Aspects of Hyperdimensional Computing for Robotics: Transfer Learning, Cloning, Extraneous Sensors, and Network Topology

Nathan McDonald, Richard Davis, Lisa Loomis
Air Force Research Laboratory, Information Directorate
Rome, NY USA
Nathan.McDonald.5@us.af.mil

Johan Kopra
University of Turku
Turku, Finland

***Abstract*— Hyperdimensional computing (HDC) is a type of machine learning algorithm but is not based on the ubiquitous artificial neural network (ANN) paradigm. Instead of neurons and synapses, HDC implements online learning via very large vectors manipulated to represent correlations among the various vectors, measured by a similarity metric. Yet this approach readily affords one-shot learning, transfer learning, and native error correction, which are standing challenges for traditional ANNs. Further, implementations using binary vectors {0,1} are particularly attractive for size, weight, and power (SWaP) constrained systems, particularly disposable robotics.**

The paper is the first to identify and formalize a method to completely clone trained hyperdimensional behavior vectors. Using shift maps, *d*-1 unique clones can be made from a parent vector of length *d*. Additionally, expeditionary robots with extraneous sensors were trained via HDC to solve a maze even when up to 75% of the sensors fed irrelevant data to the robot. Lastly, we demonstrated the resiliency of this encoding method to random bit flips and how different network topologies contribute to dynamic reprogramming of HDC robots. HDC is presented here though not to replace ANNs but to encourage integration of these complementary ML paradigms.

Keywords—hyperdimensional computing (HDC), hyperdimensional binary vectors (HBV), elementary cellular automata (ECA), swarm, robotics, network topology

I. Introduction

While artificial neural networks (ANN) are deservedly popular in the machine learning community, they do not efficiently map to traditional von Neumann computing architectures with respect to size, weight, and power (SWaP). Numerous floating-point matrix multiplications and frequently long training times incur computational costs unacceptable for resource constrained expeditionary robots. Alternative associative learning methods are therefore of increasing interest.

Hyperdimensional computing (HDC) employs very long "hyperdimensional" binary vectors (HBV) to represent symbols or concepts, where simple mathematical operations such as addition, multiplication, and permutation subsequently encode and decode associations between these concepts. The iconic example is to query "What is the dollar of Mexico?" from a database encoded via HDC [1].

HBVs comprised of binary elements {0,1} can be readily implemented in digital electronics [2, 3]. Recently, there have been a series of articles applying HDC to robotics, e.g. solving mazes [4] [5], "active perception" and multi-sensor "memory" [3], recognition of objects from multiple angles [6], and sequence processing for place recognition [6].

This work explored several areas particularly relevant for expeditionary robotics. Such robots are frequently trained in one environment but then tested against novel inputs in a novel environment. We first expand upon the foraging honeybee model [7] to demonstrate transfer of learned associations against partial matches.

After training one robot, its "experience" or learned memory may be trivially replicated across multiple robots. However, simply copying these memory vectors may make all robots vulnerable to the same attack by a malicious entity [8]. The challenge is thus to replicate the parent's set of HBVs such that all learned associations are preserved yet the clone's HBVs are approximately uncorrelated (orthogonal) with the parent's.

In practice, environments cannot be exhaustively defined beforehand, so an expeditionary robot is afforded numerous sensors with which it must learn associations between sensor patterns in an environment and rewarded (penalized) actions. There are cases then where only subsets of the available sensors are relevant for solving a given task. This work explored different encoding schemes to mitigate negative effects from extraneous sensor input.

Lastly, swarm control may include dynamically reprogramming behavior in the field. Several network topologies were evaluated demonstrating the rapidity of HDC vector convergence even over severely degraded channels.

The key contributions of this work are as follows:

a) We evaluated all 256 elementary cellular automata (ECA) rules for cloning HBVs and identified 8 rules that satisfied all of the replication requirements identified in [8], eliminating the need for a separate connection matrix. To the best of the authors' knowledge, this is the first report of complete and orthogonal replication of HBVs using ECA.

b) We identified shift maps as the underlying operation consistent across all 8 ECA rules, then mathematically proved why these maps alone satisfy the cloning requirements.

c) For scenarios where only a subset of sensors are relevant for action selection, we identified a training method resilient to the extraneous sensor input.

DISTRIBUTION STATEMENT A. Approved for public release: distribution unlimited: AFRL-2021-0158

d) We demonstrated the efficiency of dense networks for dynamically reprogramming HDC-based robot swarms via transmission of HBVs over noisy channels; however, it is critical that the first reprogrammed node not be iteratively updated.

Due to the involved nature of each task, the mathematics of HDC are described in detail in the Background but afterwards each task will contain its own Methods, Results, and Discussion.

II. Background

There are a variety of ways to generate vectors for HDC [9-14]. The Binary Spatter Codes used here involve very long vectors of length d (typically 10^4) comprised of random {0, 1} in roughly equal proportion [1]. The similarity between any two such random vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ was measured according to their normalized Hamming distance (HD), i.e. the fraction of non-identical bits:

$$HD(\boldsymbol{a}, \boldsymbol{b}) = \sum(\boldsymbol{a} \otimes \boldsymbol{b}) \,/d, \tag{1}$$

where ⊗ denotes XOR. For sufficiently large d, the Hamming distance between any two random vectors converges to 0.5, so an HD ≠ 0.5 can be used to represent learned associations among HBVs.

Note, HBVs for which HD ≈ 0.5 are said to be maximally uncorrelated, or approximately orthogonal. By this definition, the HD between a vector and itself is 0, HD($\boldsymbol{a}$, $\boldsymbol{a}$) = 0; and an HD > 0.5 indicates anti-correlation, where HD($\boldsymbol{a}$, NOT($\boldsymbol{a}$)) = 1.

Creating associative memories among HBVs typically involves three operations: addition (bundling), multiplication (binding), and cyclic shifting. Each of these operations effects the HD between the individual item HBVs in the dictionary and the resulting compositional HBVs (Table I).

TABLE I. Hamming distance between item vectors and compositional vectors

HD	$\boldsymbol{a}$	$\boldsymbol{b}$	$\boldsymbol{a}\otimes\boldsymbol{b}$	$[\boldsymbol{a}+\boldsymbol{b}+\boldsymbol{c}]$	Sh($\boldsymbol{a}$, j)
$\boldsymbol{a}$	0	0.5	0.5	0.25	0.5
$\boldsymbol{b}$	0.5	0	0.5	0.25	0.5
$\boldsymbol{a}\otimes\boldsymbol{b}$	0.5	0.5	0	0.5	0.5
$[\boldsymbol{a}+\boldsymbol{b}+\boldsymbol{c}]$	0.25	0.25	0.5	0	0.5
Sh($\boldsymbol{a}$, j)	0.5	0.5	0.5	0.5	0

Bundling is a majority bit operation over a set of vectors (Fig. 1), denoted as $[\boldsymbol{a} + \boldsymbol{b} + \boldsymbol{c}]$. The resulting compositional vector will be very similar (HD << 0.5) to the vectors added together. In practice, if bundling an even number of vectors, a random HBV is added to the set of vectors to create a majority condition. Bundling is only approximately reversible.

Binding is implemented as bitwise XOR, again denoted as ⊗. Notably, the XOR operation is reversible and is analogous therefore to the creation of a dictionary entry, with a key and value pair. For example, consider the following equation:

$\boldsymbol{a}$	0	**1**	0	0	0	**1**	**1**	**1**
$\boldsymbol{b}$	0	0	**1**	0	**1**	0	**1**	**1**
$\boldsymbol{c}$	0	0	0	**1**	**1**	**1**	0	**1**
$\boldsymbol{d}$	0	0	0	0	**1**	**1**	**1**	**1**

Fig. 1 Bundling of vectors $\boldsymbol{a}$, $\boldsymbol{b}$, and $\boldsymbol{c}$, $[\boldsymbol{a} + \boldsymbol{b} + \boldsymbol{c}] = \boldsymbol{d}$, where $\boldsymbol{d}$ is calculated as the majority bit at each position along the component vectors.

$$[\boldsymbol{a}\otimes\boldsymbol{b} + \boldsymbol{c}\otimes\boldsymbol{d}] = \boldsymbol{e}. \tag{2}$$

We can later recall the value associated with $\boldsymbol{b}$ by applying the XOR operation again,

$$\begin{aligned} \boldsymbol{b}\otimes\boldsymbol{e} &= \boldsymbol{b}\otimes[\boldsymbol{a}\otimes\boldsymbol{b} + \boldsymbol{c}\otimes\boldsymbol{d}] \\ &= [\boldsymbol{a} + \boldsymbol{b}\otimes\boldsymbol{c}\otimes\boldsymbol{d}] \\ \hat{\boldsymbol{a}} &= [\boldsymbol{a} + noise], \end{aligned} \tag{3}$$

where $\boldsymbol{b}\otimes\boldsymbol{b}$ cancels itself out and $\boldsymbol{b}\otimes\boldsymbol{c}\otimes\boldsymbol{d}$ is treated as a random HBV, that is noise. But one more step is required.

By using such large, randomly defined vectors, information is spread equally across the entire vector representation, often termed "holographic computing." That is, unlike binary encoding with a most (least) significant bit, in HDC, each bit is equally (in)significant. It is this property that allows one to successfully perform HDC operations on severely degraded vectors. The final "cleanup" step is performed by comparing the Hamming distances between this noisy $\hat{\boldsymbol{a}}$ and the HBV dictionary vectors $\boldsymbol{a}$, $\boldsymbol{b}$, $\boldsymbol{c}$, and $\boldsymbol{d}$. In this case, the smallest Hamming distance will be found with respect to $\boldsymbol{a}$.

These two operations are commutative, associative, and distributive; but there will be scenarios where we require discrimination among vectors combinations. The final operation is permutation, or rotation of the coordinates of the hypervector. This is often implemented as cyclic shift, whereby vector $\boldsymbol{a}$ is shifted by j elements, denoted as Π_j, For example,

$$\begin{aligned} (\boldsymbol{a}\otimes\boldsymbol{b})\otimes(\boldsymbol{c}\otimes\boldsymbol{d}) &= (\boldsymbol{a}\otimes\boldsymbol{d})\otimes(\boldsymbol{b}\otimes\boldsymbol{c}) \\ (\boldsymbol{a}\otimes\boldsymbol{b})\otimes \Pi_1(\boldsymbol{c}\otimes\boldsymbol{d}) &\neq (\boldsymbol{a}\otimes\boldsymbol{d})\otimes(\boldsymbol{b}\otimes\boldsymbol{c}) \end{aligned} \tag{4}$$

III. Transfer Learning

A. Method

Introduced as mini-game in [7], the foraging honey bee model is a simple demonstration of learning associations between physical shapes and/or positional relationships with a *Reward* vector. While a *Pain* vector was discussed, it was not actively used in their example. This work expands the original model to incorporate both a R*eward* and *Pain* vector. Further, we explored the effect of the scene encoding scheme upon the Hamming distance range.

Parent compositional HBVs were created via online learning, where scenes for 2 of 3 flower species were rewarded and 1 of 3 color hues were penalized. Dictionary vectors were *flower* (1x) and *color* (1x), with values *species* (3x) and *hue* (3x), respectively. For example, the first *hue* vector was used to represent "red," the second "blue," and the third "yellow." Feedback vectors were *PAIN* (1x) and *REWARD* (1x).

Each scene was encoded as

$$Scene_j = [color\otimes hue + flower\otimes species]. \tag{5}$$

Then each rewarded (penalized) scene was bundled together and bound with the respective feedback,

$$EXP_R = [Scene_{R1} + \cdots + Scene_{R3}] \otimes REWARD, \quad (5)$$

$$EXP_P = [Scene_{P1} + \cdots + Scene_{P3}] \otimes PAIN, \quad (6)$$

where there were 3 penalized, 2 rewarded, and 4 uncategorized scenes. For example, let 'rose' be rewarded and let 'red' be penalized including 'red rose.' Again, each *EXP* vector was thus only of length d, the majority bit operation distilling the salient patterns across the different scenes. For the rewarded (penalized) scenes, the critical pattern was 'rose' ('red') HBV. Lastly, a single *Behavior* composite vector likewise encoded both experiences into a single HBV,

$$BEHAVIOR = [EXP_R + EXP_P]. \quad (7)$$

Testing included the original 9 training examples as well as novel scenes introducing 3 novel *species* and 3 novel *hues* (new random HBVs). for a total of $j = 36$ flower scenes described in Eq. (5). The goal was for the "bee" to generalize the rewarded (penalized) species (hues) to include the novel hues (species) without undergoing additional training. Each testing scene $Scene_j$ was queried of *BEHAVIOR*,

$$Query_j = Scene_j \otimes BEHAVIOR, \quad (8)$$

where the resulting *Query* vector was classified as *REWARD* or *PAIN* according to the smallest HD below a threshold. If both HD values were above this threshold, then the scene was classified as unknown '*??*.' That is, HDC affords native anomaly detection, an ongoing challenge with ANNs generally.

B. Results

Testing accuracy was 99.93%, again without any additional training. Note that HDC performance does not suffer from class imbalance (5x reward, 6x punished and 25x unclassified) because each composite class HBV was independently generated.

In the field, the desired robot behavior might change, e.g. classify good (bad) flowers as bad (good). The new *BEHAVIOR* vector would be created in a similar manner then transmitted potentially over a noisy channel, leaving the "bee" with a degraded version. Applying random noise to the new *BEHAVIOR* and measuring the resulting classification accuracy, we observed over 95% accuracy even out to 50% noise (HD~0.25 difference between the clean and noisy version) (Fig. 2). The effect of noise can be further mitigated by repeatedly transmitting *BEHAVIOR*. Since each degraded transmissions will not be identically corrupted, the bundling operator can recover *BEHAVIOR*. 95% accuracy was maintained out to 80% noise with 9 copies.

IV. Cloning

Having thus trained one expeditionary robot, we want to replicate the trained HBVs across multiple robots to perform the same behavior. While these vectors can be trivially copied to another robot, if this encoding is compromised by a malicious agent, all copies are equally vulnerable.

In [8], elementary cellular automata (ECA) rule 90 was proposed to clone the parent's set of HBVs such that the

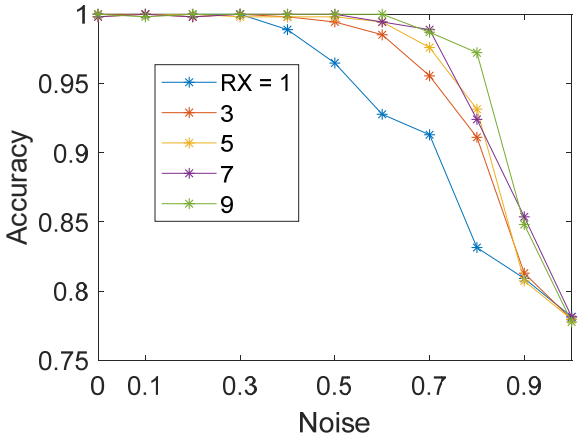


Fig. 2. Classification accuracy as a function of noise applied to *BEHAVIOR*. Accuracy is improved by bundling multiple received (RX) noisy copies.

resultant copies were maximally uncorrelated with the parent. ECA rule 90 proved to be inadequate since only two of the three typical HDC operations, binding and permute, were preserved. The third operation, bundling, was instead annotated in a separate connection matrix paired with each clone.

A. Background

Cellular automata (CA) are cell–based state machines which follow a homogeneous rule for state transitions based on local interactions between a cell and its neighboring cells. One dimensional ECA are the simplest class of CA, where each cell may only have one of two states {0, 1}, and, for every iteration i of the state update rule, a cell's new state is determined based on its own state q and those of its two neighbors p (left) and r (right). Each ECA rule is numbered according to the decimal equivalent of the rule's output (Fig. 3). Despite their simplicity, mathematical chaos and Turing complete behavior are demonstrated amongst these rules [15].

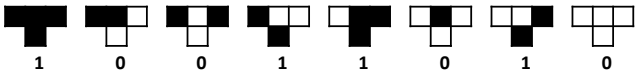


Fig. 3 All 2^3 state transitions for ECA rule $154 = 10011010_2$

B. Method

More formally, an ECA rule is viable for cloning when for each set of parent HBVs, clones resultant from the ith iteration of ECA rule R satisfy target Hamming distances (Tables II and III) [8].

Random HBVs $\boldsymbol{a}$, $\boldsymbol{b}$, and $\boldsymbol{c}$ of length $d = 10^4$ were generated. Each of the 256 ECA rules for $i = [1, 50]$ iterations were used to implement the cloning operations prescribed in Tables II and III, where varying i provided insight into a rule's cloning ability over time. The resulting HDs were then measured against their respective target HDs.

To both illustrate the quality of ECA rules identified in this way and in case the tables were incomplete, we cloned the HBVs from the foraging honeybee model and evaluated each clone's performance against the same testing scenes (reencoded using the cloned version of the item vectors) without any additional training. All 17 item vectors and the trained *BEHAVIOR* vector were cloned via each of the 256 ECA rules for i = 1-20 iterations.

TABLE II. TARGET HDS FOR BINDING AND SHIFT OPERATIONS, WHERE * DENOTES CLONING OPERATION

x	y	HD(x,y)
$\boldsymbol{a}$	$\boldsymbol{a}^*$	0.5
$\boldsymbol{a}\otimes\boldsymbol{b}$	$(\boldsymbol{a}\otimes\boldsymbol{b})^*$	0.5
$\Pi_{10}(\boldsymbol{a})$	$\Pi_{10}(\boldsymbol{a})^*$	0.5
$(\boldsymbol{a}\otimes\boldsymbol{b})^*$	$\boldsymbol{a}^*\otimes\boldsymbol{b}^*$	0
$\Pi_{10}(\boldsymbol{a})^*$	$\Pi_{10}(\boldsymbol{a}^*)$	0

TABLE III. TARGET HDS FOR BUNDLING AND SHIFT OPERATIONS, WHERE * DENOTES CLONING OPERATION

x	y	HD(x,y)
$[\boldsymbol{a}+\boldsymbol{b}+\boldsymbol{c}]^*$	$[\boldsymbol{a}^*+\boldsymbol{b}^*+\boldsymbol{c}^*]$	0
$[\boldsymbol{a}\otimes\boldsymbol{b}+\boldsymbol{b}+\boldsymbol{c}]^*$	$[\boldsymbol{a}^*\otimes\boldsymbol{b}^*+\boldsymbol{b}^*+\boldsymbol{c}^*]$	0
$[\boldsymbol{a}+\boldsymbol{b}+\Pi_{10}(\boldsymbol{c})]^*$	$[\boldsymbol{a}^*+\boldsymbol{b}^*+\Pi_{10}(\boldsymbol{c}^*)]$	0
$[\boldsymbol{b}+[\boldsymbol{a}+\boldsymbol{b}+\boldsymbol{c}]+\boldsymbol{c}]^*$	$[\boldsymbol{b}^*+[\boldsymbol{a}^*+\boldsymbol{b}^*+\boldsymbol{c}^*]+\boldsymbol{c}^*]$	0

C. Results

1) ECA cloning rules

8 ECA rules, viz. 15, 85, 154, 166, 170, 180, 210, and 240, attained all the target HDs, though not with all the same frequency i (Table IV). Only 2 rules generated a viable clone with every iteration, with the majority of rules requiring multiple iterations. For the MUX-XOR rules, 16i affords perfect cloning, yet 8i achieves extremely low HD difference values (<0.01).

TABLE IV. VIABLE ECA RULES FOR HBV CLONING

Logic Family	Literal			
Rule	240	170	15	85
Iteration (*i*)	1		2	
Boolean Expression	p	r	¬p	¬r

Logic Family	MUX-XOR			
Rule	180	166	210	154
Iteration (*i*)	8, 16			
Boolean Expression	p⊕(q∧¬r)	r⊕(¬p∧q)	p⊕(¬q∧r)	r⊕(p∧¬q)

2) ECA cloning results

Only the 8 aforementioned ECA rules demonstrated complete replication of the learned associations (Fig. 4a black dots) and approximate orthogonality with parent HBVs (not shown). Fig. 4b illustrates 8i clone behavior, which is acceptable in practice since threshold between known and unknown scenes need not be redrawn.

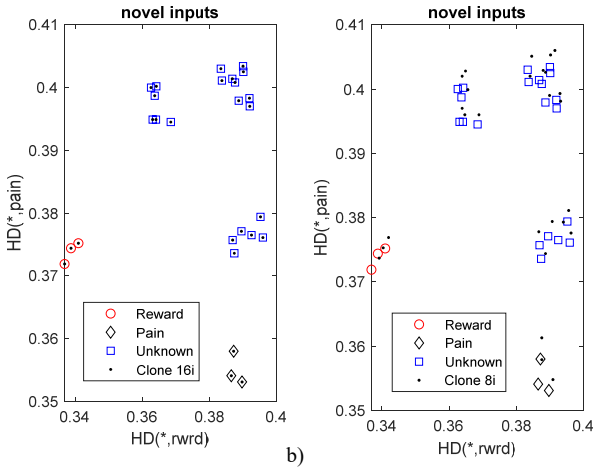


Fig. 4 Target parent behavior (shape outlines) and clone behavior (black dots) for R = 180, a) i = 16 and b) i = 8

D. Discussion

1) Maximum Number of Clones

What then is the maximum number of clones that may be generated by this method? This depends chiefly on the frequency with which an ECA rule generates a viable clone. ECA rules 240 (170) clone every iteration i through a simple left (right) circular shift. Intuitively, an HBV of length d can be circularly shifted at most d times before repeating. That is, a maximum of (d-1) unique clones can be made from an HBV of length d. For rules limited to 2i and 16i, the number of possible unique clones drops to (1/2)d and (1/16)d, respectively.

When discussing ECA rules, the literature typically remarks on the characteristic behavior of each rule over sequential iterations. It is not obvious, however, what is the characteristic behavior at arbitrary iterations. Unexpectedly, all 8 rules produce identical patterns (allowing for left/right symmetry) every 16i (Fig. 5). This is particularly surprising since these rules come from two different logic families [16]. Rules 15, 85, 170, and 240 are Literal rules, dependent on a single cell's value; whereas, rules 154, 166, 180, and 210 are non-linear MUX-XOR rules. Because of the equal density of 0's and 1's, the MUX-XOR rules functionally implement a shift map with at least period 16i. (However, for densities at either extreme, some differences emerge among these 8 rules.) In short, the simple shift map, one of the standard HDC operations, is sufficient to generate d-1 unique clones from one parent HBV of length d.

Fig. 5. Sample 16i ECA behavior observed for different initial conditions

2) Shift Rules Exclusive Viability

An exhaustive ECA search only found shift-based rules, and here we prove that indeed only shift rules are viable for HBV cloning. Criterion $(\boldsymbol{a}\otimes\boldsymbol{b})^* = \boldsymbol{a}^*\otimes\boldsymbol{b}^*$ implies the cellular automata (CA) rule * is a group homomorphism w.r.t group operator ⊗. Such CA rules on binary sequences perform modulo 2 addition of some neighboring coordinates, (folklore result, Lemma 4.1.7 [17]).

Claim: If the summation is over at least two cells, the following criterion fails:

$$\mathrm{HD}([\boldsymbol{a}+\boldsymbol{b}+\boldsymbol{c}]^*, [\boldsymbol{a}^*+\boldsymbol{b}^*+\boldsymbol{c}^*]) = 0. \qquad (9)$$

Proof by example: For concreteness let us consider the operator * defined by the Boolean expression p⊗r. If $\boldsymbol{a}$, $\boldsymbol{b}$, and $\boldsymbol{c}$ are uniformly random sequences and $\boldsymbol{a}$(k) denotes the triplet of bits occurring in $\boldsymbol{a}$ starting at position k, then the combination of the three events in the top part of the following table occurs with proportion $\varepsilon = (1/8)^3 = 1/512$:

State pqr	**Freq.**	**Operator *:** p⊗r
$\boldsymbol{a}$(k) = 101	1/8	$\boldsymbol{a}^*$(k) = ?0?
$\boldsymbol{b}$(k) = 100	1/8	$\boldsymbol{b}^*$(k) = ?1?
$\boldsymbol{c}$(k) = 001	1/8	$\boldsymbol{c}^*$(k) = ?1?

$[\boldsymbol{a}+\boldsymbol{b}+\boldsymbol{c}]$ (k) = 101

$[\boldsymbol{a}+\boldsymbol{b}+\boldsymbol{c}]^*$(k) = ?0? $\qquad$ $[\boldsymbol{a}^*+\boldsymbol{b}^*+\boldsymbol{c}^*]$(k) = ?1?

$\therefore$ HD will be $\geq \varepsilon = 1/512 \neq 0$.

A similar argument (with the choice of ε depending on the definition of *) rules out all nontrivial summation operators for this task. Therefore, only shift rules can satisfy all criteria. (As noted before, while HD need not be 0 for adequate performance in practice, this proof establishes the limits for the ideal case.)

Lastly, if a set of HDC equations do not include all the equations in Tables II and III, then there is an increase of the possible cloning operators. For example, when the requirement of shift preservation is dropped, we are allowed to use positional information. Consider a map that shifts symbols at even positions by two positions to the right and symbols at odd positions by two positions to the left. Here we use only the parity of the position, so this operation can be implemented by some kind of finite state machine.

3) Cyber security considerations

The computationally lightweight nature of HDC lends itself well to SWaP limited, "disposable" assets as opposed to resource intensive traditional ANNs. The rapid training methodology further lends itself to one-time-use programming per short-term expeditionary/reconnaissance objective versus a dedicated agent for a long-term task. Thus, while this cloning method does not obfuscate the HBVs used in each clone (desirable in long-term assets), the rapidity and diversity of clone generation protects against proliferation of traditional adversarial machine learning approaches against all clones in a time-sensitive manner.

Furthermore, as illustrated by the 8i cases, some noise may also be intentionally added to each clone's HBVs without loss of functionality, e.g. by redrawing thresholds. Alternatively, instead of using these methods for generating clones, they can be used to permute the agent's internal memory, increasing the difficulty of reverse engineering in a time-sensitive fashion.

V. Extraneous Sensors

A. Method

Based on the maze in [4], a simulated robot must "find its way out of a paper bag," navigating to the white region (Fig. 6a). The robot has 3 sensors: left light sensor (S_l), right light sensor (S_r), and center touch sensor (S_c), and two wheel motors: left (A_l) and right (A_r). Each sensor and motor has two states: ON (1) and OFF (0). In the maze, the robot moves in the 4 cardinal direction, where the left, center, and right sensor positions orient accordingly (Fig. 6b). In practice, an expeditionary robot would be equipped with z more sensors (S_z) than the 3 sufficient to navigate this maze. For example, a temperature sensor would provide no relevant information for this task. The robot was explicitly trained to map the 8 S_l, S_r, S_c sensor combinations to 4 desired actions (expanding the 5 rows in Table V to explicitly cover the "don't care" states). Success was measured as completion of the maze within 35 actions, the maximum steps necessary to complete the maze.

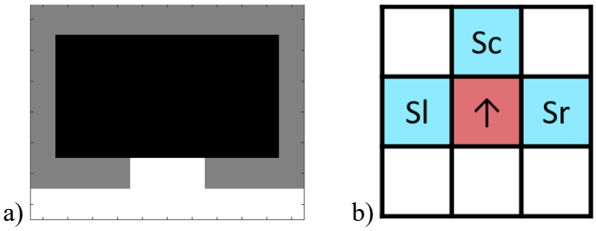


Fig. 6. a) Maze where black is navigable, grey is wall detected by S_c, and white is the target space detected by S_l & S_r. b) Robot moves in cardinal directions (red arrow), and sensors positions (blue) rotate with robot direction.

TABLE V. Sensor inputs and desired actions, where X indicates "don't care" sensor values.

Activated Sensor	S_l	S_r	S_c	$S_{1,2,\ldots,z}$	A_l	A_r	**Desired Action**
Light	1	1	X	X	0	0	Stop
Light Object	1 0	0 0	X 1	X	0	1	Turn Left
Light	0	1	X	X	1	0	Turn Right
No input	0	0	0	X	1	1	Go Straight

There is little by way of design rules for HDC encoding with robotics, so we explored 2 methods. In the first method, since each sensor has two states, each sensor's state was represented by a unique HBV, e.g. S_{lON} and S_{lOFF}. In this way, row 1 of Table V was encoded as follows:

$$Scene_1 = [S_{lON} + S_{rON} + S_{cX} + S_{1X} + \ldots + S_{zX}] \quad (10)$$

$$Scenario_1 = Scene_1 \otimes [A_{lOFF} + A_{rOFF}] \qquad (11)$$

where S_z denotes all z extraneous sensors and X is the randomly assigned "don't care" *ON* or *OFF* state.

Alternatively, states *ON* and *OFF* can themselves be represented as HBVs, which were subsequently bound to a sensor. In this way, row 1 of Table V was encoded as follows:

$$Scene_1 = [S_l \otimes ON + S_r \otimes ON + S_c \otimes X + S_1 \otimes X + \ldots + S_z \otimes X] \quad (12)$$

$$Scenario_1 = Scene_1 \otimes [A_l \otimes OFF + A_r \otimes OFF], \quad (13)$$

where X is the randomly assigned *ON* or *OFF* state.

In this work, we added up to $z = 9$ additional sensors per encoding method; that is, up to 75% of the sensors provided no useful information to the robot. Instead of training exhaustively on all 2^{z+3} possible sensor inputs, we performed training epochs comprised of the 8 training scenarios with random state values for each extraneous sensor S_z. For example, 3 training epochs affords 24 training scenarios. All these training scenarios were then bundled together to create one *BEHAVIOR* vector,

$$BEHAVIOR = [Scenario_1 + Scenario_2 + \ldots + Scenario_j]. \quad (14)$$

In addition to this *BEHAVIOR* vector, the simulated robot stored the 4 *Action* responses as dictionary entries used for deciding among the 4 possible wheel actions. For example, from row 1 of Table V, $Action_1 = [A_{lOFF} + A_{rOFF}]$ for Method 1 robots (Eq. 10-11) and $Action_1 = [A_l \otimes OFF + A_r \otimes OFF]$ for Method 2 robots (Eq. 12-13). Note, bundling requires an odd number of vectors; so we explored how using a fixed arbitrary tie-breaking vector *ODD* (1x) instead of a random one affected robot performance.

During testing, the robot's initial starting point and orientation were randomly assigned within the black field (Fig. 6a). At each time step, the scene (sensor states) was queried of *BEHAVIOR*, Eq. (14). A "clean up" step returned the most similar of the 4 *Action* dictionary entries.

The quality of Eq. (10-11) and (12-13) was measured as the *Action* classification accuracy. The number of unique sensor inputs increased exponentially with the number of extra sensors. A maximum of 1,024 unique scenes were tested against each robot configuration and equation set.

B. Results

1) Maze navigation

Each robot version was tested in 1,000 mazes in each of 5 trials. As a baseline, a robot whose *BEHAVIOR* was a random HBV attained a 6.1% success rate. Though the robot never started in the target area, it could start one cell next to it and randomly perform the rewarded action. Method 2 robots with no extra sensors S_z completed 100% of the mazes.

Method 1 robots with no S_z achieved 52.1% success. All unsuccessful robots failed for the same reason: the sensor pattern associated with the "Go Straight" action returned "Turn Left" instead (Table V). This occurred because "Go Straight" appears only once in the 8 key training examples. While the "Stop" action is also only tested in a single way in the maze (S_l, S_r, S_c = [1,1,1] being impossible), both possible sensor representations (S_c = 0, 1) were explicitly trained for. A simple quality control check ensured robots with this encoding deficiency were not used in subsequent experiments.

Note, the deficient Method 1 robots still attained 87.5% classification accuracy yet could only accidentally solve the maze. On the contrary, Method 2 robots across varying numbers of extraneous sensors and training epochs demonstrated 100% maze accuracy with less than 100% *Action* classification accuracy. Thus to aid interpretation of the extraneous sensors experiments, we note that Method 1 robots reliably achieves 100% maze success if and only if demonstrating 100% classification accuracy; however, Method 2 robots achieved 100% maze success if demonstrating >95% classification accuracy.

2) Action selection

Method 1 robots were decidedly not robust against extraneous sensor input (Fig. 7). Even training over 10 epochs (not shown) with varying "don't care" values did not noticeably improve robot performance. Method 2 robots also decreased in accuracy as the number of sensors increased, though far less dramatically (Fig. 8). However, the same robots trained over 6 epochs (48 scenes) maintained the requisite accuracies even when up to 9 (75% of the total) sensors were extraneous. No further improvement was observed after more than 6 training epochs (not shown).

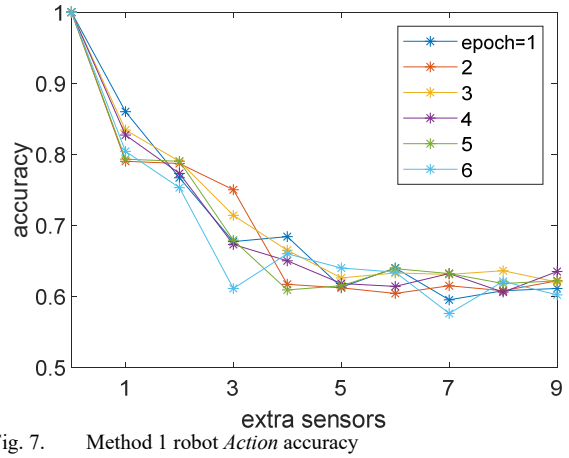


Fig. 7. Method 1 robot *Action* accuracy

When a fixed tie-breaking vector was used for bundling, Method 2 robot accuracies fell below acceptable levels even with additional training epochs (Fig. 8b). Method 1 robot accuracies showed no discernable difference (not shown).

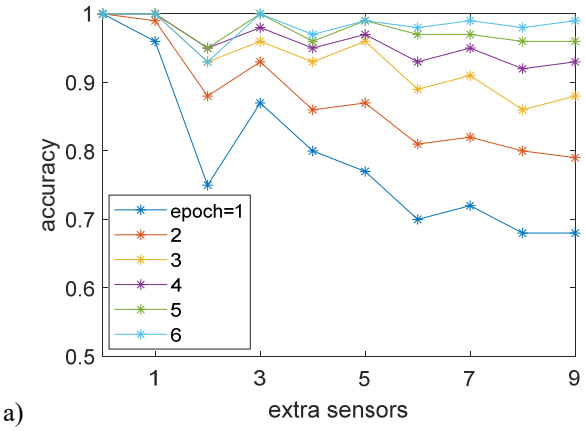
3) Noise resiliency

Lastly, a Model 2 robot (3 sensors, 1 training epoch) with 100% maze success had its *BEHAVIOR* vector degraded by random noise. For this task, the trained robot subjected to 85% bit flips (HD~0.42) still solved 92% of mazes (Fig. 9).

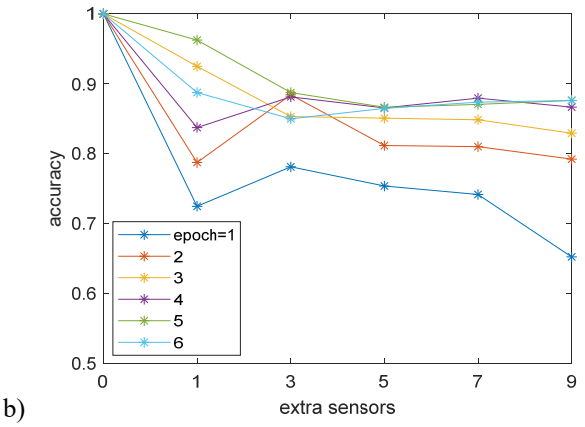
C. Discussion

Both Method 1 and Method 2 bundle a set of sensor states then bind these values with a pair of bundled motor actions. As the number of vectors bundled together increases, changing a single HBV in the set has a diminishing effect on the overall distinguishability between the two composite vectors (Fig. 10). This suggests small differences in sensor state patterns would became lost in the noise if only bundling is used, e.g. Eq. (10). Yet, as the bit overwrite experiment showed, only minimal information is necessary to preserve bound associations. Thus using binding of sensor with its state contributed to the resiliency of Eq. (12). That said, using a fixed tie-breaker vector encoded

yet another persistent pattern into *BEHAVIOR* which served no advantage during testing. A random tie-breaking vector better allowed underlying patterns to come through.



a)



b)

Fig. 8. a) Method 2 robot *Action* accuracy. b) Robot accuracy with fixed tie-breaking bundling vector (only used when z is odd).

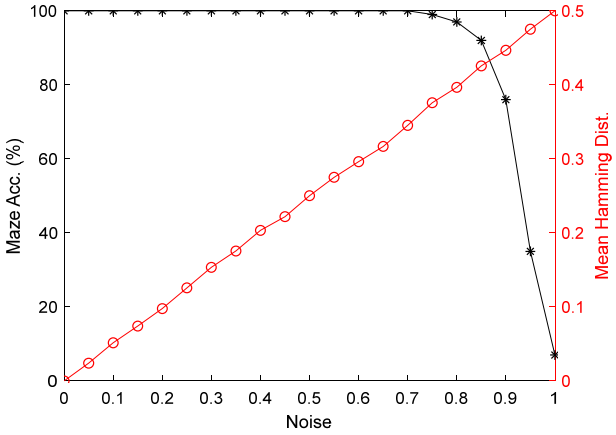


Fig. 9. Maze success (black stars) and mean Hamming distance from original vector (red circles) as function of the noise in trained Model 2

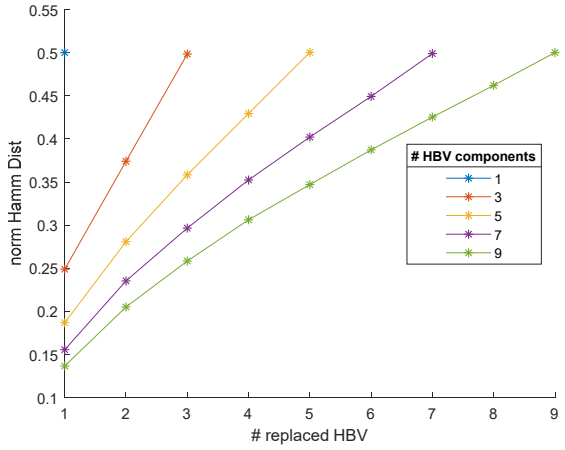


Fig. 10. Hamming distance for a set bundled HBVs [***a***+***b***+***c***+ …] as a function of the number of replaced vector elements.

VI. Network Topology

A. Method

Lastly, consider a swarm of expeditionary robots. As described in Fig. 2, noisy hyperdimensional vectors may be partially recovered by bundling multiple copies together, assuming uniform random noise across the received vector. Here we consider how the network topology effects the rapidity of convergence for a noisy HBV recovered in this way.

We created 4 network topologies of 5 robots each (Fig. 11). The Node 1 robot receives a new *BEHAVIOR* vector *B'* perfectly (HD=0), overwriting its previous version *B*. Every time step *t*, each robot transmits its current *BEHAVIOR* vector according to the network design and over a noisy channel (20% bits randomly replaced). (Note, this random noise only changes any particular bit half the time). After each robot received 2 such vectors transmissions from any node, it bundled them with its current *BEHAVIOR*, creating its new local vector. Network topology efficacy was measured as the time necessary for all robots to attain a threshold Hamming distance from *B'*.

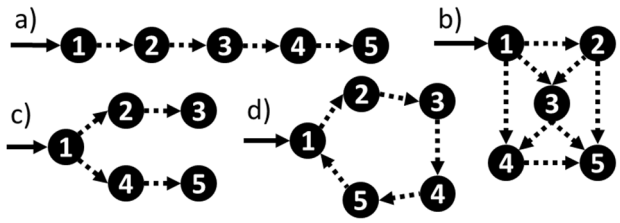


Fig. 11. Agent communication network topology: a) line, b) mesh, c) V, and d) ring. Noisy channel (20% noise) indicated by dashed line.

B. Results

All topologies but the ring converged to HD = 0.03 within 25 time steps (Fig. 12). With respect to network configurations, both proximity to Node 1 and connectivity density sped up convergence. Given the mesh network, nodes reached their update criteria (2 received messages) rapidly, resulting in the fastest convergence. The V network converged more quickly

than the linear network, since Node 1 was feeding twice as many nodes. The ring network failed to converge because, unlike all other networks, Node 1 received as well as transmitted. Since all node messages were weighted equally, Node 1's new *BEHAVIOR* quickly became lost in the majority bit updates.

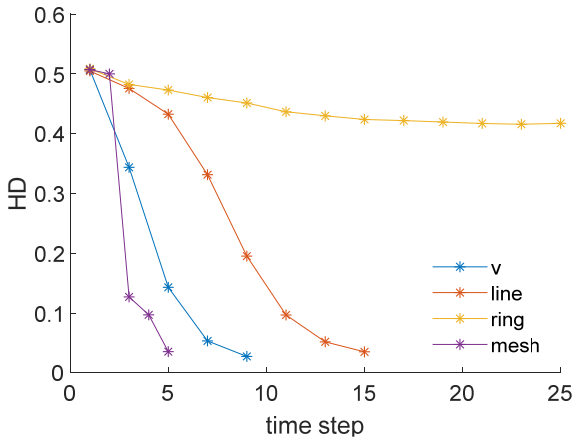


Fig. 12. Maximum Hamming distance among Node 1 and the other four nodes as a function of transmission time steps

VII. Discussion & Future Work

While ANNs have a deserved reputation, a tacit assumption may have crept into the machine learning community that neural networks are all you need. Hyperdimensional computing provides an exciting alternative to perform many of the operations difficult for traditional neural networks designs, e.g. one-shot learning, extreme noise resilience, anomaly detection, non-trivial cloning.

HDC has some significant drawbacks however. The most pressing here is all the sensor data considered here was state based. We do not consider how the robot determined the flower was "red" (Section III). Real-valued sensor values would have to be discretized and mapped to hyperdimensional vectors to perform calculations.

Conceptually, it is easy to envision an integrated ANN/HDC disposable robot where the ANN classifies raw, rea-valued sensor data. Based on this classification, a hyperdimensional state vector is submitted to a HDC decision element which constructs higher order meaning of scenes, compares said scenes against its behavior program, then determines the appropriate action.

Consider that such a hierarchical approach would allow for sensor interchangeability. Assume a scene **s** = **a*b*c*d**, where each letter corresponds to a state provided by a sensor and an trained ANN. If the sensor associated with **a** is replaced, provided the attendant ANN is retrained for the new sensor but the target categories remain the same, nothing changes as far as the hyperdimensional state representation is concerned. Such simple network reconfiguration cannot be done using traditional ANNs without retraining the entire network.

Acknowledgment

I wish to thank Jeffrey Hudack and Justice Prelow without whom this line of experiments would not have occurred. Any opinions, findings and conclusions, or recommendations expressed in this material are those of the authors, and do not necessarily reflect the views of the US Government, the Department of Defense, or the Air Force Research Lab.

References

[1] [1] P. Kanerva, "Hyperdimensional Computing: An Introduction to computing in distributed representations with high-dimensional random vectors," *Cogn. Comput.*, vol. 1, no. 2, pp. 139-159 (2009)

[2] [2] M. Schmuck, L. Benini, and A. Rahimi, "Hardware optimizations of dense binary hyperdimensional computing: Rematerialization of hypervectors, binarized bundling, and combinational associative memory," *arXiv preprint* arXiv:1807.08583 (2018)

[3] [3] A. Mitrokhin, P. Sutor, C. Fermüller, & Y. Aloimonos, "Learning sensorimotor control with neuromorphic sensors: Toward hyperdimensional active perception," *Science Robotics*, **4**, (30), (2019)

[4] [4] Levy, S.D. & Bajracharya, Sagar & Gayler, Ross. (2013). Learning behavior hierarchies via high-dimensional sensor projection. AAAI Workshop - Technical Report. 25-27.

[5] [5] P. Neubert,S. Schubert, P. Protzel, "Learning Vector Symbolic Architectures for Reactive Robot Behaviours," *IROS* (2016)

[6] [6] P Neubert, S Schubert, P Protzel, "An Introduction to Hyperdimensional Computing for Robotics," *KI-Künstliche Intelligenz*, 33 (4), 319-330, (2019)

[7] [7] D. Kleyko, et. al., "Imitation of honey bees' concept learning processes using Vector Symbolic Architectures," *BICA*, Vol 14, p 57-72, (2015)

[8] [8] D. Kleyko, E. Osipov, "No two brains are alike," *AISC 636*, pp. 91-100, (2017)

[9] [9] Gayler, Ross & Levy, Simon. (2009). A distributed basis for analogical mapping.

[10] [10] T. A. Plate. Holographic reduced representations. Neural Networks, IEEE Transactions on, 6 (3): 623-641 (1995)

[11] [11] D. A. Rachkovskij. Representation and Processing of Structures with Binary Sparse Distributed Codes. Knowledge and Data Engineering, IEEE Transactions on, 3(2):261-276 (2001)

[12] D. Aerts, M. Czachor, and B. De Moor. Geometric analogue of holographic reduced representation. Journal of Mathematical Psychology, 53: 389-398 (2009)

[13] S. I. Gallant and T. W. Okaywe. "Representing objects, relations, and sequences," *Neural Computation*, 25 (8): 2038–2078 (2013)

[14] P. Kanerva, *Sparse Distributed Memory*. MIT Press Cambridge. (1988)

[15] [15] S. Wolfram, *A New Kind of Science* (Wolfram Media, Champaign Illinois, USA) (2002)

[16] [Bricken 2019] W. Bricken, "Symmetry in Boolean Functions with Examples for Two and Three Variables," retrieved online 2019-01-17, http://iconicmath.com/mypdfs/symmetry-and-figures.020404.pdf

[17] [Salo 2014] V. Salo, "Subshifts with Simple Cellular Automata," PhD thesis, University of Turku (2014)