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

Dr. Teresa Davis
Psychology
Honors Council Representative

Dr. John Vile
Dean, University Honors College

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 me and other students a friendly and welcoming community of scholars and friends. Their opportunities help students like me 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 scientist, 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 serving 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

Introduction

The field of artificial intelligence (AI) is synergistic with a wide range of disciplines but artificial neural networks 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, most have limited application in other areas of AI where symbolic encoding 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 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, 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, 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).

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, 1995) 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 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 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 that is compatible with the distributed representations expected by neural network architectures, but is 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

Many people have tried to remember a phone number for a friend as they quickly spouted it off. "1-1-2-3-5-8-1-3-2-1," says the friend, and we struggle to remember it long enough for them to pick up the phone and to dial it. Were we to attempt to remember it now, we 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 we tried to remember this number right now, our brains would probably separate it into smaller groups of digits that are easier to remember. This gives us the "112 – 358 – 1321" that we are so familiar with. 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.

Why 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 points to remember about working memory. First, working memory can only hold small pieces of information long enough to be used, then immediately discards them. Second, this information is often grouped together in chunks of information that is similar in nature. Finally, there are only a few slots in WM in 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, 2004). Keeping these three things in mind, then, it makes sense that when trying to remember a ten-digit phone number, the

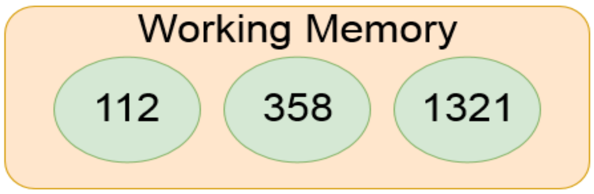


Figure 1: An example of how working memory stores task-relevant information.

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.

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, 1987). The basal ganglia (BG) also play a role the working memory system by regulating the information held by the PFC through regulation of dopamine release. Dopamine is the neurotransmitter that acts as a sort of reward chemical which the BG releases when exposed to rewarding stimuli (Shultz et al, 1988). 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, his or her 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, 1988). 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 BG regulates reinforcement learning through dopamine release while the PFC 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 that held representations of WM chunks and utilized a technique called temporal difference (TD) learning to perform reinforcement learning (Phillips and Noelle, 2005). 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 BG 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 when 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 retain. Imagine that the child has a WM capacity of one. Initially, she will be randomly selecting one of these percepts based on what her WM calculates is the most valuable. If she were 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. For this, she is rewarded with a spoonful of delicious apple sauce, and her BG will release the dopamine that tells her PFC that something that it is holding onto is good. After a variation of these steps occurs enough times, the girl's PFC and BG have 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 WM will have to retrain itself from the old code word "dada" to the new "please."

Symbolic and Distributed Encoding Methods

Before talking about holographic reduced representations and discussing the ways in which they are well-suited for concept encoding in the working memory toolkit, it is worth defining what we mean by symbolic and distributed representations. Here I will discuss how to encode concepts in AI and cognitive tasks using both symbolic and distributed formalisms, and I will conclude with a discussion on what DE/SE conversion may look like, especially in the WMtk. This chapter is specifically devoted to beginning the discussion of constructing concept representations for software use – a topic that is vital to understanding HRRs and how they work – as well as why the original DE/SE interface was complicated and difficult to use.

Symbolic representation is the simplest to understand, since this is how our brains represent concepts. A symbolic representation of a concept is exactly what it sounds like: a symbol that describes that concept. To represent a concept symbolically in a cognitive task, we would merely come up with some symbol and informally assign it the concept that we want it to represent. For example, if we wanted to represent the concept of a triangle, we could draw a picture of a closed three-sided figure, and our brains will interpret he image as the symbol representing a triangle. We could just as easily say that the word "triangle" itself is a symbol that represents the concept of a triangle, or even just the letter "T" (figure 2).

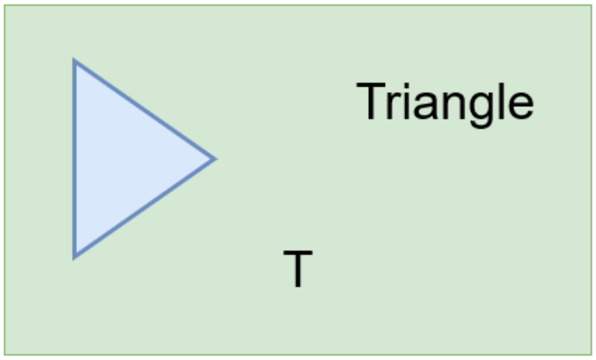


Figure 2: Different symbols representing the concept "triangle".

In the delayed saccade task mentioned earlier, we can use some symbols to represent the concepts used in the task. We will use words as our symbols to represent the concepts. We need a symbol to represent the cross' presence in the center of the screen, so we will use "center cross.” We also need a focus symbol to indicate where the agent is looking. This can be “center focus”, “northeast focus”, “northwest focus”, “southeast focus”, or “southwest focus”. Finally, we need a target symbol to indicate where the target is on the screen, such as “northeast target”, “northwest target”, “southeast target”, or “southwest target”. Visual examples of these symbols are shown in figure 3.

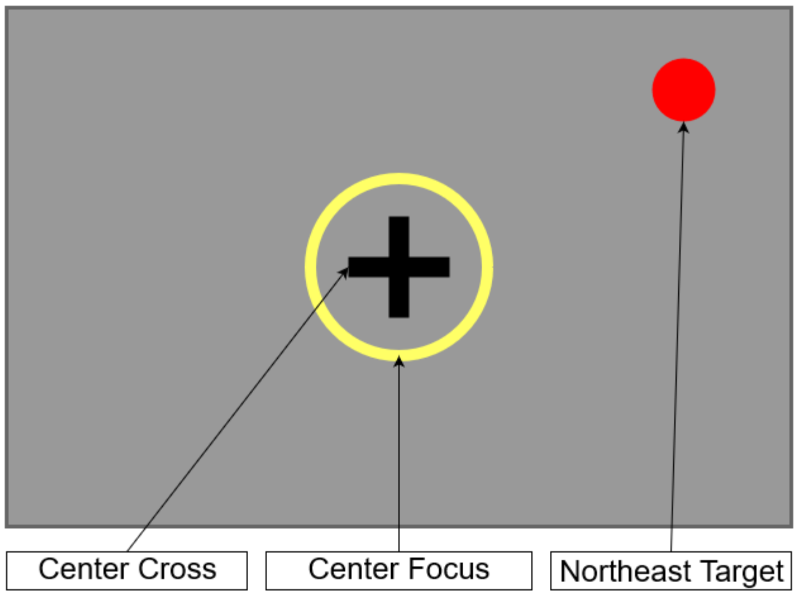


Figure 3: Visual examples of symbols used in the DST.

Distributed encoding is a method of encoding information about concepts in ways that can be expressed in mathematical terms. Where symbolic representations are good in AI tasks concerning logical decision making (in the presence of the symbol for the northeast target, do X), distributed representations are used in artificial neural networks that perform mathematical operations for decision making and learning. A distributed representation is any representation of a concept or set of concepts that can be expressed as number values in a vector. Say we wanted to represent a triangle using distributed representations, how we constructed our DE vector would depend largely on the other parameters for the task. For example, if there were four different shapes that we could represent, then we could construct a simple vector of four values, where each index was a sort of binary switch for the different shapes. Let’s suppose the indices were assigned as follows: 1 - square, 2 - triangle, 3 - circle, and 4 - cross. The vector

[0, 1, 0, 0] is a distributed representation indicating the presence of a triangle, because the index representing triangle (index 2) contains a 1.

Suppose we have four other concepts for the colors red, green, blue, and yellow. If we wanted to create a distributed representation to represent the complex concepts of colored shapes, we would extend the shape vector and assign colors to the other indices (figure 4). Using this new encoding for colored shapes, we can use the vector [0, 1, 0, 0, 0, 1, 0, 0] to represent the concept of a green triangle.

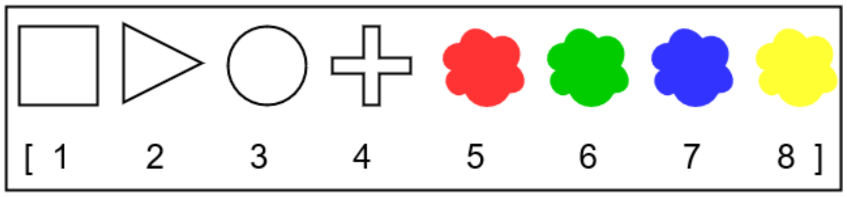


Figure 4: Making distributed encodings by assigning symbols to each index of a vector.

In the original working memory toolkit, the user developing a cognitive task not only had to write logic to handle symbolic concepts in the task environment, but also had to write functions and methods that would convert these symbolic concepts such as “red triangle” into distributed vectors such as [0, 1, 0, 0, 0, 1, 0, 0] so that the ANN embedded within the toolkit could properly learn using the TD algorithms. Unfortunately, this is not all that has to be taken into account. The encoding for colored shapes described above makes disjunctive representations, meaning that two vectors representing unique concepts can sometimes contain similarities. This means that the concepts red triangle and blue triangle appear to be similar because only one element is different in their vector representations, [0, 1, 0, 0, 1, 0, 0, 0] and [0, 1, 0, 0, 0, 0, 1, 0], respectively. Mathematically speaking, these disjunctive representations may not be orthogonal for orthogonal (or independently unique) concepts, since their dot product is higher than it would be if the vectors were truly unique. The reason this is bad is because TD learning generally only works on sparse conjunctive distributed representations.

It is important to note that when I say sparse here, I am talking in terms of sparsity of dot product. Orthogonal concepts must be represented by orthogonal vectors in order for TD learning to occur.

It is not immediately apparent why sparse conjunctive distributed representations must be used for neural nets to learn effectively using TD learning. The reason is related to how the system draws relationships between the concepts represented in the task. Remember that the vectors representing green ball and green triangle are very similar using disjunctive representations, since they both share the “green” element in their vector representations, and the only difference is the shape element that is activated in the vector. The reason we call these vectors disjunctive is because their dot products show more similarity for vectors with common features, even though the vectors may represent unique concepts themselves. Since the dot product shows similarity between two vectors, this is like saying they can either both share the same color **or** they can share the same shape **or** they can share nothing (a disjunction). Vectors built from disjunctive encodings will appear to the neural network to be fairly similar to other vectors which share common information, even though they represent independently unique concepts, such as “green ball” and “green triangle.” Even though the two concepts both describe something that is green, they are still unique concepts and must be considered orthogonal. Justin Boyan and Andrew Moore from Carnegie Mellon University propose that because of this, neural networks generally learn poorly using TD learning methods, but Richard Sutton argues that this is not the case when using sparse conjunctive distributed representations.

Conjunctive distributed representations are formed when the vectors representing orthogonal concepts are orthogonal vectors themselves, even if they contain some similar constituent pieces. For example, the vector for “green square” should not be similar in any way to the vector for “green circle”, even though they both have the constituent concept “green”. A naïve way to encode conjunctive representations is to create a matrix of *n* dimensions, where *n* is the number of constituent pieces that make up a concept (i.e., 3 dimensions for a conjunctive concept made up of a shape, a color, and a size) and a single element of 1 for the location within that matrix that represents each of the concepts on the axes that you wish. Figure 5 shows an example of a matrix that forms a conjunctive representation for “green circle” where there are two dimensions of input – shape and color – and two possible choices for each – square, circle, and red, green. We can also rearrange this into a vector form by assigning each index a unique combination of each constituent concept. From the previous example, we could assign the following indices: 1 - red circle, 2 - red square, 3 - green circle, and 4 - green square. Using this encoding, the vector [0, 0, 1, 0] represents a green circle and [0, 0, 0, 1] represents a green square. This encoding is conjunctive because every possible combination of concepts has a vector representation that is orthogonal to the others.

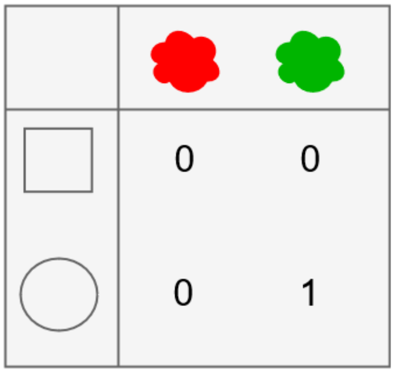


Figure 5: Using a conjunctive matrix to encode the concept for "green circle".

There are, of course, several complications and disadvantages to using conjunctive vectors constructed in this way. Perhaps most obvious is that the conjunctive vectors described above do not include individual concepts such as “green” or “circle,” and must therefore be intentionally included in the manual construction of the representation. Also, if we wanted to add a concept to our encoding scheme, we would have to take into account every possible combination of concepts that can be constructed with that concept, and include those in our manual encoding scheme. Because of this, every concept added to the encoding scheme increases the number of possible elements in the conjunctive vectors exponentially, meaning that this method of encoding is only really efficient for small conjunctive vectors.

These are only the basic factors one has to take into account when trying to develop encoding methods for representing symbolic concepts using sparse conjunctive distributed vectors. However, they are all necessary to know in order to write functions to encode symbolic concepts for a simple learning task using the original WMtk. Unless the user has a strong understanding of how representations need to be set up for learning with ANNs, they will meet much difficulty in manually writing the DE/SE conversion for their learning task. Even for those who do have the knowledge, it is a very tedious and intimidating job to have done before they can even begin writing the main logic for their cognitive task.

I hope that by this point I have demonstrated that the original toolkit's DE/SE interface – where the conversion is manually written by the user – is very complicated and difficult to use, especially to those with little knowledge of ANN-based systems using TD-learning elements. Having explained the complications and difficulties of the original toolkit's manual DE/SE encoding requirements, I can talk about a powerful method of representation that allowed me to automate the DE/SE conversion, and replace the original manual-encoding interface with a simpler SE string passing interface that is much more user-friendly. This method is called holographic reduced representation.

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 (Plate 1995). 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 a single vector can hold multiple layers of information – making them holographic.

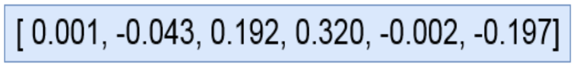
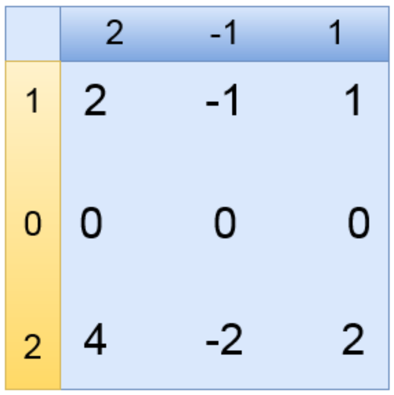

[0.001, -0.043, 0.192, 0.320, -0.002, -0.197]

Figure 6: An example HRR of length 6.

An HRR is formed by generating a vector of real values typically drawn from a Normal distribution with mean zero, and standard deviation $1/\sqrt{n}$ where n is the length of the vectors (figure 6). 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 can be performed 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).

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 7 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 8.



	2	-1	1
1	2	-1	1
0	0	0	0
2	4	-2	2

Figure 7: Forming the outer product matrix of two vectors of length 3.

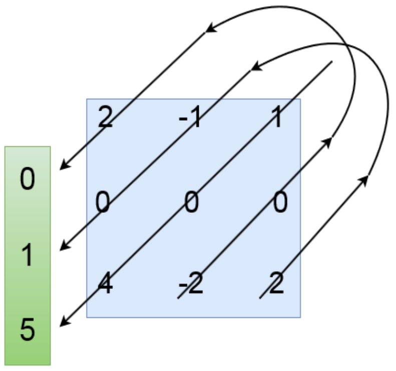


Figure 8: Summing across the trans-diagonals to calculate the circular convolution.

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 9). 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

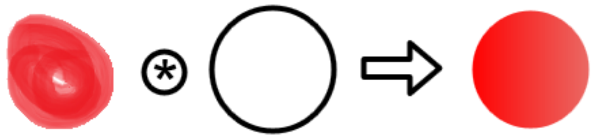


Figure 9: Convolving the concepts for "red" and "ball" yields the complex concept "red ball".

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.

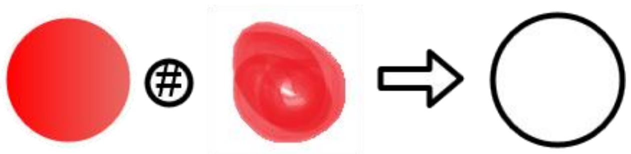


Figure 10: Correlating the complex concept "red ball" with constituent concept "red" yields the other constituent concept "ball".

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 10).

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 the exercise would have been dreadfully long and tedious. 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 fewer elements in the vectors increases the odds of randomly generating similar vectors for orthogonal concepts. In this case, the two vectors may appear to have something mathematically in common, when they might represent independent concepts. 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 11). The engine's concept memory keeps track of all concept-representation pairs that the engine has encoded. From this point forward, the term concept will be used to refer to the entity composed of a symbolic string value and its associated HRR. 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.

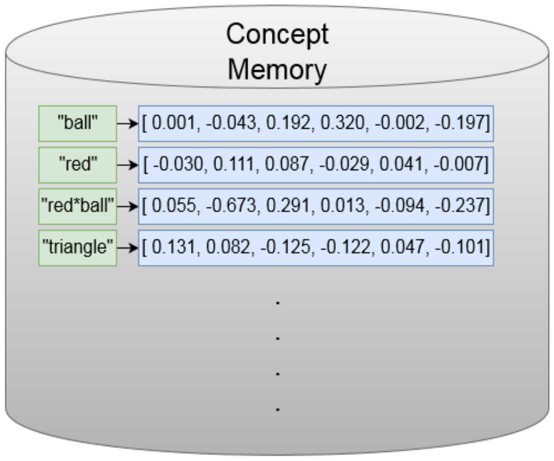


Figure 11: Concept Memory stores all "known" concepts that have been encoded by the HRR Engine.

Developing a Conjunctive Encoding Engine. After setting up the base dictionary for the engine's concept 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 into conjunctive representations using the circular convolution operation.

The main part of the HRRE's conjunctive encoding functionality is 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 in lexicographic 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.” In addition to constructing the final combination of all concepts, the engine 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 worth noting 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 already contains “ball*red*big,” which is represented by the same HRR. 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 by splitting it apart into its constituent concepts by name, and constructing each 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 correlation.

Similar to the encoding part of the engine’s construct function, the decoding part of the engine has an unpack function that separates 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 in the environment will be assembled as candidates for storing in WM.

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

Once the HRR Engine was finished, my next step was to rebuild the WMtk with the engine at its core. The three-part process for building the Holographic WMtk 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.

Researching WMtk Specifications and Making a Development Plan. The Working Memory toolkit is composed of a single-layer neural network that utilizes a working memory model inspired by the human pre-frontal cortex. This network works by passing the chunks of information in working memory and the state representation to a value function, which determines how valuable that particular set of working memory contents is in that state. All states and combinations of WM contents are equally meaningless at first, but the critic network employs temporal difference (TD) learning (Sutton, 1998, O'Reilly, 2007) to learn the 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 depending on what state it is currently seeing. At this point, it is up to the user to design 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 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).

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, from 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.

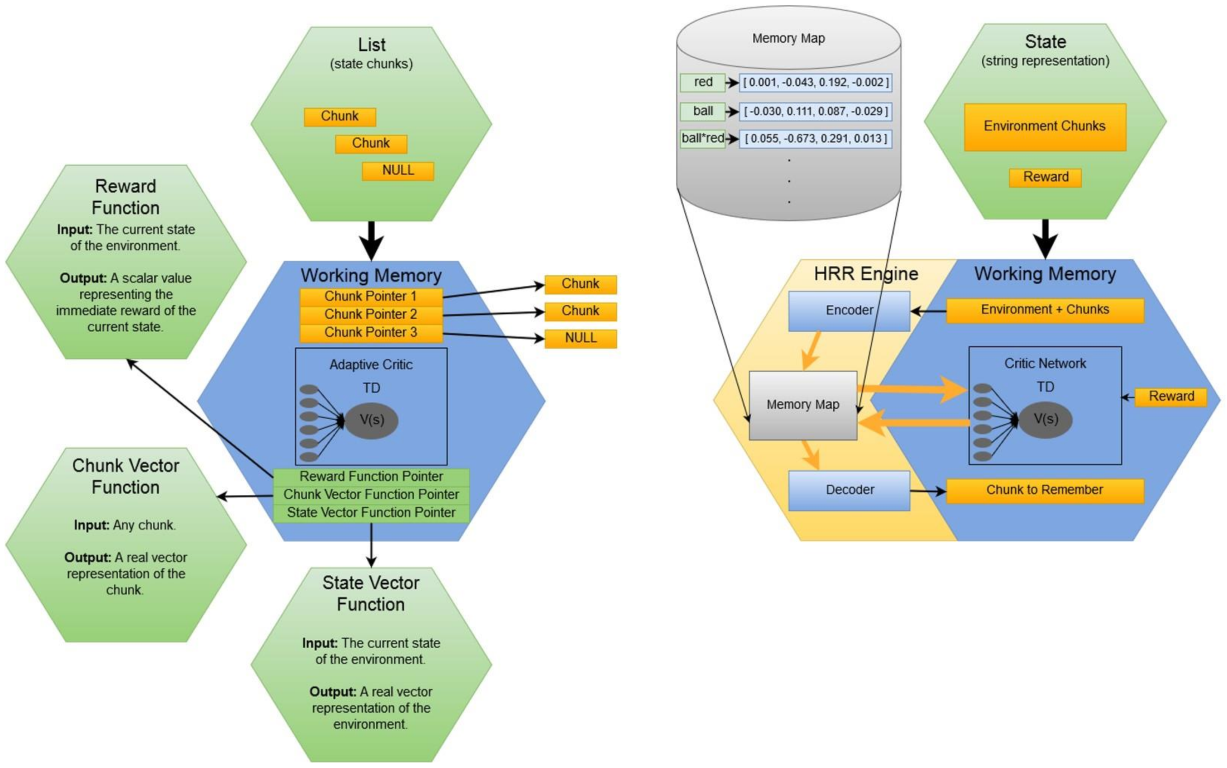


Figure 12: Comparison of the original WMtk architecture (left) to the architecture of the HWMtk (right).

A visual comparison of the basic architecture of the original and augmented toolkits is shown in figure 12. 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 pre-potent 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 Holographic Working Memory Toolkit

We developed a task for the HWMtk to determine if the user interface is indeed easier for developing new tasks compared to the original toolkit. Additionally, the task tests the basic components of working memory function: learning to store task-relevant information and ignore task-irrelevant information (distractors). For this 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 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, $\alpha = 0.1$; future reward discounting factor, $\gamma = 0.9$;

past action eligibility factor, λ = 0.1; epsilon-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 noticeable difference in learning behavior. 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.

We additionally have developed a test in the statistical language R that serves as further proof of concept for our research. This test uses the same constructs and processes as the HWMtk, making it a valid proof of concept that we plan on implementing using the full toolkit in the future. We chose to use the Wisconsin Card Sort task (WCST) – a task well suited for testing cognitive models of PFC function (Rougier et al., 2005). In our version of the WCST, a deck of cards containing objects is generated. These objects are described by 2 dimensions each with three features. A dimension would be something like color or shape and a feature would be something like green in the color dimension, and square in the shape dimension. Thus sample cards might be something like a green square, a blue triangle, or a red circle, and the deck would contain all permutations of these features per dimension. We chose a dimension-feature design for our cards for scalability. It is easier to increase the number of dimensions and features to create complex tasks than explicitly creating every permutation of the cards to add another dimension – such as the number of each shape present on each card.

At the start of the task, a rule is selected. This rule would be a single feature from one of the dimensions, e.g. blue or triangle. The agent is then shown a random card from the deck and is required to place it on one of two piles. The “match” pile is where the agent should choose to place the card if the card contains the feature specified by the rule, otherwise it should place it on the “discard” pile. Since the rule is not part of the agent’s knowledge base, it makes its decision based on the feature it decides to store in WM. A reward of 1 is given if the agent chose correctly, otherwise a reward of 0. Either way, the task is repeated episodically for 1M trials. Eventually, the agent’s WM will learn that the most valuable feature to remember is that which matches the rule, and it will always choose the correct pile for any card that it is shown.

We know that the agent has correctly learned the rule if it has chosen the correct pile for 100 contiguous cards, at which point we generate a new rule and record the number of trials since the previous switch, which we call the switch time. We found that for 1M trials, the median switch time was 3883.5 ± 327.1163, meaning it typically took the agent between 3500 and 4200 trials to correctly learn a new rule.

The parameters for the WCST are as follows: CN learning rate parameter, $\alpha = 0.9$; future reward discounting factor, $\gamma = 0.5$; past action eligibility factor, $\lambda = 0.1$; epsilon-soft random working memory selection probability, $\varepsilon = 0.05$; number of working memory slots, $s = 1$; and HRR vector length, $n = 1024$.

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 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 agent's 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 were, 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.

Effective Learning Using HRRs. The final test to determine the outcome of our project was to run the task for 100 trials and collect the data to determine whether or not the agent was learning. 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 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 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 hierarchical structure, tasks which require paying attention to hierarchical 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

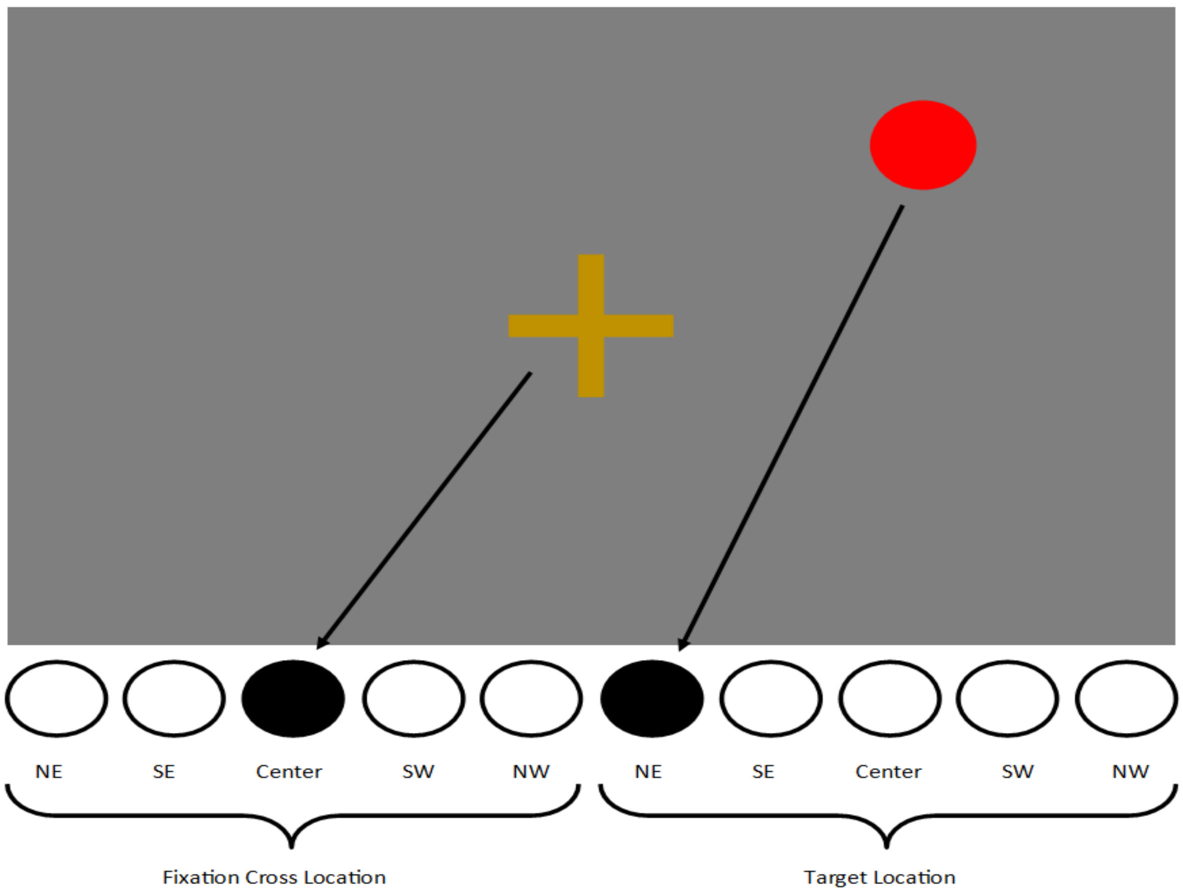


Figure 13: Example of how task-specific, sparse, distributed encoding was performed in the original WMtk. In the HWMtk, an appropriate distributed HRR representation can be built automatically without the user's aid from a symbolic description of the environment: "center*cross+northeast*target."

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 13. Second, the ability to rehearse and group items using convolution might be added to tackle tasks what 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.

Boyan, Justin A., and Andrew W. Moore. "Generalization in Reinforcement Learning: Safely Approximating the Value Function." Neural Information Processing Systems 7 (1995): n. pag. 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.

Rougier, N. P., D. C. Noelle, T. S. Braver, J. D. Cohen, and R. C. O'reilly. "Prefrontal Cortex and Flexible Cognitive Control: Rules without Symbols." *Proceedings of the National Academy of Sciences* 102.20 (2005): 7338-343. 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.

Sutton, Richard S. "Generalization in Reinforcement Learning: Successful Examples Using Sparse Coarse Coding." Neural Information Processing Systems 8 (1996): n. pag. Web.

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

Table of Figures

Appendix

ANN – Artificial Neural Network

AI – Artificial 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

WCST – Wisconsin Card Sort Task

WM – Working Memory

WMtk – Working Memory toolkit