Engine

The first step of incorporating HRRs into the WMtk was to build a software engine that would automate and handle the generation and manipulation of concept representations. Thus, I spent the spring semester of 2016 building the Holographic Reduced Representation Engine. The base data structure for this engine is a dictionary called Concept Memory, where the string name for the concept is the key and the HRR representing that concept is the value (figure 7). The engine’s concept memory keeps track of all concepts and their representations that the engine has encoded. 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. The term representation may be used interchangeably with HRR, since HRRs are the structure we use to form the digital representation of the concepts we will use.



Figure : Concept Memory stores all "known" concepts that have been encoded by the HRR Engine.

**Developing a Conjunctive Encoding Engine.** After setting up the base dictionary for the engine’s concept memoryand building the functionality for HRR generation, we needed to add aconjunctive encoding function to the engine for combining the HRRs toform complex concepts. We combine the representations for the conceptsusing the circular convolution operation.

Circular convolution is the key operation used in the main part of the HRRE’s 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 concepts for each of the previous and the combination ``ball\*big\*red'', but also the combinations ``ball\*big'', ``ball\*red'', and ``big\*red''.

It is important to note that we ensured that the HRRE always sorts the concepts into lexicographical order before working with them, to ensure that there are no duplicate representations made for the same concept. For example, we do not want the engine to see ``big\*red\*ball'', if ``ball\*red\*big'' exists. Otherwise, it could potentially perceive it as a new concept, and generate a new representation for it, even though it has a representation for ``ball\*red\*big'' that is the combination of the representations for its constituent concepts. There is an additional safeguard built into the engine, however, that protects against this as well. Whenever a concept is requested from the HRRE that it does not currently have in memory, it constructs it, splitting it apart into its constituent concepts by name, and constructs them 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 one constituent concept from a complex concept using circular involution - also sometimes called circular correlation. This is the inverse operation of circular convolution. Using circular involution, we can take the concept ``red\*ball'' and correlate it with ``ball'', and the resulting HRR would be the representation of ``red''. To perform the correlation operation, one must simply convolve the representation of the complex concept with the inverse of the representation of the constituent concept.

Similar to the encoding part of the engine’s 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.

When we finished the HRRE, it was capable of generating HRRs for new concepts, storing concepts as key-value pairs of names and representations in the concept memory dictionary, combining concepts through circular convolution, extracting concepts through circular correlation, constructing and encoding all combinations of a list of concepts, and unpacking all combinations of constituent concepts from a complex concept, and returning the resulting list to the user.

# 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 adevelopment plan for the augmented toolkit, (b) rebuilding the WMtkaround the HRRE, and (c) testing the augmented toolkit to ensure thatit still learns using the new HRR interface.

**Researching WMtk Specifications and Making a Development Plan.** The Working Memory toolkit is composed of a single-layer neuralnetwork that utilizes a working memory model inspired by the humanpre-frontal cortex (PFC). This network works by passing the chunks ofinformation in working memory and the state representation to a valuefunction, which determines how valuable that particular set of workingmemory contents is in that state. All states and combinations of WMcontents are equally meaningless at first, but the critic networkemploys temporal difference (TD) learning \cite{Sutton1998,OReilly2007}to learn the value of each WM-state combination by experiencingrepeated episodes of the learning task. In this way, theworking memory learns what information is the most valuable to retaindepending on what state it is currently seeing. At this point, it isup to the user to design their learning task in such a way that theagent decides to make an action according to what is currently held inworking memory.

We decided to start with a minimal design for our augmented toolkit. Since our aim for the project was to improve ease of use for researchers using the toolkit, providing a simple interface in addition to automating the concept encoding process was our best option, especially since more utilities can be added to the toolkit in future projects. As such, we decided to start the HWMtk with two main components: the Working Memory component, and the Critic Network (CN).

**Rebuilding the WMtk Around the HRRE.** Since we determined that our main components of the toolkit were going to be 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 HRR Engine, which serves as the store of known concepts, as well as the processor for the representations of all concepts. WM receives a string representation of the current state, which is set up as a string containing a concatenation of the concepts describing the state, delimited by a cross (``+''). 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 cross delimiter, then passes these 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, of which any are candidates for retention in WM. It is important to note that the previous contents of WM are also candidates for retention. This is what makes WM store task-relevant information in the long run, and thus what makes WM behave as it should. WM then goes through every combination of all candidate chunks that can fit in its WM slots, combining those with the representation of the state, and feeding each combination into the critic network to determine their value. The WM-state combination is calculated by convolving the WM contents into a single WM representation, then adding the representations for each concept in the state into a single state representation, and convolving the resulting two HRRs into the final WM-state representation. When passed into the critic, the set of WM 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 simulation run.

**Critic Network.** The CN component is the neural network that drives learning in the WMtk. It is passed representations from WM that it then passes through a 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 very small random values, and thus values for each representation will begin quite low. However, the CN employs TD-learning over many episodes of simulation, which will update the values in the weight vector, and thus make the value function converge 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 CN’s TD functions. Step then stores the current state’s 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 does the TD update for the previous state as well as the final state. Typically, all 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. When a new episode begins, these functions are called again, in the same order: Initialize Episode, a sequence of calls to Step, and finishing the episode with Absorb Reward. We do use eligibility traces in our TD calculations, and our epsilon soft policy is implemented by generating random WM contents, epsilon percent of the time.

A visual comparison of the basic architecture of the original and augmented toolkits is shown in Figure \ref{fig:hwmtk}. 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 and the symbols are automatically converted to appropriate vectors by the HRRE for presentation to the CN so that it learns to select task-relevant concepts that enable 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.

\begin{figure}

\includegraphics[width=85mm]{wmtk-architectures.pdf}

\caption{Comparison of the original WMtk architecture (left) to the

architecture of the HWMtk (right).}

\label{fig:hwmtk}

\end{figure}

# Testing the Holographic Working Memory Toolkit

We developed a task for the HWMtk to determine if theuser interface is indeed easier for developing new tasks compared to theoriginal toolkit. Additionally, the task tests the basic componentsof working memory function: learning to store task-relevantinformation and ignore task-irrelevant information (distractors). Forthis task, the agent isshown 7 colors in random order, and is rewarded if it remembers thecolor ``red'' at the end of the simulation. This task would beequivalent to shuffling 7 cards of different colors, and showing themall to the agent, one at a time, and asking at the end which color wewere thinking of. The task is simple, but not trivial, as the toolkitcan choose to remember nothing or any of the other colors as well.Also, the presentation order is randomized, so the agent cannotanticipate when the relevant color is being presented. The agent mustdecide to hold onto the color ``red'' and retain this concept inworking memory even while other colors (distractors) are beingpresented to the agent until the end of the episode is reached. Werepeat this process many times (each repetition being a singleepisode). The agent must learn that it is only rewarded upon rememberingred, regardless of presentation order or the number of distractorsencountered. This ability to retain task-relevant information in theface of competing distractions is the core mechanism of focusedattention 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, $\alpha$=0.1; future reward discounting factor, $\gamma$=0.9; past action eligibility factor, $\lambda$=0.1; epsilon-soft random working memory selection probability, $\epsilon$=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.

For the final task, we tested the HWMtk’s performance in the Delayed Saccade task. We chose this task because it was the main task that was used in the past to demonstrate the capability of the original WMtk. Because of this, we already had the DST source code, but more importantly, we were able to test both toolkits on the same task. This provided us with a strong comparison of performance between the original and augmented toolkits. We created a version of the DST that was wired to the HWMtk, replacing the DE-encoding functions in the old interface with simple function calls to the HWMtk, passing in a simple string representation for each state. We then run the DST 100 times on both toolkits, recording the number of episodes taken to learn the task to within a 98\% success rate within a window of 20 episodes.

Setting up the DST using the HWMtk proved simple compared to setting it up using the original toolkit. When the DST was written using the WMtk, the first parts of the task that had to be written were the DE/SE conversion functions that took the data structures describing the chunks of information and the state being passed to the toolkit and converted them into representations that the WMtk could use. An example of this kind of distributed coding is shown in Figure \ref{fig:dst-de-encoding}. Using the HWMtk, we merely had to pass a string containing that information to the WM object, which then handed it off to the HRRE for encoding into HRRs. We were able to immediately jump into writing the logic for the task without worrying about the overhead of manually writing DE conversion functions. Rather than write a reward function to hand to the toolkit, we were also able to write the performance logic directly into the task driver code, simply running a series of checks to see if the agent has been taking the proper actions, and calling Absorb Reward with the appropriate reward value upon completion of the episode. 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.

\includegraphics[width=85mm]{figures/dst-de-encoding.pdf}

\caption{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''.}

The final test to determine the outcome of our project was to run the task over 100 trials on each version of the toolkit, collecting performance data to determine whether or not the agent was learning. Our results here.

# Results and Discussion

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 learning task proved simple compared to settingup tasks using the original toolkit. Had we been using the originalWMtk, we would have had to write a function to create distributedrepresentations of each color as a chunk of information usable to WM,as well as a similar function for encoding the state, and a rewardfunction to check to provide a reward value according to the agent’sperformance. We would have had to write each of these before writingthe logic for the task itself, but using the augmented toolkit, noneof this preparation was necessary. We simply set up an array of ncolor strings, shuffled them at the beginning of each episode,initialized episode with the first color, called the Step functionwith each subsequent color less than n, and called the Absorb Rewardfunction with the nth color string. The only logic for the reward waswritten in line with the rest of the task, and it entailed a check tosee 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: (link provided in final draft).

**Effective Learning Using HRRs.** The final test to determine the outcome of our project was to run thetask for 100 trials and collect the data to determine whether or notthe agent was learning. We gathered information over every trial,keeping track of the number of episodes the agent successfullycompleted the task and recording the number of successes per every1000 episodes. We considered a 98 percent success rate per thousandepisodes an indication that the agent had effectively learned thetask. Over the 100 trials, we found that the agent learned the task toa 100 percent success rate within an average of 8000episodes. Therefore, the HWMtk meets the requirement of beingcapable of learning using holographic reduced representationsinstead of distributed representations for concepts.

**Discussions**

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, thus greatly reducing both the time and knowledge of ANNs needed to adequately set up those functions before writing the simulation. Specifically, the user no longer needs background knowledge on how to construct sparse, distributed, conjunctive codes, and does not need to rewrite encoding function when new concepts need to be proposed to WM or encoding 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 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 \ref{fig:dst-de-encoding}. 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.

# Bibliography

Baddeley, Alan. "Working Memory, Reading and Dyslexia." *Advances in Psychology* (1986): 141-52. Web.

Baddeley, Alan D., and Graham J. Hitch. "Working Memory." Psychology of Learning and Motivation (1974): 47-89. Web.

Braver, T. S., and J. D. Cohen. “On the control of control: The role of dopamine in regulating prefrontal function and working memory.” *Attention and Performance* volume 18, *Control of Cognitive Processes* (2000): 713-737. Web.

Cowan, Nelson. "The Magical Number 4 in Short-term Memory: A Reconsideration of Mental Storage Capacity." *Behavioral and Brain Sciences* 24.1 (2001): 87-114. Web.

Goldman-Rakic. P. S. “Circuitry of the prefrontal cortex and the regulation of behavior by representational knowledge.” *Handbook of Physiology* (1987): 373-417. Web.

Kriete, T., D. C. Noelle, J. D. Cohen, and R. C. O'reilly. "Indirection and Symbol-like Processing in the Prefrontal Cortex and Basal Ganglia." *Proceedings of the National Academy of Sciences* 110.41 (2013): 16390-6395. Web.

O'reilly, R. C., D. C. Noelle, T. S. Braver, and J. D. Cohen. "Prefrontal Cortex and Dynamic Categorization Tasks: Representational Organization and Neuromodulatory Control." *Cerebral Cortex* 12.3 (2002): 246-57. Web.

Phillips, Joshua L., and David C. Noelle. "A Biologically Inspired Working Memory Framework for Robots." ROMAN 2005. IEEE International Workshop on Robot and Human Interactive Communication, 2005. (2005): 599-604. Web.

Plate, Tony A. "Holographic Reduced Representations." *IEEE Transactions on Neural Networks* 6.3 (1995): 623-41. Web.

Shultz, W., P. Dayan, and P. R. Montague. “Learning to predict by the methods of temporal differences.” *Machine Learning* 3 (1988): 9-44. Web.

Waugh, Nancy C., and Donald A. Norman. "Primary Memory." *Psychological Review* 72.2 (1965): 89-104. Web.

# Table of Figures

[Figure 1: An example of how working memory stores task-relevant information. 4](file:///C:\Users\Grayson\Documents\Holographic-Reduced-Representations-for-Working-Memory-Concept-Encoding\HRRfWMCE_Honors-Thesis.docx#_Toc463829332)

[Figure 2: An example HRR of length 6. 9](file:///C:\Users\Grayson\Documents\Holographic-Reduced-Representations-for-Working-Memory-Concept-Encoding\HRRfWMCE_Honors-Thesis.docx#_Toc463829333)

[Figure 3: Summing across the trans-diagonals to calculate the circular convolution. 10](file:///C:\Users\Grayson\Documents\Holographic-Reduced-Representations-for-Working-Memory-Concept-Encoding\HRRfWMCE_Honors-Thesis.docx#_Toc463829334)

[Figure 4: Forming the outer product matrix of two vectors of length 3. 10](file:///C:\Users\Grayson\Documents\Holographic-Reduced-Representations-for-Working-Memory-Concept-Encoding\HRRfWMCE_Honors-Thesis.docx#_Toc463829335)

[Figure 5: Convolving the concepts for "red" and "ball" yields the complex concept "red ball". 10](file:///C:\Users\Grayson\Documents\Holographic-Reduced-Representations-for-Working-Memory-Concept-Encoding\HRRfWMCE_Honors-Thesis.docx#_Toc463829336)

[Figure 6: Correlating the complex concept "red ball" with constituent concept "red" yields the other constituent concept "ball". 11](file:///C:\Users\Grayson\Documents\Holographic-Reduced-Representations-for-Working-Memory-Concept-Encoding\HRRfWMCE_Honors-Thesis.docx#_Toc463829337)

[Figure 7: Concept Memory stores all "known" concepts that have been encoded by the HRR Engine. 13](file:///C:\Users\Grayson\Documents\Holographic-Reduced-Representations-for-Working-Memory-Concept-Encoding\HRRfWMCE_Honors-Thesis.docx#_Toc463829338)

# Appendix

ANN – Artificial Neural Network

AI – Aritificial Intelligence

CN – Critic Network

DE – Distributed Encoding

DST – Delayed Saccade Task

HRR – Holographic Reduced Representation

HRRE – Holographic Reduced Representation Engine

HWMtk – Holographic Working Memory toolkit

SE – Symbolic Encoding

TD – Temporal Difference

WM – Working Memory

WMtk – Working Memory toolkit