# ABSTRACT

Title of Dissertation: HYPERDIMENSIONAL COMPUTING FOR ARTIFICIAL INTELLIGENCE

Peter Sutor Jr.
Doctor of Philosophy, 2026

Dissertation Directed by: Professor Yiannis Aloimonos
Department of Computer Science

Hyperdimensional Computing (HC) is a neuromorphic and neurosymbolic computing paradigm that has attracted attention due to recent advances in Artificial Intelligence (AI) and Machine Learning (ML). By projecting information into high-dimensional spaces with certain statistical properties, HC affords a type of computational framework that is atypical to general computing. Noise becomes a feature of the system, and computation becomes noise tolerant in response. This allows computations to mirror the efficiencies and capabilities of biological organisms, under the right circumstances. Multi-modality, modal fusion, online/continuous/real-time learning, erosion of network architecture constraints, and more are directly achievable under this paradigm. In this dissertation, we analyze HC as a basis for AI and ML, studying and demonstrating its most useful properties. We show that HC is an attractive medium to serve as "lingua franca" between differing architectures, allowing both integration of existing models and downstream utilization of features derived from HC.

First, we study the properties of HC that are attractive to have in AI and ML contexts, and how HC can serve as a basis to build such systems. Then, we show case studies that demonstrate these

capabilities in practical settings, such as learning distributional semantics in a life-long and continuous process, neurosymbolic and architectural fusion of disparate network architectures in consensus, robotics applications like autonomous navigation and ego-motion, and more. Next, we demonstrate how HC is capable of achieving many properties present in machine learning that, at first glance, it may not seem to have. Namely, we show how HC can generate models, features for downstream tasks, manifold analogues for existing networks, and even perform its own form of "back-propagation". Finally, we analyze the superpositional nature of HC to show that even more efficient vector-symbolic constructions can be made by finding commonalities in superposed features and developing structures around them.

# HYPERDIMENSIONAL COMPUTING FOR ARTIFICIAL INTELLIGENCE

by

Peter Sutor Jr.

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2026

Advisory Committee:

Dr. Yiannis Aloimonos, Chair/Advisor
Dr. Shihab Shamma, Dean's Representative
Dr. Pentti Kanerva, Special Member
Dr. Cornelia Fermüller
Dr. Ramani Duraiswami
Dr. Laxman Dhulipala

# Dedication

This dissertation work is dedicated to my family, friends, loved ones, and colleagues. You were all a little piece of the puzzle that gave me the ability to conduct this research. Without you, it would not be possible. All that we are, is a sum of what has come before us. To that end, the successes in this body of work were all predicated, at least partially, on you.

Namely, I would like to dedicate this work to my parents. Whenever I felt like giving up, they were there to keep me going. Beyond that, my parents were always there to help me in whatever they could when the work demands outpaced my personal life. While they could not help conduct this research, they were always there to listen and help out in everything else. I don't think I would have succeeded without you.

Apart from that, I was aided in many aspects, from small to large, by my other family members, personal friends, and colleagues. This is work is dedicated to you, as well. Many things had to fall into place in just the right way for me to achieve what I have. Each of you has helped me, even if by just listening, humoring me, actually helping me solve my problems (research or otherwise), or just being there when things got hard. In particular, I'd like to dedicate this work to Dr. Rachel St. Clair. Much of the later topics and publications were influenced by many length discussions with her and her academic and business circles. In many ways, this dissertation is a crystallization of these things.

I would also like to dedicate this work to all the mentors I've had along the way in my education. There were many teachers in my primary education that put me down the path I have taken. I thank you all. Most crucially, I thank my high school computer science teacher, Robert Luciano. Through the many programming classes I took under his tutelage, I decided to go down this path of computing. But it wasn't just that; you challenged us all with logic puzzles, optimally playing games with smart strategies, exercising our critical thinking skills, and practical application of what we learned. Further, I

would like to dedicate this work to Professor Debra Smarkusky. When I first started college, Professor Smarkusky steered me specifically towards computer science and mathematics. Moreover, my first paper was co-authored with her. I would probably not pursue a doctorate without her influence.

Last, but not least, I dedicate this work to Professor Yiannis Aloimonos, my advisor. This research topic of Hyperdimensional Computing would not be the direction I would go in, if not for him. He has guided me through all the publications, hard work, and academic rigor that goes into a dissertation. From beginning to end, this dissertation is a product he has helped me with, every step of the way.

Finally, I dedicate this work to Dr. Pentti Kanerva, who inspired me and many others to work on Hyperdimensional Computing. None of this would be possible without his efforts to enlighten us all to a completely different way of looking at computation.

# Acknowledgments

I would like to acknowledge the efforts of all those who have helped me with my work. Namely, I would like to thank all my co-authors on the papers published for this dissertation.

I would like to acknowledge Dr. Douglas Summers-Stay, who co-authored a majority of these papers with me, and who mentored me throughout my Fellowship at the Army Research Labs in Adelphi.

I would like to acknowledge my co-author Dr. Anton Mitrokhin for helping me with many of my early papers on Hyperdimensional Computing, as well as the most successful work; Hyperdimensional Active Perception.

I would like to acknowledge my co-author Dr. Renato Faraone for all his contributions to my papers on the side of math and Category Theory. His co-authorship and the many discussions we've had has helped shape my thoughts on Hyperdimensional Computing and computing in general. Appendix A is directly his work on introducing readers to Topos Theory, as published in our work on Vector Symbolic Sub-Objects Classifiers. The credit for the contents of the appendix are all his. Additionally, all credit goes to Dr. Faraone for his rigorous work reproduced in section 4.3, from the work on our HyPE paper.

I would like to acknowledge my co-author Dr. Dehao Yuan for his work on HD-Glue, the implementation and testing of which was largely his work. I would also like to acknowledge his work on Distance Preserving Quantization, which was essential to HD-Glue.

I would like to acknowledge Dr. Rachel St. Clair for inspiring a lot of the research directions I have gone in, particularly in Hyperdimensional Computing as it pertains to neuroscience, dynamical systems, and efficient, data-agnostic, ubiquitous computing.

I would like to acknowledge my co-author Dr. Cornelia Fermüller for all the discussions, criticisms and guidance on realizing the potential of Hyperdimensional Computing.

Finally, I would like to acknowledge Professor Yiannis Aloimonos, my advisor. The entire research direction of Hyperdimensional Computing is completely credited to him. I would have likely never pursued this route without him introducing it to me. Furthermore, I'd like to acknowledge his guidance on both the academic and philosophical, when it comes to computing, computer vision, robotics, and artificial intelligence. All of the published works in this dissertation were shaped by his mentoring.

# Table of Contents

# List of Tables

# List of Figures

# List of Abbreviations

| | |
|---|---|
| AI | Artificial Intelligence |
| DM | Diffusion Models |
| DCH | Deep Cauchy Hashing Network |
| DTQ | Deep Tripet Quantization Network |
| DQN | Deep Quantization Hashing Network |
| EM | Expectation Maximization (Minimization) |
| HAP | Hyperdimensional Active Perception |
| HC | Hyperdimensional Computing |
| HD-glue | Hyperdimensional Glue |
| HIL | Hyperdimensional Inference Layer |
| HyPE | Hyperdimensional Propagation of Error |
| GAI | Generative Artificial Intelligence |
| GEM | Generalized Expectation Maximization (Minimization) |
| LLM | Large Language Models |
| ML | Machine Learning |
| PMF | Probability Mass Function |
| RBC | Reflective Binary Code |
| XOR | Exclusive-Or |

# Chapter 1: Introduction

Artificial Intelligence (AI) has become a widely debated topic in the last few years with the development, distribution, and commercialization of Large Language Models (LLMs), Diffusion Models (DMs), and other Generative AI (GAI). Such technology is now popularized to the level that it is used by students and educators alike, of all levels, in industry, corporate settings, and even private citizens. Due to this boom in demand for better and better AI, we find a bigger and bigger demand for integration of modalities that have traditionally been quite separated in both academic, engineering, and modeling settings. It is no longer enough to have an AI you can chat with by text; your AI must be capable of actual speech and inflection, capable of generating code, generating images, plans, schemas and even a level of persistence to the scale that most AI efforts up until this era of GAI have not been commonplace.

One point of contention that many of these AI solutions often arrive at is the normalization of information into a common representation space that can effectively integrate these modalities. For humans, this is as natural as can be. All sensory information, modalities, and thought processes are eventually integrated into a common space deep within the layers of the human brain. Achieving this in silicon, however, is a far more mysterious process. It is for this reason that interest in neuromorphic designs has skyrocketed in recent years.

In this work, we analyze Hyperdimensional Computing (HC), a neuromorphic, neurosymbolic framework with interesting properties in computation, particularly the type of computations of interest in AI and Machine Learning (ML). Pentti Kanerva gives a brilliant introduction to the topic in [1]. We will attempt to give a similarly careful introduction as well. There are many properties afforded by HC that are attractive to AI/ML tasks, which we will cover. Furthermore, we present case studies that aim to confirm these properties in practical settings, to demonstrate what is currently achievable.

We discuss how such properties can serve as a basis to build future systems. Next, we show how HC is capable of achieving properties that already exist in machine learning which may not be apparent at first glance. Finally, we analyze the superpositional nature of HC to show that even more efficient vector-symbolic constructions are readily available for construction by consolidating commonalities before finalizing the superposition.

## 1.1 Published Works

Published contributions in this thesis are cited below in order of publishing date, for clarity. Whenever the author's own work is cited, the language will shift to indicate this.

1. *A Computational Theory for Life-Long Learning of Semantics* (2018) [2]
2. *Learning Sensorimotor Control with Neuromorphic Sensors: Toward Hyperdimensional Active Perception* (2019) [3]
3. *Symbolic Representation and Learning with Hyperdimensional Computing* (2020) [4]
4. *Gluing Neural Networks Symbolically Through Hyperdimensional Computing* (2022) [5]
5. *Brain-inspired Hyperdimensional Computing for Ultra-efficient Edge AI* (2022) [6]
6. *Vector Symbolic Sub-objects Classifiers as Manifold Analogues* (2024) [7]
7. *HyPE: Hyperdimensional Propagation of Error* (2025) [8]

## 1.2 Outline of Thesis

In Chapter 2, we cover the basics of Hyperdimensional Computing (HC), and any background material related to HC that is relevant in subsequent sections. Namely, we begin by motivating the idea behind HC through a thought experiment. Once this is done, we mathematically define the formalisms used throughout the thesis. Finally, we cover some useful properties of HC and use cases, in order to set up the building blocks used in subsequent chapters.

In Chapter 3, we demonstrate and discuss the practical applications of the properties of HC covered in Chapter 2. This is done by outlining certain case studies and discussing their findings at length [2–5]. We use these case studies to argue that HC exhibits many qualities that are attractive to AI and ML contexts. We make the point through these case studies that HC can serve as the blood of a larger AI structure, serving as the method of communication between many disparate AI and ML subsystems that otherwise must be hand engineered to communicate with each other.

In Chapter 4, we discuss how HC can be used to achieve properties we already see in many AI and ML use cases. Namely, we discuss how HC lends itself to broadening beyond the topological trappings of most ML architectures, breaking through into more Category Theoretical contexts, naturally [7]. We give an algorithm that exemplifies this fact as a jump-off point for future research. Then, we discuss how many engineering solutions that seem specific to ML can be replicated in HC. Specifically, Vector Symbolic Architectures (VSAs) can be composed much like layers in a network, or AI subsystems that feed into each other. We demonstrate how mechanisms analogous to back-propagation can be constructed to create feedback mechanisms [8].

In Chapter 5, we unveil a novel method for analyzing the superpositional properties of HC and VSAs, called *hypermapping*. We show that further efficiencies can be gained through hypermapping.

Furthermore, hypermapping can create more properly structured aggregations of HC computations that lose less information.

In Chapter 6, we discuss future directions of research in HC that stem from the work in this thesis. We posit that there is more low-hanging fruit in HC research that has the potential to push work in AI and ML further than what is possible today. We structure these directions into three broad categories of:

1. Extension of the results in this thesis to non-binary Hyperdimensional Computing
2. Approximation of neural networks and other ML and AI architectures into fully hypermapped Vector-Symbolic Architectures.
3. Hardware designed to leverage the efficiencies of Hyperdimensional Computing

In Chapter 7, we conclude the thesis with closing remarks. It is the aim of this thesis to show that Hyperdimensional Computing will be integral to future work in AI and ML, and in conclusion we ruminate on these aspects.

# Chapter 2: Background

In this section we give background information that is necessary to understand the work presented in later chapters. It is expected that the reader will be unfamiliar with Hyperdimensional Computing (HC), so it is intended and recommended to read the background information on that topic. This section will dedicate much of itself towards simply understanding both the intuition and mathematics behind HC, so that the subsequent sections are easier to understand.

## 2.1 Hyperdimensional Computing

We break down the background information on Hyperdimensional Computing (HC) into three sections; the underlying intuition behind it, the specific mathematical formalisms, and notable properties. One should note that this thesis focuses on HC involving binary elements. The reason for this is mostly simplicity, paired with making it very clear why computation and AI/ML of all stripes can benefit from HC. If the most basic form of communication for a computer (1's and 0's) are sufficient to implement the basics of HC, then all forms of computation can benefit.

### 2.1.1 Intuition

We begin with a thought experiment, to motivate the intuition behind Hyperdimensional Computing (HC) in a very straightforward way. This enables more specific intuitions and technical descriptions to be easier to to follow.

Suppose that a human being lives their life perpetually attached to a brain scanning machine. That is, from birth, their brain is being scanned by a helmet-like mechanism. The intention here is that we have a dataset of continuous signals of ground truth to refer to that grows in real-time. Furthermore,

suppose that the brain scanning technology is specifically tracking activations of neurons in the brain, with such impressive resolution that we can actually register each individual neuron accurately. For simplicity, suppose that the neurons remain constant, neither dying off nor new neurons coming into existence. However, we will see later on that this doesn't affect the thought experiment much beyond complexity of description. We conceptualize the neural system as comprising of input neurons that correlate to the senses, output neurons that correlate to motor functions and actions, and interneurons that make up the processing structure of the brain. Once again, this is for simplicity. Refer to (a) in Fig. 2.1 for an example of what the brain scan might see at any given instance.

We would like to consider three questions in this thought experiment:

1. Can you build a new brain that will mimic the evolution of the subject's brain activations over time, given these brain scans?

2. How would it simulate the subject's behavior?

3. Can you precisely model all past behaviors perfectly and what would it take?

Consider the following: suppose we strip away all aspects of what is traditionally considered a "network". To do this, refer to the steps shown in Fig. 2.1. From the original network, since the brain scan can see every single neuron at all times and differentiate them, we can organize them from the disorganized 3D pattern in (a) to an indexed pattern in (b). We keep the edges to show how the connections transfer. Then, if we remove the edges, we are left with an array of activations, as shown in (c). Thus, a "brain state" can be represented as a binary array. We will focus on these binary arrays first, before addressing the removal of the edges connecting neurons. However, we will also see later that this does not necessarily change our conclusions.

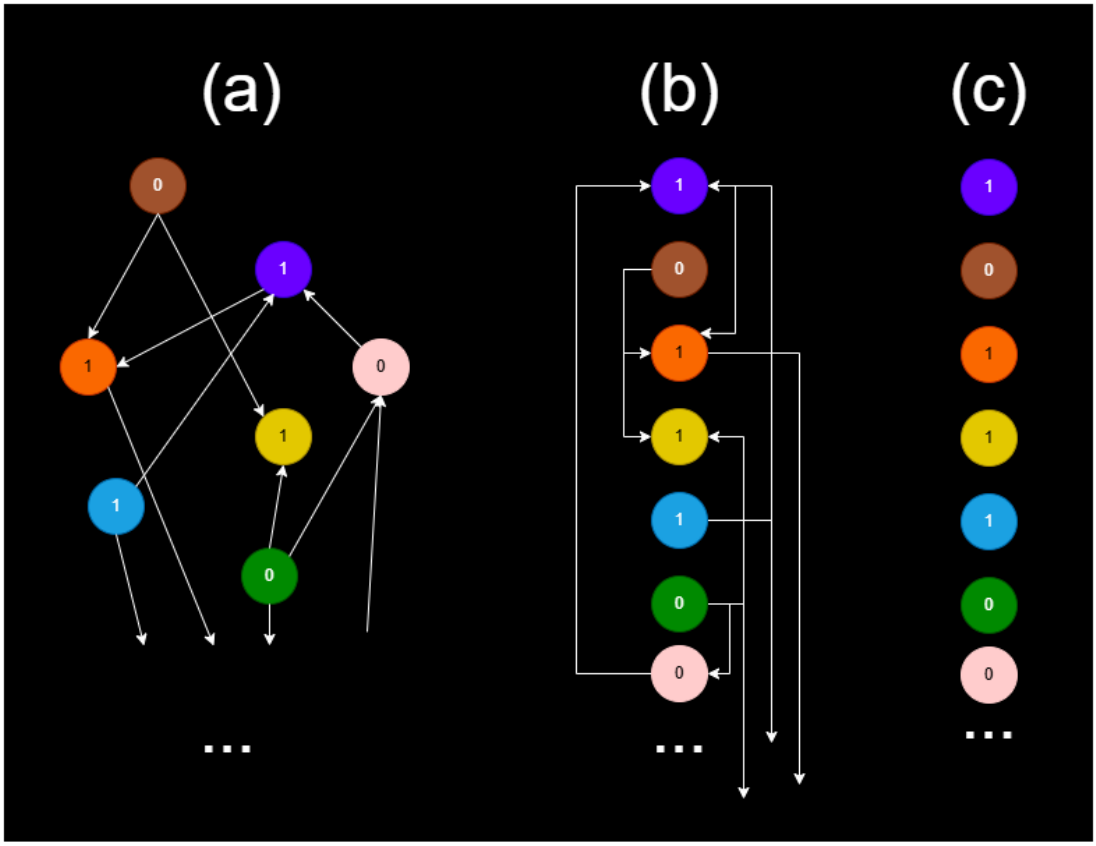



Figure 2.1: The gradual transform of a hypothetical neural network of neuron activations into simply an array of indexed neural activations, as per our thought experiment. In (a), the neurons are color-coded to indicate identity. In (b), we use the color-codings to index the neurons, thus lining up the connections along this arrangement of the graph. In (c), the connections are removed, revealing the brainstate to be just an array of 1's and 0's, denoting activations and non-activations.

In order to answer question 1, we would need a way to preserve changes in brain states over time. Without that, the information would be lost. Once a brain state is converted into the form shown in (c) in Fig. 2.1, it has effectively become a digital signal. How does one preserve a digital signal? The natural answer is to record the Exclusive-Or (XOR) of the bits of both brain states. Hence, bit-wise XOR can precisely record a brain state transfer, creating a *digital transfer function* $\Delta$, such that

$$\begin{aligned}
\Delta E_i &= (E_i \text{ XOR } E_{i+1}) \text{ XOR } E_i \\
&= E_i \text{ XOR } (E_i \text{ XOR } E_{i+1}) \\
&= (E_i \text{ XOR } E_i) \text{ XOR } E_{i+1} \\
&= \mathbf{0} \text{ XOR } E_{i+1} \\
&= E_{i+1}
\end{aligned} \tag{2.1}$$

Note, this follows as XOR is associative, commutative, and 0 is the identity. Additionally, when one of the inputs to XOR is fixed, XOR is an *involution*, meaning that it is a function that is its own inverse. This yields:

$$
\begin{aligned}
g \circ f(a) &= g(f(a) \\
&= f^{-1}(f(a)) \\
&= f^{-1}(a \text{ XOR } b) \\
&= f(a \text{ XOR } b) \\
&= (a \text{ XOR } b) \text{ XOR } b \\
&= a
\end{aligned}
\tag{2.2}
$$

where $f = g = f^{-1}$, and $a, b \in \{0, 1\}$. These properties are preserved when applied bitwise to vectors.

Consider (a) in Fig. 2.2, which shows two consecutive brain states at time $i$, represented as arrays $E_i$ and $E_{i+1}$. The bit-wise XOR shown in (b) is also a digital signal, which can be mapped back to a vector form of binary entries. As such, you can preserve a brain state transfer by constructing a brain state that performs the transfer. It is notable that no new tools were required to do this. Effectively, our formalism of using simple arrays of 1's and 0's suffices to preserve brain states.

This method seems to satisfactorily answer question 1. However, we note that the human brain does not need to record these states. It produces them directly from itself. So, we consider if there is a way to take the digital transfer functions $\Delta E_i$ we obtain at all points in time $i$ and once again map these back to a single brain state. Once again, we need a method of consolidating information. A knee-jerk response would be to yet again use XOR. However, at this point we don't actually need to perfectly recreate the digital transfer functions. We know that the human brain's activations can be quite noisy; so we should opt for a more approximating approach. We simply require the behavior of neurons to approximate digital transfer functions when aggregated with the samples of the brain so far.

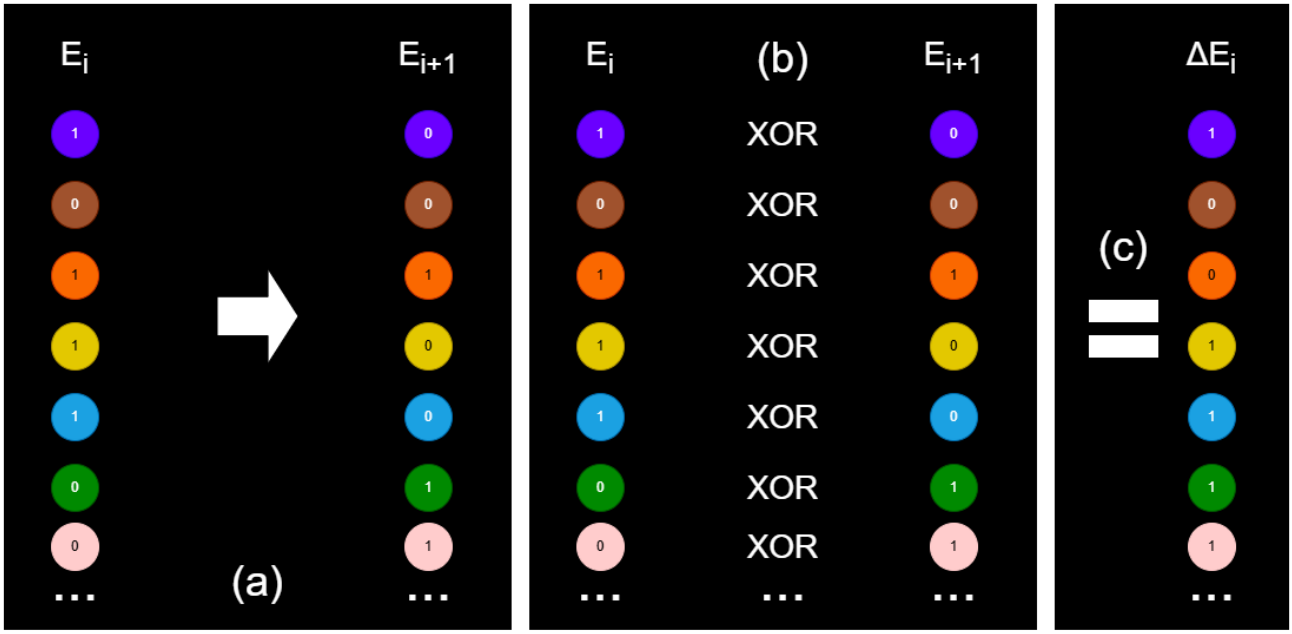



Figure 2.2: Preservation of brain state changes can be achieved via Exclusive-OR (XOR). In (a), an example of a change in brain state from moment $E_i$ to $E_{i+1}$. In (b), XOR is applied bit-wise to preserve this transform precisely. In (c), the result of the XOR is shown, equaling the digital transfer function.

Consider the following simplification: suppose there were clear clusters of $\Delta E_i$ samples. Clusters imply that the behavior required to transfer between states is similar and analogous. In such cases, we can aggregate clusters of behaviors by summation. Since we want to map brain states back to brain states, to mimic the brain, we allow a voting mechanism to find centroids of clusters. In Fig. 2.3, a list of a cluster of brain transfer functions are aggregated by finding a clear winner in majority representation of each neuron/component. If there is a tie, the result is left uncertain. Uncertainties can be resolved by simply randomly breaking the tie, thus avoiding the introduction of any bias. In the end, as shown in (d) in Fig. 2.3, a single brain state centroid can summarize the behavior of the cluster. An important thing to note here is that resolving the tie introduces random behavior into the system, when it doesn't change the result.

So far, we seem to have satisfactorily answered question 1 in our hypothetical. We now consider question 2; can we simulate behavior this way? Up until now, the observant reader may note that components of brain states appear to not interact with each other throughout the process described so far. If only some components are treated as receiving input and some produce outputs, the interneurons appear to not contribute anything. This is where we introduce the infamous "hidden layer" of our network. Our in-between components must serve as the hidden layer that transfers input signals to output. If we want to minimally change our strategy, we can replicate and randomly distribute input signals to each interneuron in a fixed, random pattern. This, effectively, projects the smaller vector of neurons into the correct, bigger size. Likewise, we can randomly choose subsets of interneurons and map them as transfers to output neurons, superposing them, using another fixed, random pattern. In effect, the input and output layer can be represented as a projection to the same dimensional space as our in-between layer. See Fig. 2.4 for an example under our input/output paradigm, with some simple subsystems for sensory inputs and output motor controls. Now, our strategy so far can be applied

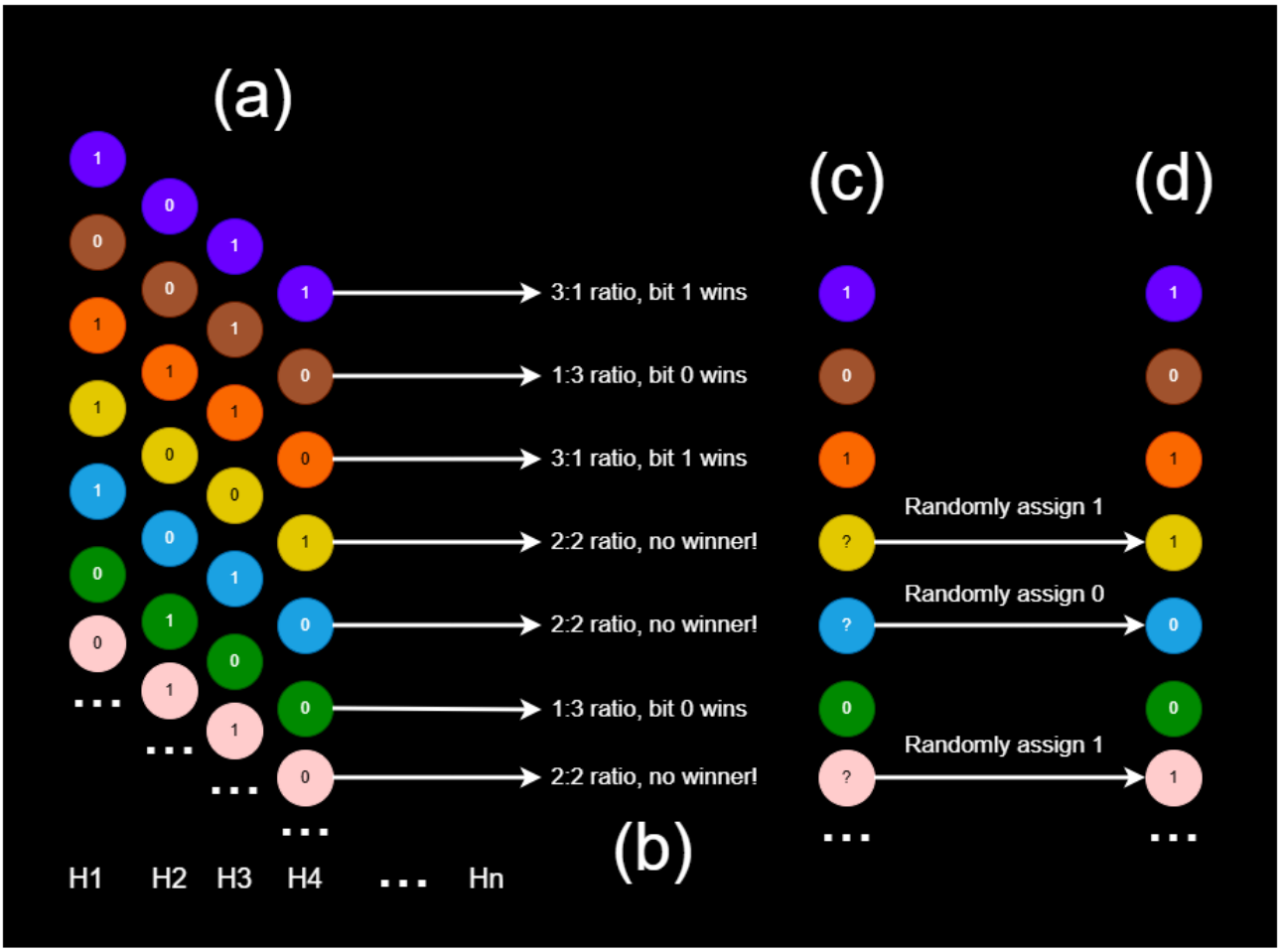



Figure 2.3: Superposition can be achieved by aggregation of brain states transfers as terms in a component-wise summation. In (a), a series of brain states transfers of interest are shown. In (b), we note the ratio of 1's to 0's, which either lead us to clear winners, or ambiguous ties, resulting in a signal of 1, 0, or "?" shown in (c). The ambiguous cases can be resolved by randomly being 1 or 0. In (d), the resulting transfer superposition is computed from a random variable to break the tied terms.

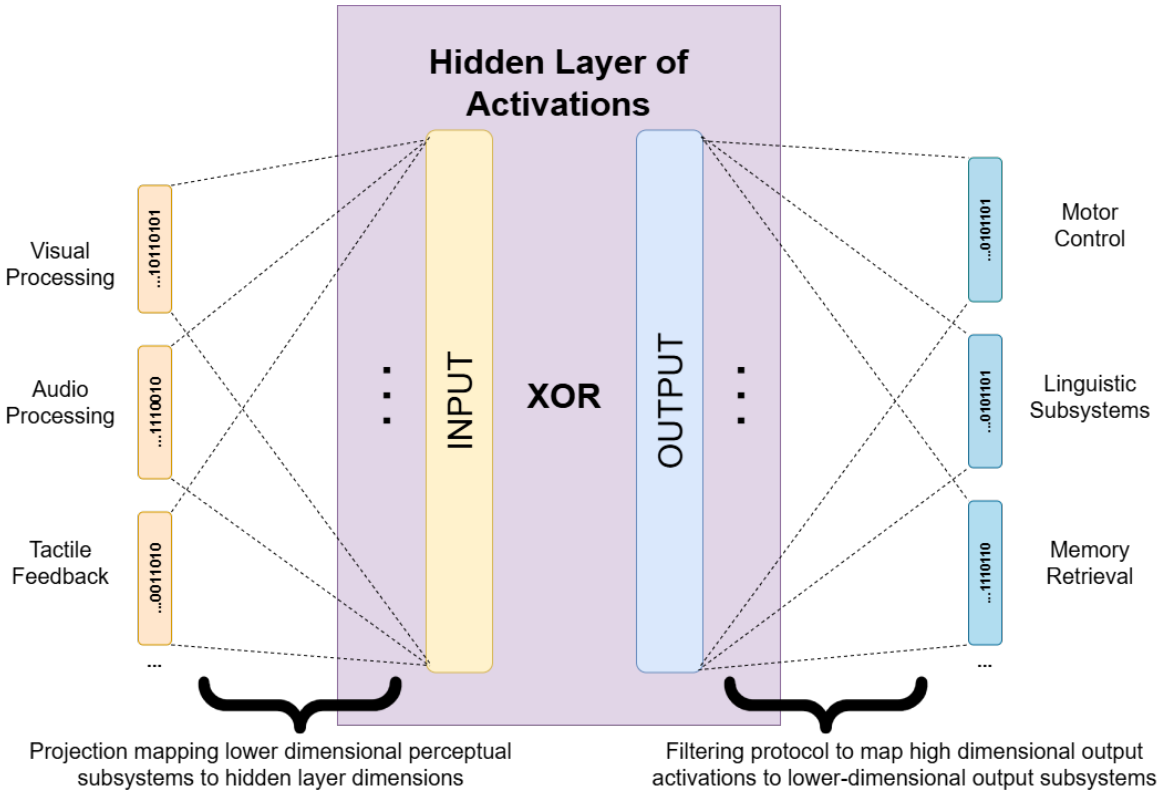



Figure 2.4: An example of the hidden layer abstraction to handle the fact that input neurons and output neurons tend to be much smaller than the hidden layer, or hidden substructure. We use replication of input neurons and random dispersal in known patterns to produce a large `INPUT` vector of the appropriate size to carry out digital transforms via XOR. Likewise, sub-vectors for output can be produced by a filtering operation. For example, subvectors can always be randomly selected in a known pattern and collated to a smaller dimension by superposition. This recovers the appropriate lengths of vectors to match output neuron counts.

properly as follows: given a previous brain state, simply XOR with the brain state transfer function $\Delta E_i$, as in Eq. 2.1, to perfectly recall the next brain state. So, we can conclude that this answers question 2. Our entire system can be mapped to operations on vectors of consistent size.

At last, we reach question 3. Is it possible to model all past behaviors perfectly, and what would it take? Our formalism of majority vote allows us to aggregate more brain states as they appear by keeping track of the deviation from expected value of 0's and 1's at each component. If we know that

$n$ brain states have occurred, each component will store how many more or less 1's were observed than the expected $\frac{n}{2}$ if they were all tie breakers. So, our binary vector becomes a signed integer vector, which gets thresholded to binary when a decision must be made. However, to overcome statistical noise, we must have sufficiently many neurons - a fact the human brain already satisfies. So, if our hidden layer is sufficiently high-dimensional, we will approximate the behavior of the human subject.

But how do we do this expansion of the hidden layer? The astute observer will note that approximation of a single neuron with just a single bit is an oversimplification. We can repeat our approximation of the brain on an individual neuron itself, by representing the input bit and output bit as fixed random patterns of 0's and 1's, as well, mapped to a concept of activation (input bit is 1) and inactivation (input bit is 0). This allows us to perform the brain state transfer technique via XOR in Fig. 2.2. We can now capture more complicated sequences of activations and non-activations in single neurons by using superpositioning like in Fig. 2.3. What we are doing now, as opposed to learning the transfers of the brain, is learning an approximating digital transfer function for sequences of activations and non-activations in time. This approximates the cellular automata of the neuron in question as a finite state automata, proportional in size to the approximating resolution desired. This expands each neuron to be comprised of multiple 0's and 1's each, but otherwise nothing else changes in our process. We can still do everything with high-dimensional vectors of the same length.

One final point of contention: what about our missing connections that were dropped when going from (b) to (c) in Fig. 2.1? Without these, the causal relationships between neurons seem to be lost! We will see near the end of the thesis how to deal with this issue when you were never given the connections to begin with. However, for now, we can construct and record the transfers into the brain state by using the aforementioned approximation of a neuron's cellular automata. Simply give each neuron a unique activation pattern that identifies it, equal in length to the input/output vector lengths

for the approximating finite state automata. Since we have indexed the neurons in (b) in Fig. 2.1, and we have color-coded the neurons in Figures 2.1, 2.2, and 2.3, in general, we are aware of the IDs of each neuron. Thus, by using XOR again to augment the output vectors of neurons with a digital transfer function mapping the output to the ID, we can flavor the sources of activations of neurons to their targets, recording the edges in our high-dimensional vectors. So, we can *still* do everything the original brain did with high-dimensional vectors.

This seems to conclude our thought experiment. We only need to increase the lengths of the vectors sufficiently to adequately approximate every neuron's cellular automata. Now, we have a single high-dimensional vector $B$ that represents the history of a brain's actions and structures. With sufficiently long vector length, $B$ approximates a global digital transfer function $\Delta E_i$. Given the first brain state $E_1$, which is computed after observing some input, we compute $\hat{E}_2 = B$ XOR $E_1 \approx E_2$ to receive an approximation of $E_2$ called $\hat{E}_2$. Now, we recursively use $\hat{E}_2$ the same way to obtain $\hat{E}_3$, and so on, obtaining all $\hat{E}_i$ at all time frames $i$. Again, if the vectors are long enough, the approximation is sufficient to carry out the output actions for the given input. In essence, we can generate predictions for what the original brain would have done. We can, of course, regularize this by obtaining new input as it arrives, curtailing a compounding runaway effect in many predictions. At some sufficient length of the vector, we will overcome the runaway effect over any constant buffer in input. So, not only have we modeled the behavior of the original brain, we also have a built in process for adjusting the behavior to sensory observations.

The purpose of this thought experiment is to motivate the intuition behind Hyperdimensional Computing (HC). Namely, the power of maintaining high-dimensional vectors that have noise tolerant operations, which can recall digital transform functions when asked. These are referred to as "hypervectors". There are, of course, blind spots to our thought experiment for the purposes

of simplifying the biological human brain. But with sufficient understanding of biological brains and sufficient computational power for long enough vectors, we can approximate the brain with a hypervector of some sort. We simply need proper engineering to account for the additional factors as we did in our thought experiment for neuronal connections and cellular automata of neurons.

### 2.1.2 Formalisms

Now that we have an intuition behind Hyperdimensional Computing (HC), we proceed to formally describe this system in algebraic terms. One thing to note before we begin is that the choice to use binary, and the particular operations of XOR and majority vote aggregation shown in Fig. 2.2 and Fig. 2.3., is purely arbitrary. There are many different algebraic systems that can be formed, with different strengths and weaknesses. We choose to use binary for simplicity in this thesis and compatibility to all computers.

Our formalism for HC is mostly derived from Pentti Kanerva's excellent introduction to the idea [1]. However, HC's origination can be traced back to Tony Plate's work [9], namely this observation of *hyperdimensionality* being key to constructing the HC framework. In this context, we use hyperdimensionality to refer to the statistically robust number of samples from a random variable to form hypervectors, which experience asymptotically diminishing returns in these properties around a very specific constant.

#### 2.1.2.1 Dense Binary Hypervectors

A dense binary hypervector is a binary vector that consists of $n$ random Bernoulli Trials, where $n$ approaches *hyperdimensional* numbers, referring to the value of $n$ being big enough that the binary

string formed by the repeated Bernoulli Trials has attained a high enough dimensionality to attain certain statistical properties in their distribution in the vector-space of all binary strings of length $n$. For a binary space, this occurs on the order of thousands of bits. Most typical use cases will not require more than $n \approx 2^{14}$. Let the number of successes - the number of on bits - be a random variable $k \sim \text{Bern}(n, \frac{1}{2})$, where $n$ is the number of trials and $\frac{1}{2}$ the probability of success. The random variable $k$ then follows the Binomial Distribution, where the probability of $k$ successes is:

$$P(k) = \binom{n}{k} \left(\frac{1}{2}\right)^k \cdot \left(\frac{1}{2}\right)^{n-k} = \binom{n}{k} \left(\frac{1}{2}\right)^n = \frac{\binom{n}{k}}{2^n} \tag{2.3}$$

Interestingly, when $k = \frac{n}{2}$, the maximal binomial coefficient value, it can be shown that $\binom{n}{k}$ is $\Theta(\frac{2^n}{\sqrt{n}})$. Thus, $P(k)$ is $\Theta((2^n/\sqrt{n})/2^n) = \Theta(1/\sqrt{n})$. However, choosing the minimal non-zero binomial coefficient value as $k = 1$ or $k = n$, yields that $\binom{n}{k}$ is $\Theta(n)$, and thus $P(k)$ is $\Theta(\frac{n}{2^n})$. This implies that $P(k)$ is $O(1)$. This extreme shows how vanishingly small it is to get near perfect matches in strings of such length. Indeed a perfect match has a probability of 0 as $n$ approaches infinity.

Furthermore, as the expected value of the Binomial Distribution is $\mu = \frac{n}{2}$ and the standard deviation is $\sigma = \sqrt{n(1/2)(1-1/2)} = \frac{\sqrt{n}}{2}$, the vast majority of possible vectors will be centered around the mean as per an almost Normal Distribution, for values approaching $n = 2^{14}$ or more. The takeaway, is that it is almost impossibly unlikely for two hypervectors to be similar by more than just a handful of standard deviations than they are to random hypervectors. Thus, similarity becomes a metric of significance. If $A \oplus B$ is taken to be the bitwise exclusive-or (XOR) of two binary vectors $A$ and $B$, then the $\oplus$ operator becomes a measure of similarity. As a result, Hamming Distance is used to measure the distance between two binary hypervectors:

$$\text{Ham}(A, B) = |A \oplus B| \tag{2.4}$$

where the magnitude represents the count of 1s in the resultant XOR. For long vectors, this can be normalized by the length $n$ as:

$$\text{Ham}_n(A, B) = \frac{|A \oplus B|}{n} \tag{2.5}$$

#### 2.1.2.2 Algebraic Interpretation

To speak about an *algebra* on hypervectors, we must first define a *field*. A field is a set equipped with an "addition" and "multiplication" operation. Thus, we can think of a field as a tuple $(F, +, \cdot)$ where $F$ is the set over which we apply our field, $+$ and $\cdot$ are binary operations $F \times F \mapsto F$. A field must satisfy the following *field axioms* for all $a, b, c \in F$:

1. **Associativity:** Both binary operations must be associative, so

$$a + (b + c) = (a + b) + c,$$

$$a \cdot (b \cdot c) = (a \cdot b) \cdot c$$

2. **Commutativity:** Both binary operations must be commutative, so

$$a + b = b + a,$$

$$a \cdot b = b \cdot a$$

3. **Additive and Multiplicative Identity:** Both operations must have an identity element, referred to as $0, 1 \in F$, such that

$$a + 0 = a,$$

$$a \cdot 1 = a$$

4. **Additive and Multiplicative Inverses:** Both operations must have an element, referred to as $-a, a^{-1} \in F$, that inverts them such that

$$a + (-a) = 0,$$

$$a \cdot a^{-1} = 1$$

5. **Distributivity:** The two operations must distribute, so

$$a \cdot (b + c) = (a \cdot b) + (a \cdot c)$$

Equivalently, one could say $(F, +, \cdot)$ is a *commutative ring* in which $0 \neq 1$ and all non-zero elements are invertible under multiplication.

If we set $F = \{0, 1\}$, and allow $+$ to be the Consensus Sum $+_C$ and $\cdot$ to be XOR, as in our thought experiment, we form what will be referred to as an *approximate* field. We give it this name to distinguish it from an unrelated, but similar concept called a $near$ field. We differ in that we are approximately distributive, not exactly, while a near field is distributive on one side of the multiplication and not the other. Additionally, our approximate field only approximates associativity in the Consensus Sum, as a different grouping of the sum could yield differences due to the tie-breaker and thresholding operations. Since $F$ only contains two elements, an approximation that differs from the expected result causes a complete destruction of information. This yields noise, much like a random variable may for a Bernoulli Trial.

Normally, binary bit fields equipped with XOR and regular addition form the Galois Field of 2 elements, $\mathbb{GF}(2)$. In such a formulation, XOR is typically $+$ and is defined as a congruence over a ring of two field elements, such that $a + b = a + b \pmod 2$. Multiplication is as in standard arithmetic. When applied to vectors of length $n \in \mathbb{N}$, $\mathbb{GF}(2)$ is extended from a field to a vector-space of possible

elements in the set $\{0,1\}^n$. This gives rise to an algebra, $(\{0,1\}^n, +, \cdot)$, where now the operations $+$ and $\cdot$ are applied to each component of the vectors.

Note that in the case of our approximate field, we have modified addition, $+$, to be Consensus Sum, $+_C$, causing it to produce noise. Thus, extending our field to the algebra $(\{0,1\}^n, +_c, \text{XOR})$ creates an *approximate* algebra, that creates noise as well. This noise occurs when algebraic manipulations do not align as they should in $\mathbb{GF}(2)$. Since there are only two elements in $\mathbb{GF}(2)$, all information is destroyed in noisy regions of any computation in this approximate algebra. Thus, dense binary hypervectors are approximately distributive, down to the noise produced under the particular grouping of Consensus Summations in our approximate algebra. Formally, this means we can generate many approximate additive and multiplicative inverses and identities for any particular element of our algebra's vector-space.

Note that this noise follows our observations on dense binary hypervectors from before. As such, our approximate algebra on dense binary hypervectors is producing noisy representations of calculations. These preserve alignments of algebraic statements and produce noise wherever the statements do not align. If the vectors are random, the noise is optimally absorbed, resulting in near orthogonal hypervectors, which appear indistinguishable from random noise. If the vectors have commonalities, the results begin to take on aspects of the additive and multiplicative inverses and identities, thus becoming uniquely identifiable regardless of context. The former is useful for optimal information storage and retrieval. The latter is useful for compression and consolidation of information to foment the former on similar queries.

#### 2.1.2.3 Binding and Bundling

In HC, the digital transfer function described in Fig. 2.2 via component-wise XOR is referred to as "binding". In effect, it ties two pieces of information together in the hyperdimensional space. Likewise, the bit-wise majority vote procedure in Fig. 2.3 is referred to as "bundling". Bundling aggregates multiple pieces of information where similarity may be present and preserves it in a single instance of information. In general, binding can be thought of desiring properties similar to multiplication and bundling similar to summation. As such, HC frameworks can be thought of as approximate algebraic structures.

There are many ways to perform binding and bundling on binary hypervectors. However, we focus on the particular algebra of XOR based binding and majority vote based bundling. The bitwise XOR, $\oplus$, is trivial. It can easily be shown that it satisfies the commutative and associative properties of multiplication. However, bundling is the more interesting operation to consider. The *Consensus Sum* of vectors is defined by a majority vote for the output bit across terms for each component of the hypervectors, as described in our thought experiment. A Consensus Summation of vectors in component i is defined as:

$$C_+^i(V_1, V_2, \ldots, V_m) = \mathbb{1}_{\sum_j V_j^i > m - \sum_j V_j^i} \tag{2.6}$$

or the indicator function that the sum of 1s is greater than the sum of 0s. For even $m$, one can always add an additional random term $V_{m+1}$ as a tie breaker.

Once more, it is clear that Consensus Summation is commutative. However, it is not associative. Grouping up terms in different ways can generate different results, even when one doesn't consider

the random tie-breaker term. Furthermore, the interaction with XOR as the binding term will cause non-distributivity, from a formal standpoint. From this point on, whenever a summation is performed on hypervectors, it can be assumed this is the Consensus Sum. We show it either explicitly via Eq. 2.6 or by using $+_C$ to represent the addition operation. Note, the grouping of terms is important when using this formalism, and non-associativity can cause slightly different results in the best cases, and completely different results in pathological cases sensitive to the number of voting terms.

#### 2.1.2.4 Distributive Properties

We now analyze the distributive properties of our chosen binding and bundling operations of XOR and Consensus Summation. Consider a tie-broken, if necessary, Consensus Summation of hypervectors that are bound to a source vector $S$:

$$S^i \oplus C^i_+(V^i_1, V^i_2, \ldots, V^i_m) = S^i \oplus \mathbb{1}_{\sum_j V^i_j > m - \sum_j V^i_j} \tag{2.7}$$

Suppose all hypervectors are completely random. The indicator function will then randomly choose 1 or 0, which will not interfere destructively in the Hamming Distance magnitude in Eq. 2.4. Instead, it will interfere randomly across all $i$ components. The odds of two random hypervectors being closer to 0 Hamming Distance than expected then precisely follow the Binomial Distribution. As such, the distribution of 1s and 0s across the $i$ output components will also be Binomial, resulting in another random string, with expectation of $\frac{n}{2}$ bits being 1 and 0, respectively.

Consider the case in which some of the hypervectors $V_j$ are not random, and correlate to all or parts of $S$. For only those terms, the random signal in Consensus Sum at component $i$ will be nudged towards 1 or 0, whichever correlates to $S^i$. If this nudging effect overcomes enough standard

deviations - or increments of $\frac{\sqrt{n}}{2}$ - to overcome the range of variance expected in the Bernoulli Trials over $m$ random trials, then it interferes destructively with Hamming Distance. So long as this occurs frequently enough over the $n$ components to stand out in Bernoulli Trials of $n \gg m$, the Hamming Distance will drop lower than expected. This occurs as the statistical stability in the $m$ trials over the terms will be much less stable than over the $n$ components. If $m$ is small enough, even one matching vector is enough to detect this with high certainty.

#### 2.1.2.5 Data Structures

There is a natural desire to represent data structures with hypervectors, to aid the projection process of input/output data into a shared hyperdimensional space. We describe the basic methods here.

The most basic form of information in a computational system is that of a known sequence of information. Sequences model many other data structures. If you can make a sequence, you can make an array. Likewise, if you can make a sequence, you can make a list of attributes. Thus, you can also represent objects. Since all the information in HC comprises of hypervectors, the distinction between objects that are statically allocated versus dynamically is just a matter of appropriate encoding. Thus, the first data structure we describe as a hypervector is that of a sequence.

Let a sequence of $m$ elements $s_1, s_2, s_3, \ldots, s_m \in D$ from a known dictionary $D$ be represented as an $m$-tuple:

$$s = (s_1, s_2, s_3, \ldots, s_m) \tag{2.8}$$

Suppose we are given a dictionary $H_D$ such that for $S_1, S_2, S_3, \ldots, S_m \in H_D$, the hypervector

$S_i$ appropriately represents $s_i$ for all $i$. Furthermore, suppose we have a dictionary $H_P$ that contains $m$ random hypervectors $P_1, P_2, P_3, \ldots, P_m \in H_P$ such that position $i$ is represented by $P_i$ for all $i$. Then, we can represent sequence $s$ as a linear combination under our XOR and Consensus Sum algebra:

$$S = C_+(S_1 \oplus P_1, S_2 \oplus P_2, \ldots, S_m \oplus P_m) = \texttt{BB}((S_1, S_2, \ldots, S_m), (P_1, P_2, \ldots, P_m)) \tag{2.9}$$

We refer to this operation in Eq. 2.9 as a *Bound-Bundle* (BB) of two ordered sets. While this formalism clearly works for a sequence or an array, it also works for object-like data. Suppose you had an ordered set of hypervector attributes $A$ in an object type. Given the ordered set of hypervectors representing the values for each attribute, $V$, the Bound-Bundle of $V$ to $A$, $\texttt{BB}(V, A)$ represents an object being instantiated. When a random hypervector is chosen to represent the NULL object, a $V$ comprised only of null vectors would represent a non-instantiated object. Hypervectors constructed similarly to Eq. 2.9 are often broadly referred to as *records*, such as in [1].

For a set of values, where order does not matter, the positional encodings should not be included, and thus a set is adequately represented by simply the Consensus Sum of elements, as in Eq. 2.6. As summations are commutative, the order does not matter in both Eqs. 2.6 and 2.9. This presents a conundrum; how exactly would a data structure be read once it is superposed? As HC is neurosymbolic, it is the choice of the user whether a hypervector should be read in its neural form or its symbolic form. Conversion into either form demands computation. Converting symbols to neural form is straightforward, as show here. But converting from neural form to symbolic is not as clear. We can pose this process as a minimization problem across a search space of outcomes. Let the ordered tuple $(S^{(1)}, S^{(2)}, \ldots, S^{(l)})$ be such that $H_D = \{S^{(1)}, S^{(2)}, \ldots, S^{(l)}\}$. That is, this tuple contains the

unique $l$ elements in the dictionary $H_D$. Given the desire to reconstruct index $i$ of the sequence $s$, the element $s_i$ is the data associated to dictionary element $S^{(j)} \in H_D$, where $j$ is the index of the matching data. Then, $j$ is the result of the following minimization:

$$j = \text{argmin}_{S^{(j)} \in H_D}(\text{Ham}(S \oplus P_i, S^{(j)})) \tag{2.10}$$

It should be noted that reconstruction of the sequence $s$ via repeated application of Eq. 2.10 is somewhat expensive. This is to be expected, as this reconstruction is equivalent to a human being verbalizing or writing down the sequence from memory, which requires substantial effort for humans too [10]. One would only ever do this if the cognitive result needed to be communicated to another, external brain, or to be saved in normal computer memory somewhere. In AI and ML settings, this would occur at the time the user requires the model to give its prediction.

### 2.1.3 Notable Properties and Constructions

In this section we list other notable properties of HC and other useful constructions that we will refer to in future sections frequently.

#### 2.1.3.1 The Dollar of Mexico

Examining Eq. 2.10, we can see that the solution to this argument minimization is similar to performing inference. In other words, a hypervector $S$ that represents a model can be probed to find the best prediction it can make. If instead of position hypervectors we were given input vectors seen beforehand, and instead of sequences we were given class labels, then finding the best matching symbol $s_j$ is equivalent to classification of the input. If the binding types were hypervectors

representing classes $c_1, c_2, \ldots, c_m$, as $C_1, C_2, \ldots, C_m \in C$, the minimizing index $j$ would represent the best matching class among these possibilities.

This gives rise to the idea of using HC to perform inference from examples. The infamous example of "what is the dollar of Mexico" in the literature on HC is a great example of this inference problem [1]. If a system maintains information in a record of a country by recording its name, currency, and capital city, then it may have entries for the US and Mexico as follows, where all hypervectors are random:

$$U = (H_{\text{Name}} \oplus H_{\text{US}}) +_C (H_{\text{Currency}} \oplus H_{\text{Dollar}}) +_C (H_{\text{Capital}} \oplus H_{\text{WashingtonDC}}) \tag{2.11}$$

$$M = (H_{\text{Name}} \oplus H_{\text{Mexico}}) +_C (H_{\text{Currency}} \oplus H_{\text{Peso}}) +_C (H_{\text{Capital}} \oplus H_{\text{MexicoCity}}) \tag{2.12}$$

Consider the problem of trying to figure out what is the analogy to the dollar for Mexico. If we bind $U$ and $M$ together, the distribution operation results in noise terms everywhere but where we constructively distribute on the same bound vectors:

$$\begin{aligned} U \oplus M = &(H_{\text{Name}} \oplus H_{\text{US}} \oplus H_{\text{Name}} \oplus H_{\text{Mexico}}) +_C \\ &(H_{\text{Currency}} \oplus H_{\text{Dollar}} \oplus H_{\text{Currency}} \oplus H_{\text{Peso}}) +_C \\ &(H_{\text{Capital}} \oplus H_{\text{WashingtonDC}} \oplus H_{\text{Capital}} \oplus H_{\text{MexicoCity}}) +_C \\ &N \end{aligned} \tag{2.13}$$

where $N \sim B(n, \frac{1}{2})^n$, a random noisy vector. Thus, though we would expect to get nine terms in

this distribution, we only get the three meaningful matching terms and noise consolidated into one hypervector $N$. This occurs because the XOR of any two random vectors will themselves be random, and thus we simply get random interference. Since XOR is commutative and associative, Eq. 2.13 can be altered to:

$$\begin{aligned} U \oplus M = &((H_{\text{Name}} \oplus H_{\text{Name}}) \oplus H_{\text{US}} \oplus H_{\text{Mexico}}) +_C \\ &((H_{\text{Currency}} \oplus H_{\text{Currency}}) \oplus H_{\text{Dollar}} \oplus H_{\text{Peso}}) +_C \\ &((H_{\text{Capital}} \oplus H_{\text{Capital}}) \oplus H_{\text{WashingtonDC}} \oplus H_{\text{MexicoCity}}) +_C \\ &N \end{aligned} \tag{2.14}$$

where we have the matching XORs next to each other. As XOR is an involution, these cancel to the identity XOR, which itself is extraneous. Thus, Eq. 2.14 can be simplified to:

$$U \oplus M = (H_{\text{US}} \oplus H_{\text{Mexico}}) +_C (H_{\text{Dollar}} \oplus H_{\text{Peso}}) +_C (H_{\text{WashingtonDC}} \oplus H_{\text{MexicoCity}}) +_C N \tag{2.15}$$

If we further bind this result to $H_{\text{Dollar}}$, in an effort to ascertain what is the currency analogy to a dollar in the US, we can use a similar distribution strategy to obtain:

$$U \oplus M \oplus H_{\text{Dollar}} \approx H_{\text{Dollar}} \oplus H_{\text{Peso}} \oplus H_{\text{Dollar}} \approx (H_{\text{Dollar}} \oplus H_{\text{Dollar}}) \oplus H_{\text{Peso}} \approx H_{\text{Peso}} \tag{2.16}$$

Thus, we are left with an approximation of the peso hypervector. We can run this result through the

argument minimization in Eq. 2.10 across the search space of all the hypervectors used in this example as our dictionary. We will be overwhelmingly likely to find $H_{\text{Peso}}$ as our best match.

#### 2.1.3.2 The Hyperdimensional Inference Layer

Building on the "Dollar of Mexico" example, we can consider the task of trying to decipher the best matching record across multiple examples of bound variables. Suppose we have a situation like in Eq. 2.7, except there are multiple such differing $S$ hypervectors bound to the Consensus Sum of terms. Namely, let these bound hypervectors be class labels, say $H_{\texttt{Cat}}$, $H_{\texttt{Dog}}$, $H_{\text{Horse}}$, and so on. Suppose the terms in the aggregation bound to this are actually hypervectors derived by an encoding of an image. We would assume examples of each class label to be similar to each other, and thus it makes sense to group them up in aggregate. One could also keep them separate and aggregate all bound examples as in our thought experiment, but aggregating known terms in such a way is a type of *factoring* of the summation by pulling out the XOR application shared in common. This factored form produces such aggregates as in Eq. 2.7. We refer to the actually Consensus Sum aggregate as a *prototypical* example of the class, or a *prototype hypervector*, in general.

We can Consensus Sum all class-bound example prototypes into a single hypervector, which models the differences between each prototype per class. This model hypervector, which we can call $H_M$, can perform inference to give the best matching class given a novel image hypervector. Say we are presented hypervector $X$. Using the argument minimization in Eq. 2.10, we can search which class best matches $X$ by executing the minimization on $X \oplus H_M$. We refer to $X \oplus H_M$ as a *trace*, indicating that it traces the outline of a class that can be found, much like the peso hypervector in Eq. 2.16. Suppose $X$ is an image of a dog; then we are incredibly likely to find $H_{\text{Dog}}$ as the best matching

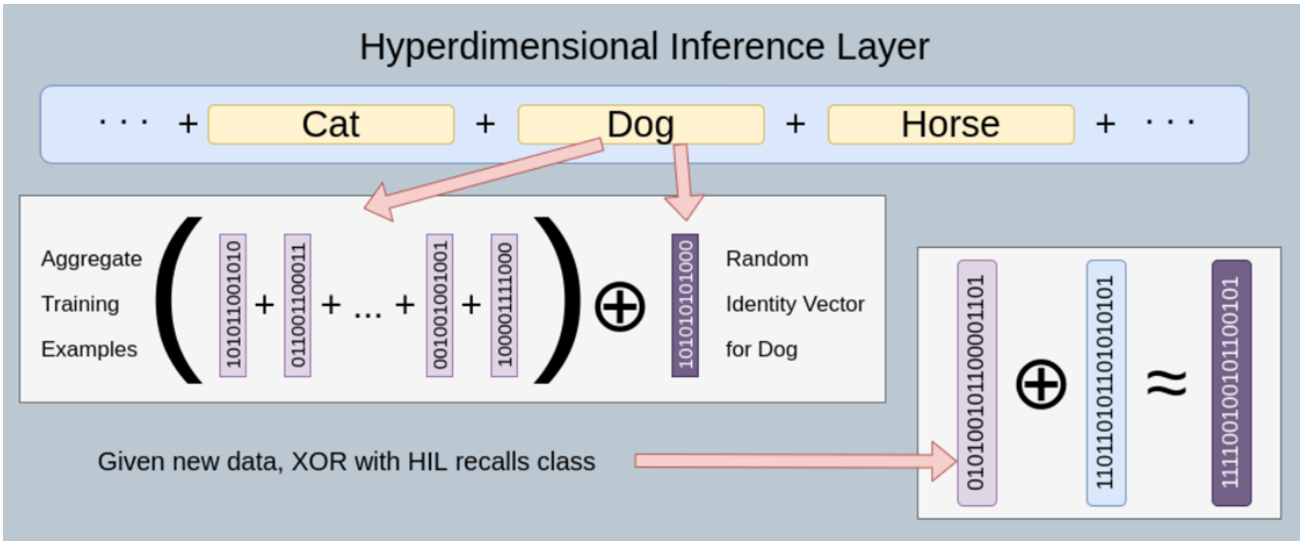



Figure 2.5: An overview of how training and inference can be performed via the Hyperdimensional Inference Layer (HIL). Training example hypervector are obtained from images and aggregated at a per class basis into prototypical hypervectors. These are bound to random hypervectors representing the class. Together, this gives a single model for predicting the presence of a class, given a test time image hypervector. Aggregation of per class classifiers forms a single HIL for predicting which class is the best fit for a new image hypervector. Given a new image hypervector, the HIL recalls the correct matching class via the argmin search in Eq. 2.10 applied to the trace of the novel hypervector and the HIL model. Boxes represent hypervectors, with color-codings to imply matches in the structure.

vector, if there are commonalities between prior dog examples in the term $H_{\text{Dog}} \oplus C_+(D_1, D_2, \ldots, D_d)$, where $D_1, D_2, \ldots, D_d$ are the $d$ examples of dog images seen beforehand. We refer to model $H_M$ as a *Hyperdimensional Inference Layer* (HIL). This is a moniker given to such a model, due to how incredibly common its usage is in AI and ML settings for HC.

Fig. 2.5 shows the construction of the HIL visually, as well as what test time inference would look like. Colors are chosen to represent similarity in hypervectors. It should be noted how similar this process is to superposition collapse in Quantum Mechanics. By considering each class in turn during Eq. 2.10, the superposition of classifiers in the HIL is disrupted. First, by performing the trace, the best classifier is isolated by finding the least noisy signal, indicating non-random behavior. The

superposition of the prototypical class hypervector is disrupted by this, isolating features from the terms in the aggregate training examples that best match the input hypervector. This produces a non-random hypervector which will be most similar to the class vector that had the most closely matching features.

So long as the noisy representation of the hypervectors produce significant enough deviations from expected behavior of random strings, they **will** select the **best** class. Furthermore, assuming the hypervector representations of training examples sufficiently differ from incorrect class labels, this process **will** select the **correct** class. Since at hyperdimensional length of vectors, the former issue of noise is not a problem, this means that the inference power of an HIL is directly correlated to the separability of the hypervector clusters of each class. Essentially, as class labels can be random, and so could dictionary elements involved in constructing hypervectors, the only non-random source of information in the HIL process is solely the structure of the raw input data, when encoded into hypervector form. Thus, the structure information is preserved in superposition if there are any similarities. This is where the "learning" occurs. Learning in HC is directly performed by the process of aggregation. The inferential power is thus solely dictated by the encoding function of the raw input data, specifically in how separable the classes are under that encoding.

In Fig. 2.6, this learning process in an HIL is shown geometrically using standard, Euclidean vectors to visually show why the encoding function is so important. As a result, HC's successful application in AI and ML contexts is directly determined by how well the data can be encoded into hypervector form. It is no suprise then, that HC is becoming popular now, when auto-encoders can effectively carry out this task for any data. The HIL, as a result, performs a perfect, algorithmic reduction for inference to relying on the encoding problem. So long as encoding can be solved, HC will produce close to entropically dense HILs that will maximize classification power from the encoding.

This is a natural integration point for HC with classical AI and ML techniques.

#### 2.1.3.3 Computing Properties

From the discussion so far, it may seem like Hyperdimensional Computing operates only on neuromorphic forms of computing. However, this is not the case. Indeed, under the right assumptions, HC itself seems to be Turing Complete [11, 12]. This fact is derived in [11] by implementing a small Turing Machine [13] and emulating a cellular automaton [12] with an HC construction. Cellular Automata are known to be Turing Complete [11]. It should be noted, however, that these results simply state that HC *can* create constructions that are Turing Complete, if needed, but there is no formal proof of this fact on a foundational level in HC.

This ties in to the neurosymbolic aspects of HC; formally, any construction in HC is rooted in symbolic manipulation, and any hypervector construction can be converted to symbolic form by probing and parsing out each symbol. Thus, one can implement any "context-free" grammar in HC by strictly using random hypervectors mapped to the production rules of the context-free grammar. So, we can extend this to a conversion of any programming language into a hypervector analogue of that programming language. So long as the dictionary is established ahead of time, a formal computing framework can be constructed in HC for any established form of computation. From a formal perspective, it suffices to implement lambda-calculus in hypervector form to establish the Turing Complete nature of HC.

Furthermore, if we treat a hypervector as a rather large machine instruction for a computer, then there is a hypervector analogue for all levels of formal computation from software down to the hardware level. In classical machine instruction paradigms, such as assembly, the instruction is parsed

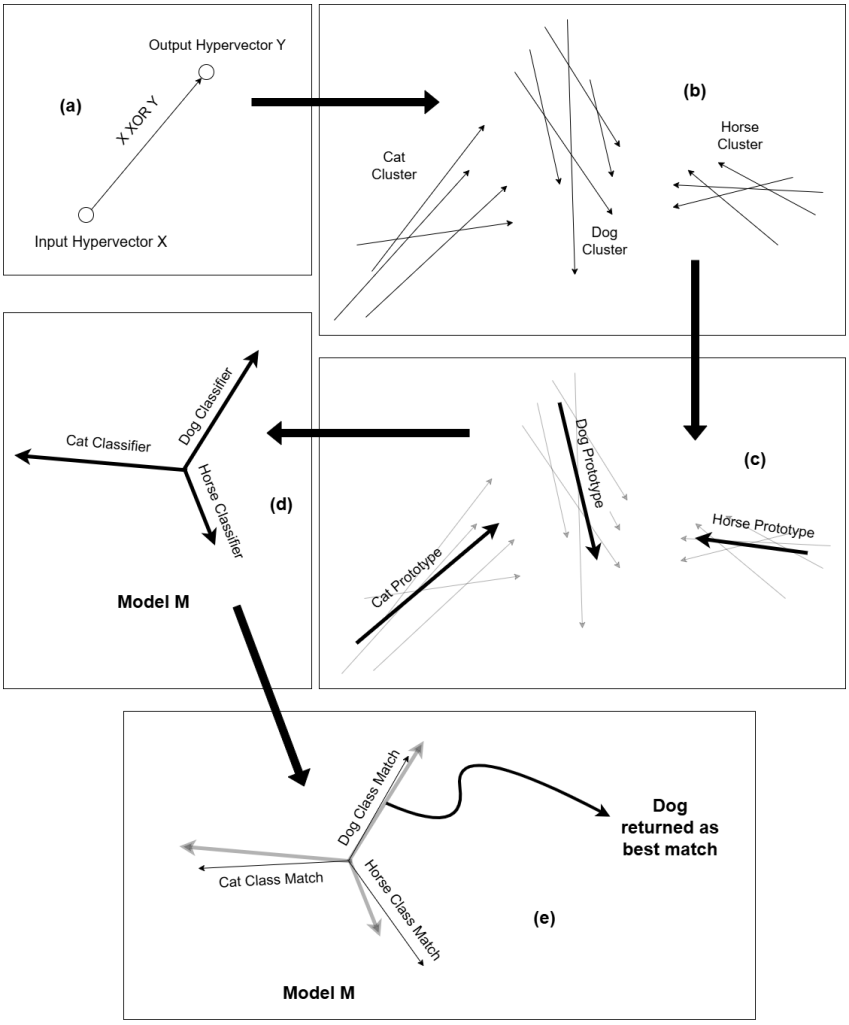



Figure 2.6: A geometric interpretation of inference through a Hyperdimensional Inference Layer (HIL). In (a), a single training example is constructed from input hypervector $X$ to output $Y$ via $X \oplus Y$, interpreting the XOR as a line pointing in space. In (b), training examples from (a) per class are shown as clusters. If there is a similarity in direction between examples for a class, in (c) the Consensus Sum will generate a prototypical hypervector for each class that is equidistant to all training examples, yet points in a very particular, non-random direction. In (d), the model is constructed for the HIL by choosing a random hypervector to represent a class and binding it to the prototypical hypervectors, thus dispersing the prototypes into random, near orthogonal directions. These are Consensus Summed into a single hypervector modeling the behavior of each cluster, called $M$, which exists in a state of superposition. In (e), when a test time example is presented and the argument minimization search from Eq. 2.10 is executed, each possible output class is considered one by one, collapsing the state of superposition in $M$ into an alignment with the prototype. The nearest match is returned as the minimizing index, which corresponds to the class predicted by $M$.

on a bit-by-bit level to construct the instruction to execute on the hardware. In hypervector form, this would require searching for the correct instruction to execute in a structured way via mechanisms like those in 2.10 on the entire vector at once. While much more expensive than classical architectures, there's nothing prohibitive about this process from the theory side. This does, however, lend itself to HC being most beneficial when multiple instructions can be carried out in parallel. The superpositional nature of HC thus gives it an affinity towards massive parallelism in hardware down to the bit-level, or perhaps the novel necessities in emerging neuromorphic hardware that do not fit well in standard computation frameworks [11].

Overall, the inefficiencies noted so far about fitting HC into standard computational frameworks underlie a **fundamental** need to radically redesign how both software and hardware is done to fully leverage the capabilities of hyperdimensional computing. At the same time, however, HC seems to be capable of handling all forms of computation that are used today, while maintaining the ability to handle those that will be used in the future. Thus, HC seems to be well suited towards the computing needs of AI and ML.

# Chapter 3: Case Studies in Hyperdimensional Computing

In this chapter, we demonstrate and discuss several practical applications of HC by outlining certain case studies and analything them at length. These are intended to showcase qualities of HC that would be attractive to AI and ML use cases.

## 3.1 **Case Study 1:** Hypervector-based Dynamical Systems [2]

In this case study, we showcase HC's capability to model arbitrary dynamical systems. This case study focuses on our work in "A Computational Theory of Life-Long Learning of Semantics and Knowledge" [2]. This paper focuses on the notion of modeling distributional semantics and general knowledge graph based information over a life-time process of learning. Primarily, as we will soon show, this is done by creating a dynamical system in which the energy is minimized over time to a resting state.

### 3.1.1 Life-Long Learning of Semantics and Knowledge" [2]

We begin with describing the work in [2] and its findings.

#### 3.1.1.1 Requirements of Life-Long Learning

We note in [2] that a life-long process of learning can be structured as depending on the following requirements, reproduced below without alteration:

1. A knowledge graph $K = (V, E)$, where $V = \{v_1, v_2, \ldots, v_{n_V}\}$ is a list of $n_v$ vertexes and $E = \{e_1, e_2, \ldots, e_{n_E}\}$, is a list of $n_e$ edges such that each $e_i = (v_a, v_b, w_i, c_i)$, denoting a directed edge from $v_a$ to $v_b$, with a weight $w_i \in (0, 1]$ derived from an occurrence count $c_i$, via

all outgoing edges starting at $v_a$. We assume the set $E_{v_a} \subseteq E$, the set of all outgoing directed edges starting at vertex $v_a$ is easily obtainable via a dictionary on $E$.

2. A matrix $X \in \{0,1\}^{b \times n_v}$, where the number of columns $b$ is a chosen value and the number of rows $n_v$ is the same as the number of vertexes in the knowledge graph $K$. Thus, each row $i$ of $X$ is a $b$-bit vector representation of vertex $i$ in $V$. Although the choice of binary vectors here as opposed to real ones may seem odd, the reasoning behind this will soon become apparent. In theory, there is no reason a real-valued vector can't be used here, but binary vectors give us access to some unique and useful properties, which we will expand upon in a later subsection.

3. A set of oracles $O = \{o_1, o_2, \ldots, o_{n_O}\}$, where $o_i(z) = \{e_1, e_2, \ldots, e_{n_j}\}$, where $e_k = (v_a, v_b)$, a directed edge from nodes $v_a$ to $v_b$ and $z$ is some data point (for example, a string or an image). These oracles can be thought of taking on a supervisory role to the learning process. In practice, these are trained to carry out some specific task (for example, labeling parts of speech in text), and the life-long learning agent is attempting to jointly compartmentalize the knowledge and semantics from all such oracles.

4. A dictionary of directors $D = \{L_1 : d_1, L_2 : d_2, \ldots, L_{n_d} : d_{n_D}\}$, where $L_i = (\ell_a, \ell_b)$ for some $a$ and $b$, and:

$$d_i(E_{L_i}, L) = \{(\ell_{n_1}, E_{\ell_{n_1}}), (\ell_{n_2}, E_{\ell_{n_2}}), \ldots, (\ell_{n_j}, E_{\ell_{n_j}})\}$$

Here, $E_{\ell_k} = \{e_1, e_2, \ldots, e_{n_l}\}$, where $e_m = (v_a, v_b)$, an edge from some vertex $v_a$ to some vertex $v_b$, and $E_{L_i}$ is some new set of edges that are observed in data and are tagged as $L_i$ type relationships. The input $L = \{\ell_1, \ell_2, \ldots, \ell_{n_L}\}$ to every $d_i$ is the set of all the hierarchical subsets

of our knowledge graph $K$, where these subsets are disjoint to one another and their union is equal to $K$. Thus, the $L_i$ in $D$ are sets of two identifiers for the location in the hierarchy of abstraction. These directors and their corresponding $L_i$ tag inform the system of how lower level abstractions come together to form higher level abstractions. For example, vertexes for "the" and "dog" are directed together to form "the dog", symbolically, by a particular director. The system then adds these units of knowledge into the appropriate layers. Note that the purpose of passing $L$ as input to each $d_i$ is so that the function returns only tuples whose identifiers exist in $L$.

As noted in [2], the oracles $O$ are necessary to inform our system of the correct mapping of pure data to vertices. The directors $D$ inform our system of how these vertices can be semantically co-occurring or related. The knowledge graph stores all these pieces of information, including the number of times each edge has been observed and with what relative frequency across all outgoing edges from the starting vertex. The matrix $X$ is the vector representation of our semantic space. As $L$ denotes a mutually disjoint set of sets whose union is $K$, and each node in $K$ corresponds to a row of $X$, we can also organize $X$ as sub-matrices for each $\ell_i \in L$. Thus, $X$ can be written as:

$$X = [X(\ell_1), X(\ell_2), \ldots, X(\ell_{n_L})]^T \tag{3.1}$$

Each of these matrices $X(\ell_i)$ are vectors occupying the same semantic location in the hierarchy, and one can directly measure how semantically close two rows are in $X(\ell_i)$ using a distance metric.

#### 3.1.1.2 Problem Statement

We formulate the problem as a dynamic system similar to a Spring-Mass system in physics, with a few differences and an abstraction that handles the binary nature of hypervectors. We envision initial random positions of hypervectors representing knowledge and semantics as high-tension states of connected springs and masses that need to be relaxed into a minimum energy state to properly position hypervectors from a topological perspective.

Let $X^{(k)} \in \{0,1\}^{m \times n}$ denote the current configuration of vectors at the $k$th step of optimization. Let $X^{(k)}$ be $X(\ell_i)$ for the current hierarchical layer $\ell_i \in L$ that is currently minimizing its tension. This is an $m$ by $n$ matrix of binary values, where $n$ is hyperdimensional, and $m$ is the number of vertices. Thus, our objective function and the minimization problem to solve is:

$$\text{argmin}_X \big(T(X^{(k)} + X)\big) \tag{3.2}$$

where $T$ is the total tension in our system in the configuration specified as input. The variable $X$ is a perturbation matrix that alters $X^{(k)}$ in such a way that it is minimized. Once found, $X^{(k+1)} = X^{(k)} + X$ is set as the new, current configuration of our hypervectors.

Next, we formalize how this tension can be represented mathematically for binary data present in hypervectors. This requires a geometric analogue of tension that is normally present from springs being stretched by mass and yet exerting pull to regain their initial spring length. We envision the repulsive force normally caused by inertia of masses as a sort of reverse gravity. The closer two hypervectors are, the more of a repulsive force is generated. This is referred to as a "proximal" force. Likewise, the pulling force of the "springs" in this analogy is the "connective" force.

The total tension of a system $A$ is the sum of unresolved forces in the system, which stem from two sources. The first is the force from the connections in our knowledge graph. The more connections two vertices share, with more frequency, the more similar the vectors want to be, as if the two points were being dragged together. When two vectors share no connection to each other, forming unconnected groups in our knowledge graph, or *islands*, they experience no connective force between them. The second source of force is the proximal force. The closer the Hamming Distance between two vectors, the more force is required to bring them together. Thus, a lot of evidence (connective force) is required to bring two vectors very close to each other. So:

$$T(A) = \sum_{i=1}^{m} \sum_{j=1}^{n} \max(F_{conn}(A, i, j) + F_{prox}(A, i, j), 0) \tag{3.3}$$

where $F_{conn}$ is the connective force and $F_{prox}$ the proximal force for a given vertex $i$ corresponding to row $i$ in matrix $A$, and the $j$th bit corresponding to column $j$ in $A$. Note that we are essentially looking at the forces each bit in each vertex experience due to the existence of other vertices and connections to them. Force, at a bit level, is a scalar value denoting the force on a bit to change its binary value. A positive value means it is capable of changing the bit, a negative value means the forces keeping the bit in the same position are greater than those trying to change it. Therefore, if the sum of $F_{conn}$ and $F_{prox}$ for a bit is negative or 0, no forces are unresolved. The $\max()$ operation between the sum of forces and 0 emulates this. The system reaches an equilibrium if $T(A) = 0$, meaning that there is no unresolved tension in the system, as the vectors have relaxed into a position where there is not enough force on any bit to cause it to change.

#### 3.1.1.3 Proximal Force

The proximal force for a given row vector $i$ in $A$ is defined as the sum over the inverse-squared Hamming Distances between each other vector, multiplied by that vector's weight:

$$F_{prox}(A, i, j) = \sum_{k=1, k \neq i}^{m} \frac{M_i M_k}{\text{Ham}(A_i, A_k)^2} C_{prox}(A_{ij}, A_{kj}) \tag{3.4}$$

where $M \in \mathbf{R}^m$ is a vector of weights (frequencies of occurrences) associated to each vertex. So, $M(i)$ is the weight of vertex $i$, generally based on the number of times it was observed in the data, and similarly for $M(k)$, for vertex $k$, serving as the *mass* of the bit. Note that the notation $A_i$ and $A_k$ is selecting the $i$th and $k$th rows of $A$, respectively. Likewise, $A_{ij}$ and $A_{kj}$ select the bits in their respective positions of matrix $A$. Finally, $C_{prox}(a, b)$ is a function that determines whether this force contributes positively or negatively to the bit $a$'s desire to change value, which depends on the values of $a$ and $b$:

$$C_{prox}(a, b) = \left\{ \begin{array}{ll} 1, & \text{if } a = b \\ -1, & \text{if } a \neq b \end{array} \right\} \tag{3.5}$$

So, if the bits match, the proximal force wants to push bit $a$ away- this creates unresolved force (positive). If they are different, the proximal force wants to keep bit $a$ away (negative force).

#### 3.1.1.4 Connective Force

The connective force is measured laterally across the binary bits in each vector. This means that a vector might experience differing forces across each of its bits, depending on its connections to other vectors and their corresponding bits. This is due to connective force being the sum of how much each bit of a vector is being *forced* to change, due to it not matching connected bits. Unlike the proximal force and its inclusion of Hamming Distance, there is no connection in forces between bits in the same row (part of a vector for a vertex). We express the connective force as:

$$F_{conn}(A, i, j) = \sum_{k=1, k \neq i}^{m} M_i W_{ik} C_{conn}(A_{ij}, A_{kj}) \tag{3.6}$$

where $A$, $i$, $j$, $k$, and $M$ are defined as in the proximal force, and $W$ is a matrix of weights on directed connections from a vertex (row) to another vertex (column). Thus, $W_{ik}$ is the number of times a connection has been observed between vertex $i$ and $k$. As before, $C_{conn}$ serves a similar purpose of deducing whether the force is unresolved or not, and is defined here as:

$$C_{conn}(a, b) = \left\{ \begin{array}{ll} -1, & \text{if } a = b \\ \\ 1, & \text{if } a \neq b \end{array} \right\} \tag{3.7}$$

So, if the bits are the same, the force is resolved as bit $a$ is already what $b$ is (justifying the connection between them), but if the bits are different, the force is unresolved as $a$ *wants* to be the same as $b$, but isn't. Note that $C_{conn} = -C_{prox}$, as the connective force pulls bits while the proximal force pushes them away. The total force $F_{conn}$ is an effective 'tug-of-war' between connected components of each

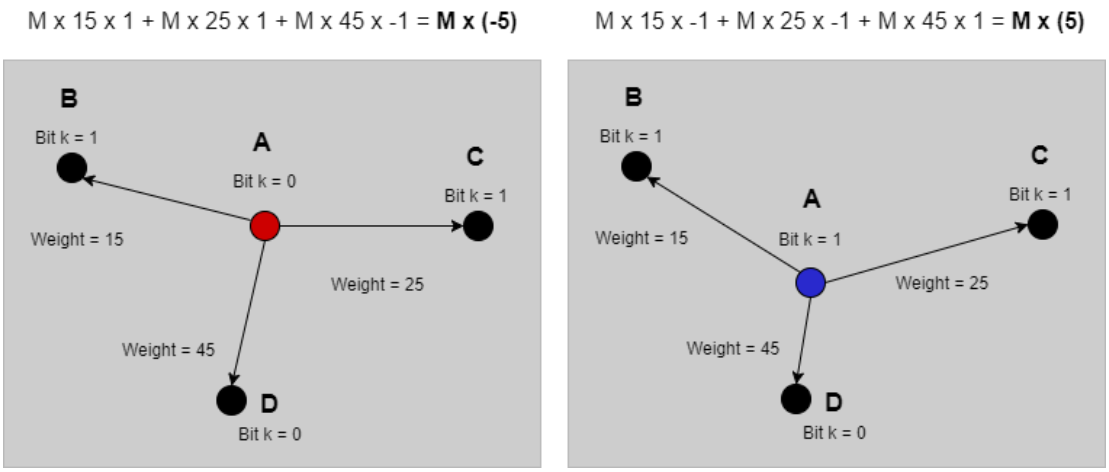



Figure 3.1: An example of how the bit value of a vector can affect the connective force. The red vertex on the left has a negative connective force, and thus does not move, while the blue vertex on the right has a positive force, and wants to move closer to D by changing its bit value.

vertex $i$. Each bit of $i$ either wins (stays the same) or loses (must change as force is positive). Figure 3.1 shows a Euclidean mock-up of how the process works. The blue vertex wants to move to a closer position to D because its bit value differs, even though it won the 'tug-of-war'.

#### 3.1.1.5 Total Tension

Given our definitions in Eq. 3.6 and Eq. 3.4, we can write the tension function as:

$$
\begin{aligned}
T(A) &= \sum_{i=1}^{m}\sum_{j=1}^{n} \max(F_{conn}(A,i,j) + F_{prox}(A,i,j), 0) \\
&= \sum_{i=1}^{m}\sum_{j=1}^{n} \max\left(\sum_{k=1,k\neq i}^{m} M_i W_{ik} C_{conn}(A_{ij}, A_{kj}) + \sum_{k=1,k\neq i}^{m} \frac{M_i M_k}{H(A_i, A_k)^2} C_{prox}(A_{ij}, A_{kj}), 0\right) \\
&= \sum_{i=1}^{m}\sum_{j=1}^{n} \max\left(\sum_{k=1,k\neq 1}^{m} M_i W_{ik} C_{conn}(A_{ij}, A_{kj}) - \frac{M_i M_k}{H(A_i, A_k)^2} C_{conn}(A_{ij}, A_{kj}), 0\right) \\
&= \sum_{i=1}^{m}\sum_{j=1}^{n} \max\left(\sum_{k=1}^{m} M_i C_{conn}(A_{ij}, A_{kj}) \left[W_{ik} - \frac{M_k}{H(A_i, A_k)^2}\right], 0\right)
\end{aligned}
$$

Thus, overall:

$$
\text{argmin}_X(T(A)) = \text{argmin}_X\left(\sum_{i=1}^{m}\sum_{j=1}^{n} \max\left(\sum_{k=1}^{m} M_i C_{conn}(A_{ij}, A_{kj}) \left[W_{ik} - \frac{M_k}{H(A_i, A_k)^2}\right], 0\right)\right)
$$

where $A = X^{(k)} + X$. The result of this is a minimizing perturbation matrix $X$, and the new configuration becomes $X^{(k+1)} = A$. Rewriting $C_{conn}$ as just $C$, for convenience:

$$
\text{argmin}_X \sum_{i=1}^{m}\sum_{j=1}^{n} \max\left(\sum_{k=1}^{m} M_i C(A_{ij}, A_{kj}) \left[W_{ik} - \frac{M_k}{H(A_i, A_k)^2}\right], 0\right) \tag{3.8}
$$

#### 3.1.1.6 Minimization

The minimization problem posed by Eq. 3.8 may seem difficult at first. However, it lends itself well to Monte-Carlo-esque solutions or Simulated Annealing. We simply change each bit by the likelihood posed by the force generated on it to simulate Monte-Carlo, or select a certain number of bits to change by the most likely to change for Simulated Annealing, for each hypervector. One

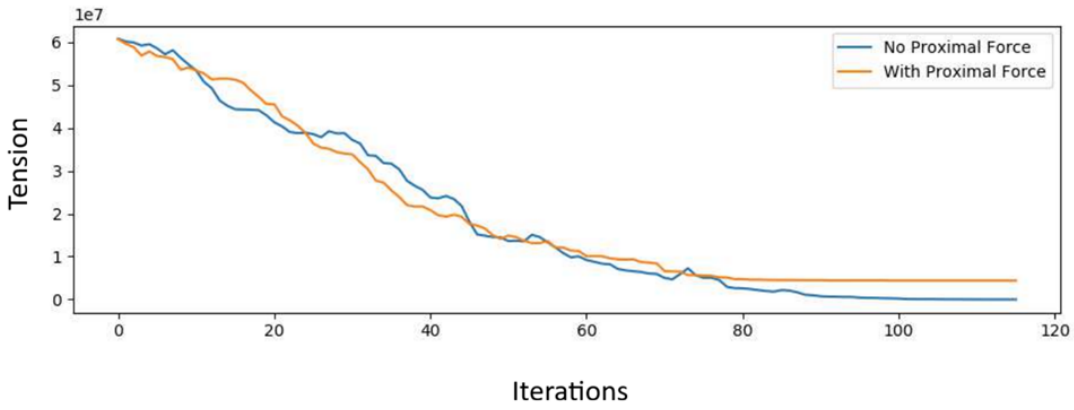



Figure 3.2: Example minimization per random row of a randomly connected 50 node graph's binary vectors via the greedy method. Without proximal force it reaches 0.

simplifying assumption, to avoid the Many-Body Problem in physics, which would require measuring the distance for every node to get the proximal forces, is to only measure proximal forces for nodes with an established connective force. Other nodes are unlikely to be close enough to be affect by reverse gravity.

Fig. 3.2 shows how a greedy Simulated Annealing strategy minimizes the tension, quite efficiently. If there is no proximal force, a tension of 0 is reached very quickly. With proximal force, a non-zero equilibrium is reached equally quickly. We can see from this minimization why proximal force is required as a regularization technique. Without proximal force, the system is allowed to collapse into a singularity, and the lowest energy solution eventually realizes this. Proximal force allows oscillatory tension in the dynamic system, which prevents the tension from ever reaching 0. We can interpret this as starting with a system of connected springs and masses in random positions, which would create a massive amount of tension, and then releasing all of the masses fixed into place. These spring into a low energy state rapidly, but the structure of the forces prevents complete collapse. This is reminiscent of orbits of celestial bodies, which reach some equilibrium but remain in motion

with unresolved energies (e.g., elliptical orbits). Another example would be complex pulley and rope systems that reach equilibrium in forces by slowly sliding into the formation that minimizes forces.

### 3.1.2 Analysis

This case study highlights how easy it is to construct hypervector analogues for dynamical systems. The formalism presented here is quite arbitrary and could be replaced by other analogues than a pseudo-Spring-Mass system. Furthermore, the minimization strategy is also quite arbitrary, lending itself to stochastic modeling of changes in the dynamical system over arbitrary steps.

Despite being trained on synthetic data, which can be quite contradictory for knowledge graphs, minimization is achieved rapidly. Recall that we can interpret the learning process in HC through HIL-type constructions as a system of linear equations, with coefficients being binary. Classically, these types of problems can be solved exactly through methods such as Gaussian Elimination when the addition operator is also classical. However, in Consensus Sum, the solution for this system of equations is allowed to be approximate through the thresholding process of the Consensus Sum. By allowing approximate solutions, the minimization process in Eq. 3.8 can rapidly develop heuristics, a central feature of HC's learning process. These heuristics take the form of approximately linear modulations in high dimensional space.

A central benefit of this process is the rapid learning of new information, even in real time. Since everything in our knowledge graph is embedded in a common, high dimensional space, data can be added and then taken into account gradually over time. This integration is of key interest to AI/ML tasks in which new information must be immediately integrated into the model, but eventually also ingested into semantic space. Much like humans can immediately integrate new information, but may

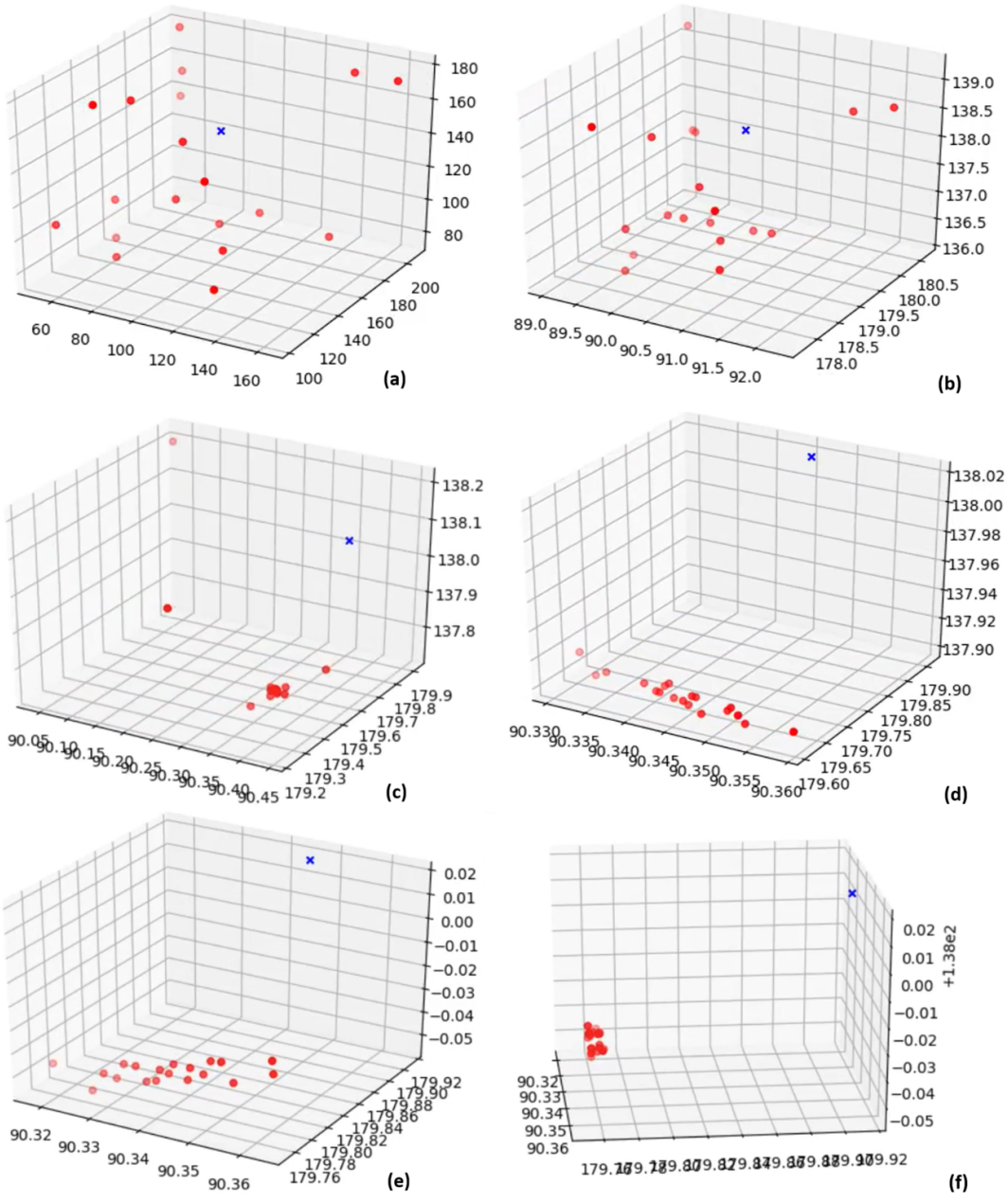



Figure 3.3: Snapshots of a video of minimization of tension in a system meant to represent a linear embedding as a knowledge graph, from our work in [3]. We can see in the 3D projection a blue "anchor point" relative to which the knowledge graph was formed. In graph (a), the system starts off as random, with high tension. Immediately in (b), the connective forces drag the connected nodes together. In (c) they appear as if they will collapse into a singularity. However, in (d), the proximal force kicks in and forces a line to form. In (e), tension reaches minimal values, enforcing order in the nodes. In (f), the line is shown from the right side, to show that it is indeed linear from the perspective of the anchor point in blue.

take time to ruminate upon the implications of it, the tension minimization serves as mechanism for developing new insight from new data.

## 3.2 **Case Study 2:** Symbolic Representation Learning with HC [4]

In this case study, we analyze HC's neurosymbolic capabilities. Namely, if HC can naturally choose between mapping information to neuronal weights or mapping neuronal weights to symbolic information, then it should be irrelevant where and in what form the symbolic data came from. This case study focuses on our work in "Symbolic representation and learning with hyperdimensional computing" [4], which studies HC's capability to understand symbolic information and maintain its correlations in hypervector form. This is done by using Hashing Networks to develop short, efficient hashcodes for complex information such as images, which maintain rankability and classification power. These hashcodes serve as analogues to classical, precise computing abstractions in symbolic form. One can think of these short hashcodes as instructions for a computer to be processed directly, intended to be very specific in what they communicate. We then study if HC can integrate hypervectors derived from these hashcodes to develop multi-perspective representations of the same symbolic information. We show that these representations have several useful properties, and can allow for consensus learning.

### 3.2.1 Symbolic representation and learning with hyperdimensional computing

In this subsection, we give an overview of the results and findings presented in [4].

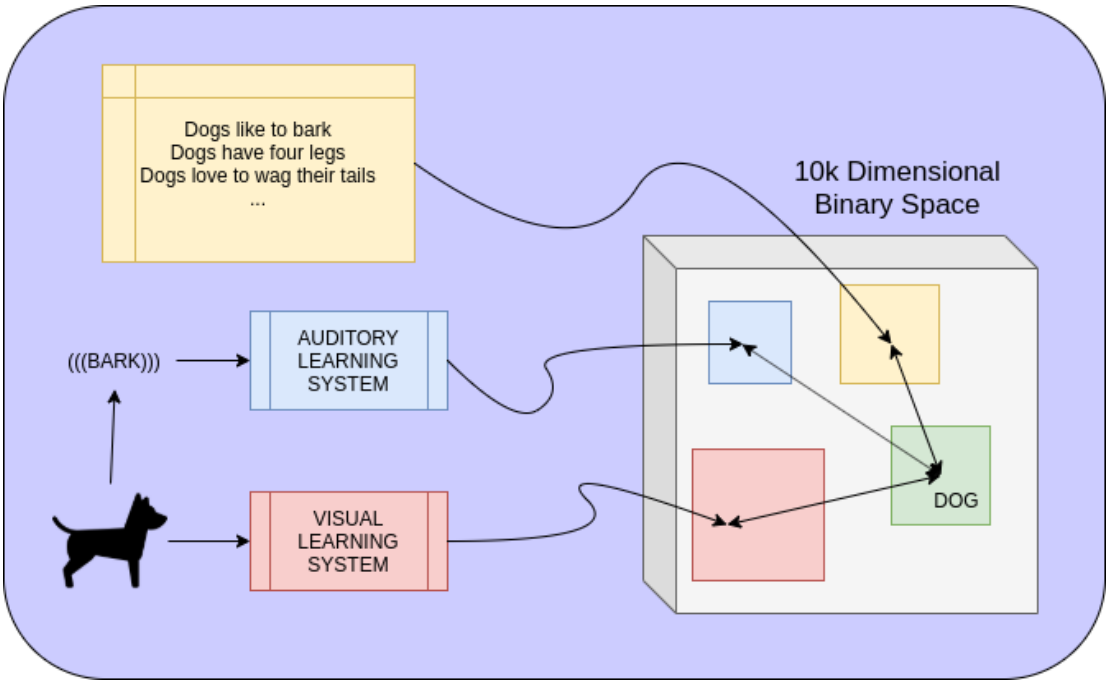



Figure 3.4: Mapping different modalities of information to the same space of long binary vectors allows knowledge of the world to coexist and combine together symbolically as well. A dog may be seen and heard, recognized by two separate data-driven learning systems. The output of each, representing the presence of a dog, is mapped to a binary vector representing the current data. The closer this mapping is to a learned representation of all dogs, the more likely it is to be a dog. In the same space, linguistic knowledge of dogs can also be mapped to symbolic representations. Combining all three modalities by purely hyperdimensional computations gives a single symbolic representation of everything pertaining to the concept of dogs.

#### 3.2.1.1 Motivation

We begin with the intuition behind this idea of multi-representational integration of symbolic data. In Fig. 3.4, we graphically show the intuition behind how multiple representation of the same information can co-exist in one hyperdimensional space. It is useful to refer to Fig. 2.6 as well to see the geometrical implication in a more familiar Euclidean space. Each modality would represent a random direction chosen for binding symbolic information. However, the specifics of encoding perceptual information into hypervector form will also choose different clusters in that direction. Thus, there would be a cluster pertaining to each sensory grounding that would be immediately different despite the same symbolic hypervector binding them. Furthermore, one can choose a more structured approach to choosing the random symbolic bindings, so that modalities are directly mapped to predetermined nearby clusters. Because the space is hypervector is so vast, we can easily choose such regions to map the clusters to without worrying about each region stepping on the others toes.

Therefore, multi-modal integration is a naturally feature of hyperdimensionality and of HC, in general. As we have discussed in the background on HC, the responsibility of maintain semantically correct positions of hypervectors is shirked to the encoding functions that produce hypervectors. Thus, in [4], we are concerned with primarily showing that this is possible to do with symbolic encodings that are more complicated than, say, assembly instructions in a computer. We study the production of generating image hashcodes and use this as suitably difficult problem to demonstrate symbolic grounding of HC. We require having multiple modalities for producing these hashcodes, so we elect to use differing neural architectures for producing hashcodes. For consistency in results and to reduce confounding factors, we demonstrate this on three vastly different hashing networks from the same authors. In general, these networks improve upon each other, but use very different architectures to

achieve this.

#### 3.2.1.2 Hashing Networks

First, we describe hashing networks. In effect, a hashing network accepts some raw input, such as an image in this case, and converts these into short binary codes known as hashcodes. The are on the order of dozens to potentially hundreds of bits or more, depending on the needs. The purpose of hashcodes are to represent the features of the data as hashable memory codes with low collision rates. In other words, the data is represented in hashcode form with minimal information required to maintain integrity of important correlations. In the case of images, there are two concerning integrities: the classification power of the hashcodes and the ability to rank relevancy of hashcodes to other hashcodes. Thus, most work in hashing networks focus on these two angles. We study three networks from the same authors behind `DeepHash`. We proceed to describe their main differences below.

**Deep Quantization Network (`DQN`) [14]:**

The *Deep Quantization Network* (`DQN`) is a hashing-by-quantization network used for efficient image retrieval [14]. The system supervises its hashing and allows statistical minimization of quantization errors from hand-crafted or machine learned features. Unlike prior traditional quantized hashing networks, `DQN` formally controls this quantization error. The system is composed of 4 main subsystems:

1. Multiple convolution-pooling layers which capture deep image representations.

2. A fully connected layer that bottlenecks deep representations and projects them into an optimal lower dimensional representation for hashing.

3. A pairwise cosine layer for learning similarity-preservation.

4. The quantization loss product that controls the quality of the hash and quantizes the bottleneck representations.

**Deep Cauchy Hashing Network (DCH) [15]:**

The *Deep Cauchy Hashing Network* (DCH) seeks to improve hash quality by penalizing similar image pairs having a Hamming Distance bigger than the radius specified by the hashing network [15]. The authors argue that hashing networks tend to concentrate related images within a specified Hamming ball due to an incorrectly specified loss function. By penalizing the network for when this happens with a pairwise cross-entropy loss based on a Cauchy distribution, the rankings become stronger. In other words, much like HC presupposes a Binomial / Bernoulli Trial distribution in binary hypervectors, the authors presuppose Cauchy distributions, which allow generalization beyond just Binomial distribution mappings. Cauchy distributions generalize in such a way as they are defined as being the ratio between two independent, normally distributed random variables with mean 0. As the number of samples grows large, the Binomial distribution approximates the Normal distribution, and so the Cauchy distribution becomes relevant to study how two Normal distributions can differ from each other. The cross-entropy allows the DCH to adjust its hashcodes for this from all angles.

**Deep Triplet Quantization Network (DTQ) [16]:**

The *Deep Triplet Quantization Network* (DTQ) further improves hashing quality by incorporating similarity triplets into the learning pipeline. By a new triplet selection approach, called *Group Hard*, triplets are selected randomly from each image group that are deemed to be "hard". Binary codes are

further compacted by use of triplet quantization with weak orthogonality at training time.

#### 3.2.1.3 Evaluation

In this section we describe the experiments performed in [4] on the three hashing networks `DQN`, `DCH` and `DTQ` [14–16]. We train the networks on pretrained features from `AlexNet` [17], which is itself trained on the `ImageNet` [18] dataset. By running the training set of ImageNet on the hashing networks, we can obtain direct hashcodes of various lengths. These are projected to hyperdimensional lengths by repetition to the correct length. For each network, we can train a HIL that will learn to understand the meaning of these hashcodes for novel classes. This is performed exactly as specified in Fig. 2.5. In Fig. 3.5, we graphically outline this pipeline, to make it clearer. By grounding the hashcodes in symbolic labels, the hypervector encodings of the images can be associated to the hypervector class labels, which are fixed hypervectors randomly selected. This, in turn, grounds the HIL symbolically as well. Thus, the overall process in Fig. 3.5 can be thought of as symbolically representing images as hypervectors from hashcodes, and then further learning symbolic prototypes which are bound to the label.

As such, the entire process relies on symbolic constructions only. If we take the hash networks to be neural analogues, then we are symbolically grounding neuronal states to language symbols, all while still being technical neuronal in hypervector form. This is reminiscent of the thought experiment described at the beginning of this thesis. Our results in [4] give a practical demonstration of this thought experiment. You can indeed, learn an analogous brain in hypervector form from another brain structure (the hashing network). Furthermore, you can also integrate these as modalities into a single hypervector space, and therefore hyperdimensional "brain".

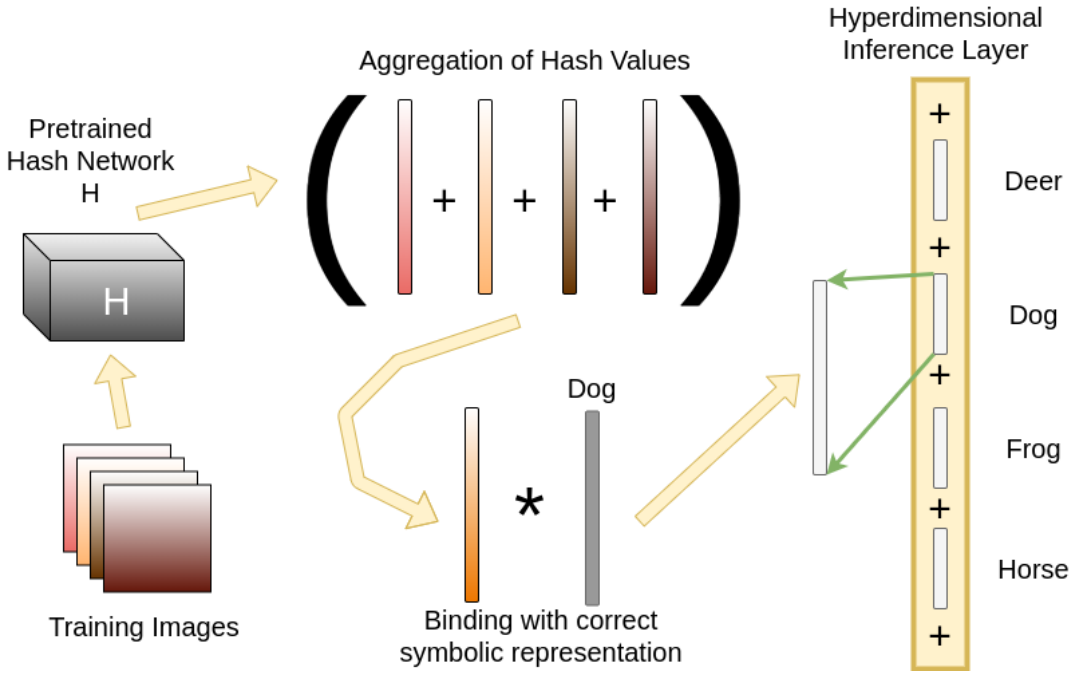



Figure 3.5: The training pipeline for a particular class "dog". First, training images are hashed into binary vectors using a pretrained network. The vectors for each image are then projected to a hyperdimensional length by randomly repeating the bits consistently. Each vector is aggregated by the Consensus Sum operation into a single vector containing all training instances for that class. A symbolic representation of the class, called "Dog" in this example, as another hypervector, is bound to the aggregated vector. This forms the association between representative images and the class itself. Once these inference vectors are computed for each class, they are aggregated by Consensus Sum into the Hyperdimensional Inference Layer, which then performs classification at testing time.

In our first experiment, we test how well the HIL analogues derived from Hashing Networks absorb the representational power of the hashcodes the networks produce. We test this effect on two datasets. The first is `CIFAR-10` [19], to test classification power. The second is `NUS-WIDE-81` [20], which is designed more to test web annotation and retrieval. Fig. 3.6 shows the F1 score on `CIFAR-10` for each network in the left column - the general accuracy of the actual Hashing Network versus the prediction from the learned HIL analogue over epochs of training in the network. At each epoch, we freeze the Hashing Network and train an HIL on the current hashcodes produced by the network. This allows us to track the progress between the network and the HIL. Immediately, it is obvious from the figure that the HIL is performing at a much higher accuracy early on in the training. Eventually, these two performances meet once the Hashing Network converges. However, it is interesting to note that the HIL seems to "arrive" at a more optimal performance immediately, producing a bootstrapping effect. Furthermore, Fig. 3.7 has similar results for `NUS-WIDE-81`. It differs only that the Hamming Distance erasure effect is not as prevalent as in `CIFAR-10`. Additionally, `DTQ` results are omitted for `NUS-WIDE-81`. Both differences are explained by the dataset itself. Since `NUS-WIDE-81` is designed for annotation and retrieval of images, the `DTQ` architecture is incompatible, and so those results are excluded. Furthermore, retrieval is less mappable to Hamming Distance in the HIL due to symbolic cross-talk (such as multiple symbols being present). Thus, Hamming Distance becomes a proper parameter to search over.

Furthermore, in Fig. 3.6 shows in the right column the effect of Hamming Distance on the F1 score's performance. Traditionally, a Hashing Network needs to perform a parameter search over Hamming Distance to find which distance between codes will best disambiguate classes. If the distance is too small, it may confuse one class with another. We see that, largely speaking, the HIL form ignores this completely. The Hamming Distance is instead directly proportional to the performance. One can

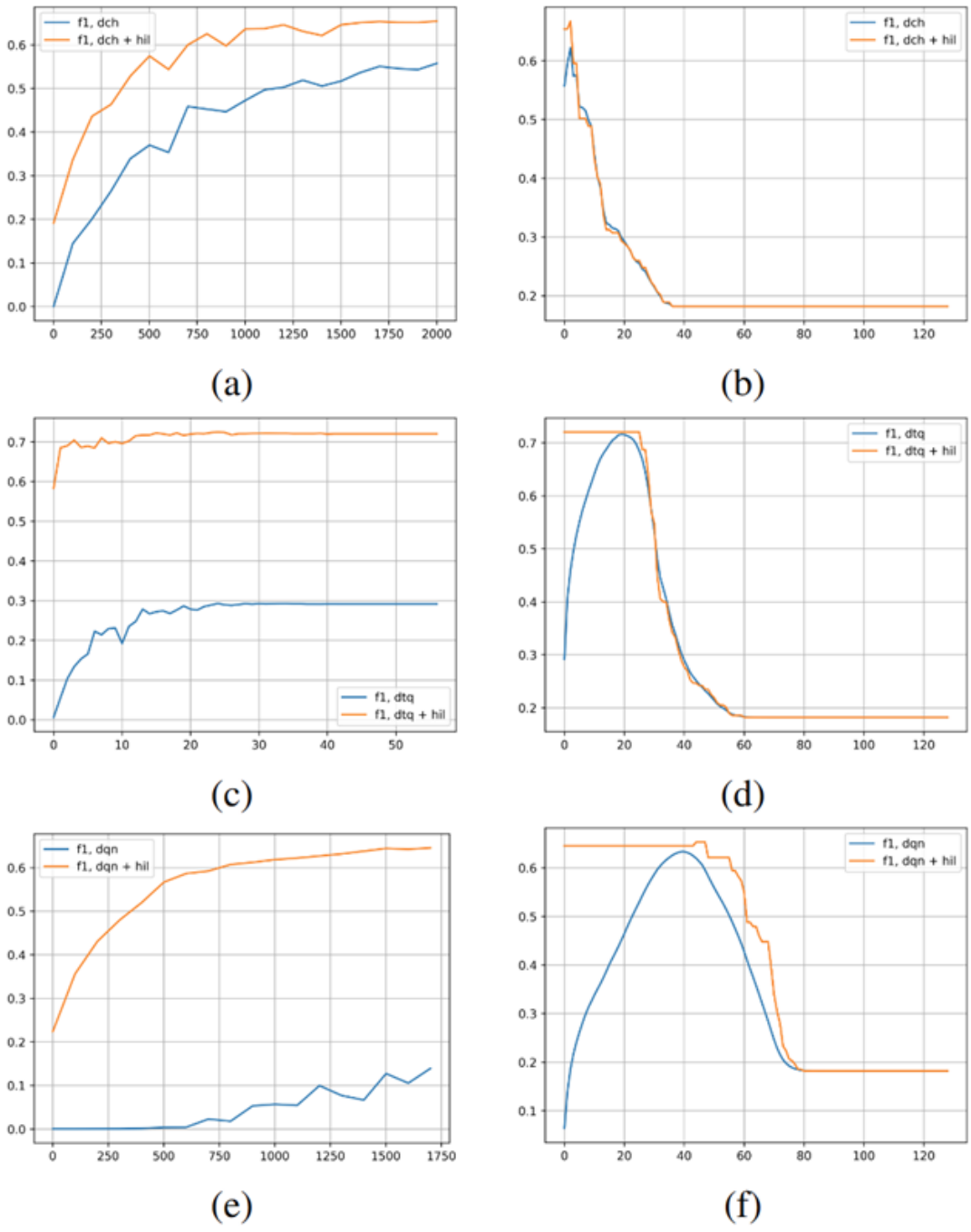



Figure 3.6: F1 Score of Hashing Networks versus the HIL analogue on `CIFAR-10`. We compare the F1 score versus epochs of training in the left column plots (a), (c) and (e). In the right column plots (b), (d) and (f), the F1 score is compared over various Hamming Distances for classification. For Hashing Networks, this is a required parameter optimization to find the typical Hamming Distance needed to tell the difference between classes. Note the boot-strapping effect of the HIL over the epochs and the parameter search erasing effect the HIL has on Hamming Distance.

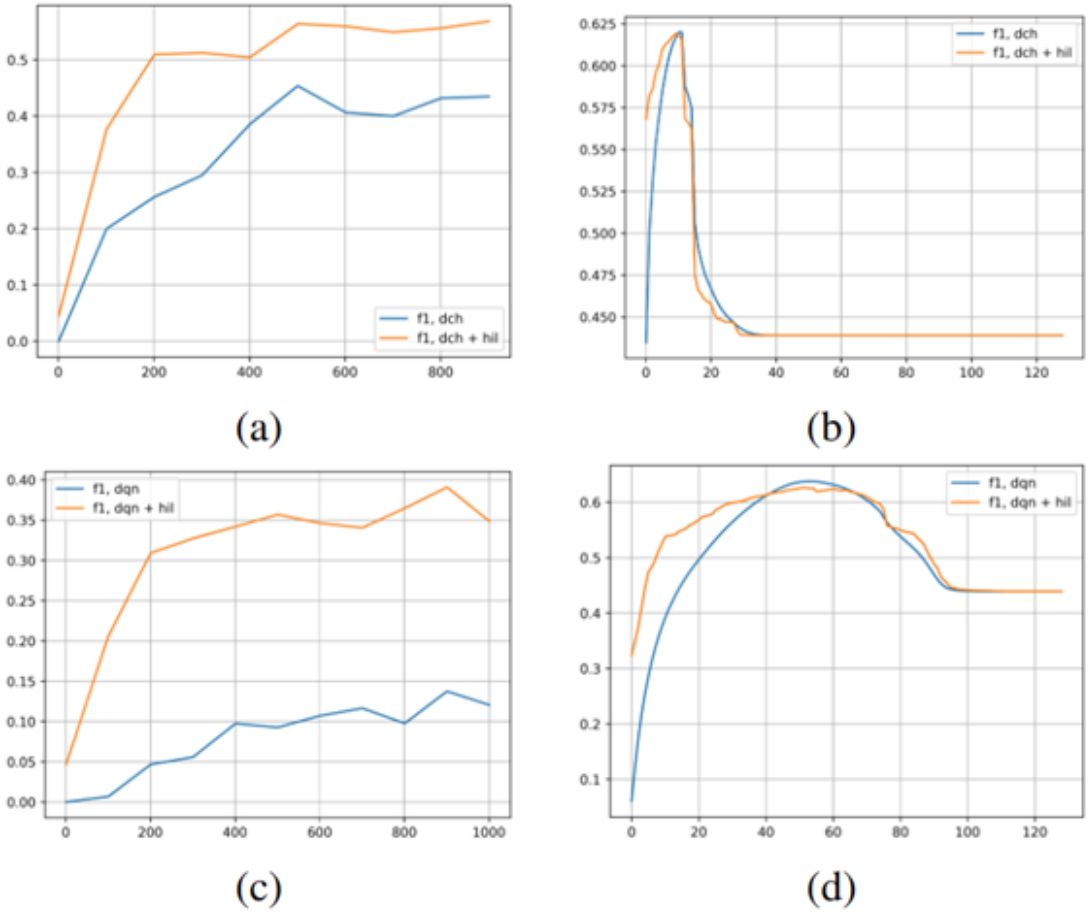



Figure 3.7: F1 Score of Hashing Networks versus the HIL analogue on `NUS-WIDE-81`. We compare the F1 score versus epochs of training in the left column plots (a) and (c). In the right column plots (b) and (d), the F1 score is compared over various Hamming Distances for classification. For Hashing Networks, this is a required parameter optimization to find the typical Hamming Distance needed to tell the difference between classes. Note the boot-strapping effect of the HIL over the epochs and the parameter search erasing effect the HIL has on Hamming Distance. This erasing effect is notable smaller on `NUS-WIDE-81` than `CIFAR-10`, likely because the dataset was designed for retrieval. Also note, the `DTQ` architecture is not applicable here, so results are omitted.

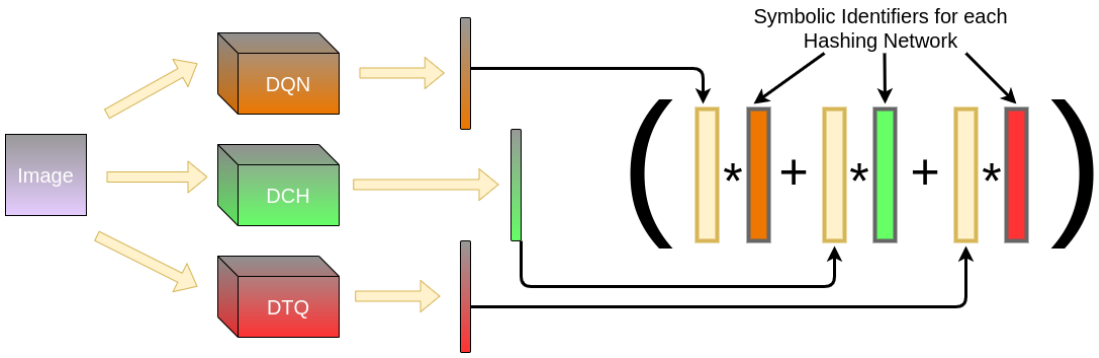



Figure 3.8: The pipeline for merging each Hashing Network's HIL into a single consensus architecture. This is done in a similar way to the "Dollar of Mexico" example shown in the background info on HC. We find the most analogous training prototypes to our novel input, and then discover the best class via the argument minimizing search in Eq. 2.10.

simply choose the smallest Hamming Distance match, and that will be the most likely one to be correct.

In a second experiment, we examine how well HC can integrate all three networks into a consensus architecture. Fig. 3.8 shows the pipeline to construct another HIL that serves as the fusion of all three networks. This is performed exactly how the "Dollar of Mexico" example works in our background section on HC. Namely, we give each Hashing Network a unique class label. Thus, any instance of a training example can be converted into a hypervector representing a record of prototypes from each Hashing Network that are bound to the network label and then Consensus Summed. At test time, when a novel image is presented, we can ask "what is the class of this novel image", forcing an analogy from our training prototypes from each network to our novel image. When we use Eq. 2.10 to probe for the correct class, each network is forced to give the best features from ever prototype, even across different networks. **Thus, the consensus architecture is choosing the best and most significant features each network is confident in.** When tested on `CIFAR-10`, `DTQ`'s performance 69% versus the consensus architecture's 79% - a full 10% increase in performance.

### 3.2.2 Analysis

This case study highlights the multi-modal fusion capabilities of HC. Namely, this is analyzed from a symbolic standpoint, rooting it in the domain of symbolic processing. The neurosymbolic capabilities of HC are manifested in how symbolic information can be freely converted and preserved in hypervector form directly. Furthermore, consensus will boost performance as the consensus HIL will allow each network to vote with features it is more confident in. If all networks are confident in the same class mapping those features, then that will be chosen. But if some are uncertain, it will prefer the most confident networks, allowing the HIL to pick and choose which features a network is best suited at resolving. This is what provides the boost in performance in consensus.

Finally, a particularly interesting result is the boot-strapping effect. In all examples shown, training of the Hashing Network in question can be concluded early, as the HIL rapidly develops good heuristics for the network early on. What can be surmised from this is that the symbolic representation gleaned from the Hashing Network is **fundamentally algebraic**. This means that HC will capture the approximate algebras much quicker than the network itself. In other words, the heuristics become more evident much quicker in hyperdimensional spaces than to the network itself. Note that this effect is more pronounced in certain networks. In `DCH`, which is already projecting the hashcodes into similar spaces (as Cauchy distributions are similar at hyperdimensional scales to Binomial ones), thus the effect of boostrapping is less pronounced. However, in `DQN`, which doesn't consider this, the boot-strapping is actually quite astonishing. This is true in both datasets tested, as shown in Fig. 3.6 and 3.7. On the other hand, `DTQ` benefits the most. In fact, after only a handful of epochs, the HIL converges on the final performance. This is likely because triplets give the HIL a far better understanding of the underlying geometrical space of the hashcodes. This is more easily absorbed into an algebra, and thus

the heuristics become apparent much quicker.

## 3.3 **Case Study 3:** "Gluing" Neural Networks Together [5]

In this case study, we extend our observations from the symbolic realm in the previous case study [4] directly to the neuronal level. Namely, we study if the neurons themselves can be embedded into a hyperdimensional space and replicate a brain structure in hypervector form. In this case, we do this by studying neural networks in a similar way to the previous case study. Can you directly absorb the features in neural network embeddings into hypervectors? And, if so, can you perform a multi-modal fusion on these as well? This case study focuses on our work in "Gluing Neural Networks Symbolically Through Hyperdimensional Computing" [5]. In this work, we study various network architectures and the ability of HC to preserve embeddings at the classification layer. These are then fused together to study how well such hypervector models can operate in consensus.

### 3.3.1 Gluing Neural Networks Symbolically Through Hyperdimensional Computing

In this subsection, we give an overview of the results and findings of our work in [5].

#### 3.3.1.1 Capturing Embeddings as Hypervectors

In order to carry out a similar experiment to the symbolic version in [4], we need a substitute for the hashcodes that were produced by Hashing Networks that is tied to neuronal representations in a neural network. We do this by recording embeddings from the output weights of the network that is presented an input, typically before the output class is determined or regression is performed, in order

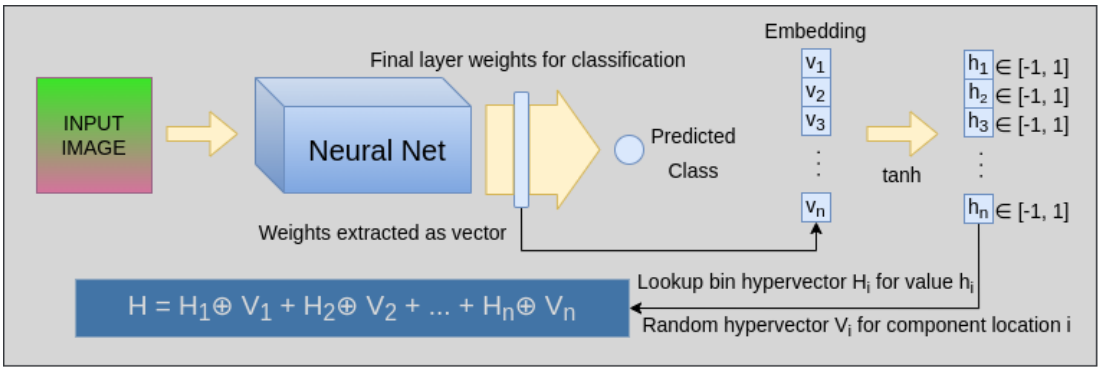
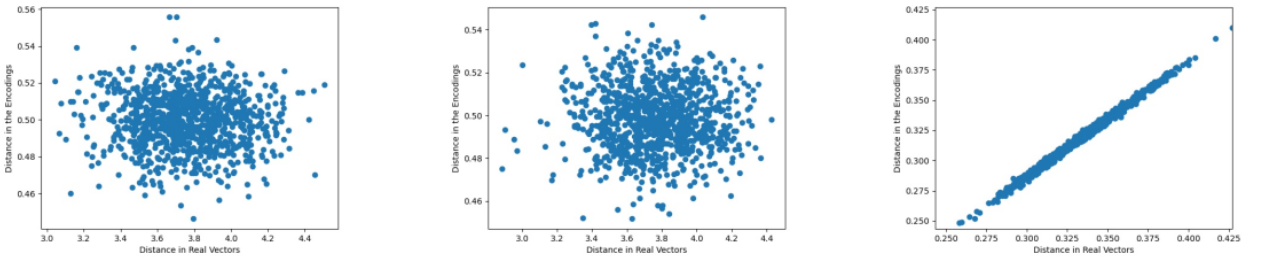



Figure 3.9: The pipeline for converting embeddings from a neural network to hypervector form. An input presented to a neural network is captured as an embedding of output weights just before classification is performed at the final layer. To normalize the weights to a predictable range, the weights are passed through a component-wise Hyperbolic Tangent ($\tanh$) function. This converts the weights to fit the range $[-1, 1]$. For each component, this range is quantized in order to compute a set of edges to denote bins. Bins are symbolically represented by a special dictionary of hypervectors. Positions are represented by a dictionary of random hypervectors mapped to the index value. Thus, a hypervector representation can be formed by the bound-bundle ($\texttt{BB}$) operation from Eq. 2.7.

to later convert these into hypervector form. As embeddings will be in the form of real-valued (more specifically, floating point) vectors, we need a consistent method to perform conversion of arbitrary embeddings to consistently sized hypervectors. Indeed this means the length of the embedding is itself a parameter.

We can achieve such a conversion by first normalizing the embedding values to a predictable range, namely using the Hyperbolic Tangent function, $\tanh$, by applying it to every component. Now, they will be in the range $[-1, 1]$. Next, we perform quantization on every component index across the training embeddings obtained from the neural network. We can use the quantized form of each component to determine the edges of bins for the data. Then, each component's value can be mapped to an appropriate bin index. Bins are assigned special hypervectors that preserve order from $-1$ to $1$. Random hypervectors are selected to represent the index of the component. Now, it



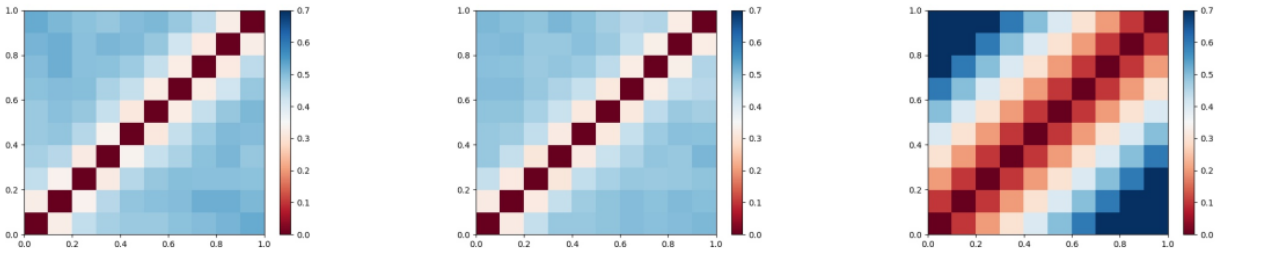

((a)) Record-based encoding. ((b)) N-gram-based encoding. ((c)) Distance-preserving quantization.

Figure 3.10: A comparison of random record based encodings (a) or $N$-gram based encodings (b) - the two most popular encoding strategies in HC - versus Distance Preserving Quantization (c). Note how much more the subplot in (c) resembles the identity function.

is relatively straightforward to hash each component's value to a hyperdimensional bin, and bind that to the hypervector representing the component index. The Consensus Sum of these produces a single hypervector that represents the embedding. This is performed by computing the bound-bundle via the BB operation in Eq. 2.7. The process of converting embeddings to hypervectors is graphically shown in Fig. 3.9.

#### 3.3.1.2 Distance Preserving Quantization

A naive implementation of the quantization step can be done by interpolating hypervectors. Suppose we want to create $k$ bins. Let $S$ and $E$ be two random hypervectors representing the beginning and end of a number line. In our case, this line would start at $-1$ and end at $1$. We can expect $S \oplus E$ to return a new vector that highlights every difference between the two hypervectors, with an expected value of the half the bits being $1$ and half $0$. Creating a dictionary of indices that contain a $1$ in $S \oplus E$, we can randomly choose $\lfloor \frac{\text{Ham}(S,E)}{k} \rfloor$ positions of $1$. These are positions that are selected to flip the bit value in $S$, creating the hypervector representation of the next bin after $-1$. We can repeat this $k-1$



((a)) Record-based encoding. ((b)) N-gram-based encoding. ((c)) Distance-preserving quantization.

Figure 3.11: A comparison of random record based encodings (a) or $N$-gram based encodings (b) - the two most popular encoding strategies in HC - versus Distance Preserving Quantization (c). Here, a heatmap of Hamming Distance is generated between each bin on a number line. Note how much more the subplot in (c) resembles the identity function across all diagonal striations, fully utilizing the range of values to disperse bin distances.

times to create $k$ hypervectors representing bins from $S$ to $E$, embedding a line into hyperdimensional space. Since floor is used to determine the number of bits to flip, we choose an extra element every time $i \cdot \frac{\text{Ham}(S,E)}{k}$ exceeds a new integer value as we create bin $i$.

Fig. 3.10 shows a scatterplot correlation to real valued distances of binning methods. We can see how such an encoding method compares to regular record based encoding with random hypervectors for the bin segments and $N$-gram based encodings for the bin segments. This is referred to as *Distance Preserving Quantization*, as it correlates the distance in real valued vectors to Hamming Distance. Furthermore, in Fig. 3.11, a heat map of Hamming Distances is generated when comparing bin $i$ to bin $j$ for all $i$ and $j$ bins, to see how well distance semantics are preserved. We can see that record and $N$-gram based encodings capture the diagonal well, but become more sporadic as the distance increases. Instead, Distance Preserving Quantization keeps the consistent striations one would expect on the diagonal entries and also perfectly utilizes the full range of Hamming Distances.

However, the calculations required in this algorithm are largely unnecessary and inefficient.

Instead, we can use binary codes to create bins that are already optimally spaced and precisely map the component value to the right bin. Since these are done on small codes that can be looked up from a code book containing the corresponding binary codes, the hypervector can be rapidly generated by stacking binary codes for each component into a hypervector. See Fig. 3.12 for a graphical pipeline describing this process. If it is important to maintain the same sized hypervector across many different embedding sizes, we can project the binary codes to hyperdimensional lengths by repetition and create the same effect, regardless of the number of components, at the cost of more computation time.

One can preserve approximate semantic distance by utilizing Gray Codes [21, 22] - also known as Reflective Binary Codes (RBCs) - to allow minimum error detection at the cost of losing magnitude of differences for large gaps. Gray Codes make only one change in bit strings for adjacent values, allowing maximal speed in implementations where switching the value is performed at a bit level for efficient representations. It should be noted that the use of Gray Codes is quite arbitrary. There exist other binary coding strategies that can be used to preserve correlations. If the goal of using a code is to preserve logical ordering and distance in a linear fashion in the whole code, alternative codes such as Thermometer Codes (shown in Fig. 3.12) - also known as Unary Codes [23] - can provide full linearity of distances. Additionally, another popular option are Scatter Codes [24], which discard the requirement of order in nodes to allow for flexibility in representations and higher capacities. These are particularly good for associative neural networks.

#### 3.3.1.3 Training the HIL

With a methodology for creating hypervectors from arbitrary embeddings in hand, the training of the HIL to approximate the embedding layer is relatively straighforward. We use the same HIL

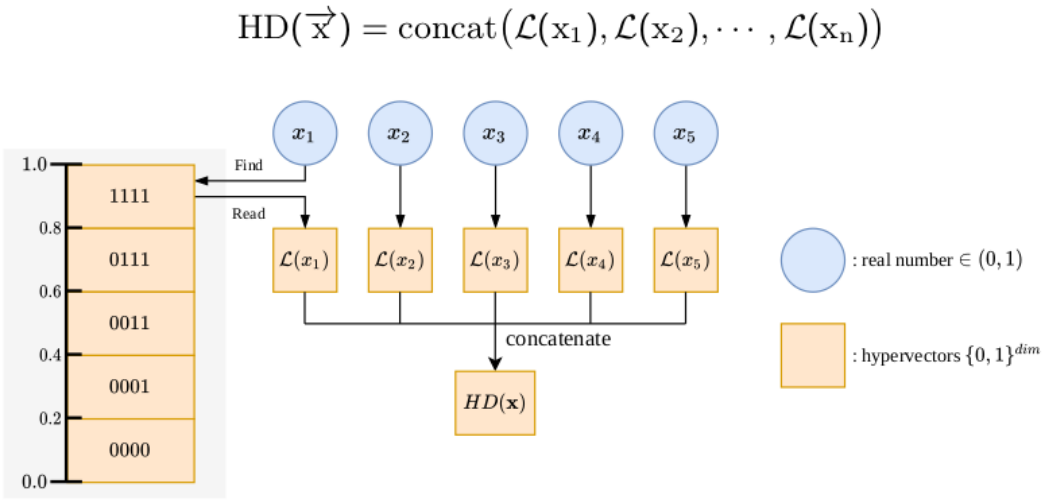



Figure 3.12: Distance preserving quantization by using binary codes to provide optimally spaced bins efficiently.

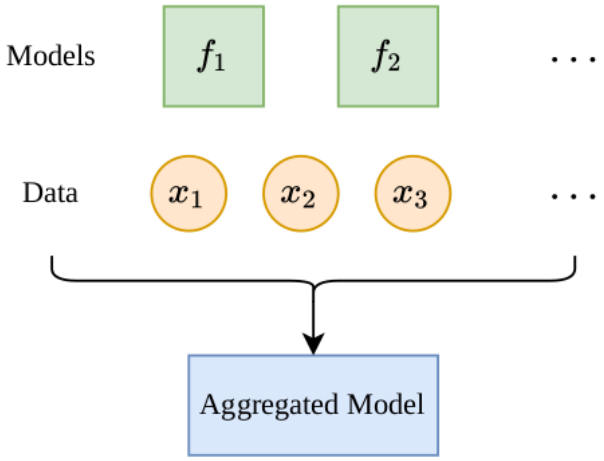



Figure 3.13: A type of consensus architecture where there is a predefined number of models that are known about ahead of time, with a dataset for each model that is also known ahead of time. Aggregating these models can be done post-hoc and at any time.

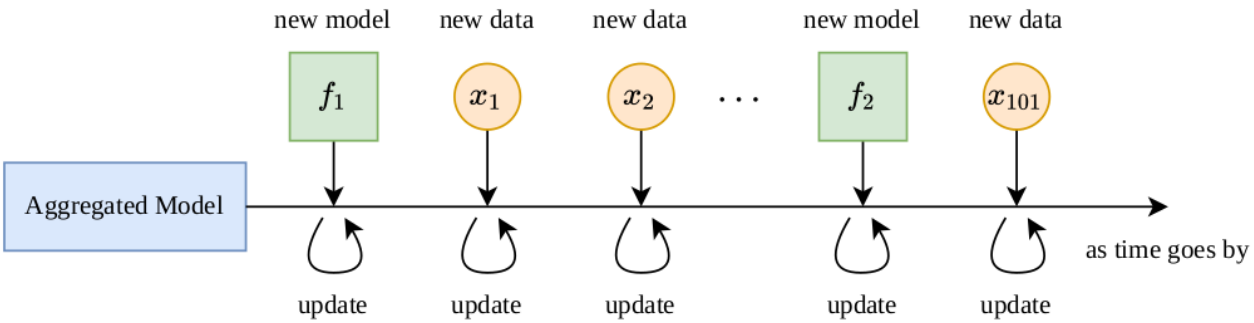



Figure 3.14: A type of consensus architecture where the number of models and amount of data is not know ahead of time. Instead, new models may pop into or out of usage at any time and data for them can arrive in an online fashion. Thus, aggregation of consensus models must be done in real time as data comes in.

training algorithm as in Case Study 2, ala Fig. 2.5, using Eq. 2.7 and Eq. 2.9. Likewise, to train a HIL for aggregating multiple differing neural networks, we can employ the same strategy used in Case Study 2, shown in Fig. 3.8.

We further build upon the HIL learning strategy and introduced in Case Study 2 by introducing the concept of *dynamic model aggregation*. We would classify Case Study 2 to be an instance of the *static aggregation model*, where the number of models and amount of data for each are known ahead of time. This is shown in Fig. 3.13. We contrast this with the dynamic case, shown in Fig. 3.14. In the dynamic case, both models and data can come into existence at any time. Models can be discontinued at any point too, and may never be seen again. Note that HC is uniquely equipped to effortlessly handle the dynamic case of learning.

#### 3.3.1.4 Evaluation

In this case study, HC is more thoroughly evaluated in comparison to other plausible methods where applicable, in order to compare the pros and cons. To carry out the experiments, we select

several pretrained networks from which to extract embeddings and convert into hypervectors. In addition to this, we also trained our own auto-encoder networks to provide novel encodings directly on the datasets used. For evaluation purposes, `CIFAR-10`, `CIFAR-100` [19] and `MNIST` [25] were used. We proceed to outline each experiment one by one.

**Experiment 1:** Gluing Encodings from Multiple Auto-Encoders on `MNIST`

In this experiment, we train five image encoder networks on `MNIST`, each trained to only handle two of the ten `MNIST` digits. This disperses the responsibility of solving the dataset to five separate, statically aggregated HIL models when we generate hypervector encodings from the embeddings of the encoders. Furthermore, we test how well classification works on increasing numbers of training examples, which represent the number of training examples shown for each digit to each encoder. Table 3.1 shows the results on this experiment for a plethora of differing strategies for aggregating embeddings. These include $k$ Nearest Neighbors [26], Linear SVM [27], Radial Basis Function SVM [27], Decision Tree [28]), Random Forest [29], AdaBoost [30], and Naive Bayes [31]. The best performing model is bolded and states how much better it is than the second best model in terms of percentages. We refer to our consensus architecture as *HD-glue*, or Hyperdimensional Glue. As the table shows, HD-glue outperforms all methods, especially so in few shot settings. This imples that HD-glue can be used to bootstrap a consensus model before the network is even done training more ideally than any other method can. This aligns with our results in Case Study 2 [4].

**Experiment 2:** Dynamic Aggregation of HC Classifiers in `MNIST`

In this experiment, we simulated the dynamic aggregation case described in Fig. 3.14. Namely, a single encoder network was used that has been trained on all of `MNIST`. Instead, the dynamic aggregation occurs at the HIL approximating the embeddings. We start with only two digits and present 100 examples of each for the HIL to learn, then measure it's performance. Then, we present another 100

Table 3.1: HD-glue performance vs. benchmark methods on the MNIST data-set

| **MNIST Data-set Benchmarking Results** | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| ***Size*** | ***KNN*** | ***L-SVM*** | ***RBF*** | ***DT*** | ***RF*** | ***Log-Reg*** | ***AB*** | ***NB*** | ***HD-glue*** |
| 10 | 33.0% | 50.8% | 50.8% | 29.5% | 43.2% | 51.9% | 28.4% | 50.8% | **56.3% (+4.4%)** |
| 50 | 53.9% | 48.4% | 48.6% | 41.9% | 45.1% | 58.1% | 24.1% | 49.7% | **63.1% (+5.0%)** |
| 100 | 54.0% | 49.2% | 54.7% | 50.0% | 55.2% | 63.9% | 27.7% | 55.0% | **64.5% (+0.6%)** |
| 500 | 66.3% | 61.3% | 70.3% | 61.1% | 62.8% | 73.4% | 22.1% | 63.6% | **74.2% (+0.8%)** |
| 1000 | 69.5% | 63.8% | 73.6% | 62.1% | 63.5% | 74.6% | 35.5% | 64.2% | **75.2% (+0.6%)** |

Results for MNIST are shown above on our benchmarks. Here, five image encoder networks were used, each trained to only handle two of the ten MNIST digits. The size column is the number of training examples shown to the neural network, thus the number of embeddings generated in the output layer. The column for Log-Reg is the logistic regression of the element-wise average of the image encoder embeddings, while HD-glue is our algorithm from the encoded hypervectors of the networks. The remaining columns are other common algorithms that run directly on the real vector embeddings, by performing an element-wise average across encoder embeddings. These include KNN ($k$ Nearest Neighbors) [26], L-SVM (Linear SVM) [27], RBF (Radial Basis Function SVM) [27], DT (Decision Tree [28]), RF (Random Forest) [29], AB (AdaBoost) [30], and NB (Naive Bayes) [31]. The best performing model is bolded and states how much better it is than the second best model in terms of percentages. As we can see, HD-glue outperforms all, especially in few shot settings - meaning HD-glue can be used to boostrap a consensus model before network are even done training. This aligns with the results in [4].

examples of each class, but introduce two more digit classes. Thus, you would have 200 example for each of the first two digits, and only 100 for the new two digits. This process continues, presenting 100 examples for each class, and introducing two new classes. Eventually, 3000 examples have been presented to the HIL across an unbalanced data set, with the original two classes having seen 1000 examples, and the last two classes only 100. Table 3.2 shows the results of this experiment.

The HIL gracefully handles the decay when new classes are introduced and doesn't experience catastrophic forgetting at any point. There are two reasons for this; one is the commutative property of the Consensus Summation, which allows the HIL to be agnostic to the order examples are presented in. The other reason is due to the innate online learning that can be done by retaining integers for counts of 1's in the Consensus Summations necessary for its construction, and the total number of examples seen so far. When inference needs to be performed, thresholding on each integer finishes the computation of the Consensus Summation, which takes constant time to achieve, proportional to the length of the hypervector. Thus, the HIL suffers no catastrophic forgetting as examples are presented, regardless of order. The same HIL is computed no matter what. One exception to this would be if training examples are grouped in Consensus Summations with different associativity. In such a situation, differences can occur either due to Consensus Summation adding a random term for tie-breakers or due to the grouping of terms changing the apparent distribution of 1's and 0's in a different association grouping. In effect, this is analogous to forming an *episodic memory*, introducing certain biases in the memories represented by each associative grouping of terms.

Table 3.2: HD-glue online learning

| **MNIST Online Learning Results** | | | | | |
|---|---|---|---|---|---|
| | **Number of classes with 100 new examples each** | | | | |
| **Class** | $\mathbf{2 \times 100}$ | $\mathbf{4 \times 100}$ | $\mathbf{6 \times 100}$ | $\mathbf{8 \times 100}$ | $\mathbf{10 \times 100}$ |
| Digit 0 | 99.9% | 99.3% | 98.1% | 94.1% | 94.3% |
| Digit 1 | 100.0% | 98.1% | 97.6% | 98.2% | 97.9% |
| Digit 2 | | 86.9% | 90.8% | 86.7% | 82.9% |
| Digit 3 | | 98.9% | 89.2% | 82.4% | 80.0% |
| Digit 4 | | | 98.0% | 95.2% | 84.4% |
| Digit 5 | | | 82.6% | 82.1% | 79.1% |
| Digit 6 | | | | 94.9% | 95.0% |
| Digit 7 | | | | 92.9% | 89.0% |
| Digit 8 | | | | | 85.5% |
| Digit 9 | | | | | 83.2% |
| All | 99.9% | 95.8% | 93.0% | 91.0% | 87.3% |

Results of simulated online learning with HD-glue on the MNIST dataset. We gradually introduce new models, each of which handles discriminating between two digits. All models are presented with 100 training examples, for each existing class - old classes retain their training. We show the performance per each class, as well as the total across existing classes. By the time all models are introduced, HD-glue has been trained on 3000 examples, unbalanced across classes. The overall performance is the same as if training HD-glue on the full 3000 examples from the beginning.

**Experiment 3:** Static Aggregation on CIFAR-10

We repeat the static aggregation test on CIFAR-10. Our results are shown in Table 3.3. In this trial, we modulate the length of the hypervector to see the effect of reducing hyperdimensionality on the results, trying 2, 4, and 8 thousand dimensional hypervectors. Predictably, HC performs worse as the length of the hypervector decreases, though surprisingly, not by much. We also study the effect of the number of encoder models used. This time, 10 models are used, and then 20. We see that the more models are used, the better HC performs. This shows that HC displays an affinity for consensus.

**Experiment 4:** Static Aggregation on Pretrained Models with CIFAR-10

In this experiment, CIFAR-10 was used in conjunction with pretrained models, namely VGG11 [32].

Table 3.3: HD-glue performance vs. benchmark methods on the CIFAR-10 data-set

| **CIFAR-10 Data-set Benchmarking Results (Note: Single Neural Network Accuracy is 54.5%)** | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **M=10** | **HD-glue** | | | **Benchmark Algorithms** | | | | | | | |
| **Size** | **d=2k** | **d=4k** | **d=8k** | ***KNN*** | ***L-SVM*** | ***RBF*** | ***DT*** | ***RF*** | ***Log-Reg*** | ***AB*** | ***NB*** |
| 10 | 52.0% | 53.4% | 54.6% (-1.0%) | 23.5% | **55.6%** | **55.6%** | 23.4% | 24.4% | 54.6% | 11.2% | **55.6%** |
| 50 | 60.4% | 61.4% | **62.3% (+1.2%)** | 58.5% | 61.0% | 35.0% | 24.3% | 39.3% | 61.0% | 17.1% | 46.0% |
| 100 | 61.5% | 62.7% | **63.7% (+1.7%)** | 60.5% | 61.3% | 34.5% | 34.1% | 41.1% | 62.0% | 10.5% | 57.9% |
| 500 | 64.1% | 64.5% | 65.1% (-0.2%) | 62.3% | **65.3%** | 24.3% | 44.2% | 49.2% | 63.3% | 28.0% | 64.3% |
| 1000 | 62.9% | 64.4% | **65.4% (+0.1%)** | 62.7% | 65.3% | 23.7% | 42.6% | 52.4% | 63.0% | 33.4% | 64.7% |
| **M=20** | **HD-glue** | | | **Benchmark Algorithms** | | | | | | | |
| **Size** | **d=2k** | **d=4k** | **d=8k** | ***KNN*** | ***L-SVM*** | ***RBF*** | ***DT*** | ***RF*** | ***Log-Reg*** | ***AB*** | ***NB*** |
| 10 | 52.6% | **56.0%** | **56.0% (+0.4%)** | 23.5% | 55.6% | 55.6% | 18.7% | 16.9% | 54.4% | 12.3% | 55.6% |
| 50 | 60.7% | 62.5% | **63.8% (+2.8%)** | 58.3% | 61.0% | 42.1% | 29.0% | 34.2% | 60.9% | 16.4% | 53.4% |
| 100 | 62.1% | 64.1% | **64.3% (+0.4%)** | 60.8% | 63.6% | 39.1% | 29.4% | 41.7% | 63.9% | 15.7% | 60.1% |
| 500 | 65.0% | 65.8% | **66.2% (+0.2%)** | 63.4% | 66.0% | 39.6% | 39.7% | 51.6% | 64.6% | 25.8% | 65.5% |
| 1000 | 64.9% | **66.2%** | **66.2% (+0.0%)** | 63.7% | **66.2%** | 34.5% | 47.0% | 54.5% | 64.7% | 41.8% | 65.8% |

Results for CIFAR-10 are shown above on benchmarks. The size column is the number of training examples shown to the neural networks, the best of which performs at 54.5% accuracy, with each example creating an embedding in the output layer of the networks. The algorithm names are the same as in Table 3.1, with NB being Naive Bayes and L-SVM being Linear SVM, shortened to conserve space, and Log-Reg being Logistic Regression. Our benchmark algorithms use the element-wise average of the real-valued embeddings. We present results for both using 10 different, image encoder networks and for 20, acting in consensus. We also present results for different dimension length of hypervectors. Bolded results are the best performing, and for our best performing HD-glue model we show how much better it is than the second best algorithm in terms of percent, and how much worse it is than the best performing one if it is not the best. As we can see, HD-glue is the best performing model overall, and is very close to the best when it is not.

Table 3.4: HD-glue performance vs. benchmark methods on the CIFAR-10 data-set with high-performance networks

| **CIFAR-10 Data-set Benchmarking Results (VGG11 #1: 92.1%, VGG11 #2: 91.1%, VGG11 #3: 90.8%)** | | | | | | | |
|---|---|---|---|---|---|---|---|
| | **HD-glue** | | | **Benchmark Algorithms** | | | |
| **Size** | **d=4k** | **d=8k** | **d=12k** | ***KNN*** | ***Linear SVM*** | ***Log-Reg*** | ***Naive Bayes*** |
| 10 | 77.4% | 80.0% | **81.2% (+0.7%)** | 28.6% | 80.5% | 80.0% | 80.5% |
| 50 | 87.5% | 88.8% (-1.6%) | 88.7% | 87.1% | 86.7% | **90.4%** | 86.3% |
| 100 | 88.9% | 89.1% | 89.4% (-0.6%) | 88.4% | 89.3% | **91.0%** | 88.8% |
| 500 | 89.0% | 90.8% | 90.9% (-0.6%) | 90.3% | 91.2% | **91.6%** | 90.3% |
| 1000 | 89.6% | 90.8% | 90.9% (-0.6%) | 90.9% | **91.6%** | **91.6%** | 90.5% |

Results for CIFAR-10 with trained VGG11 models. Much like in Table 3.1 and Table 3.3, we compare HD-glue to the benchmark algorithms. We only include competitive benchmarks here that perform similar to HD-glue. Once again, size here denotes the number of training examples presented to each algorithm. Networks utilized were 3 incarnations of VGG11 [32] obtained by training on CIFAR-10 using different initializations. Their accuracies were 92.1%, 91.1%, and 90.8%, respectively. Bolded entries are best performing, and our best performing HD-glue model includes how much better it was than the second best algorithm, or how much worse it was than the best, in terms of percent.

We train three separate, randomly initialized fine-tuned models on CIFAR-10. We also narrow the pool of benchmark algorithms to the more competitive ones for clarity. The first VGG11 model achieved a performance of $92.1\%$, the second $91.1\%$, and third $90.8\%$. This implies that, while there are differences, the networks largely have the same performance. We then repeat our experiment of statically presenting a small number of examples of each class. We also modulate by the length of the vector, trying 4, 8, and 12 thousand dimensional hypervectors. Surprisingly, HD-glue performance worse in this case, implying that perhaps a diversity of network architectures would work better than a fleet when using high-performance models.

**Experiment 5:** Static Aggregation on Differing Pretrained Models with CIFAR-100

In this experiment, we upgrade to a more difficult dataset with CIFAR-100 in order to study how increasing the number of classes affects HD-glue. Furthermore, we again modulate hypervector length

Table 3.5: HD-glue performance vs. benchmark methods on the CIFAR-100 data-set with diverse networks

| **CIFAR-100 Data-set Benchmarking Results** | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **M=1** | ***VGG11*** | ***VGG13*** | ***VGG16*** | ***VGG19*** | ***ResNet18*** | ***ResNet34*** | | |
| **F1** | 65.8% | 67.8% | 65.4% | 60.3% | 75.9% | 76.9% | | |
| | | | | | | | | |
| | **HD-glue** | | | **Benchmark Algorithms** | | | | |
| **Size** | **d=4k** | **d=8k** | **d=12k** | ***KNN*** | ***L-SVM*** | ***RBF*** | ***NNet*** | ***NB*** |
| 100 | 66.2% | 71.2% | **72.1% (+7.1%)** | 26.5% | 65.0% | 65.0% | 57.7% | 65.0% |
| 500 | 71.5% | 76.0% | **77.5% (+2.4%)** | 72.7% | 75.1% | 39.2% | 74.9% | 66.7% |
| 1000 | 75.9% | 76.9% | **77.6% (+0.9%)** | 74.1% | 76.7% | 30.8% | 76.1% | 75.8% |
| 5000 | 76.8% | 77.9% | **78.0% (+0.0%)** | 76.1% | **78.0%** | 23.3% | 76.8% | 77.6% |

Results for CIFAR-100 with diverse image classification models. Much like in Table 3.1, Table 3.3, and Table 3.4, we compare HD-glue to the benchmark algorithms. We only include competitive benchmarks here that perform similar to HD-glue. Once again, size here denotes the number of training examples presented to each algorithm. Networks utilized were VGG11, VGG13, VGG16, VGG19 [32], ResNet18 and ResNet34 [33], obtained training on CIFAR-100 using different initializations. Their accuracies are shown in the table. Note that only one of each unique mode was used, hence $M = 1$ signifies the performance (F1 score) of a single model on its own. No consensus was performed on the same models with different random initializations. Bolded entries are best performing, and our best performing HD-glue model includes how much better it was than the second best algorithm, or how much worse it was than the best, in terms of percent. As we can see, HD-glue achieved the best accuracy across all sizes.

by 4, 8, and 12 thousand dimensional lengths. This time, we also include a single layer neural network to compare as a benchmark to. Most importantly, a diversity of pretrained models are used to see what effect this has on HC. We use VGG11, VGG13, VGG16, VGG19 [32], ResNet18 and Resnet34 [33] to forms our HILs and perform consensus. Table 3.5 As we suspected with the results of experiment 4, HC prefers diverse networks to be in consensus over those with merely the same architecture but differing initializations. Note that in this case, the diverse networks all had the same embedding sizes. If embedding sizes were different, that would introduce a new stream of network diversity.

**Experiment 6:** Static Aggregation on Pretrained Models with Different Lengths on CIFAR-100

Table 3.6: HD-glue with different embedding sizes

| **CIFAR-100 Different Embedding Sizes** | | | |
|---|---|---|---|
| | **Length** | | |
| | **512** | | **256** |
| **Model** | ***VGG11*** | ***VGG13*** | ***ResNet18*** |
| **Accuracy** | 65.8% | 67.8% | 72.3% |
| **Model** | ***VGG16*** | ***VGG19*** | ***ResNet34*** |
| **Accuracy** | 65.4% | 60.3% | **74.5**% |
| | | | |
| | **HD-glue** | | |
| **Size** | **dim=4000** | **dim=8000** | **dim=12000** |
| 100 | 64.1% | 68.4% | 70.5 |
| 500 | 72.7% | **74.8**% | **75.2**% |
| 1000 | 73.1% | **75.5**% | **75.9**% |
| 5000 | 74.1% | **76.2**% | **76.3**% |

Results for CIFAR-100 with networks that have differing sizes. We present only the results of HD-glue, as other benchmarking methods cannot account for different vector lengths. Specifically, the ResNet networks have half sized embeddings (which lowers performance a bit). We mark the performance of the best individual network in bold, and also mark HD-glue models that outperform it in bold. In these instances, the HD-glue consensus outperforms individual networks. We underline the best performing HD-glue model. As we can see, for reasonable hypervector lengths, consensus performs better after a small number of examples.

In this experiment, we attempt to modulate the length of the embeddings in each pretrained network, using CIFAR-100 as our benchmarking dataset. This prevents us from using many of the benchmark methods used up until now, as they are not equipped with this capability like HC. Once again, we modulate by length of the hypervector, but this time use different embedding lengths between the VGG and ResNet models. Table 3.6 shows our results. Note that HD-Glue manages to outperform individual networks for reasonable lengths of hypervectors after just 500 examples.

### 3.3.2 Analysis

Our results verify that we can indeed encode neuronal data in addition to symbolic data. Thus, it appears HC bridges the gap between the neuronal and symbolic, as theory would suggest. Furthermore, the online learning properties were demonstrated, as well as an agnosticism to the network architecture once embeddings were converted to hypervectors. The fact that HC managed to consistently outperform other possible benchmark methods, which cannot handle embeddings of differing lengths, implies that HC is the best choice for post-hoc gluing of neural networks together into consensus architectures. All in all, HC exhibits many desirable properties in AI and ML in this case study.

## 3.4 **Case Study 4:** Hyperdimensional Active Perception [3]

In this case study, we present perhaps the most important contribution in this thesis. We show the raw efficiency of HC by applying it to the problem of Active Perception [34–36]. We show that HC can be used to form memories directly from perceptual data, to solve the problem of Ego-Motion, in order to anticipate the next movement. Furthermore, it adjust course based on these regularities, as is implied by Active Perception.

### 3.4.1 Hyperdimensional Active Perception using Neuromorphic Hardware

In this subsection, we give an overview of the results and findings of out work in [3].

#### 3.4.1.1 Active Perception

In Active Perception [34–36], an agent actively engages with its environment to gather information, rather than passively receiving sensory input. As a result, the process causes the agent to enter a feedback loop of action to perception, in which the agent selects behaviors that maximize information from the flow of data produced by those behaviors.

In the specific task of Ego-Motion, an agent attempts to determine from sensory input how it is moving through the world. This could be in the form of a velocity vector in 3D space, rotational speeds, etc. Framed as a prediction task, the agent anticipates what kind of velocity must be currently experienced. While this is not inherently a form of Active Perception, certain strategies can be employed that utilized it. In [3], we specifically solve the Ego-Motion task using hyperdimensional computing and special neuromorphic hardware that achieves the Active Perception feedback loop.

#### 3.4.1.2 Event Cameras

In Fig. 3.15, we show how a special camera, referred to as an Event Camera, differs from classic RGB cameras. Event cameras do not see the world as a dense snapshot of pixels with a specific value at all points in time. Instead, they produce "events" under specific circumstances. When a pixel of the event camera experiences a sufficiently big change in intensity over a sufficiently short amount of time, that row and column pixel position is time stamped with an event. Depending on the event camera, different information is included with this time stamp. For example, whether the intensity change was positive or negative. Events can happen at resolutions as small as microseconds, far exceeding classic framerates of regular cameras.

The basic functionality of the event camera ensures that the output stream is typically very

Figure 3.15: Sourced from Prophesee, the figure shows how a regular RGB camera compares to an Event camera, which fundamentally sees the world as events in asynchronous time that occur at specific pixel locations. Each pixel in an event camera asynchronously produces an event when pixel intensity sufficiently changes sufficiently quickly, labeling that row and column pixel with a timestamp of the event.

sparse. Events primarily occur when motion happens. If nothing in the image is moving and the agent is standing still, then few, if any, events are recorded. However, if an object moves in the view of the camera, this will produce many events near the objects, particularly on the visible outlines of the object. If the agent moves, then the entire camera may light up with events, as all visible outlines experience a homography transform. As such, event data takes the form of a cloud of event timestamps occurring on a fixed 2D slice over time. Hardware such as the event camera is often referred to as neuromorphic, in that it mimics aspects of neuronal activity in biological organisms. In this case, events represent a very specific, rapid response to change that neurons early in the brain's visual cortex are very sensitive to.

#### 3.4.1.3 Event Cameras Lead to Active Perception

If an agent uses event camera data to make an active decision in its environment, this can cause the event camera to pick up new event responses that would otherwise not happen. Even something as simple as turning to look to the left produces very specific events patterns. These can be used by the agent to deduce whether or not the response lives up to their expectations, which may prompt a new behavior in turn. This causes an Active Perception feedback loop.

In our work on [3], we use hypervectors to form memories of Ego-Motion from event data that cause an expectation of output. These are then used to predict the next motion that should occur. A smoothness constraint of motion under the rapid events of the event camera guarantee that certain regularization can occur across these memories, to constrain and alter predictions of Ego-Motion.

#### 3.4.1.4 Encoding Event Camera Data as Hypervectors

Using recent work from [37], event camera data can be binned into what are referred to as "time images". While you cannot inherently framerate event data, you can create an analogy to an "image" by projecting linearities through the event point clouds, grouping and binning them into frames that indicate what sort of motion or change in pixel intensity is occurring. In Fig. 3.16, a time image is shown with a comparison to the same image in grayscale. Time images from event data naturally sparsify the data stream, if black pixels are taken to be non-information. Indeed, we create hypervectors from time slices by encoding only non-black pixels to speed up encoding time drastically with little loss.

We already have explored most of the tools required to do this. RGB values can be quantized from a random starting hypervector representing 0 to a random ending hypervector representing 255, through gradual changes as in Case Study 3 [5]. The color channels are just a binding of the intensity with a random hypervector representing red, blue and green, respectively. The record formed by Eq. 2.9 represents a RGB sequence for a pixel event in a time image. Positional information can be encoded by a random hypervector representing the row and a random hypervector representing the column index of an event pixel. Binding these to each other gives the positional information of the event. Further binding this to the pixel intensity vector produces a representational vector for one event. To form an image, we simply perform Consensus Sum on the event hypervectors for a time image that are not just black, as per Eq. 2.6.

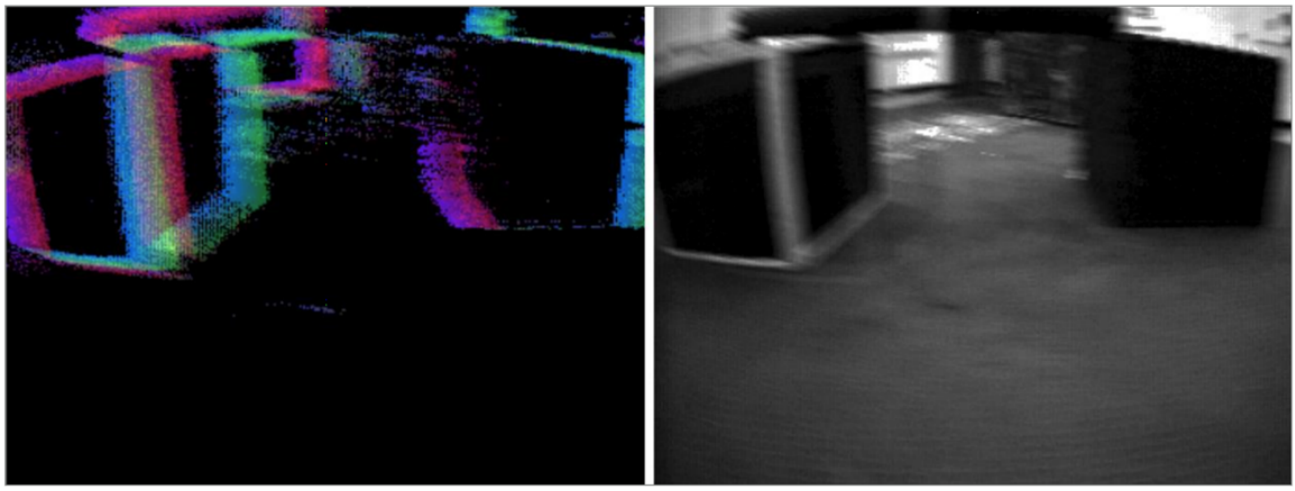


Figure 3.16: An example of what a time image might look like from event data, vs actual images, as per [37]. The coloration of event grouping indicate linear change in intensity and perhaps imply directional movement as well. For example, blue indicates older changes in pixel intensity, while green indicates more recent timestamps. Note how sparse the pixel information is in the time slice data vs the grayscale image. If black pixels represent non-information, then there are only a small portion of pixels activated at the same time.

#### 3.4.1.5 Representing Velocities

In the case of predicting 3D velocities, we can take the same approach taken with RGB values of pixel intensities. Each $x$, $y$ and $z$ component of the velocity vector functions like a color channel. The velocities are quantized in a manner similar to Case Study 3 [5]. Binning velocities under a quantization preserves their distance metric, allowing it to transfer into the hyperdimensional space as Hamming Distance. To predict a velocity, we simply learn three separate prototypes for each degree of freedom. Their values are returned by the HIL formed under this. Functionally, the solution here is no different to Case Studies 2 and 3 [4, 5]. For angular rotations and tilt and such, similar methods can be employed.

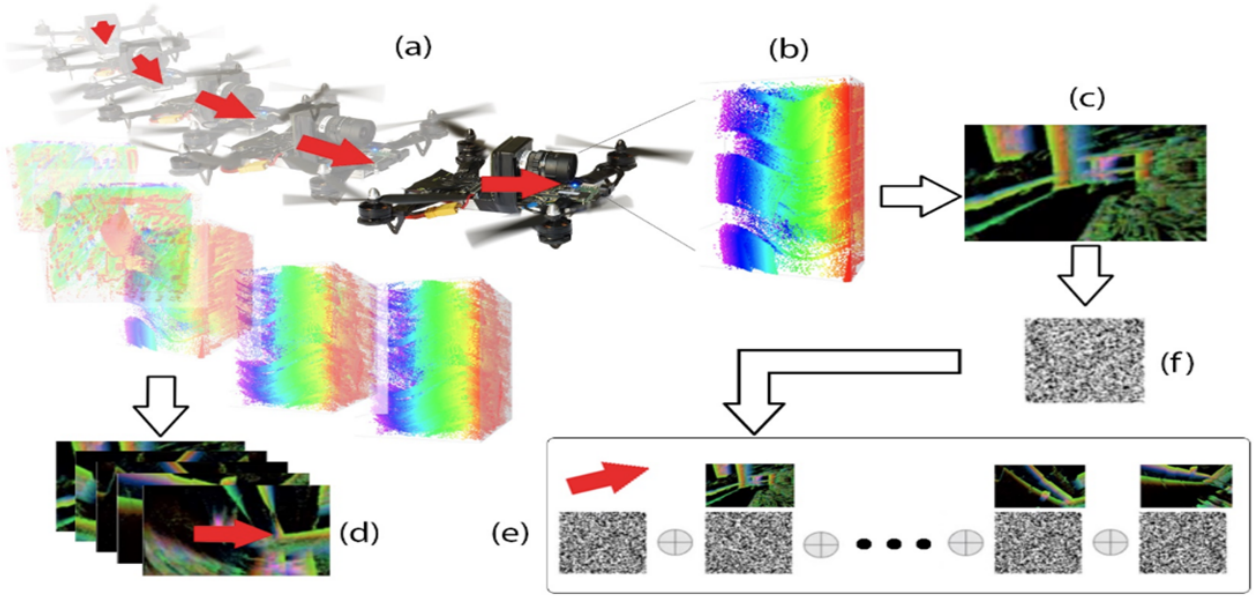



Figure 3.17: The learning and inference pipeline for Hyperdimensional Active Perception (HAP). In (a), a drone flies around and produces events on an event camera attached to it. At the same time, it experiences ego-motion in the form of 3D velocity vectors. These are recorded into training data. In (b), event data is taken and converted into time images like (c). Using the time image, hypervectors are formed by encoding the image into (f). In (d), multiple time images are assembled into "memories", referred to as "time slices", in which we bundle several time images together as a moving average. When a new time slice is shown, the last one gets popped out, and the new one is added in. These are paired with their associated velocity vectors. During training, a hypervector model such as (e) is formed by taking a hypervector encodings of the next time image, and computing the new moving average by bind it to the ground truth velocity hypervector and consensus summing it to previously seen time image and velocity bindings. Given a novel time image hypervector such as (f), we can probe a memory hypervector in (e) with it and then use Eq. 2.10 to find the best velocity bins in each degree of freedom, reconstructing the velocity vector. By using time slices, a regularization effect from previously seen time images occurs, make the predictions smoother in time.

#### 3.4.1.6 Learning Pipeline

Fig. 3.17 shows the learning pipeline for Hyperdimensional Active Perception (HAP). There is little difference to what was shown in Case Studies 2 and 3 [4,5]. But one particular difference enforces a smoothness constraint in predictions, aiding the active perception process. Time images and their velocity vectors are bundled into what are referred to as "time slices". These are memories that recall previous behavior. By forming these in a queue-like way, where the oldest time image hypervector is popped out for a long memory and the most recent one is aggregated, this regularizes the predictions of our model to smoother, less drastic velocity changes.

#### 3.4.1.7 Inference

Inference is performed by obtain the most recent time image that has been observed by the event camera, encoding that into a hypervector, and then presenting that to our memory hypervector. Much like in a HIL, this collapses the superposition of all memories to the most relevant ones. From this, we probe what the memory recall hallucinates as the next velocity vector that will occur. Before performing the next inference, we pop out the oldest time image from our time slice that is maintained over predictions, if we are starting to forget it, and then pop in our most recent time image from the prediction. This causes a regularization to the inference over which smoothing can be done.

#### 3.4.1.8 Evaluation

We test our HAP pipeline on the `MVSEC` dataset [38]. An example of what predictions look like is shown in Fig. 3.18 for Outdoor Day 1 part of the dataset. A heatmap of probabilities of selecting any particular velocity bin is shown over time in seconds. A clear line of regularized, smooth motion

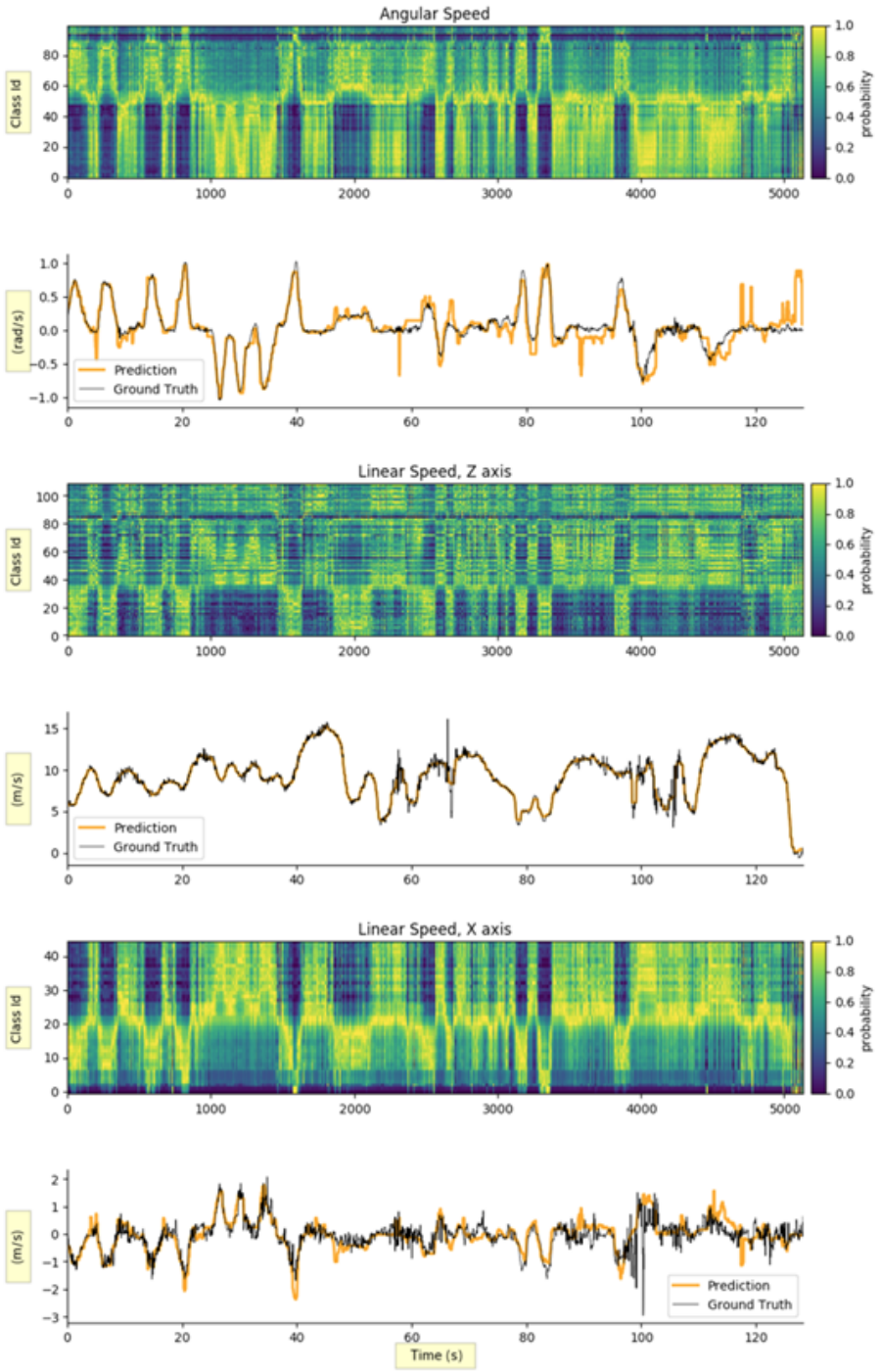



Figure 3.18: Results of HAP on MVSEC, Outdoor Day 1. Heatmaps of probabilities are presented over second in time for the angular speed, and the linear speed in $z$ and $x$ axis. Yellow is HAP's prediction, where black is the ground truth.

**Table 1. Quantitative results for the MVSEC dataset across the five subsets.** The table gives the number of frames in each subset, length in seconds, the size of the angular and linear bins used for HBVs, number of vectors in the rotation, *X* and *Z*, and the average endpoint error (AEE), average relative pose error (ARPE), and the average relative rotational error (ARRE).

| MVSEC subset | Outdoor day 1 | Outdoor day 2 | Outdoor night 1 | Outdoor night 2 | Outdoor night 3 |
|---|---|---|---|---|---|
| Frames | 5134 | 12196 | 5133 | 5497 | 5429 |
| Length (s) | 128.325 | 304.875 | 128.3 | 137.4 | 135.7 |
| Ang. bin (rad/s) | 0.02 | 0.02 | 0.02 | 0.02 | 0.02 |
| Lin. bin (m/s) | 0.08 | 0.08 | 0.08 | 0.08 | 0.08 |
| Rotation (clusters) | 104 | 101 | 40 | 74 | 87 |
| *X* (left-right, clusters) | 47 | 44 | 24 | 40 | 33 |
| *Z* (front-back, clusters) | 119 | 311 | 244 | 251 | 228 |
| AEE | 0.810 | 1.03 | 0.933 | 1.16 | 0.940 |
| ARPE | 0.122 | 0.225 | 0.243 | 0.0948 | 0.0831 |
| ARRE | 0.0994 | 0.108 | 0.0632 | 0.116 | 0.121 |

Figure 3.19: A copy of the table of results from [3] of HAP on MVSEC on all subsets.

is shown. The ground truth versus HAP's actual prediction is also shown for comparison. In MVSEC, there are 3 degrees of freedom, angular speed of the turning car, linear speed on the Z-axis, and linear speed on the X-axis. In Fig. 3.19, a table of all standard metrics on MVSEC across all subsets of data is shown.

#### 3.4.1.9 Performance

What is particularly impressive about the results of HAP, as noted in [3], is the efficiency for the performance. The results are comparable to standard convolutional network solutions that have been applied. However, the entire experiment runs on a single classical CPU. After approximately 15 seconds of training data, the hypervector memory model is already trained. The entire HAP solution occupies only several MBs of space. Training time on a single CPU of a standard modern processor takes under 1 second. Moreover, inferences run at 100s per second. All in all, as HC can learn in an online fashion, HAP can run be in real on modest hardware that is overkill even for a small drone.

### 3.4.2 Analysis

As stated earlier, this work is perhaps the most significant in the thesis. The reason for this is the sheer efficiency at which HAP runs. While at first this might seem to be due to HC entirely, the implication is actually more stunning. It is due to the neuromorphic hardware of the event camera as well. Because HC prefers sparse data already, and the event camera generates this natively, encoding can be done very rapidly and use up very little processing time or parallel computation. Most of the operations can be fit in parallel onto the registers in a standard modern CPU. What's more, the correlation between event data and Ego-Motion is so tight, that the neuromorphic hardware basically solves the encoding problem for HC. So why would something classical like a convolutional network fail to do this? It's because most modern ML solutions are not as agnostic about data representation as HC is. HC will happily learn the approximate heuristics underlying the transform from event data to velocity vector.

This highlights an important property of HC of great interest to future endeavors in AI and ML: hardware solutions for problems are easy to integrate into HC, particularly neuromorphic solutions. Dense RGB images arranged in rectangles of pixels are not how most biological organisms actually perceive the world. The computational streams are much sparser and often times rely directly on the "hardware" (e.g., how eyes are designed). This reveals an uncomfortable truth about most modern ML... it is not equipped to deal with these issues. Most neural network architectures have to be carefully hand engineered to solve novel tasks. The same goes for general AI techniques, as well. It is not so readily reducible to finding the right hardware for the problem as HC is shown to be in this case study. Furthermore, our review in [6] on HC's usage in ultra-efficient edge-computing devices highlights that this is a consistent pattern with HC.

# Chapter 4: Properties of Interest to AI/ML of Hyperdimensional Computing

This chapter is oriented around two of our works:

1. Vector symbolic sub-objects classifiers as manifold analogues [7]

2. HyPE: Hyperdimensional Propagation of Error [8]

These works discuss interesting properties of Hyperdimensional Computing that parallel many aspects of ML. Namely, [7] we show that hypervectors should have the capability to approximate any topological structure. Likewise, in [8], we show that hypervectors have their own analogous feedback mechanism to back-propagation in neural networks. We begin with an overview of the first paper and its results, then move on to the second paper. Finally we discuss some implications that stem from these two works.

## 4.1 Vector symbolic sub-objects classifiers as manifold analogues [7]

As a precaution, this work delves into aspects of Category Theory. Namely, we concern ourselves with the topic of the Topos, a Category Theoretical generalization of topological spaces. We include our Appendix from [7] in Appendix A, which gives a full, from-the-ground-up formal introduction to Topoi. We will however, try to avoid the nuances in our discussion here, and instead go for more intuitive approaches.

In this work, we explore the composability of hypervector models when they are treated as abstractions of Topoi. Namely, this implies that HC can handle any kind of geometric or topological transform by directly mapping it into the hyperdimensional space.

### 4.1.1 Category Theory

We begin by introducing Category Theory. A Category consists of:

1. Objects of the Category

2. Morphism: relationships between two objects, the target and the source

In Fig. 4.1, we show the stereotypical way of graphically showing a Category through a commutative diagram. In it, set $X$ is a source for $Y$ under a function $f$, while $Y$ is the target of $f$. Likewise, $Y$ is found to also be a source for $g$, where $Z$ is target. When composed, $g \circ f$ constructs a new function where $X$ is now revealed to be the source for $Z$ under this composition, and $Z$ is the target. This simple exercise demonstrates one thing about Category Theory; it is more interested in the relationships between objects of the Category than the Category itself. If $f(x) = x^2$, and $g(y) = \sqrt{y}$, the more interesting thing to a Category Theorist is the behavior of the composition and how the diagram commutes, and what that must imply about $f$ and $g$. Largely speaking we can think of this as caring about the actions being done, than the objects themselves, as that is often covered by Set Theory. Category Theory subsumes Set Theory and also generalizes beyond what is possible to express mathematically with just Set Theory.

### 4.1.2 Topos Theory

A Topos is a special category which aims to capture all aspects of geometry, geometric thinking, logic, and transforms. It is intended to be the end-all generalization of topological structures. If something is revealed to actually be a Topos, it is a topological structure, in other words. In Fig. 4.2, we show the basic necessities that construct a Topos. Namely, given objects $A$ and $B$, such that all $A$

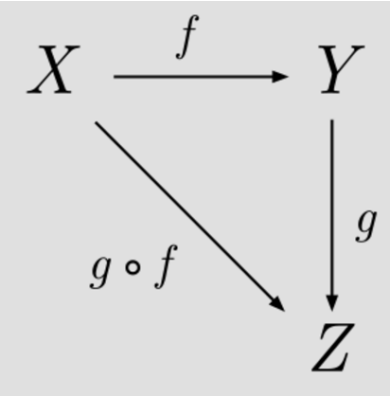



Figure 4.1: A commutative diagram showing three objects and 3 morphisms of a Category.

are also objects $B$, and given object $S$, which may or may not be an example of $A$ but must necessarily be an example of $B$, all examples $x$ of object $S$ must have their placement preserved under a transform $q$. That is, if $x$ is an example of $S$, then $q(x)$ doesn't change this fact, and all $q(x)$ are also actually objects $S$. Moreover, if $x$ is also an $A$ object, then this fact is preserved under $q(x)$ and so $q(x)$ is also an example of $A$. Actually, $q(A \bigcup S)$ must include both objects in transform $q$.

This special property preserves geometric logic across commutations in Topoi. We are allowed to make drastic jumps which may render sudden discontinuities in the spaces of $A$ and $B$, but the jump guarantees transferral of object inheritance. As a simple analogy, if one is trying to scale a mountain, you might be tempted to go straight up it in a cartoonish line that defies physical logic, but is geometrically still a geodesic (shortest path) up on that surface. However, you can reach the same result by suddenly jumping around the mountain if you are guaranteed to land at a higher elevation with every jump. Thus, both systems describe Topoi, or compositional structures that maintain a certain minimal geometry to them.

The careful observer will notice that this relationship is very dependent on $A$ being a sub-object

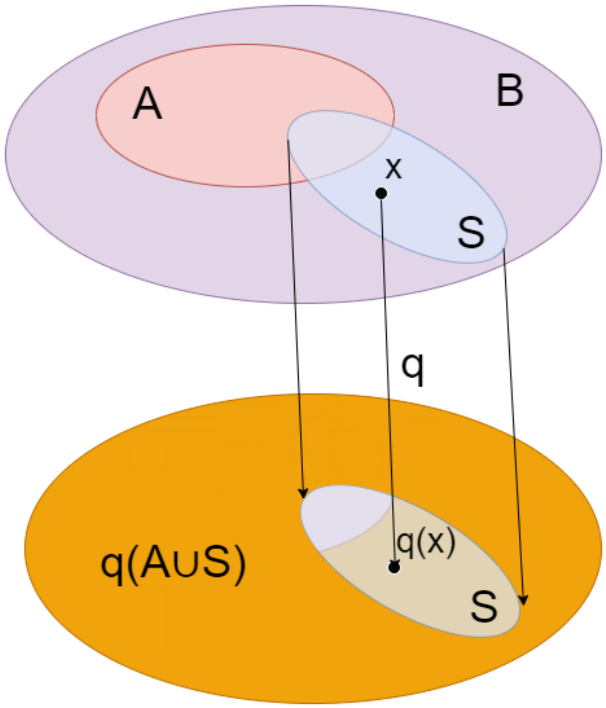



Figure 4.2: The basic building block of Topoi. A Topos must fulfill the requirements set by this diagram.

of $B$. Likewise, the fact such a transform $q$ must be required to exist also implies that there exists a strict requirement (dictated by the transform $q$) that classifies $x$ to either be *just* an example of $B$, or more specifically an example of $A$. We can leverage this fact to develop compositions of transforms such as $q$ to eventually narrow down an objects' specific sub-object classifier. In ML terms, there must be a definite way to classify $A$ vs $B$ if these are classes. This fact is very attractive for AI and ML purposes.

In [7], we explore a universal Category of Topoi in the form of arbitrarily precise classifications dictated by, you guessed it, long binary vectors. From an Information Theory perspective, a long enough vector of bits must guarantee that the sub-object classifier can be precisely defined. To do this, a Topos requires a so-classed top-down structure referred to as a *Separation* transform. Likewise, if separation is possible of sub-objects, then so must be the reverse, as per Fig. 4.2. This is referred to as the *Pushout* transform. We should have a method with which to conclude a sub-object is part of a larger object, hence the "pushout" in scope. One issue in determining separation and pushout is that they are often task-specific. This is where the high-dimensional representation of hypervectors can help us approximate separators and pushouts.

#### 4.1.2.1 A Generic Hyperdimensional Separator

In Algorithm 1, we show a general separation algorithm for use in binary, dense hypervectors. The function `AverageHypervector` computes the Consensus Summation of a set of hypervectors in $H_{set}$ via Eq. 2.6. The `Partition` function will partition $H_{set}$ around a Hamming Ball of size $ball$ around a hypervector $H^{(local)}$, keeping the "close" hypervectors $C_{set}$ within the Hamming Ball, and "far" hypervectors in the $F_{set}$ outside of it, thereby partitioning them. This is reminiscent to

**Algorithm 1** A General Separation Algorithm

**procedure** SEPARATION($H_{set}, localRoot$)
  $C_{set} \leftarrow \emptyset$
  $F_{set} \leftarrow \emptyset$
  $H^{(local)} \leftarrow$ AVERAGEHYPERVECTOR($H_{set}$)
  $localRoot.h = H^{(local)}$
  $upper \leftarrow |H^{(local)}|$
  $lower \leftarrow 0$
  $ball \leftarrow \lfloor \frac{upper+lower}{2} \rfloor$
  **repeat**
    $prev \leftarrow ball$
    $(C_{set}, F_{set}) \leftarrow$ PARTITION($H_{set}, H^{(local)}, ball$)
    **if** $|C_{set}| < |F_{set}| - 1$ **then**
      $temp \leftarrow \lfloor \frac{upper+ball}{2} \rfloor$
      $upper \leftarrow ball$
      $ball \leftarrow temp$
    **else if** $|C_{set}| > |F_{set}| + 1$ **then**
      $temp \leftarrow \lfloor \frac{lower+ball}{2} \rfloor$
      $lower \leftarrow ball$
      $ball \leftarrow temp$
    **end if**
    **if** $prev == ball$ **then**
      **break**
    **end if**
  **until** $|C_{set}| == |F_{set}| \pm 1$
  **return** $[C_{set}, F_{set}]$
**end procedure**

the relationship between $A$ and $B$ in Fig. 4.2. Effectively, the algorithm attempts to find the ideal Hamming Ball that splits our set of vectors into as equally-sized as possible partitions, recursively, via a binary-search-like loop. These maximize the information gain from one transform, making as general of a heuristic as possible.

**Algorithm 2** A General Pushout Algorithm

**procedure** PUSHOUT($N$)
    $H_{set} = \emptyset$
    **for** $child \in N.children$ **do**
        **if** ISLEAF($child$) == False **then**
            **return**
        **end if**
        $H_{set} \leftarrow H_{set} \cup \{child.h\}$
    **end for**
    $H \leftarrow$ AVERAGEHYPERVECTOR($H_{set}$)
    $D \leftarrow \mathrm{H}(H_{set}, H)$
    $ball \leftarrow$ MIN($D$)
    **if** ISDISAMBIGUABLE($H_{set}, H$) **then**
        $N \leftarrow$ NEWNODE($N.parent$)
        $N.ball \leftarrow ball$
        $N.h \leftarrow H$
        $N.local = H_{set}$
    **end if**
**end procedure**

**procedure** ISDISAMBIGUABLE($H_{set}, H$)
    **for** $h$ in $H_{set}$ **do**
        $h_{best} \leftarrow \mathrm{argmin}_{x \in H_{set}}(\mathrm{H}(x, H_{set}))$
        **if** $h_{best} \neq h$ **then**
            **return** False
        **end if**
    **end for**
    **return** True
**end procedure**

#### 4.1.2.2 A Generic Hyperdimensional Pushout

To make a generic pushout, we leverage the Consensus Summation operation. So long as the elements of a Consensus Sum can be disambiguated from each other by the HIL algorithm in Eq. 2.10, when no binding is applied. That is, this measures the compressibility of the information in a set $H_{set}$. If all elements can be disambiguated, then they belong in one group, and can be safely pushed out to be the best of our knowledge under the HC algebra. Thus, Algorithm 2 shows how a pushout will attempt to find all leaves that can be disambiguated, and thus merged into larger sub-objects than the individual constituents. Once this fails, we stop.

#### 4.1.2.3 Topos Generation

Given a separation and pushout algorithm, a Topos can be generated for a finite set of elements. Algorithm 3 performs this generation process recursively on a given set of hypervectors. The procedure `LocalNeighborhoods` reaps all hypervectors at a particular level of the tree, in an effort to aid pushout discovery. Recursively, a Topos first performs top-down separation until it cannot seperate anymore, because it either reached a leaf of the tree, which contains only one hypervector, and thus can't be separated. Then, the pushout function starts from the bottom-up, pushing out all possible hypervectors into bigger sub-objects, as much as possible, compressing the Topos as much as is possible under the HC algebra.

#### 4.1.2.4 Utility Functions

We must define several utility to functions to utilize the Topos after its creation, which are outlined in Algorithm 4. Namely, to measure distance under the new metric defined by the Topos,

**Algorithm 3** Topos Generation

```
1: procedure TOPOSGENERATOR(H_set)
2:     T ← TOPOS(H_set)
3:     T ← LOCALNEIGHBORHOODS(T)
4:     return T
5: end procedure

6: procedure TOPOS(H_set, localRoot = root)
7:     if |H_set| == 1 then
8:         localRoot.h ← h ∈ H_set
9:         return
10:    end if
11:    P ← SEPARATION(H_set, localRoot)
12:    for i from 1 to |P| do
13:        child ← MAKECHILD(localRoot)
14:        TOPOS(P[i], child)
15:    end for
16:    return
17: end procedure

18: procedure LOCALNEIGHBORHOODS(T)
19:    for i from HEIGHT(T) - 1 to 1 do
20:        N_(i) ← LEVELSET(i, M)
21:        for N ∈ N_(i) do
22:            PUSHOUT(N)
23:        end for
24:    end for
25: end procedure
```

we must use the `Hyperdesic` utility function. Given two hypervectors, the function places them into their proper sub-object by the `Find` utility, which simply narrows down which resting pushout sub-object the hypervector is placed in thanks to the separations of Hamming Balls and subsequent upwards compression through the pushout. Then, `Hyperdesic` constructs the edges required to reach from one hypervector to the other. Measuring the distance traversed along the way. Distance is measured Hamming Distance from the average vector on each traversal's corresponding partition. If the two hypervectors are actually in the same sub-object, then no traversal is required and regular Hamming Distance is computed. If they are not, then we add to the distance the distances between

**Algorithm 4** Topos Utility Functions

**procedure** HYPERDESIC($H_1, H_2$)
$R^{(1)} \leftarrow$ FIND($H_1$)
$R^{(2)} \leftarrow$ FIND($H_2$)
$d \leftarrow$ LOCALMETRIC($R^{(1)}, R^{(2)}$)
$d \leftarrow d + \min(H(H_1, R^{(1)}.h), R^{(1)}.ball)$
$d \leftarrow d + \min(H(H_2, R^{(2)}.h), R^{(2)}.ball)$
**return** $d$
**end procedure**

**procedure** LOCALMETRIC($R^{(1)}, R^{(2)}$)
$(R^{(1)}, P_1) \leftarrow$ FIND($H_1, getPath = True$)
$(R^{(2)}, P_2) \leftarrow$ FIND($H_2, getPath = True$)
$P = P_1 \Delta P_2$
**return** DISTANCE($P$)
**end procedure**

**procedure** DISTANCE($P$)
$s \leftarrow P.head$
$e \leftarrow P.tail$
$c \leftarrow s$
$sum \leftarrow 0$
**while** $c \neq e$ **do**
$sum \leftarrow sum +$
$H(c.localRoot.h, c.next.localRoot.h)$
$c \leftarrow c.next$
**end while**
**return** sum
**end procedure**

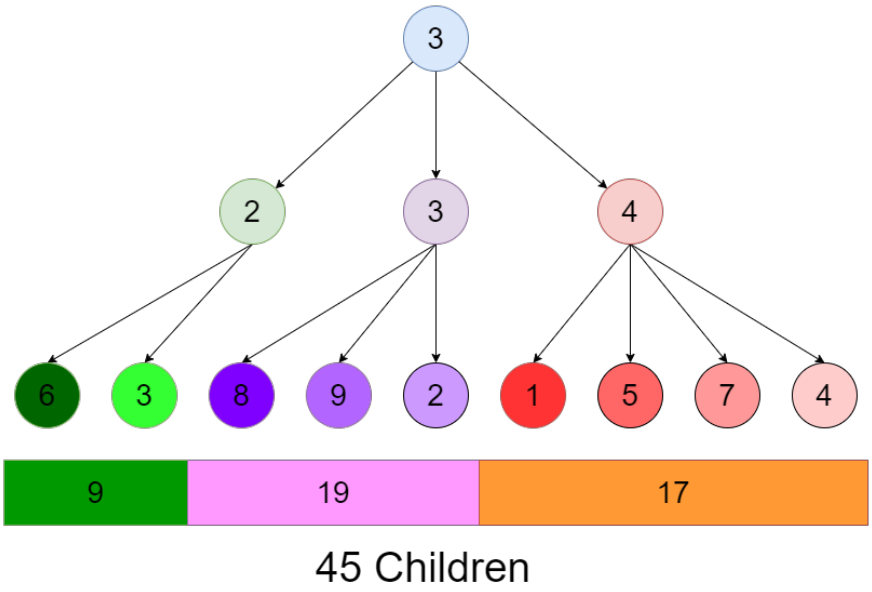



Figure 4.3: Working Example of Genetically Generated Descendants: Descendants inherit their parent's DNA by averaging their random hypervector with their parent's. Then, a random percent of bits up to a given noise ratio are flipped, so that each sibling is different. The color codes signify these alterations from ancestral hypervectors.

each traversal's average hypervector. This deforms the Hamming Distance metric. If the sub-objects are not to far away in number of traversals, they are likely to have a smaller distance. Thus, Hamming Distance is skewed to be higher than normal under a deformation.

#### 4.1.2.5 Practical Working Example

To construct a synthetic dataset in which we are guaranteed that a Topos must exist, we consider what is referred to as the "practical working example". We consider the practical problem of understanding genetic lineage in ancestors and descendants. This is conceptualized as a hypervector representing DNA, which is then passed on to descendants. Specifically, we consider dense binary hypervectors for this problem, for the sake of simplicity. Given a random hypervector $a$, its descendant

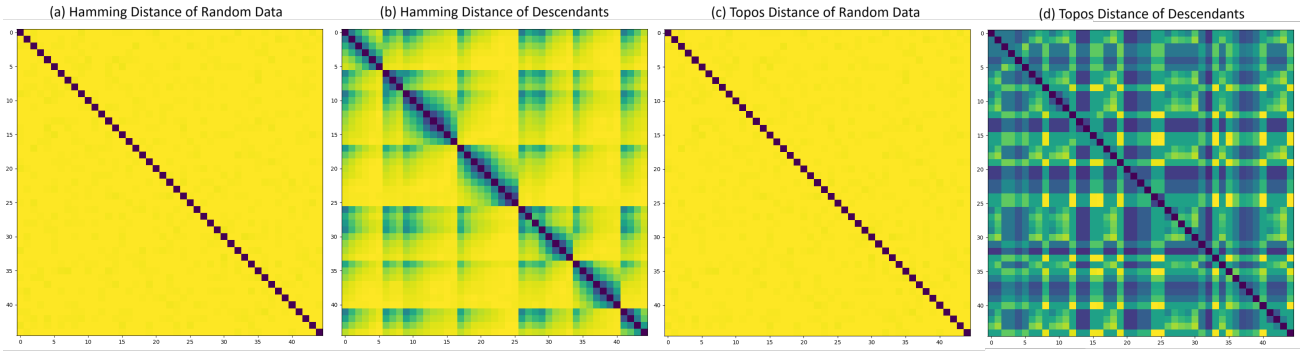



Figure 4.4: Example Descendant vs. Descendant Distances: Shows the distances between the 45 descendant of the example in Fig. 4.3. Yellow symbolizes orthogonality, and darker colors symbolize closeness. Plot (a): Randomly generated descendants and their pairwise Hamming Distances. Plot (b): Descendants and the Hamming Distances between each. Plot (c): Topos Distance between the same random descendants in plot (a). Plot (d): Topos Distance between the same descendants as plot (b). While some of the genetic structure is apparent in plot (b), many distance relatives appear random (yellow), even though they share the same ancestry. In plot (d), the Topos reveals much deeper genetic similarity than (b), without compromising the randomness of unrelated vectors as shown in plots (a) and (c).

$d(a) = \text{FLIP}(a + b, \epsilon)$, where $b$ is another random hypervector, and FLIP is a function that random flips the bits in the positions where a 1 is specified in $\epsilon$. Essentially, FLIP adds noise to the result, which can be specified by percent of noise parameter that is allowed. This parameter controls how chaotic the DNA of the descendant is.

Furthermore, we will use a working example of such genetic lineage throughout the rest of the paper, in order to demonstrate properties of the Topos. Consider the 45 children generated in Fig. 4.3. We can see through their coloration, how the descendant function $d$ would be used compositionally to generate the children and their ancestors. These children were generated using hypervectors of length $2^{14}$ and noise in up to $20\%$ of the those bits.

#### 4.1.2.6 Experimental Results

In Fig. 4.4, we see a heatmap of distances between the nodes shown in the genetic example in Fig. 4.3. We compare completely random data in subplot (a) to these 45 genetically related descendants in (b), using Hamming Distance. Note that Hamming Distance reveals in (b) that the descendants are definitely not random, and clear structure is visible in sibling subgroups. In (c) and (d), we compare the same distances but under the metric defined by the Topos constructed for this working example. In (d), you can see far richer structure. Note that yellow coloring on the heatmap indicates random distances, while darker colors indicate closeness. Under the Topos, we see that actually all the descendants are related in clear ways. The little bits of yellow are correlated to the likelihood of noise that was introduced during the creation of the descendants.

Likewise, we want to see how much genetic similarity is preserved across ancestors to the descendants. In Fig. 4.5, we perform a similar comparison of heatmaps to Fig. 4.4, except one of the axis is a parent, or a grandparent. As shown in subplots (b) and (d), the Topos reveals more genetic correlations. Interestingly, even in (c) structure is revealed, but it doesn't seem to have any predictive power as it is a constant drop in Hamming Distance. This will be relevant in the next Chapter.

Finally, we perform a large scale randomized set of trials on randomly generated descendants like those in Fig. 4.3, but with differing numbers of parents, lineage depths, noise levels and hypervector lengths to measure Hamming versus Topos distances across random data versus descendants. The goal is to generalize the results shown in Fig. 4.4. Fig. 4.6 shows the results of this over noise levels from 5% up to 50% (complete random data, effectively). We can see that the results are consistent despite varying hypervector lengths from absurdly small $32$ bits lengths up to $16184$, a large hypervector. Generally speaking, random hypervectors will have on average the expected value of

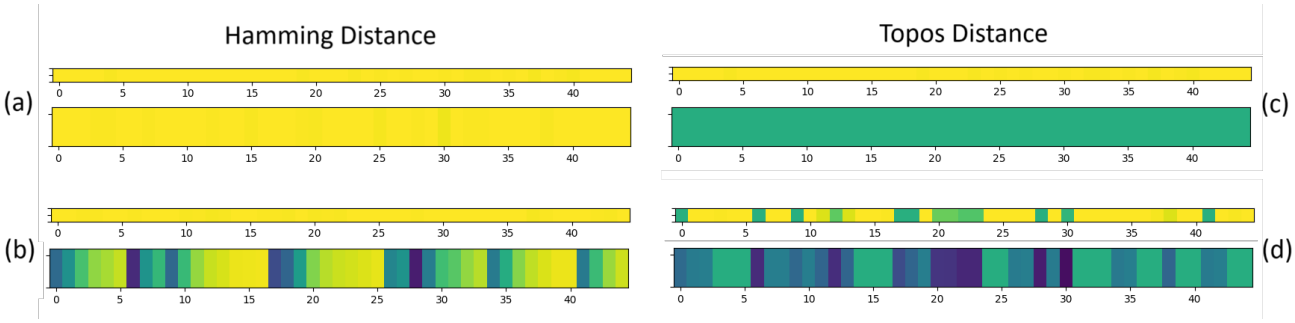



Figure 4.5: Example Ancestor vs. Descendant Distances: Shows the distances between the 45 descendants and their ancestors in Fig. 4.3. Yellow symbolizes orthogonality, and darker colors symbolize closeness. Since each descendant has 2 generations of ancestors before their parents, we show both generations. Plot (a): Randomly generated descendants and the Hamming Distance between them and the actual ancestors. Plot (b): Hamming Distance between descendants and their ancestors. Plot (c): Topos distance between the same random descendants in plot (a). Plot (d): Topos Distance between descendants and their ancestors. We can see that plot (a) is completely random, while plot (b) shows similarity in at least the closer generation of ancestors. Plot (c) shows that the Topos puts ancestors in a non random and unusual close subspace of the hypervectors. Plot (d) shows that genetic similarity is preserved better in the first generation and even in the second generation, which doesn't happen in plot (b).

Hamming Distance between each in both standard Hamming Distance and Topos Distance. However, the averages drop in Topos Distance when the working example nodes are introduced. As the legnth of the hypervector increases, the results become more significant, as multiple standard deviations of significance are captured by drops in average distance. As the noise levels approach 50%, the results become more chaotic and collapse into true noise, except in longer vectored Topoi.

Next, we perform a similar randomized trial but on distance comparisons between descendants and ancestors, like in Fig. 4.5. Our results are shown in Fig. 4.7. Interestingly, we receive the same results as in our working example. Random hypervectors no longer look random to the Topoi when you compare them to ancestor nodes, and comparisons of descendants to ancestors yields a strictly lower bound across all cases. The standard deviations are also more pronounced for ancestors than descendants. This makes sense, since ancestors serve a "genetic root", which has no noise in it. This the distances in the Topoi will be consistently lower on average than when descendants are compared to each other.

## 4.2 Analysis of Hyperdimensional Topoi Generation

Our results confirm that Topoi generated over empirical sets of correlated hypervectors demonstrate different, and more statistically significantly distance metrics than regular Hamming Distance. This implies an underlying manifold-like structure has been discovered and embedded into a hyperdimensional Topos. Considering statistically significant results were generated for "training sets" consisting of dozens to hundreds of training examples, the Topos was effective at generating a sensible metric for few shot examples. Likewise, the Topos did not affect the standard behavior of Hamming Distance for random hypervectors, taking advantage of hyperdimensional orthogonality.

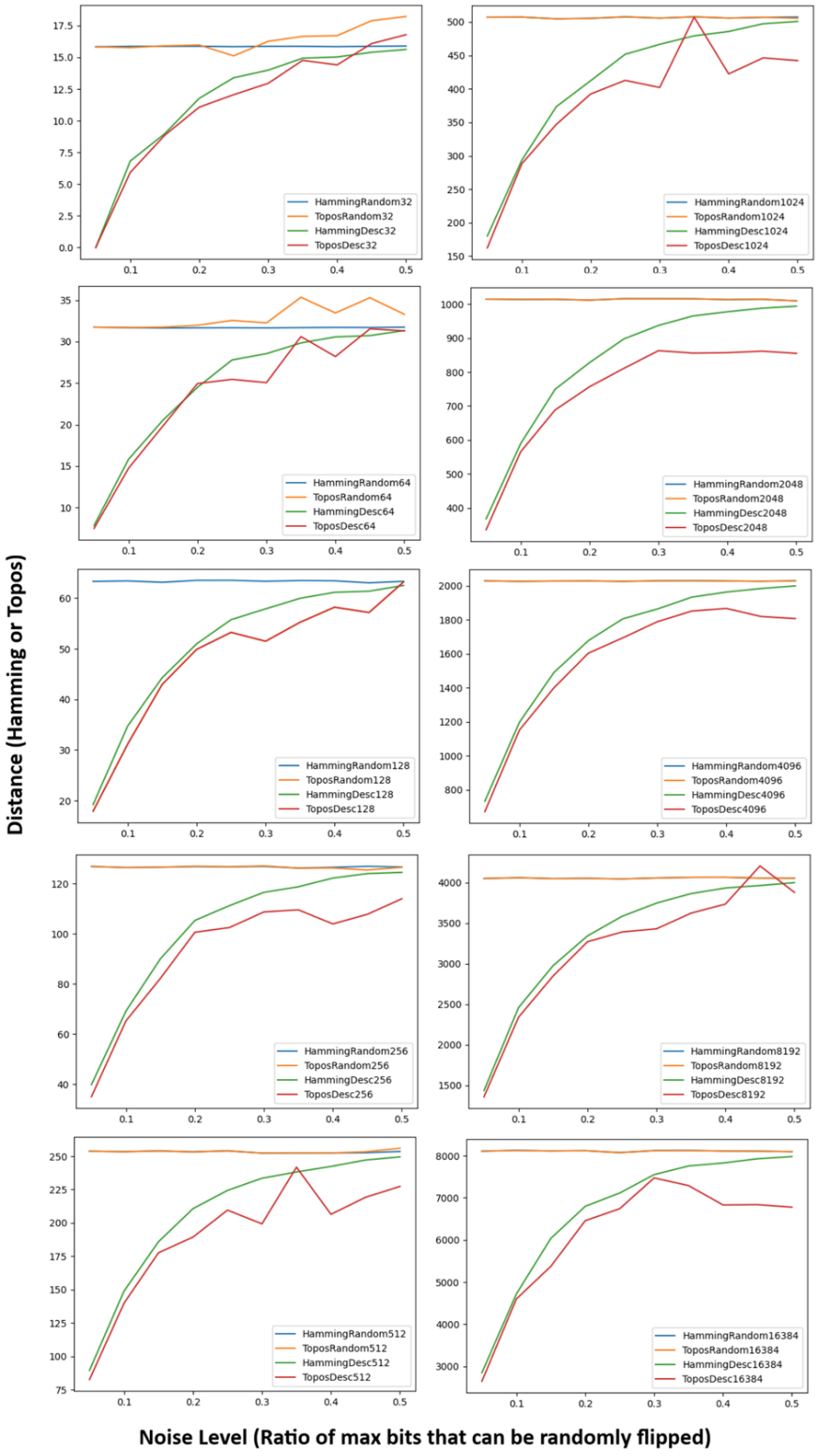



Figure 4.6: Average Trial Results: Shows the average distances across multiple randomly generated lineages between descendants, much like in Fig. 4.3.

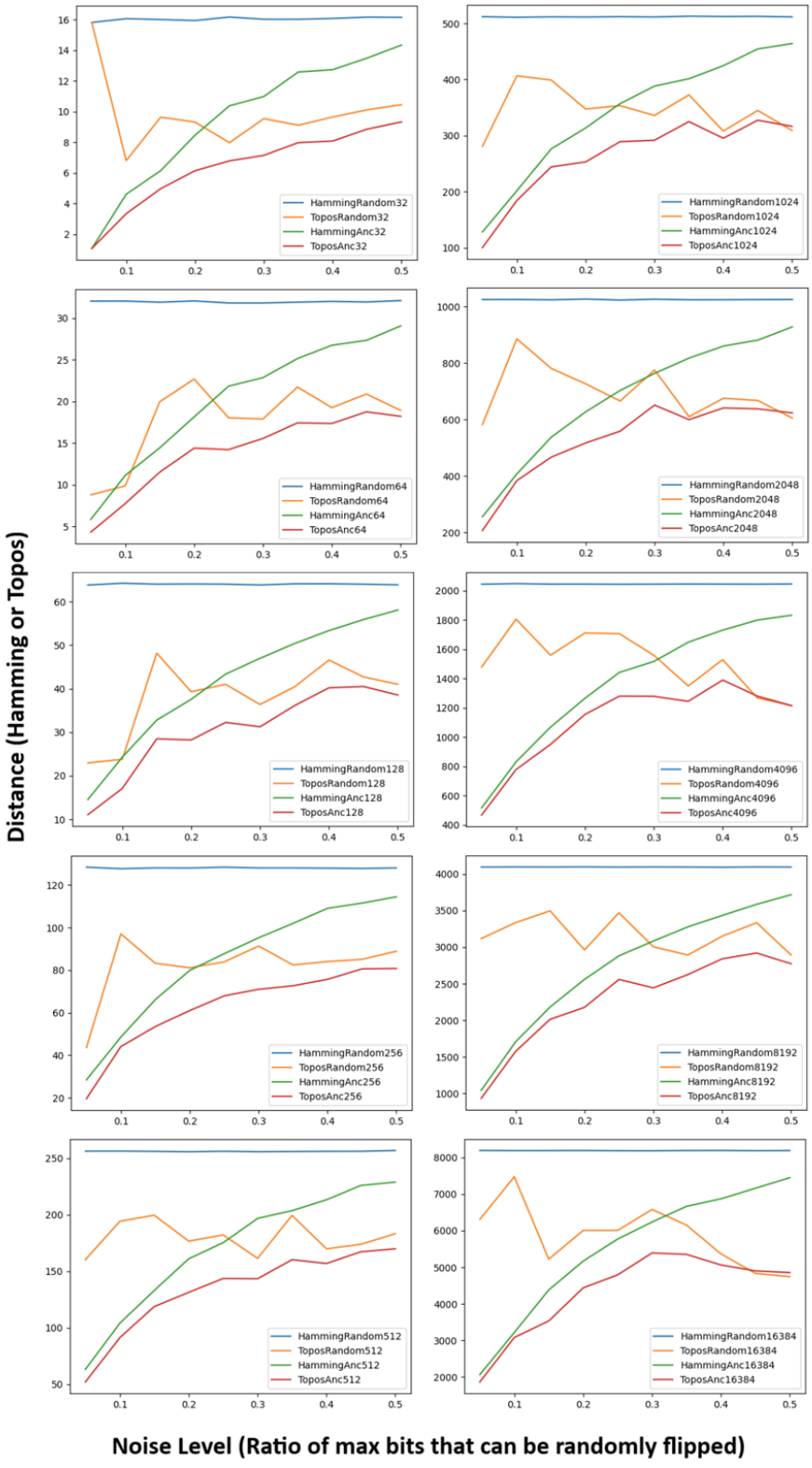



Figure 4.7: Average Trial Results: Shows the average distances across multiple randomly generated lineages between descendants and ancestors, much like in Fig. 4.3.

The effect can be likened to embedding a manifold into a massive metric space without affecting the vast majority of it. For a more practical example, it's like embedding structures into a solar system, while leaving the rest of the galaxy untouched. Thus, randomly choosing points in the galaxy will yield predictably random distances, whereas selecting points near planets from the solar system will cause disproportionately close distances between points, as they are in the same solar system. Another way to think of what the Topos is doing is it is effectively creating Russian nesting dolls, where the outer dolls are gigantic compared to the inner dolls, to the point you wouldn't even see them if you were standing inside the outer doll.

Why are these properties of interest to AI and ML? Manifolds underlie nearly every aspect of ML. One can think of neural networks are manifold learning machines. Discovering that arbitrary manifolds can be embedded into hyperdimensional spaces to create custom distance metrics means that any architecture's underlying manifold can be embedded in such a way, with enough samples. Therefore, one can construct a hyperdimensional space that will integrate all manifolds of interest, with sufficiently big hypervectors and sufficient samples from the manifold. This explains many results found in Case Studies 1-3, in which a core feature of HC was it's ability to embed heuristics into its approximate algebra.

## 4.3 HyPE: Hyperdimensional Propagation of Error [8]

In this work, we explore the concept of discovering feedback mechanisms in Hyperdimensional Computing. Namely, we explore so called *HyPE features* and leverage these to improve the performance of hypervector models. HyPE stands for Hyperdimensional Propagation of Error. Fundamentally, this is referring to an inherent error source present in hypervectors that can be utilized

to reduce error across Vector-Symbolic Architectures (VSAs). We formalize this as a Generalized Expectation Maximization (GEM) algorithm, a broader class of well-explored optimization strategies. This fact alone should give a big clue as to what the error feedback mechanism are; they are the expectations present in HC natively by construction.

Once again, there is a heavy emphasis on Category Theory in this work. Refer to Appendix A to learn Topoi from first principles. However, proving correctness for HyPE requires a bit more involved Category Theory. We cite the appropriate references for the mathematics presented here, for those interested in learning more. In the following several sections, we largely repeat the "Methodology" and "Discussion" section of [8]. In an attempt to maintain mathematical rigor, we largely reproduce this section with minor changes to adapt it to the context of the broader thesis. All credits go to co-author Dr. Renato Faraone, an expert in Category Theory, who originally wrote the Appendix, Methodology, and parts of the Discussion section in our work in [8]. Consider the following to be quoting this work and paraphrasing where needed. We also introduce some novel observations as relevant to this thesis.

### 4.3.1 Expectation Maximization in Hyperdimensional Computing

The canonical HC learning paradigm accepts a dataset $D$ of encoded hypervector pairs of the form $(T_i, C_i)$, which we refer to as the training example and class hypervectors, respectively. For each unique class in $C_i$, we create a hypervector model $M_i := \sum_{j|C_j=i} T_j$ and then define a *minimizing model* $M := \sum(M_k \oplus C_k)$. This $M$ is referred to as the minimizing model because of the conservative nature of the Consensus Summation defined in Eq. 2.6. To preserve information, the Consensus Sum throws away the least amount of information it can, by choosing the more popular bit across terms.

From one angle, this "makes the most" out of the self-mapping to a hypervector that is required by HC. Under that lens, HC performs an expectation maximization, since what is preserved is not suprising to our expectations, both statistically and informally. Next, we assume the training hypervectors are actually via a prescribed embedding operation from an original space outside of our hypervector-space, which we denote by encoder `enc`. This is itself a function operating on arbitrary finite homogeneous lists of hypervectors.

As we are concerning ourselves with dense binary hypervectors, there is little to no assumption on the nature of the data space. In this sense, such hypervectors can be quite arbitrarily constructed on the input encoding side. This is one consequence of our work in the Topos of hypervectors in [7], which allows us to deal with far more general situations than the one typically assumed by most Deep Learning paradigms. These, in some way or another, expect quite rigid topological properties, such as differentiability or, at minimum, continuity. Yet again, that is a consequence of our observations so far in this thesis, through the case studies and Topos work.

To wit, we will adhere to Lawvere's interpretation of Metric Spaces, which we briefly review here. However, see [39] for a more thorough discussion of the motivation behind these observations in the context of AI. Under this interpretation, metrics are considered to be only partially defined or, equivalently, *extended* as to allow for points to be "infinitely far apart". Additionally, symmetry is not required. When viewed as a Category, the space comes with multiple available paths connecting the same endpoints, and a generalized metric of this kind is referred to as a *pseudo*-metric. In our case, Hamming Distance serves as a lattice-based movement from two points, where a movement up is a match, and over is a difference. This phenomena of multiple path is evident in this pseudo-metric. Lastly, by computing in a distributed setting, the *identity of indiscernibles* axiom is weakened to not imply *separation* of points, and reduces to $d(x, x) = 0$, generalizing the (pseudo-)metric to a *quasi-*

(pseudo-)metric.

Now, taking our model $M$, and a train example $X$, paired with class $Y$, we compute $E_M(X, Y) := X \oplus M \oplus Y$ and define the *error* by

$$\texttt{err}_M(X, Y) := |E_M(X, Y)| = d(\bar{0}, E_M(X, Y)) \tag{4.1}$$

Here, $d$ stands for the extended-quasi-pseudo-metric of the ambient space. In our case, this is Hamming Distance. Meanwhile $\bar{0}$ denotes the hypervector consisting of all 0's, or the $0$ hypervector. As an error estimate, this indicates how far we are from a perfect match due to the noisy superpositional collapse of the individual class vectors caused when $M$ is probed with input $X$.

We leverage expectations under this metric to our advantage to formulate the HyPE algorithm. By assuming that the hypervectors are random, we expect erroneous matches to resemble samples of a binomial distribution of all possible head/tails coin tosses. This is the Bernoulli Trials interpretation described in our background section on HC. However, this will not be the case if $Y$ is indeed the correct output. Almost like a proof by contradiction, we can perfectly characterize the strength of this correlation by the reciprocal of the Probability Mass Function (PMF) for the Binomial Distribution with equal heads and tails probability. By minimizing the likelihood of this, we increase the chance that the predictive behavior is NOT random. This will "hype" up non-random features. From this we get the moniker of HyPE; we use this features to propagate non-random error signals back into our computation of a hypervector model.

Given a dataset $\mathcal{D}$, the first pass proceeds with the assumption of treating each training example as equally important. Computing $M$ proceeds in the usual fashion. A subsequent second pass will reveal where the model succeeds. This isolates a subset of $\hat{\mathcal{D}}$, which holds only the features relevant to

model $M$. In order to bias $M$ towards that data, we propagate the error per training example in $\hat{\mathcal{D}}$ into a hype feature biased model $\hat{M}$. Thus, $\hat{M}$ represents a sub-model that is a weighted subset of favorable examples, dispersing errors that could be made under a random construction.

**Algorithm 5** HyPE Single Layer

**Input:**
Training inputs $\mathcal{D} = \{(X_i, Y_i)\}$, where $X_i$ are hypervector encodings of the input and $Y_i$ of the output for the $i$th training example.
**procedure** HYPE_LAYER
    $\mathcal{L} \leftarrow \emptyset, \mathcal{D}^{(0)} = \mathcal{D}, f_1^{(1)} = 0, f_1^{(0)} = -1, k = 0$
    **while** $f_1^k > f_1^{k-1}$ **do**
        $M^{(k)} \leftarrow \texttt{HDC\_Train}(\mathcal{D}^{(k)})$
        $\hat{\mathcal{D}}^{(k)}, \mathcal{D}^{(k+1)} \leftarrow \texttt{partition}(\mathcal{D}^{(k)}, M^{(k)})$
        $\hat{M}^{(k)} \leftarrow \texttt{HyPE\_Weighted}(\hat{\mathcal{D}}^{(k)}, M^{(k)})$
        $\mathcal{L}\texttt{.append}(\hat{M}^{(k)})$
        $f_1^{(k-1)} \leftarrow f_1^{(k)}, f_1^k \leftarrow \texttt{test}(\mathcal{D}, \mathcal{L}), k \leftarrow k + 1$
    **end while**
**end procedure**

### 4.3.2 Parallel Layers of HyPE Models

Suppose we continue the HyPE process on many subsets of $\mathcal{D}$. Each time a hyped model is formed, we train a new model on $\mathcal{D} \setminus \hat{\mathcal{D}}$. These are the sets of training examples that $\hat{M}$ got incorrect. While the global accuracy on $\mathcal{D}$ monotonically increases, this process can continue until either we perfectly partition the entire dataset, or produce a fleet of models that work in tandem to do this as optimally as possible with our hypervectors. We will refer to this as a layer, defined:

$$\mathcal{L} := \{M^{(0)}, M^{(1)}, \ldots, M^{(k)}\} \tag{4.2}$$

where $M^{(0)} = \hat{M}$ for the corresponding $\hat{\mathcal{D}}$. If we want to predict an output for $X$, we probe each model and find its best match across the layer $\mathcal{L}$'s sub-models. The layer $\mathcal{L}$ will outperform the original model

$M$.

This newly constructed layer can be put in a *stack*:

$$\mathbb{S} := \{\mathcal{L}, \mathcal{L}', \dots\} \tag{4.3}$$

through repeated iteration on this process, dispersing the global error signals across parallel models. This, in turn, lowers the average Hamming Distance of the error, and increases the odds of a correct prediction. Training each model involves a classic class-wise bundle of each classes' training examples. These produces *prototypes* of each class. The prototypes reveal "useless" classifying features, which are components where all bits match across prototypes. Binding these prototypes to appropriately sized class hypervectors and bundling the prototypes together results in denser binary features, saving computation time.

Additionally, we can continually discard components that underperform. We can maintain only the best performing aspects of our HyPE models, by sorting them by the error expectation. As a result, online HyPE is possible. Alternatively, if we are just seeking to find the best performing components in number to the original hypervector length, we can also do that. This is the approach taken in our experiments. We summarize a single layer approach in Algorithm 5.

### 4.3.3 Interpretation as Generalized Expectation-Maximization

In this section we give an interpretation of the proposed algorithm as an HDC analog of *Generalized Expectation-Maximization* (GEM) techniques. An in depth overview of GEMs algorithms is covered in [41]. These are quite broad in scope. Typical usage is *maximum likelihood* estimation in the case of missing or incomplete data. The algorithmic outline of these methods tends

to follow the steps:

1. Replace missing values by their estimates
2. Estimate parameters
3. Estimate missing values according to the parameters
4. Estimate new parameters

and loop until a desired threshold condition is met.

In our case, step $1$ becomes the sparse encoding $\hat{\mathcal{D}}$ of the original dataset, step $2$ is the learning of the new minimizing model $\hat{M}$, step $3$ becomes the pruning which takes us to $\mathcal{D}^{(1)}$ while step $4.$ is simply the computation of the next minimizing model $M^{(1)}$. Note that the original EM is a Maximization-Maximization procedure, while HyPE performs Minimization-Maximization in alternating steps. Maximization of expectation forces one to strive away from the chaotic regions of the space indistinguishable from random noise. Our only assumption is that of the inherent expectation of a Binomial Distribution of hypervector strings, which is re-established through HyPE in distinctly non-random way.

### 4.3.4 Convergence and Correctness

General convergence properties of the EM algorithm were already studied in [42] using purely statistical techniques. As our models live in dense binary high-dimensional space, boundedness properties are trivially satisfied and this is enough to guarantee local convergence. Algebraically, this is immediate since Topoi are complete and admits all small limits. Instead, convergence to global optima cannot be guaranteed as we are not imposing any assumption on differentiability.

Correctness is visualized as follows. Recall (see [45]) that a **sieve** $\mathcal{S}$ is a full Subcategory (i.e. contains all the morphisms from the ambient Category between its objects) closed under precomposition: $f; g$ whenever $g$ is already in $\mathcal{S}$. Note, sieves generalize *ideals* from Monoid Theory, the choice between diagrammatic $f; g$ or applicative $g \circ f$ notation for the composite of $f : \Box f \to \Box g$ and $g : \Box g \to \blacksquare g$ somehow influence if we should consider them as so for left or right ideals.

Sieves at an object $X \in \mathtt{Ob}(\mathcal{C})$ are simply the sieves in the *slice* Category over $X$. Thus, all the morphisms have $X$ as the target. We can visualize them as the correct formalization of a collection of *covers* of $X$ (see again [44]), which is, in turn, equivalent to that of a *sub-object* of the *Yoneda Functor* $\mathtt{Hom}(\_, X)$.

Thus, the assignment of the collection of all sieves at an object is functorial and indeed provides the *sub-objects classifier* $\Omega$ for a presheaf category (such as hypervectors). The corresponding morphism $\mathtt{true} : \bot \to \Omega$ out of the *terminal* is the *natural transformation* that picks at every object the maximal sieve.

Intuitively, if one views a presheaf over a (small) category as a set evolving over time, then the sub-objects classifier may be viewed as encapsulating the ways of an element to be in that set ranging from never there to always present.

As such, correctness boils down to recognizing that layer $\mathcal{L}$ constructed with HyPE, as shown in Fig. 4.8, is trained until it behaves as a maximal sieve, the collection of models working as the covers for the objects in the encoded dataset.

This diagram sketch visualizes the single layer construction process in our HC framework, showing how training data gets transformed through successive refinements. The vertical flow shows how the original dataset gets sparsely encoded and then successfully refined into its pruned descendants through iterative use of XOR with model residuals and then performing a majority vote. The horizontal

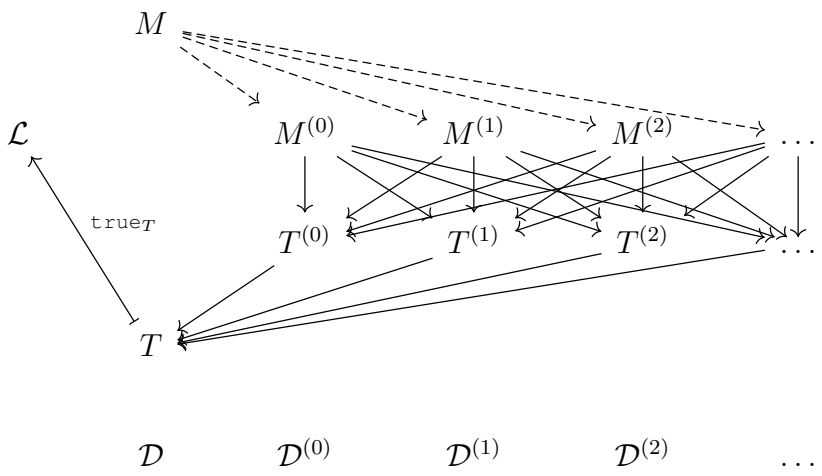



Figure 4.8: Diagram of parallel HyPE models in the case of a layer $\mathcal{L}$

flow shows how the original model gives rise to better and better representations through the EM (or Minimization EM) steps iterations (the dashed arrows are informally used to avoid the clutter one would have by displaying all the composite arrows). Finally, the truth arrow $\texttt{true}_T$ assigns to each example $T$ its maximal sieve, showing how the layer $\mathcal{L}$ acts as a collection of covering models, commutativity ensuring information preservation.

### 4.3.5 Experiments

We test HyPE on the `Fashion-MNIST` [40] Dataset; a more challenging version of `MNIST` [25] in which clothing articles must be categorized. Human level performance tends to high 80%, whereas state-of-the-art exceeds 95%. Our goal is to show that low dimensional hypervector encodings can be projected by multi-layer HyPE to a target length without losing encoding power. This means that HyPE will exceed classic HDC everywhere except at extremely high lengths, where the two begin to meet. Indeed, our results shown in Fig. 4.9 confirm this.

We try two popular encoding strategies. The first is per-pixel encoding, where every row, column

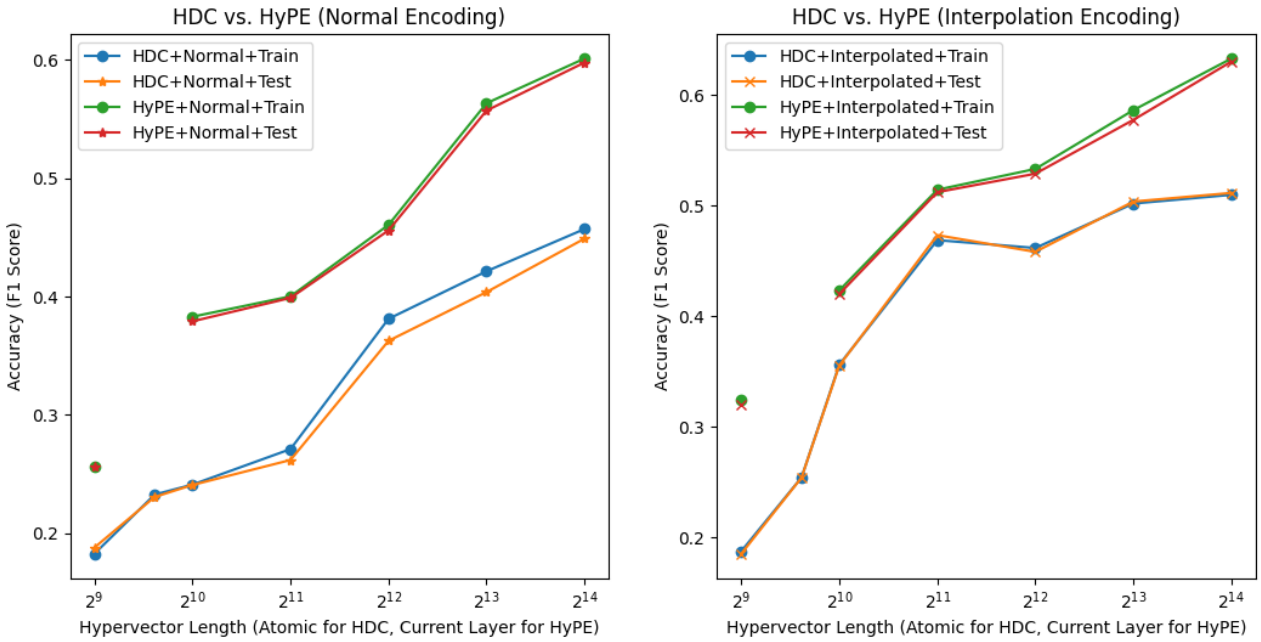



Figure 4.9: We consider both normal per-pixel encoding (left) and interpolation encoding (right). In either case, accuracy increases in HyPE-boosted models when we keep the same amount of vector components as the starting model's length $n$, choosing the top $n$ expectation minimizing components. We include the test and train accuracy, as well as performance for differing hypervector lengths $n$ to show stability of the results. Each trial was repeated 10 times with random dictionaries for encoding and classes, averaging the results.

and pixel intensity is represented by a random hypervector and an image is the binding of each pixel intensity to its row and column, and subsequent superposition of all pixels bindings. The second differs only in the atomic hypervectors. One interpolates equally between the 28 rows and columns, and the 256 pixel intensities from random hypervectors denoting the starting and end positions (row/col 0 and 27, and intensity 0 and 255).

Each run in Fig. 4.9 is repeated 10 times, taking the best performing random encodings, simulating evolution-based encoders for the "`MNIST`" environment. This prevents unfair bias due to random chance, particularly for shorter hypervector encodings. Layers are produced until the target length is reached, doubling in length each time. Typically, only a few hypervectors are generated per layer before global performance begins declining.

## 4.4 Analysis of HyPE

The formal analysis of HC through the lens of expectation maximization reveals an underlying objective function that is leveraged by HC to carry out its approximate algebraic heuristic learning. Confirming that HyPE features can be leveraged to improve on this by minimizing our expectations reveal that there is much more to be done to improve upon this approximate algebra. Repeatedly, the story that comes up is that the learning power of HC is in the Consensus Summation; aggregation is the workhorse of learning. Structuring summations to reduce information loss is beginning to sound like a valid strategy. We explore this in the next chapter.

Finally, these results pose HC as minimization strategy on par with those employed in ML and often in AI. Backpropagation-like mechanisms imply that even more sophisticated structures can be made. Packing more efficiency and prediction power into hypervectors is an attractive offer for AI and

ML purposes.

# Chapter 5: Hypermapping

In this chapter, we present a novel concept referred to as *hypermapping*. To the best of the author's knowledge, this is a novel contribution to Hyperdimensional Computing of this thesis.

## 5.1 Motivation

In Table 5.1, we show the 10 prototypical hypervectors obtained during a training run on `Fashion-MNIST` in our work in [8]. The columns represent components in the hypervector prototypes, starting at left. Namely, only the first 18 bits of the hypervectors are shown. Rows represent the class for which the hypervector was formed. We encourage the reader to stare at this table for a while, to see if they notice any patterns. The next table will try to elucidate the pattern. Once the reader is convinced they see it, they should look at Table 5.2. In this table, the cells are color coded, white for 1 and black for 0. Furthermore, certain fonts are colored red, green and blue, in order to exemplify a pattern. The reader should check and see if the pattern they noticed was confirmed or not.

To point it out explicitly, red columns all have the same value in their cell, whereas green columns are the same everywhere, except for one cell. Blue columns have equal counts of opposing bit values. Consider the fact that prototypes of hypervectors were explicitly noted in our background section on HC to be the only source of classification power in a HIL. Therefore, three things occur if we try to form a HIL using the elements in Table 5.2:

1. Firstly, the red columns will contribute nothing to the classification power of prototypical hypervectors. Since they are all the same bit value, they cannot tell the difference between stereotypically prototypical input hypervectors. One can refer to these as *useless* classifying

Table 5.1: Class Prototypes of `Fashion-MNIST`

| **Initial Components of Prototypical Hypervectors** | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Class 1** | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | $\cdots$ |
| **Class 2** | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | $\cdots$ |
| **Class 3** | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | $\cdots$ |
| **Class 4** | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | $\cdots$ |
| **Class 5** | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | $\cdots$ |
| **Class 6** | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | $\cdots$ |
| **Class 7** | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | $\cdots$ |
| **Class 8** | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | $\cdots$ |
| **Class 9** | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | $\cdots$ |
| **Class 10** | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | $\cdots$ |

The first 18 elements of prototypical hypervectors obtained during training on `Fashion-MNIST` from a debug run in [8]. Columns represent hypervector components starting at the first bit, and continuing for $2^{14}$ bits. Rows represent which class the prototype is for.

features. If this component is thought of as a neuron, it is maximally polysemous.

2. Secondly, the green columns will maximally contribute towards the classification power of prototypical hypervectors. They are special because they identify only one class. We refer to these as *perfect* classifying features. If this component is thought of as a neuron, it is monosemous.

3. Thirdly, the blue columns will partition the outcomes into two equal sized groups, an ideal situation in the case of HyPE features [8]. They are the lower bound on how fast a hypervector can classify an input. Any other partition will have a favorable side (the one with less outcomes) which can be leveraged to reduce the outcome possibilities.

All in-between cases can be considered regular polysemous neurons, which co-activate on particular classes prototypically.

Immediately, one can realize that for prototypical hypervector examples, the red columns can

Table 5.2: Class Prototypes of `Fashion-MNIST`

| Initial Components of Prototypical Hypervectors | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Class 1** | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | ··· |
| **Class 2** | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | ··· |
| **Class 3** | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | ··· |
| **Class 4** | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | ··· |
| **Class 5** | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | ··· |
| **Class 6** | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | ··· |
| **Class 7** | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | ··· |
| **Class 8** | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | ··· |
| **Class 9** | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | ··· |
| **Class 10** | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | ··· |

The first 18 elements of prototypical hypervectors obtained during training on `Fashion-MNIST` from a debug run in [8]. Columns represent hypervector components starting at the first bit, and continuing for $2^{14}$ bits. Rows represent which class the prototype is for. This time the cells are colored black and white for 0 and 1, respectively, and bits are colored red, green and blue to exemplify a pattern.

be directly removed from the algebra. This amounts to removing the corresponding component index from consideration at all. However, assuming the prototypical hypervectors have seen all training examples, these components are actually useless to the point that removing them will often improve accuracy, as they are just a weak-point for injection of noise. Thus, all dense binary hypervectors are equipped with a speed-up capability that can be leveraged by removing these components, once detected.

But how many of such components are there? In Table 5.2, we can see that 6 out of the 18 initial components are useless. The odds of 10 random coin-flips delivering all the same value are exactly 2 out of $2^c = 2^{10}$ outcomes, or $2^{-9} = \frac{1}{512} \approx \frac{1}{500}$, where $c$ is the number of classes. Finding them $\frac{1}{3}$ of the time defies expectations. Indeed, it was discovered that when HyPE features are leveraged in [8], much of the slack in expectation minimization comes from these useless classifiers. They have an approximate probability of $\Theta(\frac{n}{r})$ of occurring in non-random data, where $n$ is the length of

the hypervector and $r$ is the radix. The more structure present in the training data, the less useless classifiers are discovered. Thus, the speed-up in processing time by removing useless classifiers can be quite substantial.

Furthermore, considering the perfect classifiers, we notice from Table 5.2 that 4 out of the 18 components are perfect classifiers. The more structure present in data, the more perfect classifiers are discovered. The probability of a perfect classifier occurring is precisely $\binom{c}{1}\frac{2}{2^c} = \frac{c}{2^9} = \frac{10}{512} \approx \frac{1}{50}$, where $c$ is the number of classes. This is equivalent to combinatorially choosing the position of the non-random bit (the class index) and then selecting whether it is 1 or 0. Thus, while we have an over-representation of these, they are not as over-represented as the useless classifiers. This is where expectation can minimized as well, by targeting perfect classifier selection. Thus, HyPE in [8] has a tendency to prefer perfect classifiers during the algorithm.

Interestingly, the third case has an expectation of:

$$\frac{\binom{c}{c/2}}{2^c} = \frac{\binom{10}{5}}{2^{10}} = \frac{252}{512} \approx \frac{1}{2}$$

where $c$ is the number of classes, for which we observe an under-representation. This is equivalent to combinatorially choosing which half of the bits will be 1. The better the features in the hypervectors, the less these appear. Thus, this is another target for removal in HyPE minimization. In fact, it suffices to maintain perfect classifiers only, in which the accuracy typically does not suffer, and can potentially increase. We can interpret all in-between cases as being weaker classifiers, but still better than this case, assuming one matches better with the side with less possibilities.

### 5.1.1 Causal Factorization of the HC Algebra

Consider the perfect classifiers as one distinct group of components that can be factored out of the HC Algebra and used separately. The function is clear, in that they detect the presence of 1 prototypical feature. Consider, however, what the function must be of the useless classifiers, when they are factored out. While the moniker "useless" is tongue-in-cheek, careful reflection on the fact that these cases are so over-represented reveals an inescapable conclusion: the useless classifiers fingerprint the dataset inputs. These are the specific features which don't bias themselves towards any particular class, but instead choose or don't choose them all with equal likelihood. If the dataset inputs were constructed differently, this fingerprinting cannot occur. Thus, useless classifiers identify *the problem you are trying to solve*. A nearest Hamming Distance in useless classifiers of any input hypervector will identify which problem type the hypervector model is being asked to solve, non-verbally. They are thus prototypes of the entire dataset. Suppose we had $m$ different datasets our hypervector model $M$ can solve. For a novel input hypervector $X$, computing:

$$\text{argmin}_i(\text{Ham}_{|U_i|}(X[U_i], M[U_i])) \tag{5.1}$$

where $U_i$ is the useless classifier set of component indices for task $i$ of $m$ tasks, will find the best match the task to solve.

We have a factorization for perfect and useless classifiers. However, note that there is a clear causal order to these. Specifically, you would first consult the useless classifiers on which task to solve, and then **only** consult perfect classifiers to get the most accurate prediction for the class. What about the in-between cases? Indeed, these would be consulted last, only in the case that the Hamming

Distances between the best matching and second best matching class are close enough that noise in a prototypical example could cause a mis-classification. So, how do we further factorize the remaining cases, and is it also causal? One way of doing this is to factorize for all possible $\binom{c}{i}$ cases for each $i$, starting at $i = 2$ and ending before $i = \lfloor \frac{c}{2} \rfloor$, which is the inflection point for the equal partition case. Since choose is symmetric, if $c$ is not divisible by 2, $\lfloor \frac{c}{2} \rfloor + 1$ would simply be the same case as $\lfloor \frac{c}{2} \rfloor$. These factorizations are relaxing the perfect classifier cases to consider more and more possible classes, which can sway the determination for a perfect classifier.

Indeed, we have just reversed-engineered the remainder of HyPE [8]. HyPE will choose these features to populate the hypervector if more perfect classifiers are not available across the expectation minimization. However, note that HyPE did not consider the causal ordering of these steps. Viewed as sub-objects experiencing a pushout to their super-object, ala Topos generation in our work in [7], there exist strict super sets for which we can causally check the presence of in our factorization. This completes the factorization, into a proper, causally correlated metric augmenting traditional Hamming Distance. This implies a proper Topos not only exists during the HyPE process, but it can be computed more efficiently without HyPE in a direct way through this process of causally *hypermapping* components into causally ordered sets of subsets. This means we can always use Hypermapping to get a better distance metric than Hamming Distance which is tailored-made for our specific prototypical classes.

### 5.1.2 Clustering of Prototypes

There exist more than just prototypical cases of hypervectors. Indeed, out-of-distribution prototypes are still a possibility in HC. If clustering is performed on data in a similar way to coloring

Table 5.2 we can perform Hypermapping on any arbitrary cluster that is detected. Thus, each cluster will create its own metric to replace Hamming Distance. This is now start to sound very similar to the proces of Topos generation in our work in [7]. The primary difference is the factorization process tailors the choice of sub-hypervector lengths. Instead of choosing one length for the whole process of Topos generation as described in [7], Hypermapping goes the extra distance of tailoring the length of the hypervector throughout the Topos generation process by leveraging the causal correlations. Thus, any slack introduced by normalizing the length of the hypervector to a particular size is also accounted for in the distance metric generated for a cluster.

### 5.1.3 Analysis of Hypermapping

The factorization process of the HC algebra using hypermapping yields a startling revelation! A hypervector model was a neural network all along. Perhaps the thought experiment at the beginning of this thesis was more accurate than it seemed - you can reverse engineer the behavior of a brain through building an analogous, hyperdimensional brain. The factorization process recreates a layered architecture that is incredibly similar to a neural network, fueled by a error propagation mechanism similar to backpropagation ala HyPE, and is captured by a Topos in metric through Hypermapping that approximates the underlying manifold. Bringing it all together, this implies that all neural networks can be modeled by ideal, super efficient, hyperdimensional analogues. It is precisely for this reason that HC must be studied more.

# Chapter 6: The Future of Hyperdimensional Computing in AI and ML

In this chapter, we ruminate on future work involving Hyperdimensional Computing that is motivated by the work in this thesis. We can largely split these future direction into three separate categories (no, not that kind of Category):

1. Extension of the results in this thesis to non-binary Hyperdimensional Computing
2. Approximation of neural networks and other ML and AI architectures into fully hypermapped Vector-Symbolic Architectures.
3. Hardware designed to leverage the efficiencies of Hyperdimensional Computing

We proceed to discuss these, one-by-one.

## 6.1 Extension to Non-binary Hyperdimensional Computing

There are a dozen major HC formulations out there, perhaps even more at this point. Extension of the results in this thesis to non-binary cases can yield positive outcomes for specific situations. Namely, work on Fourier-based HC formulations can reveal far better hyperdimensional algebras than projection into binary spaces alone, especially when image data or continuous value representations are concerned. The latter is a task that Fourier representation excel at in particular. Note that in our discussion on Topos generation in our work in [7], we remark that Topos generation is fundamentally a task-specific function. While all data can eventually be abstracted into dense binary hypervectors, this may not be the cases for the early stages of computation. It may be beneficial to employ other algebras and slowly translate them into dense binary hypervector spaces.

## 6.2 Approximation of Neural Networks and Other Architectures

A very particular nagging question posed by our thought experiment at the beginning of the thesis is whether or not you can perfectly approximate a brain by construction another brain, hyperdimensionally from samples of brain scans. Our results in this thesis seem to indicate this achievable, but requires engineering particular approximations. Is it possible to extend something like the work in [5] to perform fully end-to-end knowledge distillation. In knowledge distillation, a student network is trained from a larger, pretrained teacher network, to solve a task with less resources with minimal losses in accuracy. Typically this is done by linking the loss functions between the two models and optimizing the student model through backpropagation.

The tools discussed in this thesis seem to put this idea tantalizingly in our reach. We can see how embeddings can be approximated by HD-glue [5]. We can see how HyPE features could be used to foment feedback mechanisms like back-propagation [8]. With hypermapping, we can see how to perhaps regularize the errors passed through HyPE, gaining a causal understanding of how error propagates and can be minimized. One thing we cannot see how to do, however, is embed raw data into hypervector form efficiently. There is no direct, universal way to this currently, without relying on an encoder network or some kind of external process from HC.

In our work on Hyperdimensional Active Perception [3], we noted that the hardware efficiently solves the task for the hypervectors. Once relevant sparse data is directly mapping to hypervector form, the problem becomes almost easy. However, this requires engineering new neuromorphic hardware for very specific AI and ML use cases. From an Information Theory angle, we almost need to have some knowledge of how to ideally encode data. This also doesn't exist. And so, we are stuck. There exist no general methods to convert raw, arbitrary data directly into useful hypervectors. If they existed, they

would already be used in AI and ML. The hope is, however, that this thesis bring us one step closer to achieving this. For a future research direction, it seems like hypermapping is the most promising direction to achieve this, but more work must be done to confirm this.

## 6.3 Neuromorphic Hardware Solutions for HC

In our work in Hyperdimensional Active Perception [3], we noted that the neuromorphic hardware of the event camera essentially solved the problem of Ego-Motion. HC just captured the heuristics of the solution through its algebra. HC is good at abstracting away the precision of computation and data processing that we often need in computers. However, hardware solutions like event cameras are few and far between. More neuromorphic hardware must be designed for HC, to fully leverage its capabilities. While event cameras are good for motion, fundamentally better hardware for general vision needs to be invented.

Human sight, for example, is very hyperdimensionally motivated. The hardware of the eyes is designed to keep features scale and rotation invariant, by seeing in log-polar coordinates. Translation invariance is handled by the brain, which instructs the eyes to look rapidly in very particular patterns, called saccades. We recognize object by performing these saccades. Saccades seem to be slightly order invariant. One can recognize a face by first looking at a different part than what the saccading may suggest, but that amounts to being tolerant to reordering. This is something that HC is good at handling, by performing Consensus Summation. Order does not matter, so long as you detect the set of saccades you expect, the next choice in saccadding will lead to an active perception loop. This would be an interesting area of application for HC.

# Chapter 7: Conclusion

In conclusion, this thesis has provided a comprehensive study of Hyperdimensional Computing as a suitable basis on which to build future AI and ML architectures. If nothing else has been achieved, it is our hope that the reader at least walks away with an awareness of HC as a tool in their tool-set for solving such problems. Much like the Fourier Transform enables efficiencies in so many tasks related to AI and ML, HC offers similar avenues that should be tried as first solutions to any problems. While it is tempting to simply pull the trigger on a massive neural network to solve the problem for you, that avenue has led to many inefficient solutions in the modern landscape of ML. It is not necessary to use energy-hungry Large Language Models to figure out where one wants to eat dinner tonight. There are much simpler and more efficient solutions to explore. In the best of cases, we hope this thesis serves in inspiring future work into Hyperdimensional Computing, whether for AI, ML or general computation.

# Appendix A: Topoi from first principles [7]

The monograph [45] is a standard and comprehensive introduction to Topos Theory, suited both for readers already experienced with Category Theory and its applications to Computer Sciences and readers with no previous exposure to such concepts.

The founding work of Grothendieck proposes Topoi as a unifying concept for mathematical theories, a contemporary and comprehensive account is given by [46], where their role as bridges between mathematical universes is largely explored.

Informally, underlying a Category $\mathcal{C}$ is just a pair of collections $\mathbf{Obj}(\mathcal{C}), \mathbf{Arr}(\mathcal{C})$, whose elements are *objects* and *arrows*, together with an associative *composition scheme* $\circ$. Each arrow $f$ comes equipped with the information of a *source* $f$ and a *target* $f$ objects, while each object $a$ comes in pair with a special arrow: the *identity* on that object $\iota_a$. This allows us, to a great extent, to forget about objects entirely. In order to further describe the arrows of a Category, we recall some standard taxonomy.

[Monics and Epics] An arrow $f : a \to b$ is **monic** if for all pairs of *parallel arrows* $g, h : c \rightrightarrows a$ we have

$$f \circ g = f \circ h \Rightarrow g = h. \tag{A.1}$$

Dually, $f$ is **epic** when, for all $g, h : b \rightrightarrows c$,

$$g \circ f = h \circ f \Rightarrow g = h. \tag{A.2}$$

In the realm of Sets, one concludes that an arrow (function) is monic/epic iff it is injective/surjective iff it is left/right-invertible. In Category Theory, since we do not describe objects element-wise, the notion of being isomorphic replaces the stricter requirement of having actual equality.

[Isos] An arrow $f : a \to b$ is **iso** if there exists $g : b \to a$ such that

$$f \circ g = \iota_b, \qquad g \circ f = \iota_a. \tag{A.3}$$

When this is the case, we also write $a \approx b$ and say that they are **isomorphic**.

Note that isos are always both monic and epic, but the contrary is, in general, not guaranteed: take the monoid of the Naturals $(\mathbf{N}, 0, +)$ as a one-object category with arrows representing the integers, then any arrow is monic and epic but only $0$ is an iso. Categories where the contrary holds, such as **Grp** (groups and their homomorphisms), are called *balanced*.

We pause for a second to note that while "monic/epic" and "left/right-invertible" make sense for sets but also arbitrary categories, the concepts of injectivity and surjectivity are not immediately translated to categorial semantics. Indeed, they seem entirely element-based and their arrow-theoretic formulations is one of the cornerstones of Topos Theory.

Usually, there can be many *parallel* arrows between the same source and target. Objects that are connected in precisely one way to every other one are of fundamental interest.

[Initial and Terminal Objects] An object $x$ is **initial** if there is exactly one arrow $p : x \to b$ for any object $b$. Dually, it is **terminal** if there is exactly one arrow $q : a \to x$ for each $a$. An object which is both is called a **zero** object.

Initial and Terminal objects are desirable inhabitants for any suitable notion of "Universe of

Sets". A Category with an initial object is sometimes termed *pointed*. A *degenerate* category is one where all objects are isomorphic, although in Topos Theory this terminology is typically used more generally to simply mean that it has a zero object. This precludes a Category from being a Topos, with the only exception being the *Trivial Topos* (the one comprising of one object and its identity arrow only). Observe that arrows with domain/codomain a terminal/initial object are always monic/epic and that initial/terminal/zero objects are unique only up to (unique) isomorphism.

We now turn our attention to the *Extensionality* principle of Set Theory, which asserts that sets with the same elements are actually the same set, following again [45]. Its categorial form addresses the extent to which we can distinguish parallel arrows in a given category. Note that a Category in which there is at most one arrow between each pair of objects is essentially just a *Preorder*.

[Generators] An object $s$ is called a **generator** (or **separator**) if, for each pair of arrows $f, g : a \rightrightarrows b$,

$$\forall e : s \to a (f \circ e = g \circ e) \Rightarrow f = g. \tag{A.4}$$

An initial object can be a generator only in case the Category is already just a preorder. A terminal object instead, is required to be a generator in order to have a *Well-pointed Topos*. Even though **Grp** has a zero object, it also has infinitely many non-isomorphic generators. This shows how the idea of well-pointedness makes sense for arbitrary Categories and not just Toposes, and is in fact an open research topic. Hence, the following definition is somewhat non-standard.

[Well-pointedness] A Pointed Category is Well-pointed if it has a non-zero terminal generator.

The main consequence of Well-pointedness is that one can think of the monics $e : s \to a$ as[1] *generalised elements* of $a$.

[1] One is, of course, also *free* to think the same holds in the Trivial Topos.

Our next goal is to translate the ideas of *bundling* and *binding* in categorial terms. In [47] it is given a comprehensive and in depth account of the binding problem. It turns out, that its arrow reformulation leads to the idea of *colimits*. This is one of the motivations of many works of Baas, such as [48], on the foundational role of *Hyperstructures*.

A (*Covariant*) *Functor* $F$ between Categories $\mathcal{C}, \mathcal{D}$, written $F : \mathcal{C} \to \mathcal{D}$, maps objects of $\mathcal{C}$ to objects of $\mathcal{D}$ and arrows to arrows in a way that respects the composition schemes and their units, i.e. it preserves *commuting triangles*:

$$F(f \circ g) = F(f) \circ F(g), \tag{A.5}$$

$$F(\iota_a) = \iota_{F(a)}. \tag{A.6}$$

Now, in a similar way as we can think as a set function as a special kind of relation (its *graph*), a functor is a special case of the more general concept of a *Diagram*. Instead of attempting a formal definition of diagrams, which are the very essence of Category Theory, we focus on some concrete examples that are relative to this paper and Hyper-Dimensional Computing in general.

The Theory of *Nets*, was one of the main motivations behind early Category Theory. It was developed in [49] in order to generalise the idea of *sequences* and *limit points* in Topology so that an arbitrary topological space could be entirely recovered by studying the convergence of its nets (recall that this is not possible by simply using sequences).

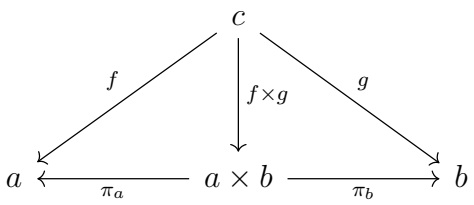
[Directed Sets] As already mentioned, any preorder $(P, \sqsubseteq)$ can be made into a category $\mathcal{P}$ with $\mathbf{Obj}(\mathcal{P}) = P$ and one arrow $f : a \to b$ whenever $a \sqsubseteq b$. Now, given two elements $a, b$ of $P$, an **upper bound** is an element $c$ satisfying $a \sqsubseteq c$ and $b \sqsubseteq c$. A preorder where upper bounds exist for all

pair of elements is called a (**upward**) **directed set**. A category where isomorphism actually reduces to equality is termed *skeletal*, so that a **Poset** is simply a skeletal preorder (i.e. a *symmetric one*). A **Totally Ordered Set** is a poset with *path-connected* objects, that is: we always either have $a \sqsubseteq b$ or vice versa (i.e. *trichotomy holds*).

[Nets] A functor $S : \mathcal{P} \to \mathcal{C}$, where $\mathcal{P}$ is a directed set, is called a **net** in $\mathcal{C}$.

[Joins and Meets] An element $c$ of a preorder $P$ is **maximal** if whenever $c \sqsubseteq a$, we also have $a \sqsubseteq c$. It is called a **greatest element** if it is terminal. Dually, we obtain **lower bounds**, **minimal elements** and **smallest elements**. Note that a preorder with a greatest/smallest element is trivially upward/downward directed. The **join/meet** of $a, b$ is, when it exists, the smallest/greatest element of the directed (sub)set of upper/lower bounds of $a$ and $b$, and is denoted by, resp., $a \vee b$ and $a \wedge b$.

In an arbitrary Category, this generalises to the idea of (co)products: given objects $a, b$, a *span* (sometimes, *internal object*) $a \times b$ is an object together with two maps (the *projections*) $\pi_a : a \times b \to b, \pi_b : a \times b \to b$. A span which is *universal*, i.e. such that given any other span $c$ with arrows $f, g : c \rightrightarrows a, b$ we have a unique $f \times g : c \to a \times b$ making




(A.7)

a *commutative diagram*, is called the *product* of $a$ and $b$. As usual, reversing the direction of all arrows leads us to the dual concept of *cospans* and, specifically, the *coproduct* of $a, b$, which is denoted $a + b$. Thanks to associativity, if a category has products for each pair of objects, it is always possible to extend the definition to any (finite) number $n$ of objects $\{a_i\}_{i \leq n}$, denoted by $\prod_{i \leq n} a_i$, with the terminal object as the limiting case when $n = 0$.

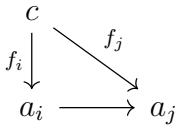
Looking back at Sets, one recognises that the product is given by the familiar notion of the *Cartesian product*. Moreover, given two sets $a, b$, the collection of arrows from $a$ to $b$ is the set $b^a$. Categories such that the families of parallel arrows carry the same structure as its objects are termed *closed*.

[Exponentials] Let $a \times b$ be the product of objects $a, b$, we say that an object $c$ together with an arrow $v : c \times a \to b$, the **evaluation**, is the **exponential** of $a$ and $b$, denoted $b^a$, if for any other $w : d \times a \to b$ there exists a unique $t : d \to c$ such that

$$v \circ (t \times \iota_a) = w. \tag{A.8}$$

[CCC] A **Cartesian Closed Category** is a Category with all finite products (including a terminal object) and all exponentials.

Now, suppose that we were given not only some objects $\{a_i\}_{i \leq n}$ but also a (finite) collection of arrows between them. A *cone over* this diagram is just an object $c$ together with one arrow $f_i : c \to a_i$ for any index $i$ such that all triangles



(A.9)

commute. Reversing directions, we find the *cocones under* a diagram.

[Completeness] A universal cone for a diagram, if it exists, is called its **limit**. A category is **finitely complete** if every finite diagram has a limit. Duality yields **colimits** as universal cocones and **finite cocompleteness**.

[Bounded Lattices] A **Bounded Lattice** is a finitely complete and finitely cocomplete poset.

In order to have a Topos, we need all finite (co)limits. For completeness, the following diagrams

will suffice in a CCC.

[Equalizers and Coequalizers] Given a parallel pair of arrows, $f, g : a \rightrightarrows b$ their (**co**)**equalizer** is, if it exists, their (co)limit.

A Cartesian Closed Category is finitely complete if and only if it has all equalizers.

[Pullback and Pushouts]

- A **pullback** is the limit of a cospan.
- A **pushout** is the colimit of a span.

A proof of the following classical result (actually, of its dual) can be found in [50].

A category is finitely cocomplete if and only if it is pointed and has all pushouts.

Thus far, we have seen how the unifying concepts of functors, diagrams and (co)limits permits us to create new objects from old ones. These constructions take the place, and generalise, the *Pairing* and *Union* axioms from Set Theory. The last ingredient needed to obtain a Topos, will take care of *Power-sets*.

First of all, as we think of elements of an object $a$ as the monics with source the terminal object (at least if the category is well-pointed), a *subobject* of $a$ is any monic with target $a$; the collection of subobjects of $a$ is denoted $\mathbf{Sub}(a)$. For sets, there is an immediate correspondence between subsets and functions with value in $2 := \{0, 1\}$, namely $\mathcal{P}(X) \approx 2^X$.

[Sub-objects Classifier] An object $\Omega$ of a category with terminal object $s$, together with an arrow $\text{TRUE} : s \to \Omega$, is called a **subobjects classifier** whenever for all monics $f : a \to b$ there is precisely

one arrow $\chi_f : b \to \Omega$, the **characteristic arrow**, making

$$\begin{array}{ccc} a & \xrightarrow{f} & b \\ \downarrow & & \downarrow{\scriptstyle \chi_f} \\ s & \xrightarrow{\text{TRUE}} & \Omega \end{array} \tag{A.10}$$

a pullback. In general, the arrows of type $s \to \Omega$ are called the **truth-values**.

With all of this in place, we can finally give a proper definition of a Topos.

[Elementary Toposes] A (**Elementary**) **Topos** is a Finitely Cocomplete Cartesian Closed Category with a subobjects classifier and all equalizers.

The explicit mention of equalizers is needed because we are following the current modern definition of CCCs, as given in [51], that requires all finite products but not all finite limits.

It is instructive to draw a comparison with the classical definition of topological space, to better understand the new perspective given by Toposes.

[General Topology] Usually, one presents a **Topological Space** as a set $T$ together with a *collection* $\mathcal{T}$ of **open** subsets containing $\emptyset, X$ and **closed** under *finite intersections* and *arbitrary unions*. We can now rephrase this as saying that $\mathcal{T}$ is a bounded lattice with union for the join and meet for the intersection with the additional presence *arbitrary* joins. The *points* of a topological space are not the objects. The connection is, of course, much deeper and, in fact, the study of sheaves and bundles from Algebraic Geometry served as the inspiration for much of the early development of the Theory of Toposes.

Now that we have sketched a path to Topoi, we conclude by giving a glimpse of how one can reason "inside" a Topos and how this relates to Manifold Learning.

According to the constructivist view of mathematics, truth is not absolute but rather context-

dependant: the truth-value of a sentence varies according to the state of knowledge regarding its matter. Indeed, the theory of topoi is deeply connected to *Kripke Frames*. An interpretation of classical *Vector Spaces* in these terms is discussed in [52], which further motivate the investigation of how models for *Modal Logic* can be realised as VSAs.

To make things easier, we can think of a frame as being simply any poset $(P, \sqsubseteq)$. Given a set $X$ and a *formula* $\phi$, the *Separation Scheme* allows us to consider the set $\{x \in X : \phi(x)\}$ of elements of $X$ that *satisfy* it. If truth is supposed to persist in time, we can instead consider the $P$-indexed collection of formulas which have been proven to hold at instant $p \in P$:

$$\{x \in X : \phi_p(x)\}.$$

The *algebras of subobjects* in the case of sets are *Boolean Algebras*, which are known to be the *models* of *Classical Propositional Logic*. The models for *Intuitionistic Logic* are instead provided by

[Heyting Algebras] A **Heyting Algebra** is a bounded lattice with all exponentials. The exponential of objects $a, b$ is referred to as **implication** and denoted by $a \Rightarrow b$.

This construction is essentially the weakest possible such that *modus ponens* is *sound*.

[Hereditary Sets] Consider a poset $(P, \sqsubseteq)$, a subset $Q \subseteq P$ is **hereditary** (or **upward closed**) if $q \sqsubseteq p$ implies $p \in Q$ whenever $q \in Q$. For any element $p \in P$, the **principal** hereditary set $\{q \in Q : p \sqsubseteq q\}$ is denoted by $[p)$.

Denoting by $P^+$ the collection of its hereditary (sub)sets, we have actually defined a topology, whose collection of open sets forms a Heyting Algebra: the exponential of $S, Q$ is obtained by taking the interior of $(P - S) \cup Q$, where $P - S$ is the complement of $S$.

The reader interested with the paradigms of *Lattice-based Computing*, always throughout the

lens of Category Theory, can find an extensive treatment in [53]. This line of research was initiated by the groundbreaking work of Lawvere, starting from [54], where a detailed description of *metric spaces* in terms of *Enriched Categories* is given. In the paper, it is also made extensive use of the notion of *Closed Categories*, where the collections of arrows carry the structure of objects themselves, and *Monoidal Categories*, which are equipped with an abstract *tensor product* (the categorial notion of bundling), both generalising CCCs.

# Bibliography


[1] Kanerva, P. (2009). Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors. *Cognitive computation, 1*(2), 139-159.

[2] Sutor Jr, P., Summers-Stay, D., & Aloimonos, Y. (2018, July). A computational theory for life-long learning of semantics. In *International Conference on Artificial General Intelligence* (pp. 217-226). Cham: Springer International Publishing.

[3] Mitrokhin, A., Sutor, P., Fermüller, C., & Aloimonos, Y. (2019). Learning sensorimotor control with neuromorphic sensors: Toward hyperdimensional active perception. *Science Robotics, 4*(30), eaaw6736.

[4] Mitrokhin, A., Sutor, P., Summers-Stay, D., Fermüller, C., & Aloimonos, Y. (2020). Symbolic representation and learning with hyperdimensional computing. *Frontiers in Robotics and AI, 7*, 63.

[5] Sutor, P., Yuan, D., Summers-Stay, D., Fermuller, C., & Aloimonos, Y. (2022, July). Gluing neural networks symbolically through hyperdimensional computing. In *2022 International Joint Conference on Neural Networks (IJCNN)* (pp. 1-10). IEEE.

[6] Amrouch, H., Imani, M., Jiao, X., Aloimonos, Y., Fermuller, C., Yuan, D., ... & Sutor, P. (2022, October). Brain-inspired hyperdimensional computing for ultra-efficient edge ai. In *2022 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS)* (pp. 25-34). IEEE.

[7] Faraone, R., Sutor, P., Fermüller, C., & Aloimonos, Y. (2024, June). Vector symbolic sub-objects classifiers as manifold analogues. In *2024 International Joint Conference on Neural Networks (IJCNN)* (pp. 1-10). IEEE.

[8] Sutor, P., Faraone, R., Fermüller, C., & Aloimonos, Y. (2025, August). HyPE: Hyperdimensional Propagation of Error. In *International Conference on Artificial General Intelligence* (pp. 241-251). Cham: Springer Nature Switzerland.

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

[10] Bick, C., & Rabinovich, M. I. (2009). Dynamical origin of the effective storage capacity in the brain's working memory. *Physical Review Letters, 103*(21), 218101.

[11] Kleyko, D., Davies, M., Frady, E. P., Kanerva, P., Kent, S. J., Olshausen, B. A., ... & Sommer, F. T. (2022). Vector symbolic architectures as a computing framework for emerging hardware. *Proceedings of the IEEE, 110*(10), 1538-1571.

[12] Cook, M. (2004). Universality in elementary cellular automata. *Complex systems, 15*(1), 1-40.

[13] Neary, T., & Woods, D. (2009, September). Small weakly universal Turing machines. In *International Symposium on Fundamentals of Computation Theory* (pp. 262-273). Berlin, Heidelberg: Springer Berlin Heidelberg.

[14] Yue, C., Long, M., Wang, J., Han, Z., & Wen, Q. (2016, February). Deep quantization network for efficient image retrieval. In *Proc. 13th AAAI Conf. Artif. Intell* (pp. 3457-3463).

[15] Zhu, H., Long, M., Wang, J., & Cao, Y. (2016, March). Deep hashing network for efficient similarity retrieval. In *Proceedings of the AAAI conference on Artificial Intelligence* (Vol. 30, No. 1).

[16] Liu, B., Cao, Y., Long, M., Wang, J., & Wang, J. (2018, October). Deep triplet quantization. In *Proceedings of the 26th ACM international conference on Multimedia* (pp. 755-763).

[17] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems, 25*.

[18] Deng, J., Dong, W., Socher, R., Li, L. J., Li, K., & Fei-Fei, L. (2009, June). Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition* (pp. 248-255). Ieee.

[19] Krizhevsky, A., & Hinton, G. (2009). Learning multiple layers of features from tiny images.

[20] Chua, T. S., Tang, J., Hong, R., Li, H., Luo, Z., & Zheng, Y. (2009, July). Nus-wide: a real-world web image database from national university of singapore. In *Proceedings of the ACM international conference on image and video retrieval* (pp. 1-9).

[21] Gray, F. (1953). Pulse code communication. *United States Patent Number 2632058*.

[22] Savage, C. (1997). A survey of combinatorial Gray codes. *SIAM review, 39*(4), 605-629.

[23] Kak, S. (2016). Generalized unary coding. *Circuits, Systems, and Signal Processing, 35*(4), 1419-1426.

[24] Smith, D., & Stanford, P. (1990, June). A random walk in Hamming space. In *1990 IJCNN International Joint Conference on Neural Networks* (pp. 465-470). IEEE.

[25] Deng, L. (2012). The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE signal processing magazine, 29*(6), 141-142.

[26] Fix, E. (1985). *Discriminatory analysis: nonparametric discrimination, consistency properties* (Vol. 1). USAF school of Aviation Medicine.

[27] Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine learning, 20*(3), 273-297.

[28] Quinlan, J. R. (1986). Induction of decision trees. *Machine learning, 1*(1), 81-106.

[29] Ho, T. K. (1995, August). Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition* (Vol. 1, pp. 278-282). IEEE.

[30] Freund, Y., & Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences, 55*(1), 119-139.

[31] Rish, I. (2001, August). An empirical study of the naive Bayes classifier. In *IJCAI 2001 workshop on empirical methods in artificial intelligence* (Vol. 3, No. 22, pp. 41-46).

[32] Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

[33] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).

[34] Bajcsy, R. (1988). Active perception. *Proceedings of the IEEE, 76*(8), 966-1005.

[35] Bajcsy, R., Aloimonos, Y., & Tsotsos, J. K. (2018). Revisiting active perception. *Autonomous Robots, 42*(2), 177-196.

[36] Aloimonos, Y. (2013). *Active perception*. Psychology Press.

[37] Mitrokhin, A., Fermüller, C., Parameshwara, C., & Aloimonos, Y. (2018, October). Event-based moving object detection and tracking. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (pp. 1-9). IEEE.

[38] Zhu, A. Z., Thakur, D., Özaslan, T., Pfrommer, B., Kumar, V., & Daniilidis, K. (2018). The multivehicle stereo event camera dataset: An event camera dataset for 3D perception. *IEEE Robotics and Automation Letters, 3*(3), 2032-2039.

[39] Renato Faraone (2025). Analogies, Metaphors, Allegories: Categorial Architectures of General Intelligence. *Università di Parma, Ph.D. thesis*.

[40] Xiao, H., Rasul, K., & Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*.

[41] Roderick J. A. Little, Donald B. Rubin (2019). Statistical Analysis with Missing Data, 3rd Ed. *John Wiley & Sons*.

[42] Jeff C. F. Wu (1983). On the convergence properties of the EM algorithm. *The Annals of Statistics, Vol. 11, No. 1, pp. 95 - 103*.

[43] Robert Goldblatt (1984). Topoi:The Categorial Analysis of Logic. *Studies in Logic and the Foundations of Mathematics Vol. 98, North-Holland*.

[44] Peter J. Freyd, Andre Scedrov (1990). Categories, Allegories. *Mathematical Library Vol. 39, North-Holland*.

[45] Goldblatt, R. (2014). *Topoi: the categorial analysis of logic* (Vol. 98). Elsevier.

[46] Caramello, O. (2018). *Theories, Sites, Toposes: Relating and studying mathematical theories through topos-theoretic'bridges'*. Oxford University Press.

[47] Ehresmann, A. C., & Vanbremeersch, J. P. (2007). *Memory evolutive systems; hierarchy, emergence, cognition* (Vol. 4). Elsevier.

[48] Baas, N. A. (1992). Hyperstructures-a framework for emergence, hierarchies and complexity. In *Proceedings du Congres sur I'emergence dans les modeles de la cognition* (pp. 67-93).

[49] Moore, E. H., & Smith, H. L. (1922). A general theory of limits. *American journal of Mathematics, 44*(2), 102-121.

[50] Herrlich, H., & Strecker, G. E. (2007). Category Theory, Sigma Series in Pure Mathematics, vol. 1.

[51] Johnstone, P. T. (2002). *Sketches of an Elephant: A Topos Theory Compendium: Volume 2* (Vol. 2). Oxford University Press.

[52] Greco, G., Liang, F., Moortgat, M., Palmigiano, A., & Tzimoulis, A. (2019). Vector spaces as Kripke frames. *arXiv preprint arXiv:1908.05528*.

[53] Hofmann, D., Seal, G. J., & Tholen, W. (Eds.). (2014). Monoidal Topology: A Categorical Approach to Order, Metric, and Topology (Vol. 153). Cambridge University Press.

[54] Lawvere, F. W. (1973). Metric spaces, generalized logic, and closed categories. *Rendiconti del seminario matématico e fisico di Milano, 43*(1), 135-166.