UNIVERSITY OF CALIFORNIA SAN DIEGO

Enhancing Deep Learning Efficiency: A Hyperdimensional Computing Approach

A dissertation submitted in partial satisfaction of the
requirements for the degree Doctor of Philosophy

in

Computer Science (Computer Engineering)

by

Rishikanth Chandrasekaran

Committee in charge:

Professor Tajana Simunic Rosing, Chair
Professor Chung-Kuan Cheng
Professor Mingu Kang
Professor Ryan Kastner

2024

The Dissertation of Rishikanth Chandrasekaran is approved, and it is acceptable in quality and form for publication on microfilm and electronically.

University of California San Diego

2024

DEDICATION

To my parents, Chandrasekaran, and Nirmala,
to my partner Nandhini,
and my friends Shreyg and Amith.

EPIGRAPH

Not all those who wander are lost.

J.R.R Tolkein

You write with ease to show your breeding,
But easy writing's curst hard reading.

Richard Brinsley Sheridan

Writing, at its best, is a lonely life. Organizations for writers palliate the writer's loneliness, but I doubt if they improve his writing. He grows in public stature as he sheds his loneliness and often his work deteriorates. For he does his work alone and if he is a good enough writer he must face eternity, or the lack of it, each day.

Ernest Hemingway

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

First and foremost, I owe an enormous debt of gratitude to my advisor, Prof. Tajana Rosing. Her decision not to kick me out when she had the chance was pivotal to my success. Her patience, understanding, and unwavering support allowed me to grow throughout this journey. This doctorate would have been impossible without her guidance. Thank you, Professor Rosing—I am forever grateful and indebted.

A big thank you goes to my colleagues in SEELab for their help, feedback, and the many shared memories. Special thanks to Anthony Thomas, Xiaofan Yu, Flavio Ponzina, Quanling Zhao, Kazim Ergun, and Michael Ostertag for always finding time to assist me. A particular shout-out to Anthony Thomas, whose brilliance in theory sparked my own appreciation for it. Anthony, you're a fantastic teacher who patiently answered all my (sometimes less-than-brilliant) questions, and I owe my theoretical knowledge to the spark you ignited.

Embarking on this PhD journey was the toughest challenge of my life, and I couldn't have done it without my partner, Nandhini. Thank you for being my rock and for your unwavering support. To my friends, who were my steadfast pillars of strength, I am eternally grateful. Shreyg, my best friend, thank you for always believing in me and injecting much-needed humor into my life. To my buddy Amith, cheers to our "mid-week crisis" escapades and for always being there when I needed you. A special shout-out to the MS Commons, ChezBob in the CSE building, and Mesa Nueva. These places are filled with memories, especially with Indu and Mithun—thank you both for those two invaluable years of fun and chasing deadlines in the "Pacific RIM".

Lastly, but most importantly, I extend my deepest thanks to my parents, Chandrasekaran and Nirmala. Your patience has been tested countless times, yet your dedication to my success never wavered. I am eternally grateful and indebted to you both.

I would also like to thank CRISP, PRISM and CoCoSys, centers in JUMP1.0 and 2.0 (SRC programs sponsored by DARPA), SRC Global Research Collaboration grants (GRC TASK 3021.001), and NSF grants 2003279, 1911095, 1826967, 2100237, and 2112167 for supporting my work.

Chapter 2 in full, is a reprint of the material as it appears in FHDnn: Communication Efficient and Robust Learning for AIoT Networks. DAC '22: Proceedings of the 59th ACM/IEEE Design Automation Conference. The dissertation author was the primary investigator and author of this paper.

Chapter 3 in full, is a reprint of the material as it appears in Federated Hyperdimensional Computing. ACM Transactions on the Internet of Things (Under review). The dissertation author was one of the primary investigators and author of this paper.

Chapter 4 in full, is a reprint of the material as it appears in Multi-Label Classification With Hyperdimensional Representations. IEEE Access, Volume 11, 2023. The dissertation author was the primary investigator and author of this paper.

Chapter 5 in full, is a reprint of the material as it appears in NeuroHD: Neuro-Symbolic Classification with Hyperdimensional Computing, submitted to ISLPED 2024. The dissertation author was the primary investigator and author of this paper.

VITA

2014	Bachelor of Engineering in Electrical and Electronics Engineering, Anna University, India
2017	Master of Science in Computer Engineering, Columbia University, New York
2024	Doctor of Philosophy, Computer Science (Computer Engineering), University of California San Diego

PUBLICATIONS

R Chandrasekaran, F Asgareinjad, J Morris, T Rosing "Multi-label classification with Hyperdimensional Representations" *IEEE Access Journal* 2023

R Chandrasekaran, K Ergun, J Lee, D Nanjunda, J Kang, T Rosing "FHDnn: Communication efficient and robust federated learning for aiot networks" *Proceedings of the 59th ACM/IEEE Design Automation Conference* 2022

A Dutta, S Gupta, B Khaleghi, R Chandrasekaran, W Xu, T Rosing "Hdnn-pim: Efficient in memory design of hyperdimensional computing with feature extraction", *Proceedings of the Great Lakes Symposium on VLSI* 2022

S Xia, R Chandrasekaran, Y Liu, C Yang, TS Rosing, X Jiang, "A drone-based system for intelligent and autonomous homes", *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems* 2021

Rishikanth Chandrasekaran, Yunhui Guo, Anthony Thomas, Masimiliano Menarini, Michael Ostertag, Tajana Rosing, "Efficient Sparse Processing for Smart Home Applications", *ACM Conference on Embedded Network Sensor Systems SenSys* 2019

Peter Wei, Xiaoqi Chen, Jordan Vega, Stephen Xia, Rishikanth Chandrasekaran, Xiaofan Jiang, "A Scalable System for Apportionment and Tracking of Energy Footprints in Commercial Buildings", *ACM Transaction on Sensor Networks (TSON)* 2018

Daniel de Godoy, Bashima Islam, Stephen Xia, Md Tamzeed Islam, Rishikanth Chandrasekaran, Yen-Chun Chen, Shahriar Nirjon, Peter R Kinget, Xiaofan Jiang, "Paws: A wearable acoustic system for pedestrian safety", *IEEE/ACM Third International Conference on Internet-of-Things Design and Implementation (IoTDI)* 2018

Peter Wei, Xiaoqi Chen, Jordan Vega, Stephen Xia, Rishikanth Chandrasekaran, Xiaofan Jiang, "ePrints: a real-time and scalable system for fair apportionment and tracking of personal energy

footprints in commercial buildings", *ACM International Conference on Systems for Energy-Efficient Built Environments (BuildSys)* 2017

Peter Wei, Xiaoqi Chen, Rishikanth Chandrasekaran, Fengyi Song, Xiaofan Jiang, "Adaptive and Personalized Energy Saving Suggestions for Occupants in Smart Buildings", *ACM International Conference on Systems for Energy-Efficient Built Environments (BuildSys)* 2016

Rishikanth Chandrasekaran, Daniel de Godoy, Stephen Xia, Md Tamzeed Islam, Bashima Islam, Shahriar Nirjon, Peter Kinget, Xiaofan Jiang, "SEUS: A Wearable Multi-Channel Acoustic Headset Platform to Improve Pedestrian Safety", *ACM Conference on Embedded Network Sensor Systems (SenSys)*, 2016

Peter Wei, Xiaoqi Chen, Rishikanth Chandrasekaran, Fengyi Song, Xiaofan Jiang, "Personal energy footprint in shared building environment", *International Conference on Information Processing in Sensor Networks (IPSN)* 2016

C Rishikanth, Harini Sekar, Gautham Rajagopal, Ramesh Rajesh, Vineeth Vijayaraghavan, "Low-cost intelligent gesture recognition engine for audio-vocally impaired individuals", *Global Humanitarian Technology Conference (GHTC)* 2014

ABSTRACT OF THE DISSERTATION

Enhancing Deep Learning Efficiency: A Hyperdimensional Computing Approach

by

Rishikanth Chandrasekaran

Doctor of Philosophy in Computer Science (Computer Engineering)

University of California San Diego, 2024

Professor Tajana Simunic Rosing, Chair

The recent advances in Deep Learning (DL) have changed the landscape of Machine Learning significantly. However, over the last few years, we observe that to enable complex applications with DL, an increasing amount of compute resources are required for deployment. This has several undesirable effects ranging from increased deployment costs and energy consumption to harmful environmental effects. Recently, as the community continues to explore alternate paradigms for machine learning, Hyperdimensional Computing (HDC) has gained popularity, due to its low-footprint, low energy consumption, and ease of acceleration on parallel hardware. However, prior work has shown that HDC is not sufficiently accurate on a few complex applications. This dissertation explores the synergy between DL and HDC for improving the

compute efficiency of learning while keeping state-of-the-art accuracy. We propose novel hybrid models that leverage the salient properties of HDC and DL to address each methods drawbacks.

We first introduce FHDnn, a hybrid architecture for federated learning that leverages HDC to improve compute efficiency and robustness to unreliable network conditions in IoT Networks. Our architecture uses a frozen pre-trained CNN for feature extraction with a HDC classifier that is trained in a federated manner. By training and transmitting only the HDC classifier, we avoid having to transmit large DNN models and consequently mitigate the robustness issues. Our experiments show that our proposed methods improve communication efficiency by $66\times$ and are $6\times$ faster than federated learning with DNNs. We then extend this work to analyze the convergence properties of FHDnn and show that FHDnn provides convergence guarantees for federated learning.

Next, we propose systematic approaches for designing hybrid architectures for different modalities of data. We leverage the binding operator in HDC to capture relationships between data and generate HDC representations that capture these relationships in a single vector. We then use these informative data representations as input to DNN models. This facilitates ease of learning as the model no longer needs to learn relationships in data from scratch. We first consider text data for multi-label classification and demonstrate on large-scale real-world datasets that our proposed architecture is $231\times$ smaller and $16\times$ faster compared to SoA models.

Finally, we propose a novel architecture for image classification that leverages the symbolic properties of HDC to structure the problem hierarchically to reduce learning complexity. We construct a label hierarchy by grouping together similar images into groups and generate label representations using HDC, that captures these group relationships. This allows us to break down the classification into 2 stages: detecting the group the image belongs to and identifying the specific label within the group. We show that this improves efficiency by up to $200\times$ compared to SoA models with minimal loss in accuracy.

Chapter 1

Introduction

The field of Machine Learning (ML) has experienced a transformative shift with the advent and rapid advancement of Deep Learning (DL). DL has enabled breakthroughs across a plethora of applications, ranging from natural language processing and computer vision to autonomous systems and healthcare. These advancements have been driven by the availability of large datasets, sophisticated neural network architectures, and powerful computational resources. However, the proliferation of DL comes at a significant cost: the ever-increasing demand for computational power. As DL models grow in complexity and size, they require substantial compute resources for both training and deployment, leading to several pressing issues.

First, the heightened computational demands translate to increased deployment costs, which can be prohibitive for many organizations. Second, the energy consumption associated with running these models is substantial [187], raising concerns about the environmental impact. The carbon footprint of large-scale DL models is non-trivial, contributing to the global climate crisis [157]. Therefore, there is a compelling need to explore alternative paradigms and methods that can mitigate these drawbacks while retaining the benefits of DL.

1.1 Hyperdimensional Computing: An Emerging Paradigm

In the quest for more efficient computational paradigms, Hyperdimensional Computing (HDC) has emerged as a promising candidate. HDC is inspired by the properties of high-

dimensional spaces and leverages the principles of vector symbolic architectures [95]. It offers several advantageous features, such as low computational footprint, reduced energy consumption, and inherent parallelism, making it suitable for deployment on parallel hardware [73]. These characteristics position HDC as a viable alternative for scenarios where computational efficiency is paramount.

1.1.1 Hyperdimensional Computing Background

Hyperdimensional computing (HDC) uses high-dimensional vectors called hypervectors [160, 55, 92] to represent data. The basic idea behind HDC is to represent structured or symbolic data using hypervectors and then provide a set of mathematical operations to manipulate these vectors like symbolic objects. These operations are associative, commutative, and distributive[123], they operate element-wise, allowing them to be performed in parallel, making HDC an attractive approach for implementing hardware-accelerated, energy-efficient computing.

Hypervectors are typically represented as binary or bipolar vectors in a high-dimensional space. Mathematically, a hypervector is represented by a vector $X \in \{+1, -1\}^D$ where D is the dimensionality of the vector space. The dimensionality of the hypervector is often much larger than the number of dimensions required to represent the data, enabling the vector to encode many concepts or attributes in a single representation. For instance, a hypervector representing an object might contain attributes such as color, shape, texture, and position.

HDC Operations

HDC provides three fundamental operations: bundling, binding, and similarity check. These operations are implemented differently in various HDC models, and we will briefly explain their usage under the Multiply-Add-Permute (MAP) model. In the MAP framework, hypervectors are bipolar and can be represented as $X = \{+1, -1\}^D$.

Bundling: The bundling operation is used to represent multiple symbolic entities using a single hypervector. This operation is denoted by the $\oplus$ symbol and can be expressed as:

$$\text{bundle}(X_1, X_2, ..., X_n) = X_1 \oplus X_2 \oplus ... \oplus X_n$$

where $X_1, X_2, ..., X_n$ are hypervectors representing the symbolic entities. The result of the bundling operation is a new hypervector that represents the combination of all the input entities. For MAP the bundling operation is a simple element-wise sum of the hypervectors. Under the similarity check metric defined below, the resultant hypervector is similar to its constituent hypervectors.

Binding The binding operation is used to associate one entity with another and is denoted by the $\otimes$ symbol. The binding operation is defined as the element-wise multiplication of two hypervectors and can be expressed as:

$$\text{bind}(X, Y) = X \odot Y$$

where X and Y are the hypervectors representing the two entities to be associated. The result of the binding operation is a new hypervector that encodes the relationship between the two input entities. By the similarity metric, the resultant hypervector is orthogonal to the input hypervectors.

Similarity check Finally, the similarity check operation is used to determine the degree of similarity between two hypervectors. The similarity check operation is defined as the dot product between two hypervectors and can be expressed as:

$$\text{similarity}(X, Y) = X \cdot Y$$

where X and Y are the two hypervectors to be compared. The result of the similarity check operation is a scalar value that represents the degree of similarity between the two hypervectors, with higher values indicating greater similarity.

Together, these operations provide a powerful and flexible approach to representing and

manipulating symbolic data in a distributed and parallel fashion, enabling the development of novel machine learning algorithms and cognitive models.

HDC Learning

The HDC learning process involves the encoding of data and its inherent relationships within hypervectors. These vectors are then subjected to a set of mathematical operations, enabling the extraction of useful patterns and relationships within the data. Learning in HDC involves three steps: encoding the data, learning on the encoded data and inference.

Encoding: The first step in learning with HDC involves encoding the input data into high-dimensional hypervectors. The goal of this step is to create a distributed representation of the input data that can capture its semantic properties.

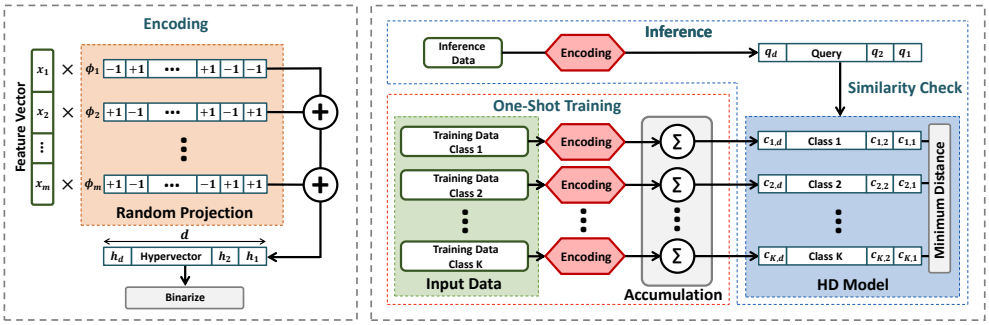
Given an input vector $X \in R^d$, we generate a set of random projection vectors $R_1, \ldots, R_K \in R^D$, where $D >> d$. The projection of X onto the k^{th} random projection vector is given by the dot product $X \cdot R_k$. The resulting set of K projections can be represented as a hypervector $H \in \{+1, -1\}^K$, where $H_i = sign(X \cdot R_i)$. Random projection encoding in HDC has been shown to preserve Euclidean distance in the original vector space, mapping it to angular distance in the high-dimensional space[194]. This similarity-preserving nature makes it suitable for encoding data by retaining complex relationships between them.

Training: In HDC, training typically involves two steps: one-shot training followed by iterative retraining. One-shot training involves representing each class with a centroid hypervector that is the average of hypervectors representing the training examples for that class. Retraining involves updating the centroid hypervectors using a simple perceptron-style algorithm [177] in an iterative process that runs until convergence. During retraining, the centroid hypervectors are updated when a sample is mispredicted, with updates applied to both the correct label and the mispredicted label.

Inference: The centroid vectors can be used to classify new data by measuring the similarity of the new data to each centroid vector. The class with the highest similarity measure

is chosen as the predicted class for the new data point. Hamming distance similarity is used for binary hypervectors, while cosine similarity can be used for any other type of data.

Fig. 1.1a shows an overview of HDC encoding and b shows the overview of training and inference.



(a) HDC encoding functionality **(b)** HDC classification overview

Figure 1.1. Hyperdimensional learning overview

1.1.2 Synergy between HDC and Deep Learning

Despite its advantages, HDC faces limitations, particularly in handling complex applications. Prior research has shown that while HDC performs well in simple tasks, its efficacy diminishes with increasing application complexity. To address these challenges, this dissertation focuses on combining HDC with DL to harness the strengths of both paradigms. By integrating the high-dimensional, symbolic nature of HDC with the powerful feature extraction capabilities of DL, we aim to develop more efficient and effective machine learning models.

One critical issue in neural networks, known as the binding problem, involves the difficulty in representing and manipulating relationships between different components of data (e.g., associating features to form a coherent object) [204]. Traditional neural networks often struggle with this problem, particularly in tasks requiring the integration of disparate pieces of information [60]. HDC, with its vector symbolic architectures, offers a potential solution by providing a mechanism for binding different features into high-dimensional vectors that can be manipulated symbolically. This allows for more robust and interpretable representations of

complex data relationships.

Existing works that attempt to combine HDC with CNNs have made strides in addressing some of these limitations. For instance, Liao and Yuan [126] introduced a method that replaces convolution operations with circular convolution to reduce model size and inference time by leveraging Holographic Reduced Representations (HRR). Similarly, Danihelka et al. [35] proposed embedding HRRs in Long Short-Term Memory (LSTM) networks by incorporating complex weights and activations. However, these approaches do not fully leverage the symbolic properties or employ the symbolic manipulations offered by the HDC framework.

In contrast, our work utilizes the symbolic properties of HDC, exploiting its ability to represent and manipulate symbolic entities. Our methods harness the powerful capabilities of DL to learn mappings between high-dimensional representations of data, thereby enabling DL to leverage the symbolic relationships embedded in these representations.

1.2 Related Works

Deep learning has revolutionized various domains, including IoT networks, multi-label text classification, and image classification. However, the high computational cost of deep learning models poses significant challenges in resource-constrained environments [145, 38]. In IoT networks, deep learning has been applied for tasks such as anomaly detection, predictive maintenance, and resource management [155, 223]. The limited computational resources of edge devices and the need for real-time processing make the deployment of deep learning models challenging [215]. Federated learning has emerged as a promising approach to address privacy and communication overhead issues in IoT networks [121, 88], but the computational efficiency of federated learning remains a concern, as the training process still requires significant resources on the edge devices [139, 19]. Similarly, in multi-label text classification, a fundamental task in natural language processing with applications in document categorization, sentiment analysis, and information retrieval [120, 218], deep learning models such as Convolutional Neural Networks

(CNNs) and Recurrent Neural Networks (RNNs) have achieved state-of-the-art performance [102, 130]. However, the large number of parameters and extensive computational resources required by these models make them challenging to deploy in resource-constrained environments [9, 175]. The high dimensionality of text data and the need to capture complex relationships between labels contribute to the computational complexity of these models [16]. In the domain of image classification, deep learning models, particularly Convolutional Neural Networks (CNNs), have dominated the field [110, 127].

While CNNs and similar models have achieved impressive performance on benchmark datasets, their computational complexity and memory requirements have grown significantly [3, 66]. The high-dimensional nature of image data and the need for deep architectures to capture hierarchical features contribute to the computational burden [216]. Efforts to develop efficient CNN architectures, such as MobileNets and ShuffleNets, aim to reduce the computational cost while maintaining competitive performance [69, 220]. However, these models still require significant resources compared to traditional machine learning approaches [31, 36]. The high computational cost of deep learning models in IoT networks, multi-label text classification, and image classification necessitates the development of more efficient approaches. Hybrid architectures that leverage the strengths of HDC and deep learning have the potential to address this challenge, enabling the deployment of powerful models in resource-constrained environments.

1.3 Research Objective and Contributions

Motivated by the computational challenges faced by deep learning models in IoT networks, multi-label text classification, and image classification, as discussed in the related works section, this dissertation aims to improve the compute efficiency of deep learning by exploring the synergy between HDC and DL. The primary objective is to develop novel hybrid models that leverage the complementary strengths of HDC and DL, thereby overcoming their individual limitations. To address the computational efficiency challenges in IoT networks, where the

limited resources of edge devices and the need for real-time processing hinder the deployment of deep learning models [215], we introduce FHDnn, a hybrid architecture designed for federated learning scenarios. FHDnn leverages the computational efficiency and robustness of HDC to improve performance under unreliable network conditions, which are common in IoT environments [155, 223]. Our experimental results demonstrate that FHDnn is up to 22$\times$ faster and up to 6$\times$ more efficient than existing models, making it a promising solution for resource-constrained IoT networks. We analyze the convergence properties of FHDnn and provide theoretical guarantees for its performance in federated learning, ensuring that the proposed architecture not only improves efficiency but also maintains reliability and robustness, addressing the concerns raised in [139, 19].

In the context of multi-label text classification, where the high dimensionality of text data and the need to capture complex relationships between labels contribute to the computational complexity of deep learning models [16], we propose systematic approaches for designing hybrid architectures tailored to this data modality. Our architecture demonstrates a significant reduction in size (231$\times$ smaller) and an increase in speed (16$\times$ faster) compared to state-of-the-art models, validated on large-scale real-world datasets. This contribution addresses the challenges of deploying deep learning models for multi-label text classification in resource-constrained environments [9, 175].

Finally, we introduce a novel architecture for image classification that leverages the symbolic properties of HDC to structure the problem hierarchically, reducing learning complexity and enhancing efficiency by up to 200$\times$ while maintaining competitive performance with state-of-the-art models. This contribution tackles the computational complexity and memory requirements of deep learning models in image classification [3, 66], which have grown significantly due to the high-dimensional nature of image data and the need for deep architectures to capture hierarchical features [216]. By leveraging the strengths of HDC, our architecture provides a more efficient alternative to existing approaches, such as MobileNets and ShuffleNets [69, 220], which still require significant resources compared to traditional machine learning approaches [31, 36].

These contributions represent a significant advancement in the field of machine learning, paving the way for more efficient and viable deep learning models, particularly in resource-constrained environments such as IoT applications. By addressing the computational challenges identified in the related works section of this thesis, this work enables the deployment of powerful machine learning models in a wider range of scenarios, where computational efficiency is of paramount importance.

Chapter 2

FHDnn: Communication Efficient and Robust Learning for AIoT Networks

2.1 Introduction

A group of distributed edge devices communicating with each other and sharing data is loosely termed the Internet of Things (IoT). These edge devices are privy to a rich source of data which when leveraged can enable various smart applications such as smart cities [131] [5] and AI-enabled farming [152]. However, often the private and sensitive nature of the data coupled with high transmission costs prevent the central aggregation of data to the cloud. Recent advances in edge computing enabled the idea of distributed computing for on device processing. One such distributed learning paradigm is federated learning (FL) [138]. FL learns a machine learning model on data distributed across various devices without having to aggregate them centrally. FL works by training models locally on the device with data visible to each device and then averages these models from all participating devices.

Transmission costs, unreliable networks, and limited on device computer power pose

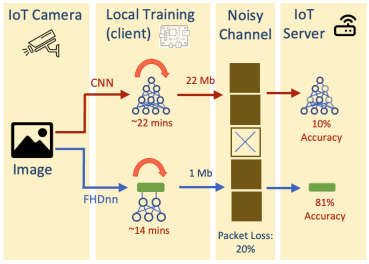


Figure 2.1. FHDnn against CNNs for federated learning

significant challenges for FL. Previous works [22, 20] have explored model compression methods and dropout techniques to reduce the communication cost by decreasing the size of the model updates. However, these methods do not factor in the non-ideality of IoT networks, assuming reliable lossless communication and subsequently are neither robust to network errors nor provide guarantees for convergence. Also, IoT often uses Low Power Wide Area Networks (LP-WAN) to conserve energy of battery operated edge devicesm but it has limited bandwidth, high packet loss and no sophisticated coding scheme making FL vulnerable to errors.

We present FHDnn a novel synergetic federated learning framework that combines 2 different learning paradigms of Deep Learning and Hyperdimensional Computing (HDC) [93]. Deep learning excels at learning a complex hierarchy of features and boasts high accuracy however at the cost of compute power often requiring GPUs to train. HDC on the other hand features lightweight training using simple operations on distributed low precision representations that are inherently robust to errors. However, they don't enjoy the same accuracy as deep learning due to their inability to learn features automatically. FHDnn combines the complimentary salient features of both learning methodologies to enable a lightweight highly robust FL framework that addresses each of the above challenges. In this work, we limit ourselves to the problem of image classification, a common application in IoT.

FHDnn uses a pre-trained Convolutional Neural Network (CNN) as a feature extractor, the output of which are encoded into hypervectors that are then used for training a federated HD learner. Specifically we utilize a CNN trained using SimCLR [27] a contrastive learning framework which learns informative representations of images in a self-supervised manner that generalizes well to several datasets. FHDnn avoids the transmission of the CNN and instead trains only the HD classifier in a federated manner. This simple strategy accelerates learning, reduces transmission cost and utilizes the inherent robustness of HDC to tackle network errors as shown in Figure 3.1. In this work, we detail the architecture of FHDnn and systematically compare the performance of FHDnn with CNN under various settings. We summarize our key contributions below:

- We propose FHDnn, a novel FL strategy that is robust to lossy network transmission, is incredibly lightweight to train, and converges faster.
- We empirically show through numerous experiments that FHDnn is robust to lossy network conditions. Specifically we evaluate FHDnn under three different unreliable network settings: packet loss, noise injection, and bit errors.
- We show that our approach reduces the communication costs by 66×, and reduces the local computations cost on the clients by up to 6× while being robust to lossy transmissions.

2.2 Background and Motivation

2.2.1 IoT networks

IoT networks often involve a large number of battery operated edge devices operating on a Low Power Wide Area Networks (LPWAN). LPWAN networks have limited bandwidth, narrow spectrum, and often lack any advanced modulation schemes due to compute cost and power constraints. Further, the network performance is worse due to the presence of packet loss which is highly prevalent in these networks [158, 196]. A study [136] shows that retransmission is non ideal as it further increases energy consumption and reduces network performance due to limited capacity. However, [71] shows that tolerating a packet loss rate of 20% allows for increased energy efficiency and network capacity.

2.2.2 Challenges in FL

Communication Efficiency: FL involves multiple rounds of communication typically until a target test accuracy is achieved. Each round, the participating clients send their models to the server. Complex models, like CNNs, contain millions of parameters resulting in large updates. FL typically takes several rounds to converge to the optimum which further exacerbates the communication cost.

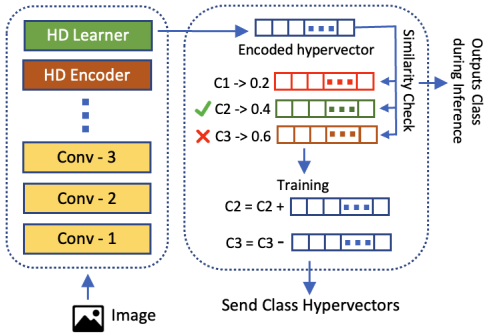


Figure 2.2. FHDnn Model Architecture

Lossy Transmission: As detailed above, IoT networks are often unreliable which adds noise to the model updates causing convergence issues. CNNs in particular are not robust to noise on weights as shown in [6]. Failure to converge results in poor accuracy while longer convergence times results in increased communication costs.

Resource constraints: Battery operated edge devices typical to IoT networks have limited power and computation budgets. CNN based FL methods require edge devices to perform on-device backpropogation during each round of training which is computationally expensive incurring high resource usage.

2.3 Proposed Method: FHDnn

2.3.1 Model Architecture

Figure 3.5 shows the model architecture of FHDnn. In the following subsections we detail the 2 components of FHDnn: i) a pre-trained CNN as a feature extractor, ii) a federated HD learner, followed by the training methodology.

2.3.2 Feature extractor

While in theory any standard CNN can be used as a feature extractor, due to its salient characteristics we use a pre-trained SimCLR Resnet model as our feature extractor. SimCLR [27] is a contrastive learning framework which learns representations of images in a self-supervised manner by maximizing the similarity between latent space representations of different

augmentations of a single image. This class agnostic framework trained on a large image dataset allows for transfer learning over multiple datasets, (as evaluated in [27]) making it ideal for a generic feature extractor. Standard CNNs learn representations that are fine-tuned to optimize the classification performance of the dense classifier at the end of the network. Since SimCLR focuses on learning general representations as opposed to classification oriented representations, it is a better choice of feature extractor. Note that We choose the ResNet architecture due to availability of pre-trained models. One could use other models such as MobileNet, which are more ideal for edge devices with resource constraints.

2.3.3 HD learner

HDC is a computing paradigm based on biologically plausible models of data representation [92]. HD works by encoding data into low precision vectors of very large dimensions, referred to as hyper vectors in literature. HD classifiers operate on these vectors using binding and bundling operations which are simple and highly parallelizable.

Here we are concerned with encoding the output of a neural network. We use an encoding method proposed in [79] based on the notion of random projection. This approach embeds the data into a high-dimensional Euclidean space under a random linear map before quantizing them. More formally, given a point $x \in \mathscr{X}$, the features $z \subset Z^n$ are extracted using the feature extractor $f : \mathscr{X} \to Z$ where f is a pre-trained neural network. The HD embedding is constructed as $\phi(z) = \text{sign}(\Phi z)$ under the encoding function $\phi : Z \to \mathscr{H}$ the rows of which $\Phi \in R^{d \times n}$ are generated by randomly sampling directions from the n-dimensional unit sphere. $\text{sign}(\Phi z)$ is the element-wise sign function returning $+1$ if $\Phi z \geq 0$ and -1 otherwise.

2.3.4 Federated Training

Figure 3.6 summarizes the federated training process for FHDnn.

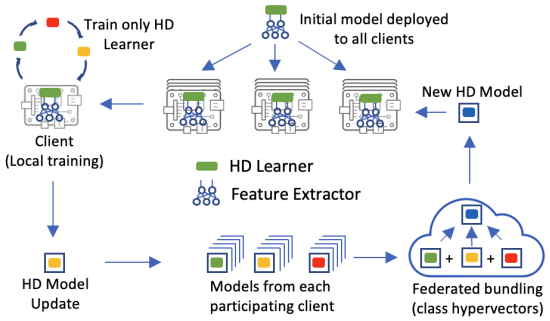


Figure 2.3. FHDnn Federated Training

Client Local Training:

Each client initially starts with a feature extractor f and an untrained HD learner. Once we get the encoded hypervectors using the method described above, we create class prototypes by bundling together hypervectors of the corresponding class using $\mathbf{c}_k = \sum_i \mathbf{h}_i^k$. Inference is done by computing the cosine similarity metric between a given encoded data point with each of the prototypes, returning the class which has maximum similarity. After this one-shot learning process we iteratively refine the class prototypes by subtracting the hypervectors from the mispredicted class prototype and adding it to the correct prototype as shown in Figure 3.5. We define the complete HD model $\mathbf{C}$ as the concatenation of class hypervectors, i.e., $\mathbf{C} = [\mathbf{c}_1^T, \mathbf{c}_2^T, ..., \mathbf{c}_n^T]$.

Federated Bundling:

In the *federated bundling* framework, each client maintains its own HD model and participates to build a global model in a distributed fashion. This is achieved via an iterative training procedure for which we describe one round (say t-th) of the algorithm below.

[noitemsep, topsep=0pt, leftmargin=*]**Broadcast:** The central server broadcasts the latest global HD model, $\mathbf{C}^t$, to all clients. **Local updates:** Each participating client $n \in [N]$ sets its model $\mathbf{C}_t^n = \mathbf{C}_t$ and then performs training for E epochs using local data as described in 2.3.4 **Aggregation:** The central server receives and aggregates the local models to produce

a new global model:

$$\mathbf{C}_{t+1} = \sum_{n=1}^{N} \mathbf{C}_{t+1}^{n}. \tag{2.1}$$

After aggregation, the server moves on to the next round, $t+1$. This procedure is carried out until sufficient convergence is achieved.

2.3.5 FL Over Unreliable Channels With FHDnn

Federated learning is often carried out over wireless channels that attenuate the transmitted signal and introduce noise followed by packet losses. The centralized server is assumed to be able to broadcast the models reliably, error-free at arbitrary rates, which is a common assumption in many recent works. For uplink communications, the channel capacity per client is more constrained and unreliable due to shared wireless medium, even at very low rates.

The bandwidth allocated per client decreases with the number of clients, so does the capacity. Accordingly, the volume of data that can be conveyed reliably, i.e, throughput, scales by $1/N$. This implies that the data rates will be small, resulting in slow training speed unless transmission power is increased, which is undesirable considering energy consumption concerns.

Instead of limiting the rate to achieve error-free communication, we admit errors for the channel output at the server as perturbations in the client models can be tolerated to an extent by FHDnn. If the model is robust to errors, then there is no need for perfectly reliable transmissions. Thus, we analyze FHDnn assuming that the clients communicate over unreliable channels and the transmitted models are corrupted.

We consider three error models at different layers of the network stack. All models are applicable in practice depending on the underlying protocol. We first explore the properties of HD computing that makes the learning robust under the considered error models, then introduce different techniques for further improvement.

Noisy Aggregation.

Conventional systems use source and channel coding to ensure reliability which are often unavailable in LPWAN networks. For noisy aggregation, as an alternative to the conventional pipeline, we assume uncoded transmission. This scheme bypasses the transformation of the model to a sequence of bits, which then need to be mapped again to complex-valued channel inputs. Instead, the real model parameter values are directly mapped to the complex-valued samples transmitted over the channel. Leveraging the properties of uncoded transmission, we can treat the channel as formulated in Equation (3.28), where the additive noise is directly applied to model parameters. The channel output received by the server for client k at round t is given by

$$\tilde{\mathbf{C}}_t^k = \mathbf{C}_t^k + \mathbf{n}_t^k \tag{2.2}$$

where $\mathbf{n}_t^k \sim \mathcal{N}(0, \sigma_{t,k}^2)$ is the $d \times n$-dimensional additive noise. Then, the signal-to-noise ratio (SNR) is:

$$SNR_{t,k} = \frac{E\|\mathbf{C}_t^k\|^2}{E\|\mathbf{n}_t^k\|^2} = \frac{P_{t,k}}{\sigma_{t,k}^2} \tag{2.3}$$

An immediate result of federated bundling is the improvement in the SNR for the global model. When the class hypervectors from different clients are bundled at the server, the signal power scales up quadratically with the number of clients N, whereas the noise power scales linearly. Assuming that the noise for each client is independent, we have the following relation:

$$SNR_t = \frac{E\|\sum_{k=1}^N \mathbf{C}_t^k\|}{E\|\sum_{k=1}^N \mathbf{n}_t^k\|} \approx \frac{N^2 P_{t,k}}{N\sigma_{t,k}^2} = N \times SNR_{t,k} \tag{2.4}$$

Notice that the effect of noise is suppressed by N times due to bundling. This claim can also be made for the FedAvg framework over CNNs. However, even though the noise reduction factor is the same, the impact of the small noise might be amplified by large activations of CNN layers. In FHDnn, we do not have such a problem as the inference and training operations are purely linear.

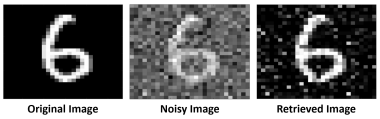


Figure 2.4. Noise robustness of hyperdimensional encodings

One other difference of FHDnn from CNNs is its information dispersal property. HD encoding produces hypervectors which have holographic representations, meaning that the information content is spread over all the dimensions of the high-dimensional space. Since the noise in each dimension can also be assumed to be independent, we can leverage the information spread to further eliminate noise.

Consider the random projection encoding described in Section 2.3.3. Let the encoding matrix $\Phi \in R^{d\times n}$ expressed in terms of its d row vectors, i.e., $\Phi = [\Phi_1, \Phi_2, ..., \Phi_d]^T$. Then, the hypervector formed by encoding information $x \in \mathscr{X}$ can be written as $\mathbf{h} = [\Phi_1^T x, \Phi_2^T x, ..., \Phi_d^T x]^T$, where $x = [x_1, x_2, ..., x_n]^T$. As implied by this expression, the information is dispersed over the hypervectors uniformly. Now consider additive noise over the same hypervector such that $\mathbf{h} + \mathbf{n} = [\Phi_1^T x + n_1, \Phi_2^T x + n_2, ..., \Phi_d^T x + n_d]^T$. We can reconstruct the encoded information from the noisy hypervector $\tilde{\mathbf{h}} = \mathbf{h} + \mathbf{n}$ as follows:

$$x \approx \left[\frac{1}{d}\sum_{i=1}^{d} \Phi_{i,1}\tilde{\mathbf{h}}_i, \frac{1}{d}\sum_{i=1}^{d} \Phi_{i,2}\tilde{\mathbf{h}}_i, ..., \frac{1}{d}\sum_{i=1}^{d} \Phi_{i,n}\tilde{\mathbf{h}}_i\right] \tag{2.5}$$

where $\tilde{\mathbf{h}}_i = \Phi_i^T x + n_i$ are the elements of the noisy hypervector. The noise variance is then reduced by the averaging operation, similar to the case in Equation (3.30). Therefore, in HD computing, the noise is not only suppressed by bundling accross models from different clients, but also by averaging over the dimensions within the same hypervector. We demonstrate this over an example where we encode a sample from the MNIST dataset, add Gaussian noise, then reconstruct it. Figure 3.7 shows the original image, noisy image in the sample space, and reconstructed image for which the noise was added in the hyperdimensional space.

Bit Errors.

We use bit error rate (BER) in conventional coded transmission as a figure of merit for system robustness. It is a measure on how accurately the receiver is able to decode transmitted data. The errors are bit flips in the received digital symbols, and are simply evaluated by the difference (usually Hamming distance) between the input bitstream of channel encoder and the output bitstream of channel decoder. Let $\hat{\mathbf{c}}$ be the binary coded model parameters that are communicated to the server. For the bit error model, we treat the channel as a binary symmetric channel (BSC), which independently flips each bit in $\hat{\mathbf{c}}$ with probability p_e (e.g., $0 \rightarrow 1$). The received bitstream output at the server for client k at round t is then as follows:

$$\tilde{\hat{\mathbf{C}}}_t^k = \hat{\mathbf{C}}_t^k \oplus \mathbf{e}_t^k \tag{2.6}$$

where $\mathbf{e}_t^k$ is the binary error vector and $\oplus$ denotes modulo 2 addition. Given a specific vector $\mathbf{v}$ of Hamming weight $\mathrm{wt}(\mathbf{v})$, the probability that $\mathbf{e}_t^k = \mathbf{v}$ is given by

$$P(\mathbf{e}_t^k = \mathbf{v}) = p_e^{\mathrm{wt}(\mathbf{v})}(1-p_e)^{m-\mathrm{wt}(\mathbf{v})} \tag{2.7}$$

The bit error probability, p_e, is a function of both the modulation scheme and the channel coding technique (assuming lossless source coding). To conclude the transmission, the corrupted bitstream in (3.32) is finally reconstructed to a real-valued model, i.e., $\tilde{\hat{\mathbf{C}}}_t^k \rightarrow \tilde{\mathbf{C}}_t^k$.

Bit errors can have a detrimental effect on the training accuracy, especially for CNNs. At worst case, a single bit error in one client in one round can fail the whole training. We give an example of how much difference a single bit error can make for the standard 32 bit floating point CNN weights. In floating point notation, a number consists of three parts: a sign bit, an exponent, and a fractional value. In *IEEE 754* floating point representation, the sign bit is the most significant bit, bits 31 to 24 hold the exponent value, and the remaining bits contain the fractional value. The exponent bits represent a power of two ranging from -127 to 128. The

fractional bits store a value between 1 and 2, which is multiplied by 2^{exp} to give the decimal value. Our example shows that one bit error in the exponent can change the weight value from 0.15625 to 5.31×10^{37}.

The bit errors are contagious because a parameter from one client gets aggregated to the global model, then communicated back to all clients. Furthermore, errors propagate through all communication rounds because local training or aggregation does not completely change the parameter value, but only apply small decrements. For instance, assume a federated learning scenario with 100 clients and one bit error in a client's model as in the above example. After 10 rounds of training, the CNN weight for the global model will be on the order of $\sim \frac{5.31\times10^{37}}{100^{10}} = 5.31 \times 10^{17}$, still completely failing the whole model. Consider ResNet-50, which has 20 million parameters, so training 100 clients even over a channel with $p_e = 10^{-9}$ BER results in two errors per round on average, making model failure inevitable.

A similar problem exists with HD model parameters, but to a lesser extent because the class prototypes use integer representations. Particularly, errors in the most significant bits (MSB) of integer representation leads to higher accuracy drop. We propose a quantizer solution to prevent this. Inspired by the classical quantization methods in communication systems, we leverage *scaling up* and *scaling down* operations at the transmitter and the receiver respectively. This can be implemented by the automatic gain control (AGC) module in the wireless circuits. For a class hypervector $\mathbf{c}_k$, $k \in \{1,...,K\}$, the quantizer output $Q(\mathbf{c}_k)$ can be obtained via the following steps:

[noitemsep, topsep=0pt, leftmargin=*]**Scale Up:** Each dimension in the class hypervector, i.e. $c_{k,i}$, is amplified with a scaling factor denoted quantization gain G. We adjust the gain such that the dimension with the largest absolute value attains the maximum value attainable by the integer representation. Thus, $G = \frac{2^{B-1}-1}{\max(c_k)}$ where B is the bitwidth. **Rounding:** The scaled up values are truncated to only retain their integer part. **Scale Down:** The receiver output is obtained by scaling down with the same factor G.

This way, bit errors are applied to the scaled up values. Intuitively, we limit the impact of the bit error on the models. Remember, from Section 2.3.4, that prediction is realized by a normalized dot-product between the encoded query and class hypervectors. Therefore, the ratio between the original parameter and the received (corrupted) parameter determines the impact of the error on the dot-product. Without our quantizer, this ratio can be very large whereas after scaling *up* then later *down*, it is diminished.

Packet Loss.

At the physical layer of the network stack, errors are observed in the form of additive noise or bit flips directly on the transmitted data. On the other hand, at the network and transport layers, packet losses are introduced. The combination of network and protocol specifications allows us to describe the error characteristics, with which the data transmission process has to cope.

The form of allowed errors, either bit errors or packet losses, are decided by the error control mechanism. For the previous error model, we assumed that the bit errors are admitted to propagate through the transport hierarchy. This assumption is valid for a family of protocols used in error resilient applications that can cope with such bit errors. In some protocols, the reaction of the system to any number of bit errors is to drop the corrupted packets. These protocols employ a cyclic redundancy check (CRC) or a checksum that allows the detection of bit errors. In such a case, the communication could assume bit-error free, but packet lossy link. We use the packet error rate (PER) metric as a performance measure, whose expectation is denoted packet error probability p_p. For a packet length of N_p bits, this probability can be expressed as:

$$p_p = 1 - (1 - p_e)^{N_p} \tag{2.8}$$

The common solution for dealing with packet losses and guarantee successful delivery is to use a reliable transport layer communication protocol, e.g., transmission control protocol

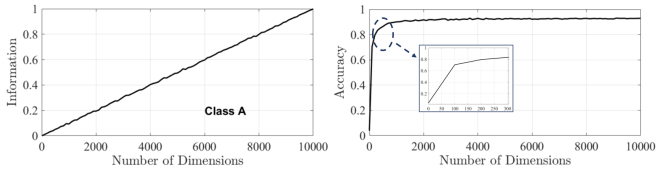


Figure 2.5. Impact of partial information on similarity check (left) and classification accuracy (right)

(TCP), where various mechanisms including acknowledgment messages, retransmissions, and time-outs are employed. To detect and recover from transmission failures, these mechanisms incur considerable communication overhead. Therefore, for our setup we adopt user datagram protocol (UDP), another widely used transport layer protocol. UDP is unreliable and cannot guarantee packet delivery, but is low-latency and has much less overhead compared to TCP.

HD computing's information dispersal and holographic representation properties are also beneficial for packet losses. Another direct result of these concepts is obtaining partial information on data from any part of the encoded information. The intuition is that any portion of holographic coded information represents a blurred image of the entire data. Then, each transmitted symbol –packets in our case– contains an encoded image of the entire model.

We demonstrate the property of obtaining partial information as an example using a speech recognition dataset [1]. In Figure 3.11(a), after training the model, we increasingly remove the dimensions of a certain class hypervector in a random fashion. Then we perform a similarity check to figure out what portion of the original dot-product value is retrieved. The same figure shows that the amount of information retained scales linearly with number of remaining dimensions. Figure 3.11(b) further clarifies our observation. We compare the dot-product values across all classes and find the class hypervector with the highest similarity. Only the relative dot-product values are important for classification. So, it is enough to have the highest dot-product value for the correct class, which holds true with $\sim 90\%$ accuracy even when 80% of the hypervector dimensions are removed.

2.3.6 Convergence Performance of FHDnn

It is commonly preferred to have smooth, strongly-convex, differentiable models to maintain a provable, analytically tractable convergence analysis for federated learning. Such models also provide faster convergence properties than the others. The learning computations in FHDnn are linear, demand low-complexity operations, and thus are favourable for resource-constrained, low-power client devices. However, in many learning tasks, linear federated learning models perform sub-optimally compared to their counterpart, CNN-based approaches. FHDnn diverges from traditional linear methods in this respect. It enjoys both the superior performance properties of non-linear models and low computational complexity of linear models. This is a direct result of HD computing, which embeds data into a high-dimensional space where the geometry is such that simple learning methods are effective. The linearity in HD training benefits convergence, and at the same time the performance does not degrade due to the properties of non-linear hyperdimensional embeddings.

FHDnn, when posed as a distributed optimization solution, has the following properties: *L-smoothness*, *strong convexity*, *bounded variance*, and *uniformly bounded gradient*. It was shown previously that the methods which satisfy these conditions converge to the global optimum solution of the learning task at a rate of $\mathcal{O}(\frac{1}{T})$ [122]. Such claims cannot be made for CNNs due to non-convexity and non-linearity.

2.4 Experimental Analysis

We demonstrate through systematic experiments the performance of FHDnn under various settings. We briefly discuss the datasets and setup for evaluating FHDnn before presenting results for FHDnn for various data distributions for reliable communication. We also compare the resource usage of FHDnn against CNNs. Finally we show evaluation for various unreliable network scenarios.

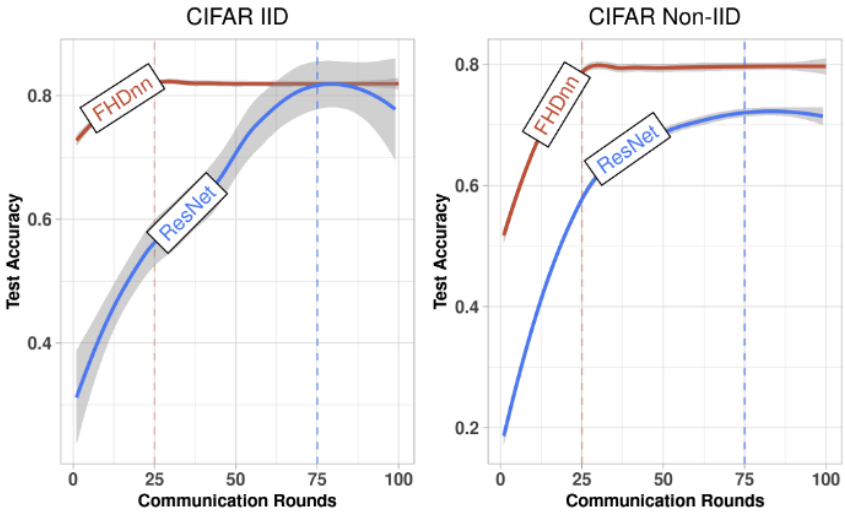


Figure 2.6. Accuracy and Number of communication rounds for various hyperparameters

2.4.1 Dataset and Platforms

We evaluate FHDnn on 3 different real world datasets: MNIST[37], FashionMNIST[210], CIFAR10 [109]. For performance evaluation we test FHDnn on Raspberry Pi Model 3b and NVIDIA Jetson mobile GPU. We use python and PyTorch for implementing all models. For MNIST, we use a simple network consisting of 2 convolution layers and 2 fully connected layers. For CIFAR10 and FashionMNIST we use the ResNet-18 model [65].

2.4.2 Accuracy and Compute

We first tune the hyperparameters for both FHDnn and CNNs and analyze their impact by experimenting with E, B, C where E is the number of local epochs, B the local batch size and C the fraction of clients participating in each round. For all experiments we use 100 clients and 100 rounds of communication in order to keep our experiments tractable. We select the best parameters for ResNet and use the same for FHDnn for all experiments in order to allow for a direct comparison.

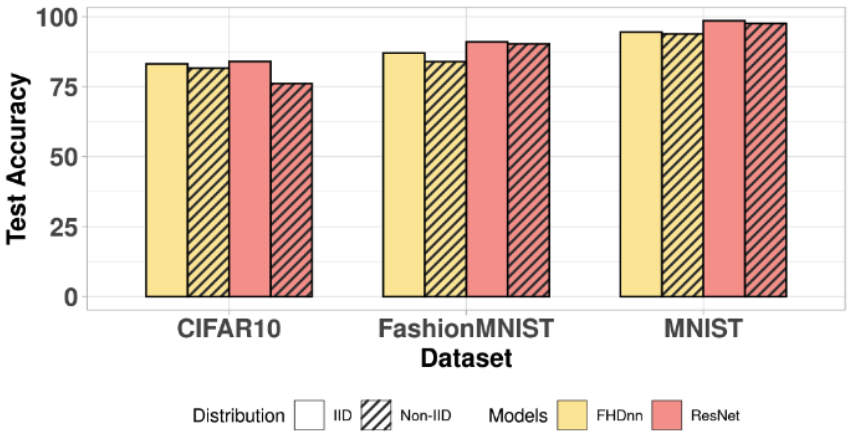


Figure 2.7. Accuracy of FHDnn and ResNet on different datasets

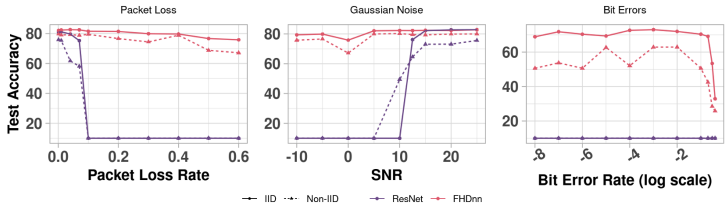


Figure 2.8. Accuracy comparison of FHDnn with ResNet under unreliable network conditions

Accuracy.

Figure 3.12 compares the accuracy of FHDnn with ResNet on 3 different datasets for 100 rounds of communication. We observe that for the same number of rounds, FHDnn achieves almost the same accuracy as ResNet and converges faster. Figure 3.13 shows the smoothed conditional mean across all different hyperparameters for both the models for iid and non-iid distributions. FHDnn reaches an accuracy of 82% in less than 25 rounds of communication whereas ResNet takes 75 rounds for both iid and non-iid data distributions. Moreover the hyperparameters do not have a big influence for FHDnn as seen by the narrow spread (gray region) in Figure 3.13. Note that the local batch size B doesn't impact FHDnn due to the nature of its training methodology. This allows us to use higher batch sizes up to the constraints of the device, allowing for faster processing whereas B affects the convergence of CNNs.

Table 2.1. Performance on Edge Devices

Device	Training Time (Sec)		Energy (J)	
	FHDnn	ResNet	FHDnn	ResNet
Raspberry Pi	**858.72**	1328.04	**4418.4**	6742.8
Nvidia Jetson	**15.96**	90.55	**96.17**	497.572

Compute Resources.

The most computationally expensive operation on the client is training. CNN training involves backpropagation for each round which is very expensive. HD on the other hand is very lightweight as featured in Table 3.4 which quantitatively compares the computation time of FHDnn and ResNet on 2 edge devices for client training. FHDnn is 35% faster and energy efficient than ResNet on RPi and 80% faster and energy efficient on the Jetson mobile GPU.

2.4.3 Unreliable Communication

In this section we analyze the performance of FHDnn and ResNet under unreliable network conditions as described in Section 2.3.5. Figure 3.14 shows the performance of models under each of these network conditions. In order to maintain a direct comparison between CNN and FHDnn we use the same hyperparameters for both models and all experiments. We use $E = 2, C = 0.2, B = 10$ and evaluate the performance on the CIFAR10 dataset. From our experiments we observe that even with fewer clients $C = 0.1$ and for other datasets, the performance of FHDnn is better than ResNet. Due to page constraints we present only the results for the settings mentioned earlier.

Packet Loss.

When the packet loss rate is extremely small, below $1e^{-2}$, ResNet has very minimal accuracy loss. However for more realistic packet loss rates such as 20% the CNN model fails to converge. A 20% packet loss rate implies 20% of the weights are zero. Moreover, this loss is accumulative as the models are averaged during each round of communication thereby giving the CNNs no chance of recovery. In contrast, FHDnn is highly robust to packet loss with almost no

loss in accuracy. For FHDnn, since the data is distributed uniformly across the entire hypervector, a small amount of missing data is tolerable. However, since CNNs have a more structured representation of data with interconnections between neurons, the loss of weights affects the performance of subsequent layers which is detrimental to its performance.

Gaussian Noise.

We experiment with different Signal-to-Noise Ratios (SNR) to simulate noisy links. Even for lower SNRs like 25dB the accuracy of ResNet drops by 8%. However it's more likely that IoT networks operating on low power wireless networks will incur higher SNRs. For such scenarios FHDnn outperforms ResNet as the latter fails to perform better than random. The accuracy of FHDnn only reduces by 3% which is negligible compared to ResNet.

Bit Errors.

Figure 3.14 shows that CNNs achieve the equivalent of random accuracy even for small bit errors. Since the weights of CNNs are floating point, a single bit flip can significantly change the value of the weights. This compounded with federated averaging hinders convergence. We observe FHDnn incurs an accuracy loss as well, achieving 72% for iid and 69% for non-iid. FHDnn uses an integer representation which is again susceptible to large changes from bit errors. However, our scaling method described in Section 3.5.2 assuages the error to some extent.

2.4.4 Communication Efficiency

So far we have benchmarked the accuracy of FHDnn for various network conditions. In this section we demonstrate the communication efficiency of FHDnn compared to ResNet.

We compare the amount of data transmitted for federated learning to reach a target accuracy of 80%. We calculate the amount of data transmitted by one client using the formula $data_{transmitted} = n_{rounds} \times update_{size}$, where n_{rounds} is the number of rounds required for convergence by each model. The update size for ResNet with 11M parameters is 22MB while that of FHDnn is 1MB making it 22$\times$ smaller. From Section 3.6.2 we know that FHDnn converges 3$\times$

faster than ResNet bringing its total communication cost to 25MB. ResNet on the other hand uses up 1.65GB of data to reach the target accuracy.

In Figure 3.13, we illustrated that FHDnn can converge to the optimal accuracy in much fewer communication rounds. However, this improvement is even more in terms of the actual *clock time* of training. We assume that federated learning takes places over LTE networks where SNR is 5dB for the wireless channel. Each client occupies 1 LTE frame of 5MHz bandwidth and duration 10ms in a time division duplexing manner. For error-free communication, the traditional FL system using ResNet can support up to 1.6 Mbits/sec data rate, whereas we admit errors and communicate at a rate of 5.0 Mbits/sec. Under this setting and for the same experiment as in Section 4.2, FHDnn converges in 1.1 hours for CIFAR IID and 3.3 hours for CIFAR Non-IID on average. On the other hand, ResNet converges in 374.3 hours for both CIFAR IID and CIFAR Non-IID on average.

FHDnn demonstrates capabilities such as quick convergence and robustness to unreliable network conditions. However, questions arise regarding the guarantees of convergence across data or applications. In the next chapter, we analyze this hybrid framework from a formal perspective.

2.5 Acknowledgements

I would also like to thank CRISP, PRISM and CoCoSys, centers in JUMP1.0 and 2.0 (SRC programs sponsored by DARPA), SRC Global Research Collaboration grants (GRC TASK 3021.001), and NSF grants 2003279, 1911095, 1826967, 2100237, and 2112167 for supporting my work.

Chapter 2 in full, is a reprint of the material as it appears in FHDnn: Communication Efficient and Robust Learning for AIoT Networks. DAC '22: Proceedings of the 59th ACM/IEEE Design Automation Conference. The dissertation author was the primary investigator and author of this paper.

Chapter 3

Federated Hyperdimensional Computing

3.1 Introduction

Recent years have witnessed an unprecedented growth of data sensing and collection by the Internet of Things (IoT). It is estimated that the number of interconnected IoT devices will reach 40 billion by 2025, generating more than 79 zettabytes (ZB) of data [72]. Empowered by this massive data, emerging Deep Learning (DL) methods enable many applications in a broad range of areas including computer vision, natural languag processing, and speech processing [115].

In the traditional cloud-centric DL approach, data collected by remote clients, e.g. smartphones, is gathered centrally at a computationally powerful server or data center, where the learning model is trained. Often, the clients may not be willing to share data with the server due to privacy concerns. Moreover, communicating massive datasets can result in a substantial burden on the limited network resources between the clients and the server. This motivated the development of distributed algorithms to allow machine learning at edge networks without data sharing. Federated learning (FL), proposed in [140], has recently drawn significant attention as an alternative to centralized learning. FL exploits the increased computational capabilities of modern edge devices to train a model on the clients' side while keeping their collected data local. In FL, each client performs model training based on its local dataset and shares the model with a central server. The models from all participating clients are then aggregated to a global model.

Learning in FL is a long-term process consisting of many progressive rounds of alternating computation and communication. Therefore, two of the main challenges associated with FL are the computation and communication bottlenecks [89]. With FL, the computation, i.e. the training process, is pushed to edge devices. However, state-of-the-art ML algorithms, including deep neural networks (DNN), require a large amount of computing power and memory resources to provide better service quality. The DNN models have complicated model architectures with millions of parameters and require backpropagation, resulting in prohibitively long training times. Besides computation, the communication load of DNN based FL suffers from the need to repeatedly convey massive model parameters between the server and large number of clients over wireless networks [124].

Another challenge arises when FL is carried out over wireless networks. The wireless channels are unreliable in nature, introducing noise, fading, and interference to the transmitted signals. Therefore, the communication in wireless FL is prone to transmission errors. The common solution for this problem is using multiple-access technologies [56] (e.g., TDMA, OFDMA) to prevent interference and error-correcting codes to overcome noise. If there still exists any errors, then a reliable transport layer protocol [111] (e.g.. TCP) is adopted, where acknowledgment, retransmission, and time-out mechanisms are employed to detect and recover from transmission failures. This reliability comes with a price; achieving error-free communication requires a lot of wireless resources, increases energy consumption, limits communication rates, and hence decreases the training speed and convergence of FL. Otherwise, in an unreliable scenario, the transmission errors will impact the quality and correctness of the FL updates, which, in turn, will affect the accuracy of FL, as well as its convergence.

This paper proposes a novel technique that enables efficient, robust, and accurate federated learning using brain-inspired models in high-dimensional space. Instead of conventional machine learning algorithms, we exploit Hyperdimensional Computing (HDC) to perform lightweight learning with simple operations on distributed low-precision vectors, called hypervectors. HDC defines a set of operations to manipulate these hypervectors in the high-dimensional

vector space, enabling a computationally tractable and mathematically rigorous framework for learning tasks. A growing number of works have applied HDC to a wide range of learning problems, including reasoning [94], biosignal processing [7], activity prediction [74], speech/object recognition [75, 78], and prediction from multimodal sensor fusion [171]. These studies have demonstrated the high efficiency, robustness and effectiveness of HDC in solving various learning problems, highlighting its potential as a powerful tool for a variety of applications.

HDC has various appealing characteristics, particularly for edge devices. It is well-suited to address the challenges in FL as:

- HDC is low-power, computationally efficient, and amenable to hardware-level optimization [99],
- it is fault tolerant, providing strong robustness in the presence of errors [146],
- HDC models are small, thus both memory-efficient and communication-efficient [101],
- HDC encoding can transform non-linear learning tasks into linear optimization problems [195], and
- HDC enables fast and light-weight learning with its simple operations [101].

These features make HDC a promising solution for FL using today's IoT edge devices with constrained storage, battery, and resources, over wireless networks with latency concerns and limited bandwidth.

We address several technical challenges to enable *federated hyperdimensional computing* at the IoT edge. Although HDC is inherently suitable for FL, current HDC algorithms fail to provide acceptable accuracy for complex image analysis [79, 226], which is one of the key FL applications. Recently published work [40] combines convolutional neural networks (CNNs) with HDC to learn effectively on complex data. It leverages convolution-based feature extraction prior to the HD encoding step. Unfortunately, such a configuration (DNN+HDC)

possesses the aforementioned computation and communication drawbacks of DNNs for FL. The other challenge is the communication of HDC models over unreliable wireless channels. While the robustness of HDC encoded data to noise and bit errors was demonstrated by prior work [146, 195], similar claims were not investigated for an entire HDC model itself. Finally, HDC models have a lot of redundancy that still can put a burden on communication efficiency, even though they are much smaller than DNN models.

SecureHD [76], HDnn [41] and our work, FedHDC, are all approaches that leverage high-dimensional computing for various tasks. However, there are distinct differences between these methods. FedHDC focuses on federated learning, which enables collaborative model training across multiple decentralized devices while maintaining data privacy. In contrast, SecureHD [76], emphasizes secure high-dimensional computing, specifically designed to handle classification tasks with a focus on security.

HDnn [41], on the other hand, is a hybrid approach that combines high-dimensional computing with convolutional neural networks (CNNs). This method aims to harness the strengths of both HDC and CNNs to improve classification performance, particularly in image recognition tasks. While FHDnn and HDnn both utilize high-dimensional computing, the primary difference lies in their objectives: FedHDC targets federated learning, whereas HDnn [41] focuses on enhancing classification performance by integrating HDC with deep learning techniques. Unlike our proposed FHDnn, HDnn [41] trains the feature extractor to learn representations amenable to learning in the high-dimensional vector space.

In this paper, we first present federated hyperdimensional computing, FedHDC, an extension of the HDC paradigm into the federated learning domain. Next we design a novel synergetic FL framework, called FHDnn, that enables FedHDC to also perform complex image classification by combining contrastive learning framework as a feature extractor with FedHDC, but still keeping model updates to only HDC portion, resulting in fast and accurate model updates via federated learning. In the following, we summarize the main contributions of the paper:

i) We present FedHDC to address the computation and communication problems in

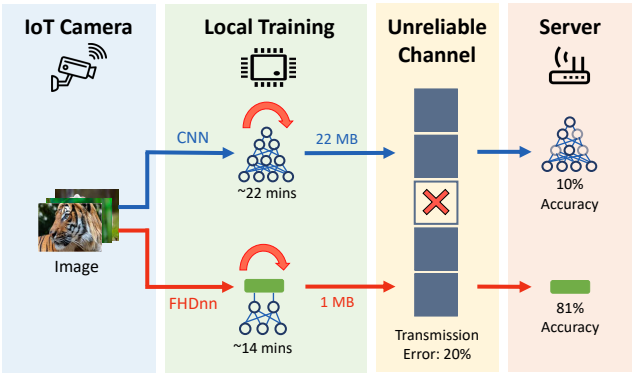


Figure 3.1. FHDnn against CNNs for federated learning

standard DNN based FL approaches. The simple and highly efficient operations of HDC allow for fast and low-weight local training on clients between communication rounds. FedHDC incurs very low communication overhead as HD models are very small in size and training requires many fewer rounds of communication to converge compared to DNNs, resulting in at least 66*times* lower communication overhead as we shown in our results.

ii) We analyze HDC training process using the language of gradient methods from statistical learning and optimization. This viewpoint helps us provide a formal treatment of FedHDC as a general framework for federated learning, and precisely study its convergence properties. FedHDC can achieve $\mathcal{O}(\frac{1}{T})$ convergence rate, with T representing the number of communication rounds, but such claim is not possible for non-convex and non-linear DNNs. As HD encoding embeds data into a high-dimensional space and can transform non-linear distributed learning tasks into linear optimization, FedHDC enjoys simpler training and faster convergence compared to DNNs as it uses only HD computing, while having the superior performance properties of non-linear models.

iii) We present FHDnn, a novel synergetic FL framework that combines pre-trained CNN as a feature extractor with HDC. Specifically, we utilize a CNN trained using SimCLR [27], a contrastive learning framework which learns informative representations of data self-supervised. FHDnn avoids the transmission of the CNN and instead trains only the HD learner in a federated manner. This strategy accelerates learning, reduces transmission costs, and utilizes the robustness

of HDC to tackle network errors as shown in Fig. 3.1.

iv) HD-based federated learning provides reliability for learning over unreliable wireless network with no additional cost. Unlike existing FL approaches, there is no need for multiple-access technologies to prevent interference or error-protection on the transmitted models. Due to such techniques, FL can have very limited communication rates, and hence low training speeds. We leverage the robustness of HDC and allow errors during transmission instead of limiting the rate to achieve error-free communication. We analyze FHDnn under three different unreliable network settings: packet loss, noise injection, and bit errors, and show that the perturbations in the client models can be tolerated by the HDC learner. A quantizer method with scaling is additionally proposed to enhance the resilience to bit errors.

v) We also propose various strategies to further improve the communication efficiency of FedHDC and FHDnn. The HDC models have redundancy which we exploit to reduce their sizes for more efficient communication. We examine three approaches: binarized differential transmission, subsampling, and sparsification & compression. We show their trade-offs between performance and efficiency through experiments.

We evaluate HDC-based federated learning by numerical experiments on different benchmark datasets and compare their performance with CNN based FL under various settings. We both theoretically and empirically show that the proposed approaches are robust to lossy network conditions. Based on our evaluations, FHDnn converges 3x faster than CNN, reduces the communication costs by 66$\times$ and the local computation cost on the clients by up to 6$\times$. The communication efficiency of FedHDC and FHDnn is further improved by various strategies up to 32$\times$ with minimal loss in accuracy.

3.2 Related Work

Communication and computation bottlenecks of FL have been widely studied in the literature and various solutions were proposed targeting improvement at different parts of

the overall process. FL involves many rounds of communication with the participation of numerous clients, typically at low rates over wireless links. These considerations have led to a significant interest in communication-efficient design of FL systems. Previous research has primarily focused on decreasing the size of the model updates [106] and reducing the number of communication rounds or communicating clients [176]. In addition, during each round of communication, participating clients train models locally on device for multiple epochs. Deep learning models that are commonly used tend to be expensive to train requiring backpropogation algorithm which is compute heavy. Efficient computation is also of great importance as clients are usually not equipped with powerful hardware. This is addressed in prior work by reducing the model complexity to alleviate local training [85]. On the other hand, there is often a trade-off between communication and computation; one strategy for lowering the frequency of communication is to put more emphasis on computation. The lightweight nature of HDC models make them suitable for running on edge devices with constrained resources.

3.2.1 Communication Efficiency

A prototypical FL approach named FedAvg [140] enables flexible communication and computation trade-off. The work follows from the seminal research in distributed stochastic gradient descent (SGD). Improvement in communication-efficiency is achieved by allowing for the clients to run multiple local SGD steps per communication round. Many succeeding studies have pursued the theoretical understanding of FedAvg in terms of communication-computation trade-offs and have carried out rigorous analysis of the convergence behavior depending on the underlying assumptions (e.g., IID or non-IID local datasets, convex or non-convex loss functions, gradient descent or stochastic gradient descent) [122, 96]. Another approach that directly affects local training is to modify model complexity. Some examples are pruning [225], restricting the model weights to be numbers at a certain bitwidth [34], and bounding the model size[153]. These methods also lower computation complexity along with communication overhead.

As the models for FL can get very large—especially in the case of DNNs—a different line

of work explored methods to reduce the communicated model (or gradient) size, without altering the original local models. Existing schemes typically perform a form of compression, that is, instead of transmitting the raw model/gradient data, one transmits a compressed representation with fewer bits, for instance by means of limiting bitwidth (quantization) or enforcing sparsity (sparsification). Particularly, a popular class of quantization operators is based on random dithering [68]. Sparsification methods decrease the number of non-zero entries in the communicated data to obtain sparse vectors [208]. Structured and sketched updates are also proposed in [105], which can be further supported by lossy compression and federated dropout [21]. Some other approaches include randomized techniques such as stochastic rounding [189], subsampling [106], and randomized approximation [107].

In FL, a group of clients might often provide similar, and hence redundant, model information during communication rounds. Orthogonal to the compression-based approaches, one can dismiss the updates of some clients as communicating all model updates would be an inefficient use of resources. Early works have attempted simple client selection heuristics such as selecting clients with higher losses [11], sampling clients of larger update norm with higher probability [29], and sampling clients with probabilities proportional to their local dataset size [140], but the similarity or redundancy of the client updates are not exploited in these methods. Ideally, a diverse and representative set of clients should be selected that contribute different, informative updates. In consideration of this, several selection criteria have been investigated in recent literature, some of which are diversity-based selection [12], importance sampling [29], and selection by update significance [26].

FL is often carried out over wireless channels that attenuate the transmitted signal as well as introduce noise, and thus the communication is unreliable, prone to transmission errors. All the aforementioned approaches assume reliable links and ignore the wireless nature of the communication medium. The inherent assumption is that independent error-free communication "tunnels" has been established between the clients and the server by some existing wireless protocol. A common way to achieve this is to divide the channel resources among clients with

multiple-access technologies (e.g., TDMA, CDMA, OFDMA) to mitigate interference, and utilize powerful error correcting codes to overcome noise [62]. However, the communication rates and consequently the overall training speed suffer due to the limited channel resources that can be allocated per client.

3.2.2 Computation Efficiency

The clients in FL are typically resource-constrained, battery-operated edge devices with limited power and computation budgets, unlike the powerful servers used in cloud-centric learning. DNN-based FL methods require clients to perform on-device backpropagation during each round of training which is computationally expensive and is incurring high resource usage. To overcome this challenge, prior works mainly explored low complexity NN architectures and lightweight algorithms suitable for edge devices. A lot of the 'local methods' for improving communication efficiency fall into this category, e.g, pruning [85] and using quantized models [176], which are also helpful for reducing computation.

A small subset of the proposed approaches specifically devote their attention to resolving the computational issues in FL. In [212], a "soft-training" method was introduced to dynamically compress the original training model into a smaller restricted volume through rotating parameter training. In each round, it lets different parts of model parameters alternately join the training, but maintains the complete model for federated aggregation. The authors of [205] suggested dividing the model into sub-models, then using only a few sub-models for partial federated training while keeping the rest of the parameters fixed. During training, sub-model capacities are gradually increased until it reaches the full model. Along similar lines, federated dropout [21] is a technique that enables each client to locally operate on a smaller sub-model while still providing updates that can be applied to the larger global model on the server. Finally, the technique presented in [193], called splitfed learning, combines the strengths of FL and split learning by splitting a NN into client-side and server-side sub-networks during federated training.

Our federated hyperdimensional computing approach is orthogonal to the most of the

existing communication-efficient FL methods. For instance, it can be used in tandem with compression, subsampling, and client selection or techniques that reduce model complexity. In fact, in Section 3.5.4, we include some strategies for further improving the communication cost leveraging the statistical properties of hypervectors, even though HD models are much smaller (around hundred thousand parameters vs millions/billions), thus more communication-efficient, compared to DNNs. Furthermore, different from the aforementioned works, we account for unreliable communication scenarios. We use the robustness of HDC to tolerate communication errors and carry out accurate training. Finally, there are studies that aim at making the compute intensive DNN-based FL methods more efficient as summarized above. In contrast, HDC itself is a very light-weight framework with low computational cost. It was shown in previous work that HDC provides $3\times$ reduction in training time, $1.8\times$ in energy comsumption compared to optimized DNNs on NVIDIA Jetson TX2 low-power edge GPU [99]. An ASIC implementation of HDC for edge devices further improves the energy consumption by $1257\times$ and training time by $11\times$ over DNNs.

3.3 Hyperdimensional Computing

In the following, we analyze hyperdimensional computing classification algorithm, then express it in a standard mathematical framework from statistical learning and optimization. The goal of this section is to provide an in-depth formal treatment of HDC as a general 'learning' method. Leveraging the analysis presented here, we later study the convergence properties of federated hyperdimensional computing in Section 3.4.1,

3.3.1 Hyperdimensional Learning

Many learning tasks can be implemented in the HD domain. Here, we focus on classification, one of the most popular supervised learning problems. Suppose we are given collection of labeled examples $\mathscr{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$ where $\mathbf{x}_i \in \mathscr{X} \subset R^m$ and $y_i \in \mathscr{C}$ is a categorical variable indicating the class label of a particular data sample $\mathbf{x}_i$. For HD learning, we first encode the en-

tire set of data samples in $\mathscr{D}$ into hyperdimensional vectors such that $\mathbf{h}_i = \phi(\mathbf{x}_i)$ is a hypervector in the d-dimensional inner-product space $\mathscr{H}$. These high-dimensional embeddings represent data in a way that admits linear learning algorithms, even if the data was not separable to begin with. In other words, simple linear methods applied on HD encoded data can capture nonlinear decision boundaries on the original data [195].

The common approach to learning with HD representations is to *bundle* together the training examples corresponding to each class into a set of "prototypes", which are then used for classification. The bundling operator is used to compile a set of elements in $\mathscr{H}$ and assumes the form of a function $\oplus : \mathscr{H} \times \mathscr{H} \to \mathscr{H}$. The function takes two points in $\mathscr{H}$ and returns a third point similar to both operands. We bundle all the encoded hypervectors that belong to the k-th class to construct the corresponding prototype $\mathbf{c}_k$:

$$\mathbf{c}_k = \bigoplus_{i \text{ s.t. } y_i = k} \mathbf{h}_i \tag{3.1}$$

Given a query data $\mathbf{x}_q \in \mathscr{X}$ for which we search for the correct label to classify, we take the encoded hypervector $\mathbf{h}_q \in \mathscr{H}$ and return the label of the most similar prototype:

$$\hat{y}_q = k^* = \underset{k \in 1,\ldots,K}{\operatorname{argmax}} \; \delta(\mathbf{h}_q, \mathbf{c}_k) \tag{3.2}$$

where δ is a similarity metric.

One-Shot Training. The bundling operator $\oplus$ is often chosen to be element-wise sum. In this case, the class prototypes are obtained by adding all hypervectors with the same class label. Then, the operation in Equation (3.1) is simply calculated as:

$$\mathbf{c}_k = \sum_{i \text{ s.t. } y_i = k} \mathbf{h}_i \tag{3.3}$$

This can be regarded as a single pass training since the entire dataset is only used once—with no

iterations—to train the model (class prototypes).

Inference. The similarity metric δ is typically taken to be the cosine similarity which is a measure of angle between two vectors from an inner product space. Equation (3.2) is rewritten using a dot-product and a magnitude operation as follows under cosine similarity:

$$\hat{y}_q = k^* = \underset{k \in 1,\ldots,K}{\operatorname{argmax}} \frac{\langle \mathbf{c}_k, \mathbf{h}_q \rangle}{\|\mathbf{c}_k\|} \tag{3.4}$$

Retraining. One-shot training often does not result in sufficient accuracy for complex tasks. A common approach is to fine-tune the class prototypes using a few iterations of retraining [168, 78, 92, 99]. We use the perceptron algorithm [**?**] to update the class hypervectors for mistpredicted samples. The model is updated only if the query in (3.4) returns an incorrect label. Let $y_q = k$ and $\hat{y}_q = k'$ be the correct and mispredicted labels respectively. Then, the new class prototypes after the retraining iteration are:

$$\begin{aligned} \mathbf{c}_k &= \mathbf{c}_k + \alpha \mathbf{h}_q \\ \mathbf{c}_{k'} &= \mathbf{c}_{k'} - \alpha \mathbf{h}_q \end{aligned} \tag{3.5}$$

where α is the HD learning rate, controlling the amount of change we make to the model during each iteration. Figure 1.1b shows an overview of HDC for classification.

3.3.2 Hyperdimensional Linear Discriminant

The single pass training and dot-product based inference approach of the HD algorithm bears a strong resemblance to Fisher's linear discriminant [44]. Assume that each sample $\mathbf{x} \in \mathscr{X}$ belongs to a class with binary label $y \in \{-1, 1\}$ for notational convenience. The assumption of a binary classication task is primarily for clarity of exposition, and our results can be extended to support multi-class problems via techniques such as "one-versus-rest" decision rules. Fisher's linear discriminant on HD space finds the line $z = \mathbf{w}^T \mathbf{h}$ that best separates the two classes. The

goal is to select direction $\mathbf{w}$ so that after projecting along this direction,

1. the separation between classes are high with their means as far away as possible from each other, and

2. the scatter within the classes is as small as possible with low variance.

A criterion that quantifies the desired goal is the *Rayleigh quotient*:

$$\begin{aligned} J(\mathbf{w}) &= \frac{\mathbf{w}^T \mathbf{S}_B \mathbf{w}}{\mathbf{w}^T \mathbf{S}_W \mathbf{w}} \\ \mathbf{S}_B &= (\mu_1 - \mu_{-1})(\mu_1 - \mu_{-1})^T \\ \mathbf{S}_W &= \Sigma_1 + \Sigma_{-1} \end{aligned} \tag{3.6}$$

where $\mu_{\pm 1}$ and $\Sigma_{\pm 1}$ are the mean vector and the covariance matrix respectively. $\mathbf{S}_B$ is defined as the between-class scatter which measures the separation between class means, while $\mathbf{S}_W$ is the within-class scatter, measuring the variability inside the classes. Our goal is achieved by maximizing the Rayleigh quotient with respect to $\mathbf{w}$. The corresponding optimal projection direction is then given as

$$\mathbf{w}^* = (\Sigma_1 + \Sigma_{-1})^{-1}(\mu_1 - \mu_{-1}) \tag{3.7}$$

One can use Fisher's linear discriminant method as a classifier where the decision criterion is a threshold on the dot-product (projection):

$$z = (\mu_1 - \mu_{-1})^T (\Sigma_1 + \Sigma_{-1})^{-1} \mathbf{h}_q + T \begin{cases} > 0, \, \hat{y}_q = \ \ 1 \\ < 0, \, \hat{y}_q = -1 \end{cases} \tag{3.8}$$

In HD computing, the procedure of one-shot training followed by inference, described by (3.3) and (3.4), is equivalent to above decision criterion. For two classes, the "similarity check" step

in (3.4) can be rewritten in the form of a decision function as follows:

$$\hat{y}_q = \begin{cases} 1, \text{ if } \frac{\langle \mathbf{c}_1, \mathbf{h}_q \rangle}{\|\mathbf{c}_1\|} > \frac{\langle \mathbf{c}_{-1}, \mathbf{h}_q \rangle}{\|\mathbf{c}_{-1}\|} \\ -1, \text{ if } \frac{\langle \mathbf{c}_1, \mathbf{h}_q \rangle}{\|\mathbf{c}_1\|} < \frac{\langle \mathbf{c}_{-1}, \mathbf{h}_q \rangle}{\|\mathbf{c}_{-1}\|} \end{cases} \tag{3.9}$$

which can be further simplified as:

$$\hat{y}_q = \begin{cases} 1, \text{ if } (\frac{\mathbf{c}_1}{\|\mathbf{c}_1\|} - \frac{\mathbf{c}_{-1}}{\|\mathbf{c}_{-1}\|})^T \mathbf{h}_q > 0 \\ -1, \text{ if } (\frac{\mathbf{c}_1}{\|\mathbf{c}_1\|} - \frac{\mathbf{c}_{-1}}{\|\mathbf{c}_{-1}\|})^T \mathbf{h}_q < 0 \end{cases} \tag{3.10}$$

Since the class prototypes are normalized sums of hypervectors with the same labels, they relate to the respective class means by a scalar multiplication, i.e, $\mathbf{c}_{\pm 1} = \frac{\|\mathbf{c}_{\pm 1}\|}{N_{\pm 1}} \mu_{\pm 1}$. Here, $N_{\pm 1}$ denotes the total number of samples in classes. We obtain the below decision rule after plugging in $\mu_{\pm 1}$ into (3.10), then dividing both sides of the inequalities by $\frac{\|\mathbf{c}_{\pm 1}\|}{N_{\pm 1}}$.

$$\hat{y}_q = \begin{cases} 1, \text{ if } (\mu_1 - \mu_{-1})^T \mathbf{h}_q > 0 \\ -1, \text{ if } (\mu_1 - \mu_{-1})^T \mathbf{h}_q < 0 \end{cases} \tag{3.11}$$

Note that this is the same classifier as in (3.8) for the special case when $\Sigma_1 = \Sigma_{-1} = \Sigma = \frac{1}{2}\mathbf{I}$. HD encoding maps data points to a hyperdimensional space such that different dimensions of the hypervectors are uncorrelated, i.e, $\Sigma_{ij} \approx 0,\ i \neq j$. Therefore, one-shot training followed by inference in HD computing is equivalent to applying Fisher's linear discriminant and classifying sample encoded hypervectors. The above result shows the HD algorithm explicitly optimizes the discrimination between the data points from different classes. We first project data via HD encoding such that it becomes linearly separable, then find a linear discriminant.

3.3.3 A Gradient Descent Perspective on HDC

A retraining step is required to fine-tune the HD model for tasks where one-shot training does not suffice. The goal is to update the class prototypes until finding the model that best

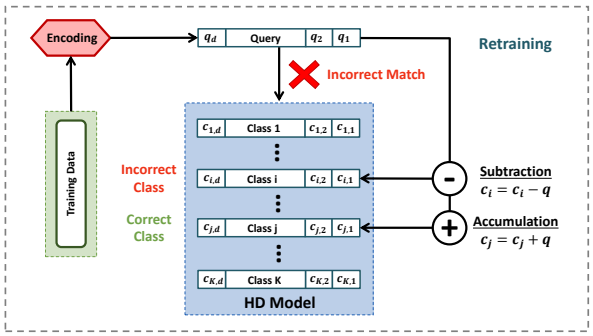


Figure 3.2. HDC retraining

separates classes. In the following, we analyze HD retraining process using the language of gradient methods from statistical learning and optimization. This viewpoint helps us provide a formal treatment of FedHDC as a general framework for federated learning, and precisely study its convergence properties.

Without loss of generality, we continue our analysis using binary class labels as in Section 3.3.2. Let $\mathbf{w} \in R^d$ be a vector of weights that specifies a hyperplane in the hyperdimensional space with d dimensions. We define this vector in terms of class prototypes, such that $\mathbf{w} = \mathbf{c}_1 - \mathbf{c}_{-1}$. Then, after inputing in the weight vector and simplifying the equations in (3.10), classification of a query data $\mathbf{x}_q$ is made through the following decision function:

$$\hat{y}_q = \begin{cases} 1, \text{ if } \mathbf{w}^T \mathbf{h}_q > 0 \\ -1, \text{ if } \mathbf{w}^T \mathbf{h}_q < 0 \end{cases} \tag{3.12}$$

This can be interpreted as a *linear separator* on the HD representations of the data. It divides $\mathscr{H}$ into two half-planes, where the boundary is the plane with normal $\mathbf{w}$. The goal is to learn the weights such that all the positive examples ($y_i = 1$) are on one side of the hyperplane and all negative examples ($y_i = -1$) on the other. For the optimal set of weights, the linear function $g(\mathbf{h}) = \mathbf{w}^T \mathbf{h}$ agrees in the sign with the labels on all training instances, that is, $\text{sign}(\langle \mathbf{w}, \mathbf{h}_i \rangle) = y_i$ for any $\mathbf{x}_i \in \mathscr{X}$. We can also express this condition as $y_i \langle \mathbf{w}, \mathbf{h}_i \rangle > 0$.

Recall that HD retraining, in the event of a misclassification, subtracts the query hyper-

vector from the incorrect class prototype and adds it to the one that it should have been matched with. The two possible retraining iterations are illustrated below for binary classification.

Misclassifying $\mathbf{x}_1$	**Misclassifying $\mathbf{x}_{-1}$**	
$\mathbf{c}_1 = \mathbf{c}_1 + \alpha\mathbf{h}_1$	$\mathbf{c}_1 = \mathbf{c}_1 - \alpha\mathbf{h}_{-1}$	
$\mathbf{c}_{-1} = \mathbf{c}_{-1} - \alpha\mathbf{h}_1$	$\mathbf{c}_{-1} = \mathbf{c}_{-1} + \alpha\mathbf{h}_{-1}$	(3.13)

For both cases, the difference of class prototypes, i.e. $\mathbf{c}_1 - \mathbf{c}_{-1}$, is updated as a function of the misclassified class label. A unified update equation that covers both cases is as follows:

$$\mathbf{c}_1 - \mathbf{c}_{-1} = \mathbf{c}_1 - \mathbf{c}_{-1} + 2\alpha y_i \mathbf{h}_i \tag{3.14}$$

Inputing in the weight vector in the above equation, we have:

$$\mathbf{w} = \mathbf{w} + 2\alpha y_i \mathbf{h}_i \tag{3.15}$$

$\mathscr{D}\ P_{ed}^{ref}$ set $t = 0, \mathbf{w}_t = 0$ convergencerandom index $i \in \{1, ..., n\}$ $y_i \langle \mathbf{w}_t, \mathbf{h}_i \rangle < 0$ $\mathbf{w}_{t+1} = \mathbf{w}_t + \eta y_i \mathbf{h}_i$ t = t + 1

A simple algorithm that implements HD retraining with the above notion of linear separators is described by Algorithm 1. Here, η is a positive scalar called the learning rate and t denotes iteration number. We now show that HD retraining can be represented as an instance of Empirical Risk Minimization (ERM). Particularly, we frame the retraining step as an optimization problem with convex loss function, then we argue that the updates in Algorithm 1 are equivalent to stochastic gradient descent (SGD) steps over an empirical risk objective.

Our ultimate goal is to find the discriminant function $g_{\mathbf{w}}(\mathbf{h})$ which minimizes the empirical risk on the embedded training set $\mathscr{D}_{\mathscr{H}} = \{(\mathbf{h}_1, y_1), ..., (\mathbf{h}_n, y_n)\}$. Empirical risk is defined as

follows:

$$R_{emp}(g_{\mathbf{w}}) = \frac{1}{n}\sum_{i=1}^{n} \ell(g_{\mathbf{w}}(\mathbf{h}_i), y_i) \tag{3.16}$$

where $\ell : \mathscr{H} \times \mathscr{H} \rightarrow R$ is a *loss function* that describes the real-valued penalty calculated as a measure of the discrepancy between the predicted and true class labels. Zero empirical risk can be achieved if HD encoding admits a linearly separable representation. Otherwise, zero risk is not possible, but we search for the optimal weights that minimizes it:

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}} R_{emp}(g_{\mathbf{w}}) \tag{3.17}$$

The "no error" condition, $y_i\langle \mathbf{w}, \mathbf{h}_i \rangle > 0 \; \forall i$, provides a very concise expression for the situation of zero empirical risk. It allows for the formulation of the learning problem as the following function optimization:

$$\text{minimize } \; J(\mathbf{w}) = -\sum_{i=1}^{n} y_i \mathbf{w}^T \mathbf{h}_i \tag{3.18}$$

The solution can be found by doing gradient descent on our cost function $J(\mathbf{w})$ where the gradient is computed as $\nabla J(\mathbf{w}) = -\sum_{i=1}^{n} y_i \mathbf{h}_i$. Another optimization method is the stochastic gradient descent that picks a random example at each step and makes an improvement to the model parameters. Then, the gradient associated with an individual example is $-y_i \mathbf{h}_i$. Given a loss function $\ell(\cdot)$, the stochastic gradient descent algorithm is defined below:

Stochastic Gradient Descent:

Given: starting point $\mathbf{w} = \mathbf{w}_{init}$, learning rates $\eta_1, \eta_2, \eta_3, ...$

(e.g. $\mathbf{w}_{init} = 0$ and $\eta_t = \eta$ for all t, or $\eta_t = 1/\sqrt{t}$).

For a sequence of random examples $(\mathbf{h}_1, y_1), (\mathbf{h}_2, y_2), ...$

1. Given example $(\mathbf{h}_t, y_t)$, compute the gradient $\nabla \ell(g_{\mathbf{w}}(\mathbf{h}_t), y_t)$ of the loss w.r.t. the weights $\mathbf{w}$.

2. Update: $\mathbf{w} \leftarrow \mathbf{w} - \eta_t \nabla \ell(g_{\mathbf{w}}(\mathbf{h}_t), y_t)$

To present an equivalent formulation to (3.18), consider the loss function $\ell(g_{\mathbf{w}}(\mathbf{h}), y) = \max(0, y\langle \mathbf{w}, \mathbf{h} \rangle)$ for the empirical risk in (3.16). If $g_{\mathbf{w}}(\mathbf{h})$ has the correct sign, then we have a loss of 0, otherwise we have a loss equal to the magnitude of $g_{\mathbf{w}}(\mathbf{h})$. In this case, if $g_{\mathbf{w}}(\mathbf{h})$ has the correct sign and is non-zero, then the gradient will be zero since an infinitesimal change in any of the weights will not change the sign. So, the algorithm will not make any change on $\mathbf{w}$. On the other hand, if $g_{\mathbf{w}}(\mathbf{h})$ has the wrong sign, then $\frac{\partial \ell}{\partial \mathbf{w}} = -y\mathbf{h}$. Hence, using $\eta_t = \eta$, the algorithm will update $\mathbf{w} \leftarrow \mathbf{w} + \eta y \mathbf{h}$. Note that this is exactly the same algorithm as HD retraining. We observe that empirical risk minimization by SGD with the above loss function gives us the update rule in Algorithm 1.

3.4 FedHDC: Federated HD Computing

We study the federated learning task where an HD model is trained collaboratively by a loose federation of participating *clients*, coordinated by a central *server*. The general problem setting discussed in this paper mostly follows the standard federated averaging framework from the seminal work in [140]. In particular, we consider one central server and a fixed set of N clients, each holding a local dataset. The k-th client, $k \in [N]$, stores embedded dataset $\mathscr{D}_k = \{(\mathbf{h}_{k,j}, y_{k,j})\}_{j=1}^{n_k}$, with $n_k = |\mathscr{D}_k|$ denoting the number of feature-label tuples in the respective datasets.

The goal in FL is to learn a global model by leveraging the local data at the clients. The raw datasets cannot be shared with the central server due to privacy concerns, hence the training process is apportioned among the individual clients as described by the following distributed optimization problem:

$$\min_{\mathbf{w}} \left\{ F(\mathbf{w}) \sum_{k=1}^{N} p_k F_k(\mathbf{w}) \right\} \tag{3.19}$$

where p_k is the weight of the k-th client such that $p_k \geq 0$ and $\sum_{k=1}^{N} p_k = 1$. A natural and common approach is to pick $p_k = \frac{n_k}{n}$. Similar to Section 3.3.3, we represent our HD model by a vector of

parameters $\mathbf{w} \in \mathscr{H} \subseteq R^d$. If the partition $\mathscr{D}_k$ is formed by randomly and uniformly distributing the training examples over the clients, then we have $E_{\mathscr{D}_k}[F_k(\mathbf{w})] = F(\mathbf{w})$, where the expectation is over the set of examples assigned to the client. This is the IID assumption that usually does not hold in FL setting; F_k could be an arbitrarily bad approximation to F under non-IID data.

To define the learning objective and measure the fit of the model to data, we introduce a loss function as in (3.16). We denote $\ell\big(\mathbf{w}; (\mathbf{h}_{k,j}, y_{k,j})\big)$ for the loss of the prediction on example $(\mathbf{h}_{k,j}, y_{k,j})$ made with an HD model parametrized by $\mathbf{w}$. For the k-th client, the local objective $F_k(\cdot)$ is defined in the form of local empirical loss as follows:

$$F_k(\mathbf{w}) = \frac{1}{n_k} \sum_{j=1}^{n_k} \ell\big(\mathbf{w}; (\mathbf{h}_{k,j}, y_{k,j})\big) \tag{3.20}$$

For ease of notation, we do not explicitly use $g_{\mathbf{w}}(\mathbf{h})$ to denote the learning model, instead substitute $\mathbf{w}$ which parametrizes it. The local empirical loss F_k measures how well the client model fits the local data, whereas the global loss F quantifies the fit to the entire dataset on average. We have shown above that the loss function $\ell = \max(0, y\langle \mathbf{w}, \mathbf{h} \rangle)$ captures the behavior of the HD algorithm for an equivalent optimization problem formulation solved by SGD. The objective is to find the model $\mathbf{w}^*$ that minimizes the global loss, i.e., $\mathbf{w}^* = \operatorname{argmin}_{\mathbf{w}} F(\mathbf{w})$.

Algorithm. In the *federated bundling* framework, each client maintains its own HD model and participates in building a global model that solves (3.19) in a distributed fashion. This is achieved via an iterative training procedure for which we describe one round (say t-th) of the algorithm below.

1. **Broadcast:** The central server broadcasts the latest global HD model, $\mathbf{w}_t$, to all clients.

2. **Local updates:** Each client $k \in [N]$ sets its model $\mathbf{w}_t^k = \mathbf{w}_t$ and then performs training for

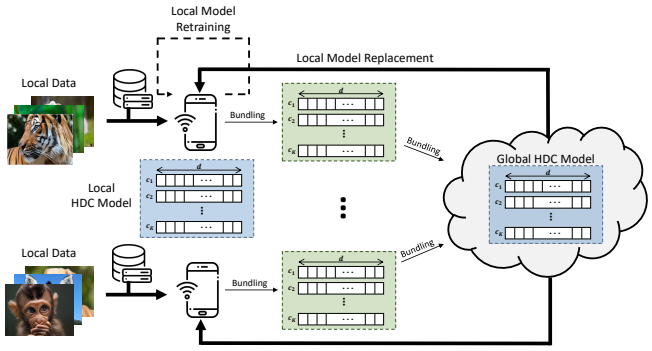


Figure 3.3. FedHDC Training

E epochs using local data:

$$\begin{aligned}
&\mathbf{w}_{t,0}^k = \mathbf{w}_t^k, \\
&\mathbf{w}_{t,\tau+1}^k \longleftarrow \mathbf{w}_{t,\tau}^k - \eta_t \nabla F_k(\mathbf{w}_{t,\tau}^k, \xi_\tau^k),\ i = 0, 1, ..., E-1, \\
&\mathbf{w}_{t+1}^k = \mathbf{w}_{t,E}^k,
\end{aligned} \tag{3.21}$$

where η_t is the learning rate and ξ_τ^k is a mini batch of data examples sampled uniformly from local dataset $\mathscr{D}_k$.

3. **Aggregation:** The central server receives and aggregates the local models to produce a new global model:

$$\mathbf{w}_{t+1} = \sum_{k=1}^{N} p_k \mathbf{w}_{t+1}^k. \tag{3.22}$$

After aggregation, the server moves on to the next round, $t+1$. This procedure is carried out until sufficient convergence is achieved. Fig. 3.3 summarizes the federated training process for FedHDC. The overall update in one round of federated bundling is similar to a gradient descent step over the empirical loss corresponding to the entire distributed dataset across clients.

3.4.1 FedHDC Convergence Analysis

In this section, we first specify the objective functions and the corresponding gradient computations in FedHDC, for whose general forms were discussed above. We then analyze the convergence behavior of FedHDC, showing that it converges to the global optimum at a rate of $\mathcal{O}(\frac{1}{T})$, where T is the number of communication rounds.

For federated learning with HD algorithm, the optimization problem in (3.19) is cast as follows:

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{k=1}^{N} \frac{p_k}{n_k} \sum_{j=1}^{n_k} \max(0, y_j \langle \mathbf{w}, \mathbf{h}_j \rangle), \tag{3.23}$$

and the local gradient $\mathbf{g}_k = \nabla F_k(\mathbf{w})$ is computed at client $k \in [N]$ as:

$$\mathbf{g}_k = \frac{1}{n_k} \sum_{j=1}^{n_k} y_j \mathbf{h}_j \tag{3.24}$$

As Equation (3.24) suggests, the gradient computations are linear, demand low complexity operations, and thus are favourable for resource-constrained, low-power client devices. However, in many learning tasks, linear federated learning models perform sub-optimally compared to their counterpart, DNN-based approaches. FedHDC diverges from traditional linear methods in this respect. It enjoys both the superior performance properties of non-linear models and low computational complexity of linear models. This is a direct result of HD computing, who embeds data into a high-dimensional space where the geometry is such that simple learning methods are effective. As we show in the following, linearity in HD training benefits convergence, at the same time the performance does not degrade due to the properties of non-linear hyperdimensional embeddings. Such convergence claims are not possible for non-convex and non-linear DNNs.

The functions $F_k(\cdot)$ and the gradients $\nabla F_k(\cdot)$ have the following properties:

1. (**L-smoothness**). Each local function $F_k(\cdot)$ is L-smooth where the gradients $\nabla F_k(\cdot)$ are

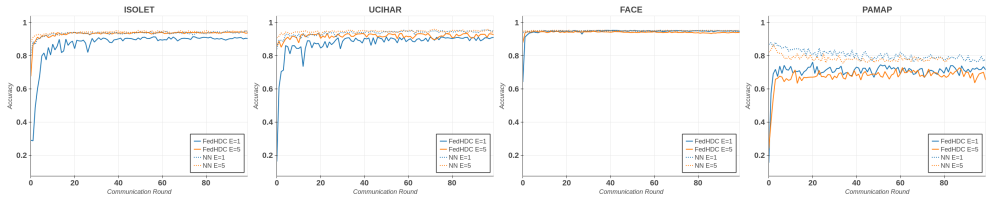


Figure 3.4. Accuracy and convergence of FedHDC and NN for various epochs (E)

Lipschitz continuous: *There exists a parameter* $L > 0$ *such that for all* $\mathbf{v},\mathbf{w} \in R^d$,

$$\|\nabla F_k(\mathbf{v}) - \nabla F_k(\mathbf{w})\| \leq L\|\mathbf{v} - \mathbf{w}\|.$$

2. (**Strong convexity**). Each local function $F_k(\cdot)$ is μ-strongly convex and differentiable: *For all* $\mathbf{v},\mathbf{w} \in R^d$,

$$F_k(\mathbf{v}) \geq F_k(\mathbf{w}) + (\mathbf{v} - \mathbf{w})^T \nabla F_k(\mathbf{w}) + \frac{\mu}{2}\|\mathbf{v} - \mathbf{w}\|^2.$$

3. (**Bounded variance**). The variance of stochastic gradients for each client k is bounded: *Let* ξ^k *be sampled from the k-th client's dataset uniformly at random, then there exists constants* σ_k *such that for all* $\mathbf{w} \in R^d$,

$$E\|\nabla F_k(\mathbf{w}, \xi^k) - \nabla F_k(\mathbf{w})\|^2 \leq \sigma_k^2.$$

4. (**Uniformly bounded gradient**). The expected squared norm of stochastic gradients is uniformly bounded: *for all mini-batches* ξ^k *at client* $k \in [N]$ *and for* $\mathbf{w} \in R^d$,

$$E\|\nabla F_k(\mathbf{w}, \xi^k)\|^2 \leq G^2.$$

These conditions on local functions are typical and widely used for the convergence analysis of different federated averaging frameworks [122, 186].

Theorem 1. *Define* $\kappa = \frac{L}{\mu}$, $\gamma = max\{8\kappa, E\}$ *and choose learning rate* $\eta_t = \frac{2}{\mu(\gamma+t)}$. *Then,*

the convergence of FedHDC with Non-IID datasets and partial client participation satisfies

$$E[F(\mathbf{w}_T)] - F^* \leq \frac{2\kappa}{\gamma + T}\left[\frac{B}{\mu} + \left(2L + \frac{E\mu}{4}\right)\|\mathbf{w}_0 - \mathbf{w}^*\|^2\right] \tag{3.25}$$

where

$$B = \sum_{k=1}^{N} p_k^2 \sigma_k^2 + 6L\Gamma + 8(E-1)^2 G^2 + \frac{N-K}{N-1}\frac{4}{K}E^2 H^2 \tag{3.26}$$

Here, T is the number of communication rounds (or SGD steps), The term Γ is used to quantify the degree of Non-IID [122]. Let F^* and F_k^* be the minimum values of F and F_k, respectively, then $\Gamma = F^* - \sum_{k=1}^{N} p_k F_k^*$. As shown in Theorem 1, FedHDC can achieve $\mathcal{O}(\frac{1}{T})$ convergence rate. Such claim does not hold for non-convex and non-linear DNNs. This result follows from the standard proof on the convergence of FedAvg on Non-IID data [122]. The proof is given in Appendix A.

3.4.2 FedHDC Experiemental Results

We implemented FedHDC on Python using a custom HDC library for the PyTorch framework. For FedHDC, we use hypervectors with dimension 10,000. For comparison, we use a NN with a fully connected layers with 128 units and ReLU activation, and a final output layer with softmax.

To observe the performance of our approach focusing on the real-world use-cases, we evaluated FedHDC on a wide range of benchmarks shown in Table 3.1 that range from relatively small datasets collected in a small IoT network to a large dataset that includes hundreds of thousands of face images. The data include: **ISOLET:** recognizing audio of the English alphabet, **UCIHAR:** detecting human activity based on 3-axial linear acceleration and angular velocity data, from different people, **PAMAP2:** classifying five human activities based on a heart rate and inertial measurements, **FACE:** classifying images with faces/non-faces, and **MNIST:** recognizing handwritten digits by different people.

Table 3.1. Datasets (n: Feature Size, K: Number of Classes)

Dataset	n	K	**Train** (Size)	**Test** (Size)	**Description**
ISOLET	617	26	6,238	1,559	Voice Recognition
UCIHAR	561	12	6,213	1,554	Activity Recognition (Mobile)
PAMAP2	75	5	611,142	101,582	Activity Recognition (IMU)
FACE	608	2	522,441	2,494	Face Recognition
MNIST	784	10	60,000	10,000	Handwritten Digit Recognition

Accuracy and Convergence

We run our experiments for 100 clients and 100 rounds of communication. We first tune the hyperparameters for both FedHDC and CNNs, then experiment with different federated learning parameters. Fig. 3.4 shows the accuracy and convergence of both FedHDC and the CNN for various number of local epochs E and local batch sizes B. For all experiments, $C = 0.2$ fraction of clients are randomly picked in every communication round. For all datasets, the best convergence is achieved with low number of epochs ($E = 1$) and moderate batch sizes ($B = 10, 20$).

Table 3.2. Impact of Dimensionality on FedHDC Accuracy

d	**1000**	**2000**	**4000**	**8000**	**10000**
ISOLET	90.79%	93.36%	95.07%	95.37%	94.59%
UCIHAR	90.60%	93.98%	93.63%	93.54%	94.46%
PAMAP2	74.9%	76.88%	76.10%	77.85%	77.98%
FACE	95.05%	95.2%	95.74%	95.86%	96.17%
MNIST	92.24%	93.81%	95.37%	96.34%	96.80%

Hypervector Dimensionality Study

Table 3.2 demonstrates the influence of hypervector dimensions on the FedHDC classification accuracy. A modest increase in accuracy is observed as the dimensionality grows. This outcome aligns with expectations, as the robustness of HDC is known to improve with increasing dimensions [195][170] [91]. Thomas [195] showed that dimensionality is directly proportional to the bandwidth of the noise in HDC classification problems, thus providing a guideline for a

Table 3.3. HDC on image data

Model	CIFAR10	CIFAR100	Flowers	dtd	GTSRB
HD-linear	26.94	8.98	19.58	6.41	83.63
HD-non linear	41.98	20.35	25.68	8.13	84.11
HD-id level	26.56	9.45	15.97	6.25	44.86
CNN	90.1	78.4	81	98.7	94.6

tradeoff between noise and the hypervector size. It is essential to consider the trade-off between performance and resource usage, as the computational cost rises with increasing dimensions.

In essence, the HD encoding dimension exhibits a linear relationship with the number of categorical features, while it depends logarithmically on the alphabet size. As previously mentioned, the separation quality of the problem is associated with factors such as the class separability and the encoding dimension. Intuitively, when the classes are well separated, a smaller encoding dimension can be employed to achieve satisfactory performance. This is because the inherent separability of the data aids in reducing the required dimensionality for efficient classification. Conversely, when the classes are poorly separated, a larger encoding dimension is necessary to enhance the robustness and accuracy of the classification process. Consequently, understanding the relationship between the HD encoding dimension and the problem's complexity is crucial for optimizing the performance of high-dimensional computing methods in various classification tasks.

3.5 FHDnn: Federated Hyperdimensional Computing with CNN Feature Extraction

FedHDC gives great results for many datasets in a federated setting, but it does not have acceptable accuracy when doing complex image analysis due to inherent inaccuracy of HDC on larger images. Table 3.3 summarizes accuracy of various state of the art encoding methods for HDC when running image classification tasks [40]. The current HD encoding methods are not able to match state of the art accuracy. In this section, to overcome this issue, we present FHDnn, a synergetic FL framework which combines CNNs and HDC. FHDnn uses a pre-trained CNN as

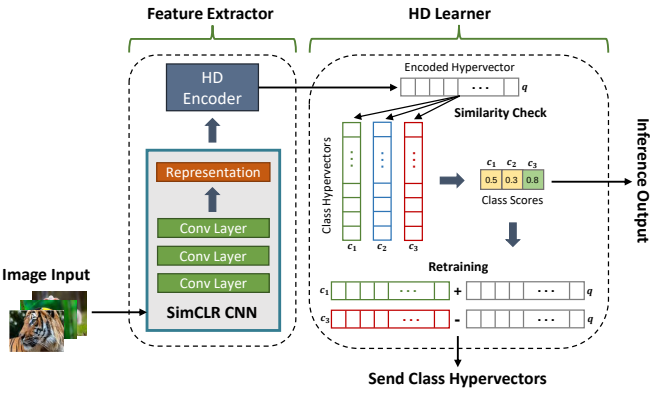


Figure 3.5. FHDnn Model Architecture

a feature extractor, whose outputs are encoded into hypervectors and then used for training. It avoids the transmission of the CNN and instead trains only the HD learner in a federated manner. The CNN excels at learning a complex hierarchy of features and boasts high accuracy, whereas HDC provides efficient and robust training. Therefore, FHDnn enjoys the complimentary salient properties of both HDC and CNN to enable a lightweight, communication-efficient, and highly robust FL framework.

3.5.1 Model Architecture

FHDnn consists of two components: i) a pre-trained CNN as a feature extractor and ii) a federated HD learner. Fig. 3.5 shows the model architecture of FHDnn. The pre-trained feature extractor is trained once and not updated at run time. This removes the need for costly CNN weight updates via federated learning. Instead, HD Computing is responsible for all the federated model updates. Since its training only requires simple operations, it is much more efficient and scalable. In the next subsections we describe both components.

Feature Extractor: While in theory any standard CNN can be used as a feature extractor, we use a pre-trained SimCLR ResNet model as our feature extractor due to its proven success in prior studies. SimCLR [27] is a contrastive learning framework which learns representations of images in a self-supervised manner by maximizing the similarity between latent space representations of different augmentations of a single image. This class-agnostic framework

trained on a large image dataset allows for transfer learning over multiple datasets, (as evaluated in [27]) making it ideal for a generic feature extractor. Standard CNNs learn representations that are fine-tuned to optimize the classification performance of the dense classifier at the end of the network. Since SimCLR focuses on learning general representations as opposed to classification oriented representations, it is a better choice of a feature extractor. We choose the ResNet architecture due to availability of pre-trained models. It is possible to use other models such as MobileNet [70].

HD Learner: FHDnn encodes the outputs of the feature extractor into hypervectors. More formally, given a point $\mathbf{x} \in \mathscr{X}$, the features $\mathbf{z} \subset Z^n$ are extracted using the feature extractor $f : \mathscr{X} \to Z$ where f is a pre-trained neural network. The HD embedding is constructed as $\mathbf{h} = \phi(\mathbf{z}) = \text{sign}(\phi\mathbf{z})$ under the encoding function $\phi : Z \to \mathscr{H}$. HD learner then operates on these hypervectors using binding and bundling which are simple and highly parallelizable. The goal of such configuration is to avoid the transmission of the CNN and instead train only the HD learner in a federated manner. An HD model is formed by bundling all encoded hypervectors with the same class level together. We perform bundling by the element-wise addition of those hypervectors, which generates corresponding class prototoypes. Then, the HD model is simply a set of hypervectors with the number of classes in the dataset. We use the HD learner in federated training that we discuss in the following.

3.5.2 Federated Training

Fig. 3.6 summarizes the overall federated training process for FHDnn. We separate the whole process into two steps, client local training and federated bundling. These two steps work in a cyclical fashion, one after the other, until convergence.

Client Local Training: Each client initially starts the process with a feature extractor f and an untrained HD learner. Once we get the encoded hypervectors using the method described above, we create class prototypes by bundling together hypervectors of the corresponding class using $\mathbf{c}_k = \sum_i \mathbf{h}_i^k$. Inference is done by computing the cosine similarity metric between a

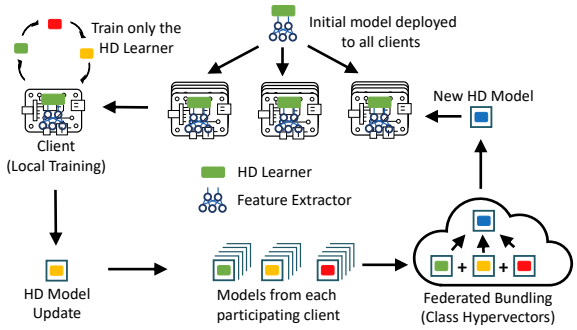


Figure 3.6. FHDnn Federated Training

given encoded data point with each of the prototypes, returning the class which has maximum similarity. After this one-shot learning process, we iteratively refine the class prototypes by subtracting the hypervectors from the mispredicted class prototype and adding it to the correct prototype as shown in Fig. 3.5. We define the complete HD model $\mathbf{C}$ as the concatenation of class hypervectors, i.e., $\mathbf{C} = [\mathbf{c}_1^T, \mathbf{c}_2^T, ..., \mathbf{c}_l^T]$.

Federated Bundling: In the federated bundling framework, each client maintains its own HD model and participates to build a global model in a distributed fashion. This is achieved via an iterative training procedure for which we describe one round (say t-th) of the algorithm below.

1. *Broadcast:* The central server broadcasts the latest global HD model, $\mathbf{C}^t$, to all clients.

2. *Local updates:* Each participating client $k \in [N]$ sets its model $\mathbf{C}_t^k = \mathbf{C}_t$ and then performs training for E epochs using local data.

3. *Aggregation:* The central server receives and aggregates the local models to produce a new global model:

$$\mathbf{C}_{t+1} = \sum_{k=1}^{N} \mathbf{C}_{t+1}^k. \tag{3.27}$$

After aggregation, the server moves on to the next round, $t+1$. This procedure is carried out until sufficient convergence is achieved.

3.5.3 FL Over Unreliable Channels With FHDnn

Federated learning is often carried out over wireless channels that attenuate the transmitted signal and introduce noise. Thus, the communication between clients and the server is unreliable, prone to transmission errors followed by packet losses. In this section, we show how FHDnn and FedHDC provide reliability for learning over unreliable wireless network at no overhead.

We consider different models for the uplink and the downlink channels. The centralized server is assumed to be able to broadcast the models reliably, error-free at arbitrary rates, which is a common assumption in many recent works [122, 96, 193, 176, 21]. For uplink communications, the channel capacity per client is notably more constrained as the wireless medium is shared, so transmissions can be unreliable even at very low rates. We next describe the considered communication setup over such multiple access channels (MAC).

The mutual interference between the transmissions of multiple participating clients can lead to erroneous aggregation of models at the server. A common approach in FL to deal with interference is to use an orthogonal frequency division multiple access (OFDMA) technique [56]. The resources of the shared-medium are partitioned in the time–frequency space and allocated among the clients. This way, each of the N clients occupies one dedicated resource block, that is, channel's spectral band and time slot.

Even though each client model can be recovered separately due to the orthogonality, the distinct channels are still inherently noisy. The individual, independent uplink channels should be rate-limited to be treated as error-free links under the Shannon capacity theorem. However, the bandwidth allocated per client decreases with the number of clients, so does the capacity. Accordingly, the volume of data that can be conveyed reliably, i.e, throughput, scales by $1/N$. This implies that the data rates will be small, resulting in slow training speed unless transmission power is increased, which is undesirable considering energy consumption concerns.

Instead of limiting the rate to achieve error-free communication, we admit errors for the

channel output at the server. The intuition is that the perturbations in the client models can be tolerated to a certain extent by the learning algorithm. If the learning model is robust to errors, then there is no need for forcing perfectly reliable transmissions. Thus, we analyze our FHDnn scheme assuming that the clients communicate over unreliable MAC and the transmitted models are corrupted.

In the following, we consider three error models at different layers of the network stack. All models are applicable in practice depending on the underlying protocol. We first explore the properties of HD computing that makes the learning robust under the considered error models, then introduce different techniques for further improvement.

Noisy Aggregation

In conventional systems, the transmitter performs three steps to generate the wireless signal from data: source coding, channel coding, and modulation. First, a source encoder removes the redundancies and compresses the data. Then, to protect the compressed bitsream against the impairments introduced by the channel, a channel code is applied. The coded bitstream is finally modulated with a modulation scheme which maps the bits to complex-valued samples (symbols), transmitted over the communication link.

The receiver inverts the above operations, but in the reverse order. A demodulator first maps the received complex-valued channel output to a sequence of bits. This bitstream is then decoded with a channel decoder to obtain the original compressed data; however, it might be possibly corrupted due to the channel impairments. Lastly, the source decoder provides a (usually inexact) reconstruction of the transmitted data by applying a decompression algorithm.

For noisy aggregation, as an alternative of the conventional pipeline, we assume uncoded transmission [52]. This scheme bypasses the transformation of the model to a sequence of bits, which are then need to be mapped again to complex-valued channel inputs. Instead, the real model parameter values are directly mapped to the complex-valued samples transmitted over the channel. Leveraging the properties of uncoded transmission, we can treat the channel as

formulated in Equation (3.28), where the additive noise is directly applied to model parameters. The channel output received by the server for client k at round t is given by

$$\tilde{\mathbf{w}}_t^k = \mathbf{w}_t^k + \mathbf{n}_t^k \tag{3.28}$$

where $\mathbf{n}_t^k \sim \mathcal{N}(0, \sigma_{t,k}^2)$ is the d-dimensional additive noise. The signal power and noise power are computed as $E\|\mathbf{w}_t^k\|^2 = P_{t,k}$ and $E\|\mathbf{n}_t^k\|^2 = \sigma_{t,k}^2$, respectively. Then, the signal-to-noise ratio (SNR) is:

$$SNR_{t,k} = \frac{E\|\mathbf{w}_t^k\|^2}{E\|\mathbf{n}_t^k\|^2} = \frac{P_{t,k}}{\sigma_{t,k}^2} \tag{3.29}$$

An immediate result of federated bundling is the improvement in the SNR for the global model. When the class hypervectors from different clients are bundled at the server, the signal power scales up quadratically with the number of clients N, whereas the noise power scales linearly. Assuming that the noise for each client is independent, we have the following relation:

$$SNR_t = \frac{E\left[\sum_{k=1}^N \mathbf{w}_t^k\right]}{E\left[\sum_{k=1}^N \mathbf{n}_t^k\right]} \approx \frac{N^2 P_{t,k}}{N\sigma_{t,k}^2} = N \times SNR_{t,k} \tag{3.30}$$

Notice that the effect of noise is suppressed by N times due to bundling. This claim can also be made for the FedAvg [105] framework over CNNs. However, even though the noise reduction factor is the same, the impact of the small noise might be amplified by large activations of CNN layers. In FHDnn, we do not have such problem as the inference and training operations are purely linear.

One other difference of FHDnn from CNNs is its information dispersal property. HD encoding produces hypervectors which have holographic representations, meaning that the information content is spread over all the dimensions of the high-dimensional space. In fact, no dimension in a hypervector is more responsible for storing any piece of information than others. Since the noise in each dimension can be also assumed independent, we can leverage the information spread to further eliminate noise.

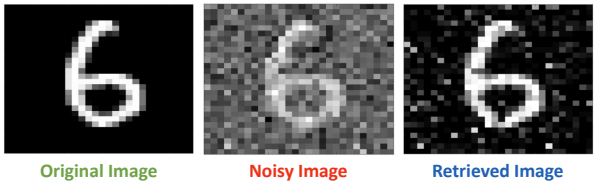


Figure 3.7. Noise robustness of hyperdimensional encodings

Consider the random projection encoding described in Section 1.1.1, which is also illustrated by Fig. 1.1a. Let the encoding matrix $\phi \in R^{d\times n}$ expressed in terms of its d row vectors, i.e., $\phi = [\phi_1, \phi_2, ..., \phi_d]^T$. Then, the hypervector formed by encoding information $\mathbf{x} \in \mathscr{X}$ can be written as $\mathbf{h} = [\phi_1^T\mathbf{x}, \phi_2^T\mathbf{x}, ..., \phi_d^T\mathbf{x}]^T$, where $\mathbf{x} = [x_1, x_2, ..., x_n]^T$. As implied by this expression, the information is dispersed over the hypervectors uniformly. Now consider additive noise over the same hypervector such that $\mathbf{h}+\mathbf{n} = [\phi_1^T\mathbf{x}+n_1, \phi_2^T\mathbf{x}+n_2, ..., \phi_d^T\mathbf{x}+n_d]^T$. We can reconstruct the encoded information from the noisy hypervector $\tilde{\mathbf{h}} = \mathbf{h}+\mathbf{n}$ as follows:

$$\mathbf{x} \approx \left[\frac{1}{d}\sum_{i=1}^{d}\Phi_{i,1}\tilde{\mathbf{h}}_i, \frac{1}{d}\sum_{i=1}^{d}\Phi_{i,2}\tilde{\mathbf{h}}_i, ..., \frac{1}{d}\sum_{i=1}^{d}\Phi_{i,n}\tilde{\mathbf{h}}_i\right] \tag{3.31}$$

where $\tilde{\mathbf{h}}_i = \phi_i^T\mathbf{x}+n_i$ are the elements of the noisy hypervector. The noise variance is then reduced by the averaging operation, similar to the case in Equation (3.30). Therefore, in HD computing, the noise is not only suppressed by bundling accross models from different clients, but also by averaging over the dimensions within the same hypervector. We demonstrate this over an example where we encode a sample from the MNIST dataset, add Gaussian noise, then reconstruct it. Fig. 3.7 shows the original image, noisy image in the sample space, and reconstructed image for which the noise was added in the hyperdimensional space.

Finally, there is a "flying under the radar" principle for federated learning over noisy channel. The analysis in [209] shows that since SGD is inherently a noisy process, as long as the channel noise do not dominate the SGD noise during model training, the convergence behavior is not affected. As the noise is immensely suppressed in FHDnn, we can claim such principle holds true in our case.

Bit Errors

We use bit error rate (BER) in conventional coded transmission as a figure of merit for system robustness. It is a measure on how accurately the receiver is able to decode transmitted data. The errors are bit flips in the received digital symbols, and are simply evaluated by the difference (usually Hamming distance) between the input bitstream of channel encoder and the output bitstream of channel decoder. Let $\hat{\mathbf{w}}$ be the binary coded model parameters that are communicated to the server. For the bit error model, we treat the channel as a binary symmetric channel (BSC), which independently flips each bit in $\hat{\mathbf{w}}$ with probability p_e (e.g., $0 \rightarrow 1$). The received bitstream output at the server for client k at round t is then as follows:

$$\tilde{\hat{\mathbf{w}}}_t^k = \hat{\mathbf{w}}_t^k \oplus \mathbf{e}_t^k \tag{3.32}$$

where $\mathbf{e}_t^k$ is the binary error vector and $\oplus$ denotes modulo 2 addition. Given a specific vector $\mathbf{v}$ of Hamming weight $\mathrm{wt}(\mathbf{v})$, the probability that $\mathbf{e}_t^k = \mathbf{v}$ is given by

$$P(\mathbf{e}_t^k = \mathbf{v}) = p_e^{\mathrm{wt}(\mathbf{v})}(1-p_e)^{m-\mathrm{wt}(\mathbf{v})} \tag{3.33}$$

The bit error probability, p_e, is a function of both the modulation scheme and the channel coding technique (assuming lossless source coding). To conclude the transmission, the corrupted bitstream in (3.32) is finally reconstructed to a real-valued model, i.e., $\tilde{\hat{\mathbf{w}}}_t^k \rightarrow \tilde{\mathbf{w}}_t^k$.

Bit errors can have a detrimental effect on the training accuracy, especially for CNNs. At worst case, a single bit error in one client in one round can fail the whole training. In Fig. 3.8 we give an example of how much difference a single bit error can make for the standard 32 bit floating point CNN weights. In floating point notation, a number consists of three parts: a sign bit, an exponent, and a fractional value. In *IEEE 754* floating point representation, the sign bit is the most significant bit, bits 31 to 24 hold the exponent value, and the remaining bits contain the fractional value. The exponent bits represent a power of two ranging from -127 to 128. The

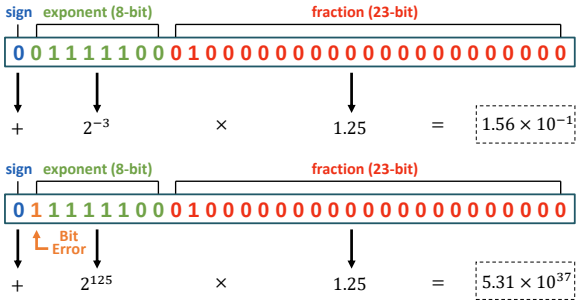


Figure 3.8. Single bit error on a floating-point number

fractional bits store a value between 1 and 2, which is multiplied by 2^{exp} to give the decimal value. Our example shows that one bit error in the exponent can change the weight value from 0.15625 to 5.31×10^{37}.

The bit errors are contagious because a parameter from one client gets aggregated to the global model, then communicated back to all clients. Furthermore, errors propagate through all communication rounds because local training or aggregation does not completely change the parameter value, but only apply small decrements. For instance, assume a federated learning scenario with 100 clients and one bit error in a client's model as in the above example. After 10 rounds of training, the CNN weight for the global model will be on the order of $\sim \frac{5.31 \times 10^{37}}{100^{10}} = 5.31 \times 10^{17}$, still completely failing the whole model. Consider ResNet-50, which has 20 million parameters, so training 100 clients even over a channel with $p_e = 10^{-9}$ BER results in two errors per round on average, making model failure inevitable.

A similar problem exists with HD model parameters, but to a lesser extent because the hypervector encodings use integer representations.

Fig. 3.10 implies that the parameters can also change significantly for the HD model. Particularly, errors in the most significant bits (MSB) of integer representation leads to higher accuracy drop. We propose a quantizer solution to prevent this.

The adopted quantizer design is illustrated in Fig. 3.9. Inspired by the classical quantization methods in communication systems, we leverage *scaling up* and *scaling down* operations at the transmitter and the receiver respectively. This can be implemented by the automatic gain

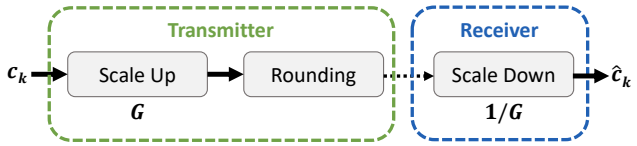


Figure 3.9. Quantizer scheme

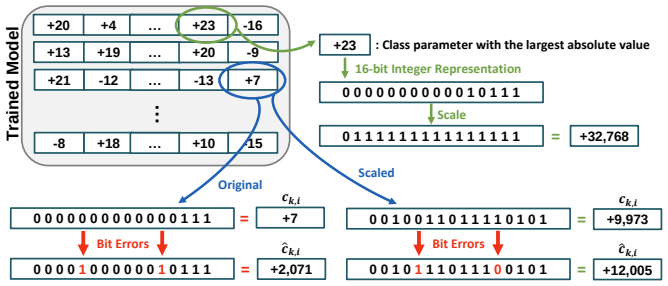


Figure 3.10. An example scaling up operation

control (AGC) module in the wireless circuits. For a class hypervector $\mathbf{c}_k$, $k \in \{1,...,K\}$, the quantizer output $Q(\mathbf{c}_k)$ can be obtained via the following steps:

1. **Scale Up:** Each dimension in the class hypervector, i.e. $c_{k,i}$, is amplified with a scaling factor denoted quantization gain G. We adjust the gain such that the dimension with the largest absolute value attains the maximum value attainable by the integer representation. Thus, $G = \frac{2^{B-1}-1}{\max(c_k)}$ where B is the bitwidth.

2. **Rounding:** The scaled up values are truncated to only retain their integer part.

3. **Scale Down:** The receiver output is obtained by scaling down with the same factor G.

This way, bit errors are applied to the scaled up values. Intuitively, we limit the impact of the bit error on the models. Remember, from Equation (3.4), that prediction is realized by a normalized dot-product between the encoded query and class hypervectors. Therefore, the ratio between the original parameter and the received (corrupted) parameter determines the impact of the error on the dot-product. Without our quantizer, this ratio can be very large whereas after scaling *up* then later *down*, it is diminished. Fig. 3.10 demonstrates this phenomenon. The ratio between the corrupted and the original parameter is $\frac{\hat{c}_{k,i}}{c_{k,i}} = \frac{2,071}{7} \approx 295.9$. The ratio decreases to

only $\frac{\hat{c}_{k,i}}{c_{k,i}} = \frac{12{,}005}{9{,}973} \approx 1.2$ between the scaled versions.

Packet Loss

At the physical layer of the network stack, errors are observed in the form of additive noise or bit flips directly on the transmitted data. On the other hand, at the network and transport layers, packet losses are introduced. The combination of network and protocol specifications allows us to describe the error characteristics, with which the data transmission process has to cope.

The form of allowed errors, either bit errors or packet losses, are decided by the error control mechanism. For the previous error model, we assumed that the bit errors are admitted to propagate through the transport hierarchy. This assumption is valid for a family of protocols used in error resilient applications that can cope with such bit errors [206]. In some protocols, the reaction of the system to any number of bit errors is to drop the corrupted packets [111]. These protocols employ a cyclic redundancy check (CRC) or a checksum that allows the detection of bit errors. In such a case, the communication could assume bit-error free, but packet lossy link. We use the packet error rate (PER) metric as a performance measure, whose expectation is denoted packet error probability p_p. For a packet length of N_p bits, this probability can be expressed as:

$$p_p = 1 - (1 - p_e)^{N_p} \tag{3.34}$$

The common solution for dealing with packet losses and guarantee successful delivery is to use a reliable transport layer communication protocol, e.g., transmission control protocol (TCP), where various mechanisms including acknowledgment messages, retransmissions, and time-outs are employed. To detect and recover from transmission failures, these mechanisms incur considerable communication overhead. Therefore, for our setup we adopt user datagram protocol (UDP), another widely used transport layer protocol. UDP is unreliable and cannot guarantee packet delivery, but is low-latency and have much less overhead compared to TCP.

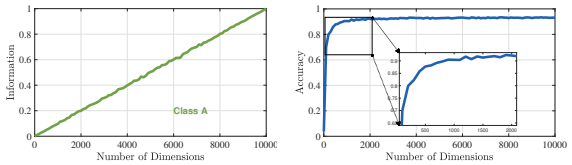


Figure 3.11. Impact of partial information on similarity check (left) and classification accuracy (right)

HDC's information dispersal and holographic representation properties are also beneficial for packet losses. Another direct result of these concepts is obtaining partial information on data from any part of the encoded information. The intuition is that any portion of holographic coded information represents a blurred image of the entire data. Then, each transmitted symbol–packets in our case–contains an encoded image of the entire model.

We demonstrate the property of obtaining partial information as an example using a speech recognition dataset [1]. In Fig. 3.11a, after training the model, we increasingly remove the dimensions of a certain class hypervector in a random fashion. Then we perform a similarity check to figure out what portion of the original dot-product value is retrieved. The same figure shows that the amount of information retained scales linearly with number of remaining dimensions. Fig. 3.11b further clarifies our observation. We compare the dot-product values across all classes and find the class hypervector with the highest similarity. Only the relative dot-product values are important for classification. So, it is enough to have the highest dot-product value for the correct class, which holds true with $\sim 90\%$ accuracy even when 80% of the hypervector dimensions are removed.

3.5.4 Strategies for Improving Communication Efficiency

The simplest implementation of FHDnn requires that clients send a full model back to the server in each round. Even though HDC models are much smaller than DNN models, it can still put a burden on communication. The structure and the characteristics of class hypervectors allow us to leverage certain techniques for improving communication efficiency of FHDnn.

We propose three approaches: i) binarized differential transmission, ii) subsampling, and iii) sparsification & compression.

Binarized Differential Transmission

At the beginning of each round, the central server broadcasts the latest global HD model, $\mathbf{C}_t$, to all clients. Then, before performing local updates, each client makes a copy of this global model. Instead of sending the local updated models $\mathbf{C}_{t+1}^k$ at the aggregation step, the clients send the difference between the previous model and the updated model, i.e., $\mathbf{C}_{t+1}^k - \mathbf{C}_t$. We call this operation *differential transmission*. As shown in (3.35), we binarize the difference to reduce the communication cost by 32x, going from 32-bit floating point to 1-bit binary transmission.

$$\Delta\mathbf{C}_{bin}^k = \text{sign}(\mathbf{C}_{t+1}^k - \mathbf{C}_t),\ \forall k \tag{3.35}$$

The central server receives and aggregates the differences, then adds it to the previous global model as:

$$\mathbf{C}_{t+1} = \mathbf{C}_t + \sum_{k=1}^{N} \mathbf{C}_{bin}^k \tag{3.36}$$

This global model is broadcasted back to the clients. Such binarization framework is not possible for the original federated bundling approach where clients communicate their full models. Binarizing the models itself instead of the 'difference' results in unstable behavior in training. Therefore, we utilize binarized differential transmission whose stability can be backed by studies on similar techniques. In [15], it is theoretically shown that transmitting just the sign of each minibatch stochastic gradient can achieve full-precision SGD-level convergence rate in distributed optimization.

Subsampling

In this approach, the clients only send a subsample of their local model to the central server. Each client forms and communicates a subsample matrix $\hat{\mathbf{C}}_{t+1}^k$, which is formed from

a random subset of the values of $\mathbf{C}_{t+1}^k$. The server then receives and averages the subsampled client models, producing the global update $\mathbf{C}_{t+1}$ as:

$$\hat{\mathbf{C}}_{t+1} = \frac{1}{N} \sum_{k=1}^{N} \hat{\mathbf{C}}_{t+1}^k \tag{3.37}$$

The subsample selection is completely randomized and independent for each client in each round. Therefore, the average of the sampled models at the server is an unbiased estimator of their true average, i.e., $E\|\hat{\mathbf{C}}_t\| = \mathbf{C}_t$. We can achieve the desired improvement in communication by changing the subsampling rate. For example, if we subsample 10% of the values of $\mathbf{C}_{t+1}^k$, the communication cost is reduced by 10x.

Sparsification & Compression

The goal of this approach is to drop the elements (class hypervector dimensions) of each individual class that have the least impact on model performance. As discussed in Section 3.3.1, given a query hypervector, inference is done by comparing it with all class hypervectors to find the one with the highest similarity. The similarity is typically taken to be the cosine similarity and calculated as a normalized dot-product between the query hypervector and class hypervectors. The elements of a query hypervector are input dependent and changes from one input to another one. Due to the randomness introduced by HDC encoding, the query hypervectors, on average, have a uniform distribution of values in all dimensions. Under this assumption, we need to find and drop the elements of class hypervectors that have minimal impact on cosine similarity. Indeed, the elements with the smallest absolute values are the best candidates as they have the least contribution to the dot-product computation of cosine similarity.

We find the elements of each class hypervector with the smallest absolute value and make those elements zero. For example, for the i^{th} class hypervector, we select S elements with the minimum absolute value as follows.

$$\min\{c_d^i, ..., c_2^i, c_1^i\}_S \tag{3.38}$$

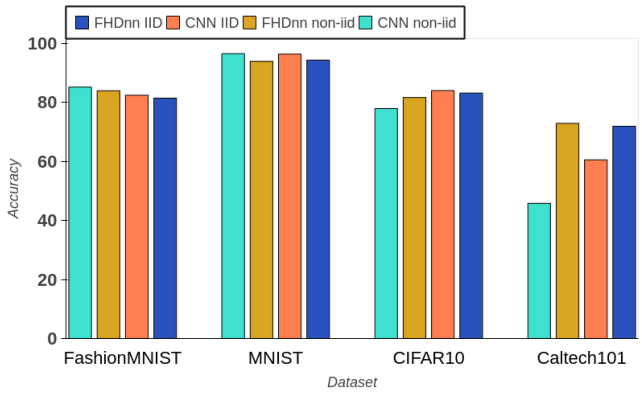


Figure 3.12. Accuracy of FHDnn and ResNet on different datasets

To make a model with $S\%$ sparsity, we make $\frac{S}{100} \times d$ elements of each class hypervector zero. Then, we employ the Compressed Sparse Column (CSC) [132] to compress the sparse model. CSC stores only the non-zero data values and the number of zero elements between two consecutive non-zero elements.

3.6 FHDnn Results

We demonstrate through systematic experiments the performance of FHDnn under various settings. We first briefly discuss the datasets and setup for evaluation, and present our results for different data distributions under the reliable communication scenario. We then compare the resource usage of FHDnn against CNNs. The strategies for improving communication efficiency are also evaluated in this section. Lastly, we analyze FHDnn under three different unreliable network settings: packet loss, noise injection, and bit errors.

3.6.1 Experimental Setup

We evaluate FHDnn on 3 different real world datasets: MNIST[37], FashionMNIST[210], CIFAR10 [109] and Caltech101 [119]. For the MNIST dataset, we use a CNN with two 5x5 convolution layers, two fully connected layers with 320 and 50 units and ReLU activation, and a final output layer with softmax. The first convolution layer has 10 channels while the second

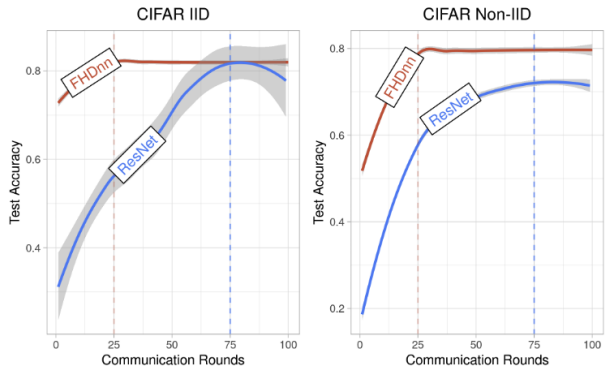


Figure 3.13. Accuracy and Number of communication rounds for various hyperparameters

one has 20 channels, and both are followed by 2x2 max pooling. While for the CIFAR10 and FashionMNIST datasets, the well-known classifier model, ResNet-18 with batch normalization proposed in [65], is used. We run our experiments on Raspberry Pi Model 3b and NVIDIA Jetson for the performance evaluations. All models are implemented on Python using the PyTorch framework. We consider an IoT network with $N = 100$ clients and one server. The simulations were run for 100 rounds of communication each in order to keep our experiments tractable.

We first tune the hyperparameters of both FHDnn and CNNs, and analyze their performance by experimenting with three key parameters: E, the number of local training epochs, B the local batch size, and C, the fraction of clients participating in each round. We select the best parameters for ResNet and use the same for FHDnn for all experiments in order to allow for a direct comparison. We study two ways of partitioning the datasets over clients: **IID**, where the data is shuffled and evenly partitioned into all clients, and **Non-IID**, where we first sort the data by their labels, divide it into a number shards of a particular size, and assign the shards to each of clients.

We test FHDnn on two different types of edge devices: Raspberry PI 4 (RPi)[159] and NVIDIA Jetson[33]. The RPi features a Broadcom BCM2711 quad-core Cortex-A72 (ARM v8) 64-bit SoC, running at 1.5GHz, and 4GB RAM. The NVIDIA Jetson uses a quad-core ARM Cortex-A57 CPU, 128-core NVIDIA Maxwell GPU, and 4GB memory.

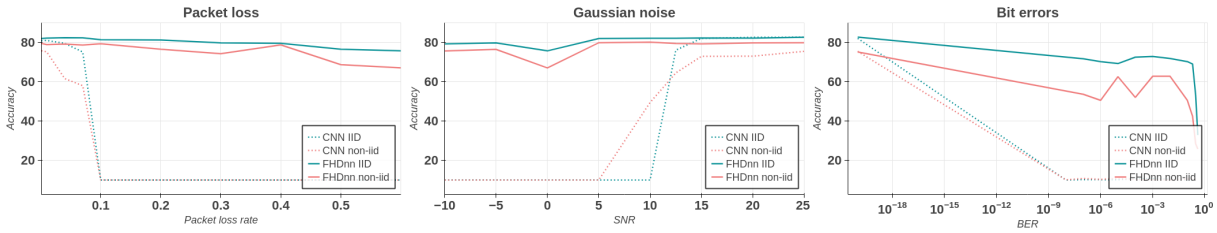


Figure 3.14. Accuracy comparison of FHDnn with ResNet with noisy network conditions

3.6.2 FHDnn Accuracy Results

Fig. 3.12 compares the test accuracy of FHDnn with ResNet on MNIST, CIFAR-10, FashionMNIST and Caltech101 datasets after 100 rounds of federated training. We observe that FHDnn achieves accuracy comparable to the state of the art, even though it trains a much smaller and less complex model. We depict how test accuracy changes over communication rounds for CIFAR-10 in Fig. 3.13. The plot illustrates the smoothed conditional mean of test accuracy across all different hyperparameters (E,B,C) for IID and Non-IID distributions. FHDnn reaches an accuracy of 82% in less than 25 rounds of communication whereas ResNet takes 75 rounds on average for both IID and Non-IID data distributions. Moreover the hyperparameters do not have a big influence for FHDnn as seen by the narrow spread (gray region) in Fig. 3.13. Note that the local batch size B doesn't impact FHDnn at all due to the linear and additive nature of its training methodology. This allows us to use higher batch sizes up to the constraints of the device, allowing for faster processing, and going over the dataset in less rounds. On the other hand, the batch size B affects the convergence of CNNs.

3.6.3 FHDnn Performance and Energy Consumption

Local training is computationally expensive for constrained IoT devices, which was one of the main drivers for centralized learning over many years. Particularly, CNN training involves complicated architectures and backpropagation operation that is very compute intensive. In addition, this has to be repeated for many communication rounds. HD on the contrary is lightweight, low-power, and fast. Table 3.4 quantitatively compares the computation time and

energy consumption of FHDnn and ResNet local training on 2 different edge device platforms. FHDnn is 35% faster and energy efficient than ResNet on Raspberry Pi and 80% faster and energy efficient on the Nvidia Jetson.

Table 3.4. Performance on Edge Devices

Device	Training Time (Sec)		Energy (J)	
	FHDnn	ResNet	FHDnn	ResNet
Raspberry Pi	**858.72**	1328.04	**4418.4**	6742.8
Nvidia Jetson	**15.96**	90.55	**96.17**	497.572

3.6.4 FHDnn in Unreliable Communication

In this section, we analyze the performance of FHDnn and ResNet under unreliable network conditions as described in Section 3.5.3. We obtained similar results for FedHDC, which is why we focus on FHDnn results in this section. Fig. 3.14 shows the performance of models under packet loss, Gaussian noise, and bit errors. To maintain a direct comparison between ResNet and FHDnn, we use the same hyperparameters for both models and all experiments. We set $E = 2, C = 0.2, B = 10$ and evaluate the performance on the CIFAR10 dataset. From our experiments, we observe that even with fewer clients at $C = 0.1$, and for other datasets, the performance of FHDnn is better than ResNet. Here, we present only the results for the settings mentioned earlier to keep it concise.

Packet Loss As shown in Fig. 3.14a, if the packet loss rate is extremely small, e.g., below 10^{-2}, ResNet has very minimal accuracy loss. However, for more, realistic packet loss rates such as 20% the CNN model fails to converge. When there is packet loss, the central server replaces the model weights from the lost packets with zero values. For example, 20% packet loss rate implies 20% of the weights are zero. Moreover, this loss is accumulative as the models are averaged during each round of communication thereby giving the CNNs no chance of recovery. In contrast, FHDnn is highly robust to packet loss with almost no loss in accuracy. For FHDnn, since the data is distributed uniformly across the entire hypervector, a small amount of missing

data is tolerable. However, since CNNs have a more structured representation of data with interconnections between neurons, the loss of weights affects the performance of subsequent layers which is detrimental to its performance.

Gaussian Noise We experiment with different Signal-to-Noise Ratios (SNR) to simulate noisy links, illustrated in Fig. 3.14b. Even for higher SNRs such 25dB the accuracy of ResNet drops by 8% under Non-IID data distribution. However it's more likely that IoT networks operating on low-power wireless networks will incur lower SNRs. For such scenarios, FHDnn outperforms ResNet as the latter fails to perform better than random classification. ResNet performance starts to completely deteriorate around 10dB SNR. The accuracy of FHDnn only reduces by 3%, even at -10dB SNR, which is negligible compared to ResNet.

Bit Errors Fig. 3.14c shows that CNNs completely fail when bit errors are present. ResNet achieves the equivalent of random classification accuracy even for small bit errors. Since the weights of CNNs are floating point numbers, a single bit flip can significantly change the value of the weights. This, compounded with federated averaging, hinders convergence. We observe FHDnn incurs an accuracy loss as well, achieving 72% for IID and 69% for Non-IID data. FHDnn uses integer representations which is again susceptible to large changes from bit errors to some extent. However, our quantizer method with scaling described in Section 3.5.3 assuages the remaining error.

3.6.5 FHDnn Communication Efficiency

So far we have benchmarked the accuracy of FHDnn for various network conditions. In the following, we demonstrate the communication efficiency of FHDnn compared to ResNet. We compare the amount of data transmitted for federated learning to reach a target accuracy of 80%. The amount of data transmitted by one client is calculated using the formula $data_{transmitted} = n_{rounds} \times update_{size}$, where n_{rounds} is the number of rounds required for convergence by each model. The update size for ResNet with 11M parameters is 22MB while that of FHDnn is 1MB making it 22$\times$ smaller. From Section 3.6.2 we know that FHDnn converges 3$\times$ faster

than ResNet bringing its total communication cost to 25MB. ResNet on the other hand uses up 1.65GB of data to reach the target accuracy.

In Fig. 3.13, we illustrated that FHDnn can converge to the optimal accuracy in much fewer communication rounds. However, this improvement is even higher in terms of the actual *clock time* of training. We assume that federated learning takes places over LTE networks where SNR is 5dB for the wireless channel. Each client occupies 1 LTE frame of 5MHz bandwidth and duration 10ms in a time division duplexing manner. For error-free communication, the traditional FL system using ResNet can support up to 1.6 Mbits/sec data rate, whereas we admit errors and communicate at a rate of 5.0 Mbits/sec. Under this setting and for the same experiment as in Section 4.2, FHDnn converges in 1.1 hours for CIFAR IID and 3.3 hours for CIFAR Non-IID on average. On the other hand, ResNet converges in 374.3 hours for both CIFAR IID and CIFAR Non-IID on average.

3.6.6 FHDnn: Effect of Communication Efficiency Strategies

Even though FHDnn is much smaller than CNN models and the training converges faster, it's communication efficiency can be further improved. We use the MNIST dataset and the parameters from Section 3.6.2 for our experiments. Table 3.5 shows the final accuracy after 100 rounds of training and the improvement in communication cost for the respective approaches.

Table 3.5. Simulation results

Method	Final Accuracy	Improvement
Baseline	94.1%	-
Binarized Differential Transmission	91.2%	32x
50% Subsampling	91.1%	2x
50% Sparsification & Compression	90.0%	2x
10% Subsampling	90.7%	10x
90% Sparsification & Compression	91.6%	10x

The differential transmission approach binarizes the model difference, going from 32-bit floating point to 1-bit binary transmission to reduce the communication cost by 32x. For

subsampling and sparsification & compression approaches, the communication improvement depends on the percentage of the model values that are subsampled or sparsified. For example, if we subsample 10% of the model, than the communication cost is reduced by 10x. Or, if we sparsify the model by 90%, the reduction is 10x again. We present the final accuracy and improvement in communication cost at different subsampling and sparsification percentages.

3.7 Conclusion

In this work we introduced methods to implement federated learning using hyperdimensional computing to enable communication efficient and robust federated learning for IoT networks. We first formalize the theoretical aspects of hyperdimensional computing to perform federated learning, presented as our first contribution called FedHD. To combat the inability of HDC to extract relevant features which consequently leads to poor performance of FedHD on large image classification, we propose FHDnn. FHDnn complements FedHD with a fixed contrastive learning feature extractor to compute meaningful representations of data that helps the HDC model better classify images. We described the federated hyperdimensional computing architecture, described the training methodology and evaluated FedHDC and FHDnn through numerous experiments in both reliable and unreliable communication settings. The experiment results indicate that FHDnn converges $3\times$ faster, reduces communication costs by $66\times$, local client compute and energy consumption by 1.5 - $6\times$ compared to CNNs. It is robust to bit errors, noise, and packet loss. Finally, we also showed that the communication efficiency of FedHDC and FHDnn can be further improved up to $32\times$ with a minimal loss in accuracy.

Next, we focus on systematically designing architectures to leverage the symbolic manipulation properties of HDC within Deep Learning. To this end, we first look at a large-scale text classification task that is a multi-label classification problem. We detail methods to represent text data as hyperdimensional representations and propose using convolution operators to manipulate these representations to learn a mapping between the inputs and the target.

3.8 Acknowledgements

Chapter 3 in full, is a reprint of the material as it appears in Federated Hyperdimensional Computing. ACM Transactions on the Internet of Things (Under review). The dissertation author was one of the primary investigators and author of this paper.

Chapter 4

Multi-Label Classification Using Hyperdimensional Representations

4.1 Introduction

Hyperdimensional computing (HDC) is an emerging paradigm of computing that offers a promising alternative to traditional machine learning approaches. In recent years, HDC has garnered significant interest due to its low computational overhead and hardware-friendly nature [90, 95]. HDC employs low-precision sparse representations and simple arithmetic operations to manipulate high-dimensional vectors, making it amenable to hardware acceleration. The independent and identically distributed (iid) nature of these representations further enables efficient parallelization, leading to improved computational efficiency. There is a large body of works showing benefits of HDC acceleration in hardware for various applications in IoT [99, 98, 100] and Machine Learning [40, 61, 81]. As a result, HDC has gained popularity in various domains, including natural language processing [53], [54], biomedical applications like DNA pattern matching [103] and protein alignment [179] and robotics [154]. Despite its success, the applicability of HDC for complex tasks like multi-label classification, which has real-world applications in recommender systems and document classification, has not been explored.

Our research presents the first comprehensive exploration of multi-label learning problems utilizing HDC representations. We introduce three novel approaches that strike an optimal balance between computational efficiency and accuracy. Our first approach, Power Set HD, is a

transformation method that achieves exceptional accuracy and efficiency on datasets with small label spaces and a limited subset of possible label combinations. For small datasets with larger label cardinality (≥ 4), we propose One-vs-All HD, which reduces the exponential complexity scaling of Power Set HD to a linear scale on the label set size, making it ideal for datasets with label sizes up to 30. In addition, we present TinyXML HD, a neural approach to learning mappings between hypervectors by decomposing the problem into multiple sub-problems. TinyXML HD fixes the output dimensionality of the model independent of the label size or cardinality, making it an ideal candidate for extreme multi-label classification problems.

HDC leverages neurally plausible representations of data and associates abstract concepts with high-dimensional vectors to perform complex cognitive tasks. The two fundamental operations of HDC are "bundling" and "binding" [95]. Bundling (denoted by $\oplus$) is used to represent multiple symbolic entities (hypervectors) using a single hypervector, while binding (denoted by $\otimes$) associates one entity with another.

We leverage the Multiply Add Permute (MAP) architecture proposed by Gayler [55], which uses bipolar representations for HDC. MAP represents data using high-dimensional vectors $X \in \{+1,-1\}^D$ called hypervectors. Gayler demonstrated that by assigning hypervectors with a conceptual meanin, we can represent conceptual relationships using these operators. For example, the sentence *"Yoda is a Jedi and Leia is a princess"* can be represented as $H = \text{Yoda} \otimes \text{Jedi} \oplus \text{Leia} \otimes \text{princess}$. HDC also allows us to query and reason about expressions. For instance, to find who is a Jedi, using the inverse operator $\tilde{\text{Jedi}}$ we can simply compose $H \otimes \tilde{\text{Jedi}} \approx \text{Yoda}$, which results in a hypervector that is approximately equal to Yoda.

Gradient-based neural methods have demonstrated tremendous success in various learning tasks [115]. They provide a systematic approach to finding the minima of a function [178] and can be efficiently computed provided the function is differentiable. The MAP framework uses element-wise products or additions that are themselves differentiable, however, the quantization step that follows is not differentiable. There are other HDC models with fully differentiable operations such as Holographic Reduced Representations (HRR)[160] which have been studied

by using a neural gradient-based approach in the context of multi-label classification like [49]. However, the HRR framework requires Fast Fourier Transform (FFT) operations which increase complexity. The MAP model in contrast, uses simple operations that can be accelerated using efficient bit-wise operations.

In [49], the authors utilized symbolic hypervectors to represent the labels and employed neural methods to learn a mapping from the instance space to the label space. To simplify the learning problem, we instead embedded both inputs and labels in the same high-dimensional vector space, which can be learned more easily than mapping across different vector spaces [67, 13, 14]. This is because the model does not need to learn complex transformations to map between spaces, reducing the complexity of the learning problem. After embedding inputs and labels as hypervectors in the same vector space, TinyXML HD learns a mapping between the two using a 1-D convolutional neural network designed for processing hypervectors.

In this work we introduce three methods that show the potential of HDC to solve multi-label classification problems across the entire spectrum of complexity - small, medium, and large, as described below:

- The first approach, Powerset HD, is suitable for small-scale multi-label classification, where each possible label combination is instantiated as a separate binary learning problem, resulting in exponential scaling over label size. This approach yields high-accuracy models that scale well for datasets with a few label combinations.

- The second approach, One-vs-All HD, is another transformation method that relaxes the exponential scaling of Powerset HD to linear scaling over label size, resulting in models that are efficient and accurate for datasets with a label set size of up to 30. Beyond this limit, the training time increases significantly, making this method less suitable.

- For extreme-scale multi-label problems, we propose TinyXML HD, which utilizes a 1-D convolutional neural network to learn hypervector representations. By having a fixed output dimensionality independent of the label complexity, TinyXML HD achieves remarkable

speedups in training. However, due to the relatively expensive convolution operations, the first two approaches provide a better trade-off between computational efficiency and accuracy for smaller size datasets.

- Through rigorous evaluations on real-world datasets, we demonstrate the superiority of our proposed methods. Powerset HD and One-vs-All HD offer up to 60x speedup on small-scale datasets, while TinyXML HD is 56x smaller compared to the state-of-the-art on medium-scale datasets and up to 836x smaller on extreme-scale datasets, all while maintaining comparable accuracy.

The rest of the article is organized as follows. We first review related work in 4.2, highlighting the differences between our approach and existing methods. In section 1.1.1, we provide an overview of HDC helpful for understanding the rest of the article. We split the problem into two variants: micro multi-label classification and extreme multi-label classification. We first tackle the micro multi-label classification in 4.3, where we discuss two simple problem transformation techniques and examine their performance on trivial learning problems in 4.5.1. We provide an overview of the Extreme Multi-Label Classification in 4.4, followed by Section 4.4.1 where we detail a new encoding method for representing text data as hypervectors. We then present our novel HDC convolution operator and neuro-symbolic approach in 4.4.2, detailing its formulation and demonstrating its effectiveness in Section 4.5.2.

4.2 Related Work

Hyperdimensional computing (HDC) is an emerging field that aims to address the limitations of traditional computing paradigms by leveraging high-dimensional vector representations to perform complex cognitive and machine learning tasks. This section presents a brief summary of various HDC works, highlighting their contributions to both cognitive tasks and machine learning tasks. We also present a brief survey of gradient based algorithms and multi-label classification for the readers benefit.

4.2.1 HDC for Cognitive & Learning Tasks

Kanerva introduced the foundational concept of a ”hyperdimensional computer”, which efficiently stores and retrieves information using large, sparse binary vectors [92]. This model exhibited robustness and efficiency in cognitive tasks, inspiring further research in HDC. Gallant et al. explored HDC in natural language understanding and reasoning, successfully capturing semantic relations in text and showcasing its potential for large-scale knowledge representation [48]. Rachkovskij expanded HDC's application to image processing and recognition, demonstrating pattern recognition capabilities with high accuracy and noise robustness [166]. In [194] Anthony et al. develops a theoretical framework for HDC and details the mathematical properties of HDC encoding methods.

In recent years, hyperdimensional computing (HDC) has emerged as a promising paradigm for machine learning, such as classification, regression, and reinforcement learning. Lai et al. employed HDC for classification tasks, developing a high-dimensional classifier that achieved competitive performance with reduced computational complexity [112]. Imani et al. applied HDC to regression problems, proposing a high-dimensional computing framework that provided accurate and efficient regression models with minimized computational overhead [80]. Goudarzi et al. explored HDC in reinforcement learning, developing a state representation and policy learning approach that demonstrated effectiveness in various environments [58]. Imani et al. proposed HDCluster, an accurate clustering algorithm for high-dimensional datasets using hyperdimensional computing [77]. In GENERIC [99], Khaleghi et al. proposed a novel and efficient method for learning on edge devices using hyperdimensional computing for a wide range of applications. The method utilizes hardware-friendly hyperdimensional vector representations and an optimized training algorithm to reduce computation and storage requirements while maintaining high accuracy. Guo et al. [61] proposed using hypervectors to represent users and items and performs a set of associative and distributive operations on these vectors to compute recommendations. The paper presents three different methods for generating recommendations,

including one that combines hyperdimensional computing with matrix factorization. Asgarinejad et al. [7] developed a method for epilepsy detection using EEG signals. They demonstrate using real-world data that HDC approaches outperforms state-of-the-art methods like Support Vector Machines (SVM) [203] and Convolutional Neural Networks (CNN) [114].

4.2.2 Gradient based HDC methods

Gradient-based methods in hyperdimensional computing (HDC) have garnered interest due to their potential for addressing optimization challenges in high-dimensional spaces. These methods extend the capabilities of HDC by incorporating gradient information to guide the learning process. For instance, Frady and Sommer introduced a gradient-based HDC framework, which allowed the use of optimization algorithms such as gradient descent and backpropagation in HDC settings [46]. In a subsequent work, Frady et al. proposed a method for gradient-based learning in HDC that utilized iterative projections and local linearizations to facilitate learning in high-dimensional spaces [47]. Building upon these developments, Wang et al. presented a gradient-based HDC algorithm for clustering and classification tasks, which employed a convex optimization formulation to enhance HDC's performance in these applications [207]. Moreover, Su et al. developed a gradient-based HDC algorithm for deep learning, illustrating the potential of gradient-based methods in improving the robustness and expressiveness of HDC models [188]. Recently, Zhou et al. presented a gradient-based HDC framework for unsupervised learning, focusing on clustering and dimensionality reduction tasks [222]. These studies highlight the increasing importance of gradient-based methods in HDC and their potential in addressing various learning tasks in high-dimensional spaces.

While these methods have shown promise in addressing optimization challenges in high-dimensional space, they introduce additional complexity in order to facilitate backpropogation through the HDC operations. For example, Frady and Somer's work involves iterative projections and local linearizations which can be expensive. Similarly Wang's convex optimization formulation for clustering and classification tasks can result in increased computational overhead

[207]

Prior works have also explored the use Holographic Reduced Representations (HRR), a family models for gradient based learning tasks. A notable attempt to capitalize on the symbolic properties of HRR was made by Nickel et al. [151], who utilized binding operations to link elements within a knowledge graph. Their approach served as an embedding mechanism that merged two vectors of information without increasing the dimensionality of the representation, as opposed to concatenation which doubles the dimension. In a more recent study, Liao and Yuan [125] employed circular convolution as a substitute for standard convolution to decrease model size and inference time, albeit without leveraging the symbolic properties inherent to HRRs. Although Danihelka et al. [35] claimed to incorporate HRR into an LSTM, their methodology simply augmented an LSTM with complex weights and activations, and did not genuinely implement HRR due to the absence of circular convolution.

4.2.3 Multi-Label Classification

The seminal works of multi-label classification emerged in the early 2000s with the introduction of the problem and initial approaches [201]. Since then a wide range of algorithms and techniques have been proposed to tackle this problem such as problem transformation methods [219], and ensemble methods [173, 134].

Among the various methods for multi-label classification, problem transformation methods have gained considerable attention. These techniques transform the multi-label problem into one or more single-label problems, which can then be addressed using traditional machine learning classifiers. One popular approach is the Binary Relevance (BR) method [198], which independently trains a binary classifier for each label. Another problem transformation method is the Label Powerset (LP) method [198], which treats each unique combination of labels as a single class in a multi-class problem. To address the shortcomings of BR and LP, researchers have proposed various ensemble and hybrid techniques. These include the Random k-Labelsets (RAkEL) method [202], which constructs multiple LP classifiers on random label subsets, and

the Classifier Chains (CC) method [174], which constructs a chain of binary classifiers while preserving label correlations. Apart from problem transformation methods, other multi-label classification techniques include algorithm adaptation methods, which modify single-label algorithms to handle multi-label data directly. Examples of such methods are the Multi-Label k-Nearest Neighbors (ML-kNN) algorithm [217], and the Multi-Label Decision Trees (MLDT) [141].

Extreme multi-label classification (XML) is a specialized form of multi-label classification, characterized by a large number of labels and instances. XML has attracted significant research attention due to its relevance in numerous real-world applications, such as large-scale document classification [17], image annotation [57], and gene function prediction [148]. Early approaches for XML include the FastXML algorithm [163], PfastreXML algorithm [83], and the Parabel algorithm [162]. Embedding-based methods, such as the SLEEC algorithm [17] and the AnnexML algorithm [190], have also been proposed for XML.

Deep learning approaches have shown considerable promise in XML tasks. Convolutional Neural Networks (CNNs) [128], Recurrent Neural Networks (RNNs) [149], and Transformer models [39] have been adapted for XML problems, demonstrating improved performance compared to traditional methods. Specifically, BERT [39] and its variants have been successfully applied to large-scale text classification tasks.

4.2.4 Motivation and Our Contributions

In the existing literature on hyperdimensional computing (HDC), the majority of studies have focused on small-scale learning problems. Ganesan et al. [49] examined the extreme multi-label text classification task, but other works have yet to explore the scalability of HDC techniques in addressing large-scale machine learning problems in real-world applications. Our research aims to bridge this gap by investigating the application of HDC to a demanding, industrial-scale learning problem.

SoA deep learning models for multi-label classification, such as LightXML [84] and

X-Transformer [24], comprising millions of parameters, necessitate days of training to achieve optimal performance. Our objective in this work is to examine HDC's potential for reducing this training time, thereby offering a more balanced trade-off between computational efficiency and accuracy.

Ganesan et al.[49] proposed a method for extreme multi-label text classification that replaces the final classification layer of AttentionXML[213] and XML-CNN[129] with a fully connected layer that outputs a hypervector encoding the relevant label information for an instance. While they demonstrated that their proposed method achieves accuracy similar to the baseline implementations of AttentionXML and XML-CNN, there are two key areas to improve upon. First, their method uses the HRR binding operation, which is a circular convolution requiring Fast Fourier Transform[32], an expensive operation. Second, their method learns a mapping from the instance space (represented as one-hot encoding) to the label space (represented as HRR hypervectors), which is a harder learning problem requiring learning the projection across the vector spaces.

In contrast, our approach is based on the Multiplicative Addition Perturbation (MAP) model introduced by Gayler[53], which uses bi-polar representations with simple element-wise arithmetic operations that can be easily accelerated and parallelized on hardware. Additionally, we embed the inputs and labels both in the same high-dimensional vector space, thereby avoiding the need to learn complex transformations across vector spaces. Our proposed neural approach for learning high-dimensional representations not only avoids increased computational complexity but also reduces the compute cost by a factor of 200.

4.3 Multi-Label OvA & PowerSet HD

Multi-label classification is a machine learning problem where an instance can belong to multiple classes simultaneously. Mathematically, it can be defined as follows: Let $\mathbf{x}$ be a feature vector representing an instance and $\mathbf{y}$ be a binary vector indicating the presence or absence of

L possible class labels, where $y_i = 1$ indicates the instance belongs to the i^{th} class and $y_i = 0$ indicates otherwise. The goal of multi-label classification is to learn a mapping function $f(\cdot)$ that takes as input an instance $\mathbf{x}$ and outputs a binary vector of length L indicating the classes the instance belongs to.

The complexity of this task can be influenced by various factors, such as the number of labels, the label dependencies, and the label cardinality. To distinctly refer to the class of problems with relatively small label spaces, we define a small scale variant of multi-label learning. This variant deals with datasets where the label space is simple and the number of instances is relatively small, resulting in label set sizes of less than 100. In Section 4.4 we discuss the characteristics of the most challenging variant of multi-label classification that focuses on very large problem sizes. Our prior work [147] laid the foundations of hyperdimensional multi-label classification by combining HDC with two well studied problem transformation techniques, One-vs-All[224] and Label Powerset[133], to solve micro size problems.

Problem transformation methods[172, 42, 198] have been proposed where the original multi-label problem is transformed into multiple single-label problems. Each transformed problem corresponds to one of the L class labels and involves training a binary classifier to distinguish instances that belong to that class from those that do not. The output of each binary classifier is then combined to obtain the final multi-label prediction. These methods can be further classified into three categories: 1) One-vs-All[224], 2) Label Powerset[133], and 3) Classifier Chains[172], each with their own advantages and disadvantages. We consider the first two methods due to their simpler nature which is appropriate for the micro size problems.

PowerSet & OvA HD involve learning multiple binary classifiers, and hence, share a common implementation strategy. The difference between the two approaches lies in the way the class hypervectors are set up. We begin by encoding each instance in the dataset into a symbolic hypervector using Random Projection Encoding [194] as explained in Section 1.1.1. We then perform one-shot learning, which involves learning the centroid hypervectors, followed by iterative fine-tuning, as detailed in Section 1.1.1. The specific differences between the powerset

and OvA approaches in this implementation are explained below.

4.3.1 PowerSet HD

The label powerset transformation method defines each unique combination of labels as a distinct class, represented by a binary class vector. This makes it possible to use standard single-label classification algorithms to train models on multi-label data. Formally, given an instance $\mathbf{x}$ with L possible class labels, this method creates a new binary class vector $\mathbf{y}$ of length 2^L, representing all possible label combinations. For each unique combination of labels C_j, a binary label is assigned based on whether the combination is a subset of the original class labels of the instance $\mathbf{x}$, as shown in Equation 1:

$$\mathbf{y}_i = \begin{cases} 1 & \text{if } S_j \subseteq C_i \\ 0 & \text{otherwise} \end{cases} \tag{4.1}$$

where $C_j \subseteq \mathbf{y}$ indicates that the class combination C_j is a subset of the original class labels of the instance $\mathbf{x}$. For example, if an instance has three possible class labels A, B, and C, then there are $2^3 = 8$ possible combinations of labels: $\{\emptyset, A, B, C, AB, AC, BC, ABC\}$.

While the label powerset method is simple and easy to understand, it suffers from the issue of class imbalance and scaling with the number of class labels, making it less practical for problems with large numbers of class labels. Nevertheless, it is still widely used as a baseline method for evaluating the performance of other more advanced multi-label learning methods.

To implement power set transformation with HDC, We create a centroid hypervector for every label combination resulting in 2^L centroid hypervectors. Retraining is done on each centroid hypervector individually as an independent binary classifier. During inference, we encode the test instance and compare it with each of the centroid hypervectors using a similarity check function. The closest centroid indicates the relevant label combination.

Compute Realization Cost of PowerSetHD: To estimate the storage size of the HDC

model, we need to calculate the total number of hypervectors required to represent all possible label combinations, and then multiply that by the size of each hypervector in bits. The number of possible label combinations for a dataset with L labels is 2^{2L}, since each label can either be present or absent in a given combination. Let's consider the case of Delicious dataset where number of labels is $L = 983$, then the number of possible label combinations is $2^9 83$. To represent each hypervector as a 16-bit integer, we need 16 bits or 2 bytes per element. Since each hypervector has 1024 elements, the size of each hypervector in bytes is $2 \times 1024 = 2048$ bytes. Multiplying the number of hypervectors by the size of each hypervector gives us the total storage size required for the HDC model: $2^{983} \times 2048$ bytes/hypervector, which is equivalent to 1.4×10^{269}Terabytes. This is an enormous amount of storage, far beyond what is currently feasible with modern computing technology. It highlights the scalability issues of the label powerset method, which becomes impractical for problems with large numbers of class labels. In addition to this high RAM requirements, to get the full ranking we would have to evaluate 2^{983} classifiers which would take many CPU cycles for even a single data point.

4.3.2 One-vs-All (OvA) HD

One-vs-all is a problem transformation method used in multi-label classification where the problem is transformed into multiple binary classification problems. In this method, a separate binary classifier is trained for each label, where each classifier predicts whether the instance belongs to the corresponding label or not. Formally, given an instance $\mathbf{x}$ with L possible class labels, the one-vs-all method creates L separate binary class vectors $\mathbf{y}_1, \mathbf{y}_2, ..., \mathbf{y}_L$ of length 2 that represent the presence or absence of each class label. For each binary classification problem i, a binary label is assigned as follows:

$$\mathbf{y}_i = \begin{cases} 1 & \text{if } y_j = i \\ 0 & \text{otherwise} \end{cases} \tag{4.2}$$

where y_j is the label vector of the instance $\mathbf{x}$. Given an instance $\mathbf{x}$ with L possible class labels, the OVA method creates L binary class vectors $\mathbf{y}_1, \mathbf{y}_2, ..., \mathbf{y}_L$, where $\mathbf{y}_i$ indicates whether the instance belongs to the i^{th} class or not. The i^{th} classifier is trained using the binary class vector $\mathbf{y}_i$ as the target variable, and the output of the i^{th} classifier is interpreted as the probability of the instance belonging to the i^{th} class.

The one-vs-all method is computationally efficient and scales well with the number of class labels, making it suitable for larger-scale multi-label classification problems. However, it suffers from the issue of label correlation as it treats each label independently, ignoring any correlations that may exist between them.

OvA HD approach involves the creation of two centroid hypervectors for each label, resulting in $2L$ labels. The two hypervectors for each class denote the positive and negative associations of that label. Together, the pair of hypervectors represent the binary classifier for a single label. During inference, we encode the test instance and evaluate it using our L binary classifiers, each of which predicts the relevance of its corresponding label. The predictions of all classifiers are then combined to give the final inferred label vector.

Compute Realization Cost of OvA HD: For the OvA HD approach, we need to create two centroid hypervectors for each label, resulting in $2L$ labels. For the example of Delicious dataset, there are 983 labels, so we need to create 1966 hypervectors in total. For a hypervector dimensionality of 1024, each hypervector will require 256 bytes of storage. Therefore, the total storage required for loading the HDC model can be calculated as 1966 hypervectors $\times$256 bytes which is 491.5 kilobytes, which is significantly smaller than PowerSetHD.

The complexity analysis for classifying a single data point using HDC depends on the number of labels and the dimensionality of the hypervectors. Since we are using the OvA HD approach with 983 labels and a hypervector dimensionality of 1024, the time complexity for classifying a single data point can be expressed as $O(LD)$, where L is the number of labels and D is the hypervector dimensionality. In practice, the complexity may be higher due to the need to compute distances between the test instance and all hypervectors, as well as the need to combine

the predictions of all binary classifiers. However, the OvA HD approach is computationally efficient and scales well with the number of class labels, making it more suitable for larger-scale multi-label classification problems compared to PowerSet HD.

4.4 TinyXML HD: Extreme Multi-Label Classification

Extreme multi-label classification (XMLC) [18, 51] represents a challenging variant of multi-label classification, where the task involves predicting a large number of labels for each instance in a dataset. The scale of the label space in XMLC can range from thousands to millions, making it extremely challenging for traditional multi-label classifiers to handle efficiently. This presents significant scalability and computational challenges, particularly compared to small multi-label classification problems, such as those described in the previous section, where the label set is relatively small. One specific variant of XMLC that has gained traction in various real-world applications, such as text categorization [201] and recommendation systems [180], is Extreme Multi-Label Text Classification (XMTC) [164]. The goal of XMTC is to classify documents into a potentially large number of labels. In the rest of this paper, we describe TinyXML HD, which solves XMTC problem by leveraging hyperdimensional representations.

4.4.1 Hypervectors for Textual Data

The XMTC datasets offer text data in two forms: bag-of-words representation or raw-text. The bag-of-words (BoW) is a widely used text representation approach in natural language processing (NLP) that represents a document as a collection of words with the frequency of their occurrences, disregarding the order of the words. In our TinyXML HD, if raw-text data is available, we leverage the Word2Vec [142, 143] embeddings for representing text; otherwise, we use bag-of-words. **TinyXML HD BoW encoding** projects BoW feature vector into a hypervector using Random Projection Encoding [194], as described in Section 1.1.1.

Raw text data poses a challenge. A simple and meaningful strategy is to consider the

compositional distributional semantics approach. Compositional distributional semantics is a method of representing the meaning of a sentence as a function of the meanings of its constituent words. This approach is based on the distributional hypothesis, which posits that words that appear in similar contexts tend to have similar meanings [64]. Given a sentence S consisting of n words, represented as d-dimensional vectors $w_1, w_2, ..., w_n$, we can combine these vectors using a composition function f to obtain a sentence vector s:

$$s = f(w_1, w_2, ..., w_n) \tag{4.3}$$

The composition function f takes the word vectors as input and returns a single vector representing the meaning of the sentence. There are various ways to define the composition function, such as averaging the word vectors and concatenating them[144, 184].

One approach for representing a sentence as a composition of words is to assign random symbolic hypervectors to each word in the dataset and then use compositional distributional semantics to obtain a sentence vector. Previous studies [90, 54] have employed this approach with varying degrees of success in various NLP tasks. However, a key issue with this approach is that it ignores the structural relationships between words. Models like word2vec [142, 143] address such issues by generating vector representations that capture semantic relationships between words in a meaningful way.

Word2Vec embeddings for TinyXML HD: We leverage Word2Vec with Hyperdimensional encoding for learning in TinyXML HD. Word2Vec is a powerful method for creating distributed vector representations of words that capture semantic and syntactic aspects of natural language processing tasks [142, 143]. Traditional approaches to representing words rely heavily on sparse one-hot vector representations, which are high-dimensional and lack the ability to capture the subtle nuances of word meanings. In contrast, Word2Vec's distributed vector representations encode semantic relationships between words by placing words with similar meanings closer together in the vector space [142, 143].

To encode an instance in TinyXML HD, we first obtain the Word2Vec embeddings of all its constituent words. We then project these embeddings into hypervector using Random Projection encoding described in Section 1.1.1. Finally, to employ compositional distributional semantics, we *bundle* ($\oplus$) the resultant hypervectors into a single hypervector that represents an instance. As mentioned in Section 1.1.1, Random Projection Encoding preserves the euclidean distance, so this enables the generation of symbolic hypervectors that capture semantic information between words through their cosine similarity scores, resulting in expressive high-dimensional representations. Consequently, hypervectors for words that are similar will be proportionally similar and those of semantically dissimilar words would be dissimilar. In this way we are able to capture the complex relationships between words and obtain a richer representation that conveys more information about their semantic context.

TinyXML HD Label Representation: We leverage HDC algebra to map and combine multiple labels in hyperdimensional space. Let L be the number of labels or symbols in the dataset, and $\mathscr{H}^D$ be a D-dimensional hyperdimensional space with $H = \{+1, -1\}$ for the MAP HDC model. We map each label to a hypervector in this space. The initialization of the label space $\mathscr{Y}_{1...L}$ involves assigning a random hypervector from a Binomial distribution to each label. Specifically, we initialize each label $\mathscr{Y}_i$ by sampling from $\mathscr{B}(0.5) \cdot 2 - 1$. To obtain a high-dimensional representation of a label, we bundle the corresponding hypervectors of the labels present for an instance, denoted by $\mathscr{Y}^p$. We then combine these hypervectors using the hyperdimensional operator $\oplus$ to obtain a single hypervector representation for the instance $\mathbf{x}_i$, given by Eq. (4.4.1). This operator is commutative, which allows us to bundle the hypervectors in any order without affecting the final result. $\mathbf{y}_i = \bigoplus_{j \in \mathscr{Y}^p} \mathscr{Y}_j$

4.4.2 Learning with TinyXML HD

With both the inputs and outputs embedded in the same high-dimensional space, the next step is to learn a mapping $f : \mathbf{x_i} \in \mathscr{H} \rightarrow \mathbf{y_i} \in \mathscr{H}$, where $\mathbf{x_i}$ is the input hypervector and f outputs the hypervector that represents all the labels present for that instance. In this section, we present

our proposed neural network based approach to learn this mapping function f. Our proposed approach presents a linear formulation over the HDC operators of *binding* and *bundling*, which allows for effective and efficient optimization using gradient-based methods.

Objective formulation: We decompose the original learning problem into multiple sub-problems to enhance its learnability. Let's consider an example where we break down the learning problem into two sub-problems, which we rewrite as

$$f : \mathbf{H_1} \in \mathscr{H}^D \rightarrow \mathbf{H_2} \in \mathscr{H}^D \tag{4.4}$$

$$f = f_1(f_2) \tag{4.5}$$

$$f_1 : \mathbf{H_1} \in \mathscr{H}^D \rightarrow \mathbf{H}_x \in \mathscr{H}^D \tag{4.6}$$

$$f_2 : \mathbf{H}_x \in \mathscr{H}^D \rightarrow \mathbf{H_2} \in \mathscr{H}^D \tag{4.7}$$

where f_1 maps the input instance to an intermediate hypervector $\mathbf{H}_x$, and f_2 maps $\mathbf{H}_x$ to the output label hypervector $\mathbf{H_2}$.

We define f_1 and f_2 using the HDC arithmetic operations of *binding* and *bundling*. In particular, we parameterize f_1 as

$$f_1 = \mathbf{H_1} \otimes H_{\text{conv1}} \tag{4.8}$$

where H_{conv} is a hypervector to be learned. Similarly, we define f_2 as

$$f_2 = \mathbf{H_x} \otimes H_{\text{conv2}} \tag{4.9}$$

This approach considers the mapping between two hypervectors as a series of geometric transformations where the input hypervector is bound sequentially with the intermediate hypervectors induced by the sub-problems. The hyperparameters of the number of sub-problems to induce and the dimensions of the learned hypervectors are chosen based on the complexity of the dataset.

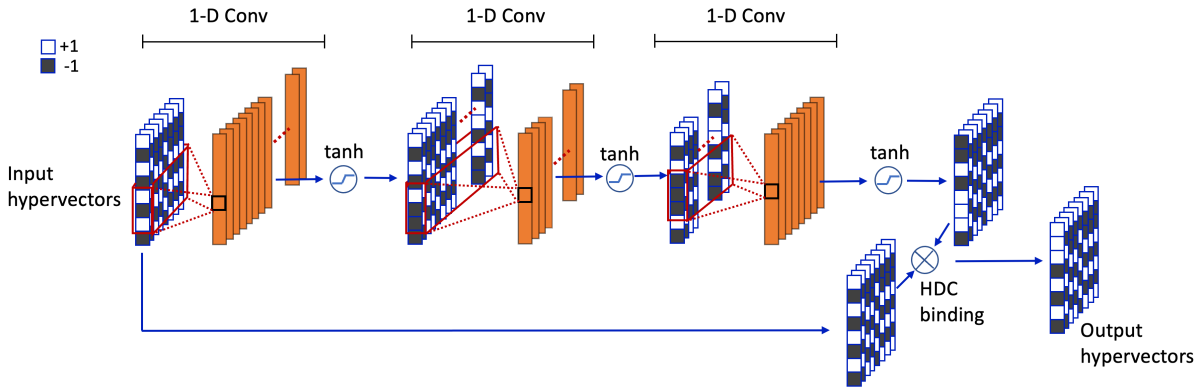


Figure 4.1. ConvHD Operator

1D Convolutions as Hypervector Operators: In developing a neural architecture, it is crucial to adhere to the principles of high-dimensional computing (HDC), which dictate that representations should be distributed, with individual coordinates devoid of semantic information. Consequently, our neural architecture should interpret inputs as distributed representations rather than feature vectors containing discrete semantic entities. To achieve this, we use one-dimensional convolutional operators as hypervector operators. A one-dimensional convolution involves applying a filter across an input, using a single set of weights to process the entire hypervector. This operation treats each vector region independently with the filter, synthesizing a vector that encapsulates information from the input. In contrast, a fully connected (FC) network utilizes an interconnected network of connections to process all coordinates of the input vector, treating the coordinates as dependent entities, which contravenes the principles of HDC representations.

Figure 4.1 details the architecture and operations of our proposed ConvHD block. Our ConvHD block consists of three layers parameterized by C, which represents the expansion factor and F, the filter size. The block consists of three convolutional layers: the first layer (X_1) takes input hypervectors and generates C hypervectors, the second layer (X_2) processes these C hypervectors to produce $C/2$ hypervectors, and the third layer (X_3) combines these $C/2$ vectors

into a single hypervector. A single convolutional unit can be defined as follows:

$$g : \mathscr{H}^D \rightarrow \mathscr{H}^D \tag{4.10}$$

$$g = X_3 \tanh\left(X_2 \tanh\left(\tanh\left(X_1 H_{\text{input}}\right)\right)\right) \tag{4.11}$$

The ConvHD block is be represented by:

$$\text{ConvHD} = g(H_{\text{input}}) \otimes H_{\text{input}} \tag{4.12}$$

Hence a model f using 2 ConvHD blocks is represented by:

$$f = \mathscr{H}^D \rightarrow \mathscr{H}^D \tag{4.13}$$

$$f = \text{ConvHD}\left(\text{ConvHD}(H_{\text{input}})\right) \tag{4.14}$$

We formulate the sub-problems as a single learning problem, where we optimize the parameters X_1, X_2, X_3 using gradient-based methods. To enhance our architecture, we incorporate the idea of dilated convolutions [214] to increase the receptive field of the convolutional layers [30]. We also set the filter size F to be large, approximately a quarter of the hypervector dimensionality D. These details are crucial, as they increase the effective receptive field with every 1-D convolution operation. That is, they increase the number of hypervector coordinates in the input that influence the synthesis of a single coordinate in the output of the last convolution layer. By using a large filter size, we increase the number of coordinates in the input hypervector that are considered to produce a single coordinate in the resultant hypervector. Similarly, dilation helps to increase the receptive field by allowing deeper layers to infer coordinates based on a larger area of the input hypervector. Since the ConvHD operator uses three 1-D convolutional layers, the receptive field increases progressively with each layer looking at a larger section of the input hypervector to make a decision.

The expansion factor C spawns more sub-problems parallely. For instance, in the above example of breaking down the learning problem into 2 parts, if we set $C = 2$, then each layer estimates 2 sets of sub-problems. The first layer will parallely solve two sub-problems similar to Equation 8 and similarly the second layer will solve two sub-problems similar to Equation 9. The results of the 2 sub-problems will then be combined through the bundling operation.

In order to learn the mapping to solve these sub-problems, we use the loss function detailed in [49], which aims to minimize the cosine distance between the predicted hypervector and the ground-truth label hypervector.

4.5 Evaluation of OvA, PowerSet & TinyXML HD

This section of our research paper presents the results of our proposed multi-label classification approach on various real-world datasets. Through a series of experiments, we demonstrate the trade-off between compute efficiency and accuracy of our approach across a range of complexity levels, from small-scale (less than 20 labels) to extreme-scale (greater than 5000 labels). Our findings indicate that, in low-complexity scenarios with datasets of low cardinality, the One-vs-All HDC approach achieves high accuracy and efficiency. Conversely, the PowerSet HDC approach provides poor trade-offs, yielding benefits only when the label cardinality is very low, with efficiency degrading exponentially as complexity increases.

We evaluate the effectiveness of our proposed approach, TinyXML HD, on extreme size datasets. Our experiments demonstrate that TinyXML HD produces models that are 231x-836x smaller than state-of-the-art models while still achieving reasonable accuracy. Furthermore, our approach can efficiently train on large text datasets in just a few hours providing a speed up of up to 16x. These results highlight the potential of our proposed approach for solving extreme-scale multi-label classification problems while greatly reducing the computational resources required.

We evaluate the small scale problems on an Intel Xeon 24-core CPU while for the larger datasets we use a single Nvidia V100 GPU.

4.5.1 Evaluation of OvA & PowerSet HD

OvA and PowerSet HD Experimental Setup: We tested our OvA HD and PowerSet HD multi-label methods on smaller size datasets by running on an optimized C++ implementation on an Intel Xeon 24-core CPU. We compare our HDC-based methods with multi-label versions of k-nearest neighbors (kNN) [217], Sequential Minimal Optimization – SMO[161], C4.5[165], and Naive Bayes – NB[217], all of which are appropriate for smaller datasets. We utilized Java-based open-source Mulan [200] multi-label package with 3 small datasets for comparison:
Genbase [135] contains protein classes of 27 most important protein families, with 662 samples, each with 1186 attributes.
Scene [113] contains images with their characteristics and classes. One image can belong to up to 6 categories. It has 2407 samples, each with 294 attributes.
Yeast [43] has information about a set of yeast cells. The task is to determine the localization site of each cell amongst 14 possible sites. It has 2417 samples, each with 103 attributes.

OvA and PowerSet HD Accuracy: Figure 4.2 shows that OvA and PowerSet HD achieve comparable accuracy to state-of-the-art multi-label classifiers. PowerSet HD consistently outperforms state-of-the-art methods on all three datasets. OvA HD is slightly less accurate on the Genbase dataset but performs better on the Scene and Yeast datasets, likely due to their better separability of HD space compared to low-dimensional space.

OvA and PowerSet HD Performance & Efficiency: While PowerSet HD achieves higher accuracy, Figure 4.2 demonstrates that this comes at a significant cost in terms of execution time. This is due to the exponential increase in class hypervectors as discussed earlier. Figure 4.2 also shows that both OvA and PowerSet HD training are significantly faster than most other multi-label classifiers, with OvA HD being 60.8 times faster on average. PowerSet HD is only 3.5 times slower than OvA HD on datasets with a large portion of label combinations. Power Set HD is 24 times faster than state-of-the-art multi-label classifiers on average, or approximately two times slower than OvA HD, but offers 13% higher accuracy. For small datasets, where only

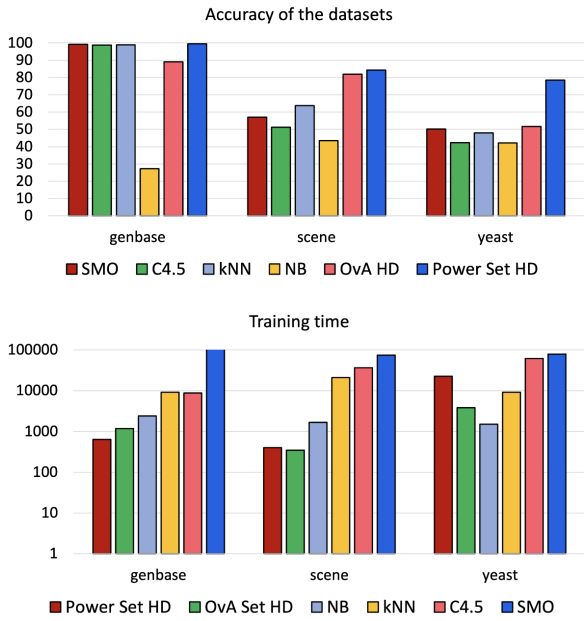


Figure 4.2. Efficiency of training

a small subset of possible label combinations appear in the dataset, PowerSet HD can potentially be more efficient and accurate. However, for datasets with more number of possible label combinations, OvA HD is the clear choice as it offers a trade-off between compute efficiency and accuracy compared to PowerSet HD. These results indicate that the OvA HD approach is an ideal candidate for small scale multi-label classification tasks.

4.5.2 Experimental setup for TinyXML HD

We evaluated TinyXML HD HD on real-world, large-scale datasets from Extreme Multi-Label Text Classification (XMTC). Our objective is to maximize the compute efficiency of learning while achieving comparable precision to the state-of-the-art. For the XMTC dataset, we evaluate our proposed TinyXML HD on Nvidia V100 GPU.

Evaluation metrics: We consider Precision@k with $k = 1, 3, 5$ as our metric for evaluating the performance of TinyXML HD on multi-label classification, where k represents the top k predictions. This is a widely accepted and used evaluation metric by other works in literature [24, 84]. In addition, we evaluate the computational efficiency of TinyXML HD against the

Table 4.1. Dataset Metadata

Dataset	Feature	# Labels	# Train	# Test	Avg. Points per Label
Mediamill [183]	BoW	101	30,993	12,914	1902.15
Bibtex [97]	BoW	159	4,880	2,515	111.71
Delicious [199]	BoW	983	12,920	3,185	311.61
Eurlex-4K [118]	Text	3,993	5,000	3,993	25.73
Wiki10-31K [227]	BoW & Text	30,938	14,146	6,616	8.52
Amazon-13k [137]	Text	13,330	1,186,239	306,782	448.57

following start-of-the-art models: XT [4], Bonsai [25], SLEEC [17] and Parabel [221] for BoW datasets. For Raw text datasets, we consider these SoA models: AttentionXML [213], LightXML [84] and X-Transformer [24]. Given that previous research has not given a comprehensive account of the compute cost associated with these models, it is difficult to establish a standardized metric for comparison. To address this issue, we have considered two distinct metrics: the count of trainable parameters and the training time. The former serves as an indicator of the cost of training, since a model with a higher parameter count requires more gradients to be calculated and optimized, and is also indicative of greater model size. The latter is a direct measure of the time required to train the model. These two metrics offer a meaningful evaluation of compute cost in the context of real-world applications.

Datasets: In order to evaluate the expressiveness of our high-dimensional representations of text data, we select six datasets from Extreme Multi-Label Text Classification (XMTC) dataset, a widely accepted benchmark in literature [164, 49, 117, 18]. The datasets are described in Table 4.1. In addition to the scalability and computational challenges, the XMTC dataset poses an additional challenge which is the label sparsity issue. Bhatia et al [18] divided the datasets according to the number of labels per sample into small scale and large scale. Small scale datasets contain at most 5000 labels. Although pre-processed BoW features are available for all datasets, the original text is not. Consequently, we use the original text when available and BoW for all others.

TinyXML HD HD Architecture Specifics: We use Random Projection Encoding

as discussed in Sec. 1.1.1 for BoW feature representations, while for raw text datasets, we utilize the combination of Random Projection Encoding with Word2Vec as described in Sec. 4.4. We employ ConvHD blocks with expansion factor $C = 128$, filter size $F = 255$ with dilation set to 7 and a hypervector dimensionality of 1024. To optimize the model, we use the loss function proposed by Ganesan et al. [49] but we remove the negative loss component, which was intended to ensure that the output hypervector from the model $f(.)$ is orthogonal to the labels that are not present for that instance. Since all labels are initialized with random hypervectors that are orthogonal to each other, enforcing the similarity to the present labels alone will automatically satisfy the orthogonality condition with the labels not present. Therefore, we only retain the positive component in the loss function, and we discard the additional positive $\mathbf{p}$ vector used in [49] as it does not improve results. The final loss function is as follows: $\mathscr{L} = \sum_{c^p \in \mathscr{Y}^p} (1 - \cos(\mathbf{y_i}, c^p))$ where $\mathbf{y_i}$ is the final hypervector output by our model $f(\mathbf{x_i})$ for the i-th instance, and c^p represents a present label. The loss function aims to minimize the cosine distance between $\mathbf{y_i}$ and all present labels, thereby encouraging the model to produce a hypervector that is more similar to the labels that are present in the instance.

Comparison baselines: Our evaluation comprises two parts, with datasets divided by the type of features used. We consider different baselines for each part. For BoW datasets, we benchmark against other state-of-the-art models that use the same features, such as Bonsai [25], Parabel [221], and PFastreXML [87]. For raw-text datasets, we compare TinyXML HD's performance against state-of-the-art deep learning approaches, including AttentionXML [213], X-Transformer [24], and Light-XML [84]. These deep learning models employ powerful architectures like transformers, with hundreds of millions of parameters, enabling them to extract highly expressive embeddings from text data. As a result, TinyXML HD is inherently disadvantaged due to the significant disparity in parameter count. The primary objective of this research is to optimize size, speed and accuracy tradeoffs of such constrained HDC models to evaluate their viability as a lightweight paradigm. Hence, we aim to achieve reasonable accuracy with respect to the state-of-the-art, within a 10% margin.

Table 4.2. TinyXML HD on Small Scale Multi-Label Classification (normalized to TinyXML HD)

Model	Genbase [135]			Scene [113]		
	Acc	# params	Train time	Acc	# params	Train time
TinyXML HD	**100**	**1**	1	76.0	1	1
PowerSet HD	99.5	30.4K	**0.54**	**84.2**	1.99	0.14
OvA HD	89.1	1.677	0.6	81.9	**0.03**	**0.12**

4.5.3 Evaluation of TinyXML HD HD

TinyXML HD, PowerSet & OvA HD Comparison: To gain insight into the trade-off between performance and accuracy, we have evaluated TinyXML HD on small-scale multi-label classification tasks. In this study, we compare the performance of TinyXML HD to that of PowerSet and OvA HD on three small datasets, as described in Section 4.5.1. Given the lower complexity of the task at hand, we have scaled down TinyXML HD by utilizing a depth of 1, a block size of 8, and a filter size of 255. As the datasets used have low cardinality, we have evaluated our approaches using overall accuracy since the precision@K metrics are inapplicable for $K = 3,5$, due to the limited number of labels per instance. In addition, considering the low complexity of the task, we evaluate performance only on CPU and do not use any specialized hardware for acceleration.

Our results in Table 4.2 show that TinyXML HD gets 100% accuracy on Genbase[135]. The performance drops by 8% on Scene[113] and 3% on Yeast[43]. The most likely reason for the lower accuracy on the two datasets is the scarcity of training data. Problem transformation techniques were trained faster than TinyXML HD, despite the latter's smaller size. The only exception to this was the Yeast[43] dataset, on which TinyXML HD was significantly faster (1.2x over OvA HD and 7.4x over Power Set HD). This is due to the disparity in label cardinality across the datasets. Genbase[135] and Scene[113] have label cardinalities of 1.25 and 1.07, respectively, meaning that only a single centroid vector needs to be updated for Power Set HD and OvA HD. However, the Yeast[43] dataset has a label cardinality of 4.2, requiring OvA

HD to update 4 centroid hypervectors while Power Set HD needs to update $2^4 = 16$ centroid hypervectors for each instance, resulting in increased training time.

The higher training time of TinyXML HD can be attributed to the convolution operations, which are computationally intensive compared to the HDC operations of bundling, binding, and similarity check used by the transformation methods. These operations reduce to simple element-wise additions and multiplications, making them easier to compute. Consequently, Power Set HD and OvA HD can be easily parallelized and accelerated in hardware, while the convolution operation of TinyXML HD would be harder to accelerate.

The dissimilarity in parameter count between the models is attributed to their respective architectures. The parameter count for PowerSet HD increases exponentially with the size of the label set. Conversely, the parameter count for the OvA HD scales linearly with the label set size, resulting in a parameter count that is twice the size of the label set for our implementation of the one-vs-all classifier. In contrast, TinyXML HD leverages only one hypervector to represent each label, with the additional parameters solely corresponding to convolution filters. These parameters are minimal in comparison to the label set size, further underscoring the efficiency of the TinyXML HD architecture.

The current study has revealed that the HDC-based problem transformation approaches offer a significantly superior trade-off between training time and accuracy for small-scale multi-label classification tasks compared to TinyXML HD. Specifically, for datasets where only a limited subset of possible label combinations appear in the dataset, PowerSet HD exhibits the potential to be both more efficient and accurate. In contrast, for datasets with a larger number of possible label combinations, albeit less than at the extreme scale, OvA HD proves to be a more promising candidate. While TinyXML HD boasts a smaller parameter count, the parameter count of OvA HD remains comparable and is sufficiently small for the complexity scale under investigation.

Due to linear and exponential scaling of PowerSet HD and OvA HD, these methods are unsuitable for extreme-scale multi-label classification tasks. PowerSet HD is too large

Table 4.3. Multi-Label Classification Performance on BoW datasets: Comparison with State-of-the-Art

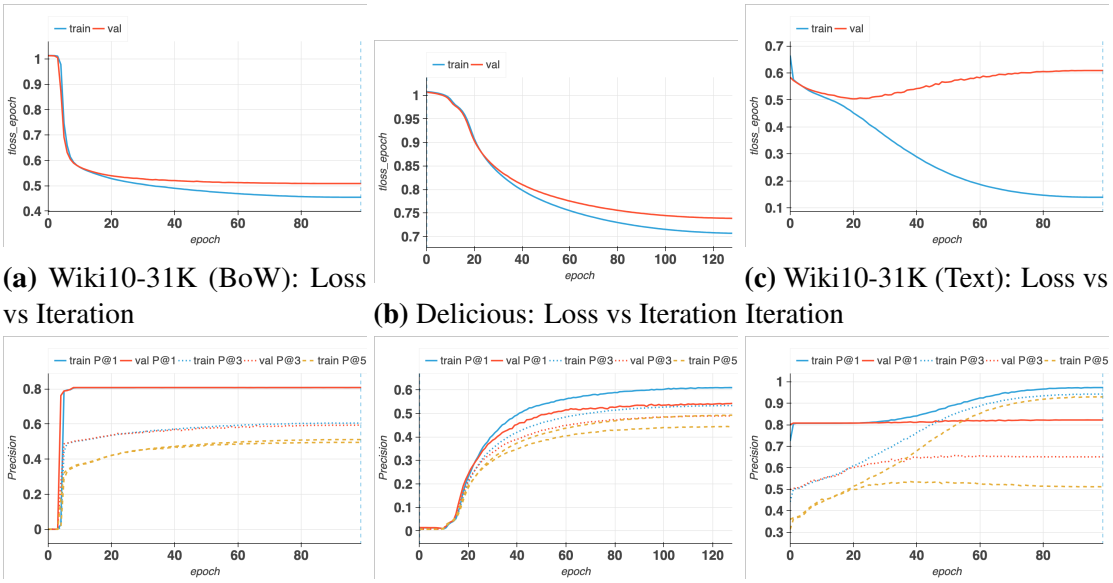
Dataset		Ours	FastXML [59]	PfastreXML [28]	SLEEC [17]
Medialmill [183]	p@1	82.1	83.5	84.2	84.0
	p@3	64.4	65.7	67.3	67.2
	p@5	50.0	49.9	53.0	52.8
Delicious [199]	p@1	62.7	69.6	67.1	67.5
	p@3	55.7	64.1	62.3	61.3
	p@5	51.4	59.2	58.6	56.5
Wiki10-31K [227]	p@1	80.8	83.0	83.5	85.8
	p@3	50.5	67.47	68.6	72.9
	p@5	44.3	57.7	59.1	62.7

Table 4.4. Multi-Label Classificaiton Performance on Real Text Datasets: Comparison with State-of-the-art

Dataset		TinyXML HD	AttentioinXML [213]	XTransformer [63]
Eurlex-4K [118]	p@1	61.3	87.1	87.2
	p@3	51.8	73.9	75.1
	p@5	43.7	61.9	62.9
Wiki10-31K [227]	p@1	83.3	87.4	88.5
	p@3	66.2	78.4	78.7
	p@5	60.7	69.3	69.6
Amazon-13K [137]	p@1	86.2	95.9	96.7
	p@3	60.4	82.4	83.8
	p@5	44.6	67.3	68.5

to implement, while OvA HD takes too long to train. We next evaluate TinyXML HD on extreme-scale multi-label classification.

TinyXML HD Multi-label Accuracy: We investigate the performance of TinyXML HD on extreme scale datasets next. Table 4.3 presents the performance of TinyXML HD along with its respective baselines on the BoW dataset. Our findings reveal that for Mediamill and Wiki10-31K BoW, TinyXML HD's precision at top one (p@1) is within 5% of the state-of-the-art (SoA). However, we note that barring Mediamill, precision at top three (p@3) and precision at top five (p@5) is lower across all datasets when compared to the SoA.



(a) Wiki10-31K (BoW): Loss vs Iteration

(b) Delicious: Loss vs Iteration

(c) Wiki10-31K (Text): Loss vs Iteration

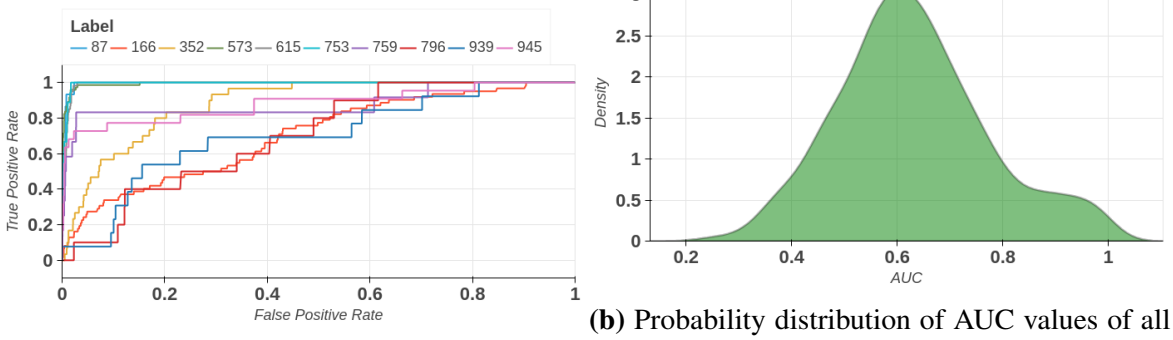
(d) Wiki10-31K (BoW): Precision vs Iteration

(e) Delicious: Precision vs Iteration

(f) Wiki10-31K (Text): Precision vs Iteration

Figure 4.3. Loss and Precision vs Iteration for Three Datasets

Table 4.4 shows the performance of TinyXML HD and its respective baselines on the raw text datasets. Our findings reveal that TinyXML HD's precision is relatively lower for these datasets. While we observe that TinyXML HD achieves comparable performance on the Wiki10-31K dataset, the precision drops for Amazon-13k by 9%. As we attempt to retrieve more labels from the hypervector, the retrieval becomes less robust. Considering that Wiki10-31K has 31,000 labels with only 8 samples per label available, the performance of TinyXML HD (83%) is remarkable. While the performance of TinyXML HD is not extraordinary compared to the state-of-the-art, it is remarkable considering the fact that TinyXML HD relies on a simple encoding scheme. In contrast, the state-of-the-art models employ highly complex architectures with millions of parameters. This observation validates the potential of HDC to provide expressive representations of data at extraordinarily low compute costs. Moreover, unlike other models where the model size scales almost linearly with label size, TinyXML HD ensures that the output size of the model is fixed to the dimensionality of the hypervector D independent of label set size.



(a) ROC Curves for 10 different labels

(b) Probability distribution of AUC values of all Delicious labels

Figure 4.4. (Left) ROC plots for 10 different labels (Right) Density of AUC values across all labels from the Delicious dataset considering each label as a binary classification problem

TinyXML HD Convergence: Figure 4.3 showcases the convergence plots of TinyXML HD on three distinct datasets: Wiki10-31K (BoW) [227], Delicious [199], and Wiki10-31K (Text) [227]. When examining the BoW datasets, namely Wiki10-31K [227] and Delicious [199], a notable trend emerges. The loss function exhibits a smooth decrease, punctuated by a slight, yet discernible, initial drop for Wiki10-31K. In stark contrast, the Wiki10-31K (Text) variant converges in fewer than 30 epochs and seemingly starts to overfit, but, the precision plots provide further insights into this phenomenon. While P@1 and P@3 seem to have converged, a closer analysis reveals that P@5 continues to improve. This observation suggests that the model is still assimilating new information from the data. Although unable to enhance P@1 and P@3 further, the model's focus shifts to effectively ranking two additional labels.

TinyXML HD ROC and AUC For the readers' benefit, we also provide additional insights in the form of Receiver Operator Characteristics (ROC) Curves in Fig. 4.4a and the corresponding Area Under the Curve (AUC) values as shown in Fig.4.4b. Although these metrics are typically employed for evaluating multi-class classifiers, considering mutually independent labels, we adapt them to our multi-label setting by treating each label as an independent binary classification problem. Due to computational constraints, we focus on the Delicious dataset for these metrics, as it contains a large number of labels. To ensure plot coherence, we present the

ROC curve for 10 randomly selected labels. Furthermore, in order to comprehend the AUC, we visualize the distribution of AUC values obtained for each label across the dataset's 983 labels. These results show expected accuracy when treating multi-label problem as an independent binary classification problem.

TinyXML HD Overfitting To address the issue of overfitting, we explored the utilization of BatchNorm-2D [82], L1/L2 Regularization, and Dropout techniques [185]. However, neither of these approaches proved to be effective. Additionally, we conducted experiments to further reduce the parameter count, but since the model already consisted of only 2.5M parameters, any additional reduction led to a decrease in accuracy.

TinyXML HD Robustness To briefly examine the ability of the method to perform when few labels are missing, for every sample we dropped one label with probability p. Our experiments show that up to $p = 0.2$ there is no accuracy degradation, showing robustness of TinyXML HD.

TinyXML HD Computing Efficiency: We compare the efficiency of TinyXML HD to the following state-of-the-art models: XT [4], Bonsai [25], SLEEC [17] and Parabel [221]. Table 4.6 compares the training time and model size of TinyXML HD against the state-of-the art listed above on the Wiki10-31K BoW dataset. Remarkably, TinyXML HD trains in 10 mins with a minuscule model size of 19.8MB while achieving comparable precision on the dataset. We observe that TinyXML HD is 6.5x smaller than the smallest SoA (Bonsai [25]) and 56x smaller than the largest SoA (SLEEC [17]). Methods such as Parabel [221], Bonsai [25] build multiple probabilistic label trees and perform classification on each node which becomes computationally expensive very quickly. Consequently, TinyXML HD is 1.25x quicker than the fastest SoA (Parabel [221]) and 4x quicker than the slowest SoA (Bonsai [25]).

Raw text training time and the number of parameters needed for Amazon-670K dataset is shown in Table 4.5. All the deep learning models necessitate several days to train on this extensive dataset. In stark contrast, TinyXML HD showcases a remarkable training speed of merely six hours, even though the dataset has over 670K labels and 130K training samples.

Table 4.5. Raw text model size & training time

Model	Model size	Training time
AttentionXML [213]	16.56 GB	26.30 hrs
Xtransformer [24]	>5GB	>35 hrs [84]
LightXML [84]	4.59GB	28.75 hrs
TinyXML HD	**19.8MB**	**6 hours**

TinyXML HD provides a speedup of 4x over the fastest SoA (AttentionXML [213]) and 16x over the slowest SoA (X-Transformer [24]) This exceptional speedup can be attributed to two crucial factors. First, the deep learning models rely on complex transformer models like BERT and RoBERTa, to extract highly expressive feature embeddings from data. In contrast, TinyXML HD employs a simple encoding scheme, that decomposes into highly parallelizeable operations. The bulky feature extractor is replaced by our lightweight HDC-based encoding, which demonstrates the expressiveness of these representations when used to encode relevant features. Second, the output dimensionality of deep learning models typically scales with the label set size (L). However, TinyXML HD ensures that the output size of the model is fixed to the dimensionality of the hypervector (D), where $D << L$, irrespective of the label size. This unique feature allows for the reduction of the number of trainable parameters, thereby improving training efficiency and reducing the computational load.

These results clearly demonstrate the strength of HDC when it comes to computational cost of learning. HDC has enormous potential to make learning computations tractable and to dramatically cut down on training time with good accuracy. TinyXML HD is 836x smaller than the largest SoA (AttentionXML [213]) and 231x smaller than the smallest SoA (LightXML [84]). Considering that X-Transformer [24] uses an ensemble of transformers we suspect that X-Transformer would be larger than 5GB and would require 100 hours of effort to train [84] making it infeasible to compare with.

Table 4.6. BoW model compute efficiency

Model	Train time	Size
Parabel [221]	0.20 hrs	180MB
SLEEC [17]	0.21 hrs	1.13GB
Bonsai [25]	0.64 hrs	130MB
XT [4]	0.39 hrs	370MB
TinyXML HD	**0.16hrs**	**19.8MB**

4.6 Conclusion

In this work, we presented novel approaches to Multi-Label classification using Hyperdimensional Computing (HDC), addressing the full complexity spectrum. For small-scale Multi-Label classification, we proposed using HDC for two problem transformation methods: PowerSet and One-vs-All transforms. Rigorous evaluation showed that OvA HD provides up to 60x speedup in low cardinality datasets, while PowerSet HD is up to 24x faster than SoA with comparable accuracy, especially in low cardinality scenarios. For extreme multi-label classification, where label size is very large, we introduced TinyXML HD, a neuro-symbolic approach that breaks down learning into sub-problems using hyperdimensional arithmetic and optimizes them with gradient methods. TinyXML HD significantly reduces the computational complexity of multi-label learning on large-scale datasets while maintaining good accuracy. TinyXML is 836x smaller than the largest SoA and 231x smaller than the smallest for text datasets, and up to 16x faster to train. For BoW datasets, TinyXML is 6.5x - 56x smaller than SoA models while training up to 4x faster.

Next, we look at another common problem in IoT – image classification. Examples include industrial monitoring, surveillance, security systems, autonomous driving etc. We look at incorporating HDC into image classification architectures for improving the efficiency of classification. We propose a principled approach to structure the classification problem as a hierarchical problem to exploit the symbolic manipulation properties offered by HDC.

4.7 Acknowledgements

Chapter 4 in full, is a reprint of the material as it appears in Multi-Label Classification With Hyperdimensional Representations. IEEE Access, Volume 11, 2023. The dissertation author was the primary investigator and author of this paper.

Chapter 5

NeuroHD: Neuro-Symbolic Classification with Hyperdimensional Computing

5.1 Introduction

Machine Learning (ML) has undeniably revolutionized the Internet of Things (IoT), enabling a myriad of applications across diverse domains. A critical bottleneck in realizing the full potential of ML within IoT lies in the constrained computational budget of IoT devices. Traditionally, these devices rely on centralized cloud computing for ML tasks, incurring significant bandwidth usage, latency issues—critical in applications such as autonomous driving—and raising privacy concerns, notably in smart healthcare solutions [2, 8]. This backdrop has catalyzed a shift towards on-device learning, a paradigm that promises enhanced privacy, reduced latency, and decreased bandwidth consumption.

Hyperdimensional Computing (HDC) emerges as a symbolic computing framework characterized by its lightweight and efficient computational footprint, making it an attractive candidate for on-device learning. Despite these advantages, HDC's applicability to complex tasks, such as image classification, remains limited, often lagging behind more conventional methods in accuracy [92]. In contrast, Deep Learning (DL) boasts exemplary performance across a spectrum of ML tasks, including those with high complexity. However, the computational intensity of DL models often renders them impractical for deployment on resource-constrained IoT devices [115].

The dichotomy between symbolic methods like HDC and connectionist approaches such as DL has traditionally been viewed through a lens of mutual exclusivity. Nonetheless, an emerging body of research, inspired by cognitive science theories suggesting that human cognition employs a blend of both symbolic and connectionist mechanisms, posits that a hybrid architecture integrating these methodologies could yield superior outcomes [181, 10].

In this work, we introduce a novel neuro-symbolic architecture that harmonizes the complementary strengths of HDC and DL. Previous endeavors in this space, such as [50] and [116], have primarily utilized DL as a feature extractor, subsequently encoding these features into hyperdimensional vectors for symbolic processing [160, 86]. While these approaches significantly reduce model size and highlight the benefits of HDC-DL integration, they do not fully address the inherent limitations of connectionist models, notably the binding problem identified in DL architectures [60].

Our proposed architecture seeks to ameliorate these challenges by leveraging DL for its feature extraction capabilities, thereby augmenting HDC's performance in complex classification tasks, while simultaneously utilizing HDC to resolve the binding issue prevalent in DL models. Empirical evaluations on standard image classification benchmarks reveal that our architecture not only achieves competitive accuracy, comparable to state-of-the-art models, but does so with a significantly reduced model size, underscoring the efficacy and efficiency of our approach.

5.2 Related Works

5.2.1 Neuro-symbolic Architectures

The debate between symbolic versus connectionist models for learning has been long-standing [45]. A niche yet growing body of research advocates for a hybrid approach, suggesting that human cognition leverages a combination of these models. This insight has spurred research into neuro-symbolic architectures, finding notable success in natural language processing tasks with TPRs [182], despite their computational demands.

Holographic Reduced Representations (HRRs) have been embedded within LSTM networks [35] and have also been utilized to replace conventional convolutional operations, reducing model size and inference time [160]. However, these implementations often overlook the symbolic properties of HRRs, and their binding operations, involving the computation of the Fast Fourier Transform (FFT), are computationally intensive. Conversely, there exists limited exploration of gradient-based learning within the MAP-C HDC framework, which utilizes bi-polar hypervectors. Works such as HDnn-PIM [41] and NSHD [116] have demonstrated the integration of CNNs for feature extraction, encoding outputs into hypervectors without fully leveraging HDC's symbolic properties. More recently, [23] used CNNs to learn compositions of symbols for Multi-Label text classification.

Differing from these approaches, our work introduces a novel neuro-symbolic architecture that not only addresses the inherent feature extraction challenges of HDC but also resolves the binding issues [60] prevalent in connectionist deep learning methodologies. Our proposed framework significantly enhances model size efficiency and reduces training times, presenting a pivotal advancement in the intersection of HDC and neuro-symbolic computing.

5.3 Methodology Overview

Our methodology decomposes the traditional classification task into a two-tiered hierarchical process. Initially, we reformulate the classification problem within the HDC framework, transforming it into a task of recognizing associations among entities. This reformulation allows us to address the problem in two distinct stages. Subsequently, we employ CNNs, under the guidance of HDC framework, to extract and learn symbolic information from unstructured sensory data.

Overall, our proposed methodology reduces the gap between symbolic and connectionist models of machine learning. By thoughtfully integrating HDC and CNNs, we unveil a comprehensive approach that promises significant advancements for classification.

This section is divided into three parts: First we detail the development of a novel encoding method to map images to hypervectors, then, we explain how to construct the hierarchy and finally, we detail the architecture to orchestrate the hierarchical classification.

5.3.1 Patch Encoder

The encoding of sensory data into a meaningful representation is fundamental in computational models. Traditional methods that operate on raw pixel values often overlook the spatial distribution of information, which is pivotal for understanding complex structures within the data. To address this, we introduce a novel encoding method that not only considers the intensity values of each pixel but also their relative positions within the image, preserving essential semantic information within the high-dimensional framework of hyperdimensional computing (HDC).

Encoding Intensities

At the core of our approach is the representation of a pixel's intensity across the Red (R), Green (G), and Blue (B) channels as a vector of intensities, formally denoted as $\vec{x_{0,0}} = (r,g,b)^T$. To encode this information into the HDC space, we employ quantized Random Fourier Features (RFF)[169], transforming each pixel $\vec{x_{i,j}}$ into a high-dimensional vector space through a random projection ϕ, sampled from a normal distribution. This transformation is followed by quantization using the $sign(\cdot)$ function to generate bi-polar hypervectors.

Encoding Relative Positions

The spatial distribution of pixels plays a crucial role in our perception of images. To encode this aspect, we consider the position of pixels in an $(m \times m)$ image grid, denoted by

$$\mathbf{P} = \begin{bmatrix} (0,0) & (0,1) & \cdots & (0,m-1) \\ (1,0) & \cdot & \cdots & \cdot \\ \vdots & \vdots & \ddots & \vdots \\ (m-1,0) & \cdot & \cdots & (m-1,m-1) \end{bmatrix}. \tag{5.1}$$

To encode these positions into hypervectors, we utilize a method inspired by the fractional binding [104] of Semantic Spatial Pointers, substituting the circular convolution with a randomly sampled permutation. Recursive binding, achieved through repeated permutation of the input hypervector, allows for a nuanced representation of pixel coordinates. Specifically, for each axis $(\hat{x}, \hat{y})$ (We denote hypervectors using$\hat{}$on top), we sample a bi-polar hypervector and apply recursive permutations, the count of which is determined by the pixel's coordinates (i, j).

Formally, if $P \in \{0,1\}^{D \times D}$ represents a permutation matrix, the hypervector for coordinates $p_{i,j}$ is computed as

$$\hat{p_{i,j}} = \left(P^i \hat{x}\right) \otimes \left(P^j \hat{y}\right).$$

This process encodes each pixel's relative position through the number of permutations, associating them with the symbolic representation in HDC space.

Divide and Conquer

To further process the image, we divide it into a grid of size $K \times K$, where K is a predetermined hyperparameter. Each grid cell, or patch, covers an area of $l \times l$ pixels, with $l = \lfloor m/K \rfloor$. Within each patch, pixels are encoded as previously described in Section 5.3.1, resulting in a set of l^2 hypervectors. The positional hypervectors for these pixels are then generated and combined through HDC binding operations. Finally, the hypervectors representing all pixels within a patch are bundled, producing a comprehensive representation for each patch.

To accommodate hardware implementations, particularly for MAP-C (bi-polar hypervectors), permutation is executed via circular shifting, a method conducive to efficient hardware execution. The shift magnitude is determined by the pixel coordinates.

This encoding process is applied uniformly across all K^2 patches, generating a set of hypervectors that collectively represent the entire image. By encoding both the intensity values and the spatial distribution of pixels, our method offers a robust framework for processing and

analyzing image data through the principles of hyperdimensional computing.

5.3.2 Constructing the Hierarchy

The crux of our proposed method lies in articulating the classification challenge as a hierarchical problem, fundamentally segmented into two distinct levels. This hierarchical perspective facilitates a more nuanced understanding and processing of the classification task, especially when dealing with semantically rich and diverse datasets.

Grouping Labels

At the heart of our approach is the strategic grouping of labels, an essential step that underpins the hierarchical classification process. To achieve this, we embark on learning symbolic representations for each class label, a process that begins with the encoding of input images into their corresponding hypervectors via the previously delineated patch encoder.

Subsequent to the encoding phase, hypervectors pertaining to images within the same class are bundled, culminating in a singular hypervector that symbolically represents the class in its entirety. The affinity between these class-specific hypervectors is quantified through similarity scores, which are then leveraged to construct a confusion matrix. This matrix serves as the foundation for our label grouping mechanism, where a greedy algorithm is employed to identify and pair the most semantically similar classes, thereby delineating the coarse categories within the hierarchy.

Representing Coarse and Fine Labels

Within the framework of our hierarchical classification, each image is associated with two labels: a "*Coarse*" label, denoting the broader category to which it belongs, and a "*Fine*" label, representing its specific class within that category. To facilitate this dual labeling system, we assign to each label a randomly sampled bi-polar hypervector, ensuring that these vectors serve as distinct and identifiable representations for the fine labels. The coarse labels, emblematic of

the broader categories, are derived by binding together the hypervectors of class pairs identified during the grouping phase.

Utilizing this method of representation, our convolutional neural network (CNN) is designed to initially deduce the group hypervector corresponding to an image. Following this, the network discerns and subsequently eliminates the incorrect class within the identified group through a process of unbinding from the group hypervector. This operation effectively isolates the correct class hypervector, which is then forwarded as the final prediction.

This hierarchical approach not only enhances the interpretability of the classification process but also significantly improves the accuracy and efficiency of the model by leveraging the inherent semantic structure of the data.

5.3.3 Model Architecture

In addressing the hierarchical classification challenge, our architecture employs Convolutional Neural Networks (CNNs) augmented with multiple residual blocks, a design choice inspired by their ability to facilitate deeper network structures without succumbing to vanishing or exploding gradients. Each residual block (named *"ConvBlock"*) within our architecture comprises two convolutional layers, each equipped with N (3×3) kernels, where N represents a hyperparameter determined empirically based on the specific demands of the dataset and the classification task at hand. Figure 5.1 illustrates the CNN architecture.

Hierarchical CNN

The crux of our proposed model lies in its hierarchical structuring, consisting of two shallow CNNs designated for *"Coarse"* and *"Fine"* classification tasks, respectively. This hierarchical arrangement allows for an initial broad categorization of inputs, subsequently refined through a more detailed analysis. Each CNN is composed of a sequence of residual blocks, with the hyperparameter N tailored for each block to optimize performance.

To translate the convolutional feature maps into a format amenable to hyperdimensional

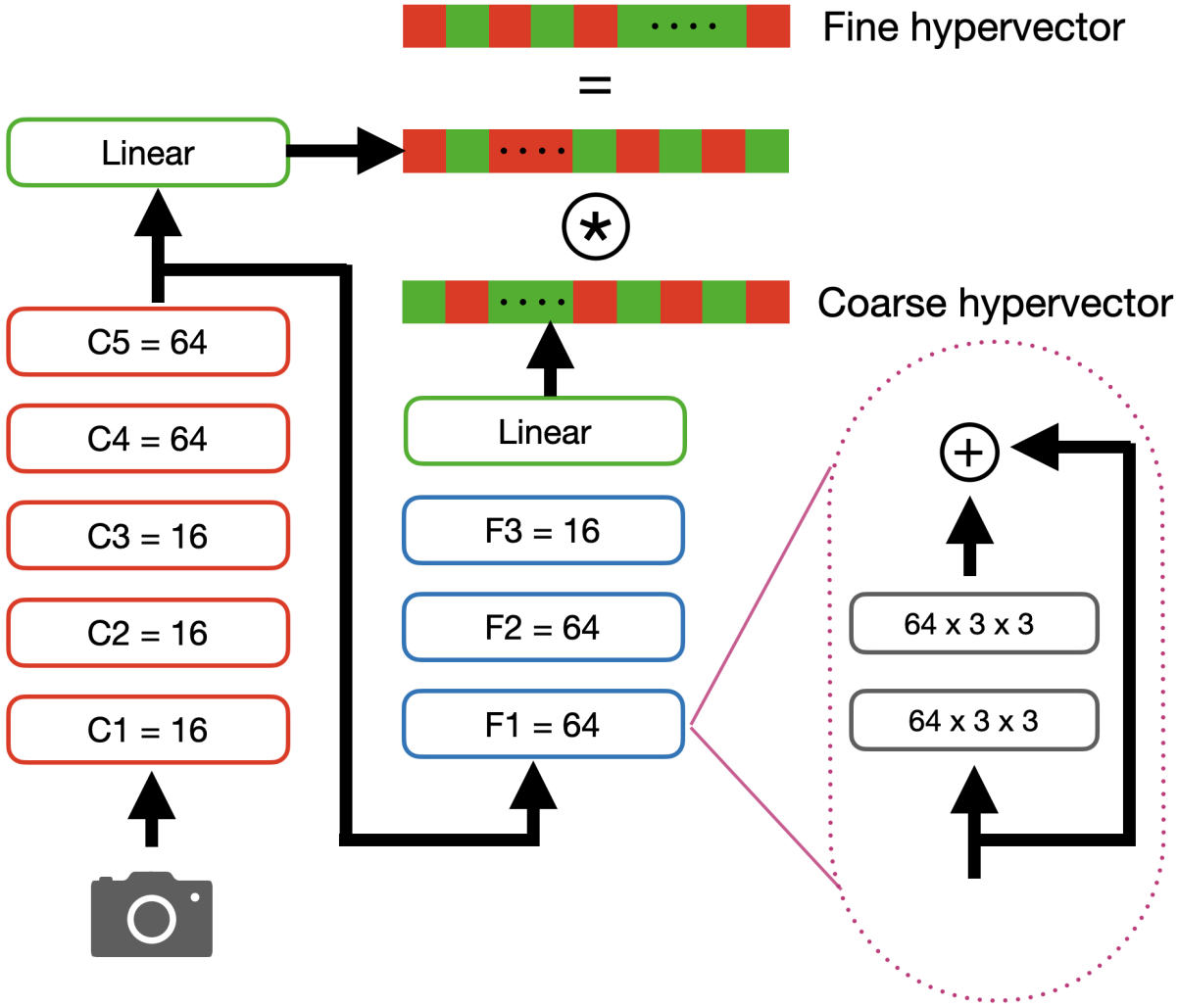


Figure 5.1. NeuroHD CNN architecture. ”C” represents the *ConvBlocks* corresponding to the *”Coarse classifier”* and ”F” that of *”Fine classifier”*

computing, the output feature maps from the final residual block of each CNN are subjected to average pooling and flattening operations. This process yields a vector representation for each image, which is then projected into a high-dimensional vector space (D-dimensional) through a linear layer. Given the non-differentiability of the sign$(\cdot)$ function, which is traditionally used for generating bi-polar hypervectors, we employ the tanh$(\cdot)$ function as an approximation, facilitating the backpropagation of gradients through this layer.

The *"Coarse"* classifier CNN diverges slightly in its final stages by outputting its feature maps prior to the pooling step, in addition to the generated hypervector. These feature maps are then used as inputs by the *"Fine"* classifier, enabling a seamless transition from coarse to fine classification within the hierarchical framework.

Loss Functions

Our training process is guided by three distinct loss functions, each serving a unique purpose in the learning hierarchy. Initially, we employ a cosine similarity loss ($\mathscr{L}_{\text{Sim}}$) to direct the *"Coarse"* classifier towards generating accurate symbolic representations for the broad categories. This is formally represented as:

$$\mathscr{L}_{\text{Sim}} = 1 - \cos(\hat{y}_{\text{pred}}, \hat{y}_{\text{true}})$$

where $\hat{y}_{\text{pred}}$ and $\hat{y}_{\text{true}}$ denote the predicted and true hypervector representations, respectively.

To reinforce the symbolic differentiation between classes within the same group, an inverse cosine similarity loss ($\mathscr{L}_{\text{I-Sim}}$) is applied, aiming to increase the angular distance for incorrect classifications within the identified category:

$$\mathscr{L}_{\text{I-Sim}} = \cos(\hat{y}_{\text{pred}}, \hat{y}_{\text{true}})$$

Lastly, the *"Fine"* classifier is trained using a second cosine similarity loss, this time fine-tuned to distinguish between the finer subclasses within the correctly identified broader

category. This multi-faceted loss structure ensures a balanced emphasis on both hierarchical levels of classification, promoting a nuanced understanding that mirrors the complex nature of real-world data.

By integrating these elements into a cohesive architecture, our proposed model addresses the inherent challenges of both HDC and DL synthetically.

5.4 Experimental Analysis

In this section, we delve into the evaluation of our neuro-symbolic model, emphasizing its application to complex image classification tasks. While our primary focus is on image classification, it's pertinent to note that the methodologies delineated herein are broadly applicable to a variety of multi-class classification problems, contingent upon the feasible representation of data in hyperdimensional computing (HDC) space. The strategic selection of HDC types and encoding strategies is aimed at harnessing the potential for hardware acceleration, particularly on specialized hardware architectures.

5.4.1 Experimental Setup

Our implementation leverages Python and the PyTorch framework [156], for acceleration using GPUs. We assess the performance of our architecture on four different image classification datasets: CIFAR-10 [108], CIFAR-100 [108], Fashion-MNIST [211] and SVHN [150].

To benchmark our model's efficiency and accuracy, we juxtapose its performance against several state-of-the-art (SoA) models for image classification. Compute cost evaluations are conducted using parameter counts, model size and FLOPs as primary metrics.

The architecture-specific parameters, namely the number of kernels per ConvBlock and the total number of ConvBlocks for both the fine and coarse classifiers, were empirically optimized. The goal was to minimize model size without compromising accuracy significantly. Further, we introduce three hyperparameters, α, β, and γ, serving as weights for the loss functions, respectively. Through extensive experimentation, the weights for the cosine-similarity

Table 5.1. Accuracy of HDC methods on CIFAR-10. D is hypervector dimensionality

Method	**Accuracy (%)**	**D** ($\times 10^3$)
SearHD	22.6	10
BRIC	26.9	4
QuantHD	28.4	8
RFF	29.4	10
PatchEncoder (Ours)	**36.8**	**1**

losses were set to $\alpha = 0.4, \beta = 0.3$, and $\gamma = 0.4$ for the inverse cosine-similarity loss (I-Sim). Models underwent training over 100 epochs, with a batch size of 128. Techniques such as weight decay and Batch Normalization [82] were employed for regularization, aiming to mitigate overfitting and enhance model generalizability.

5.4.2 Performance of Encoder

Accuracy

The efficacy of the patch encoder was assessed through its performance on the image classification task using the symbolic training methodology outlined in Section **??**. Table 5.1 presents a comparative analysis of the encoder's accuracy on the CIFAR-10 dataset against other HDC-based image encoding techniques.

As evidenced by the data in Table 5.1, our encoding strategy demonstrates a notable improvement in accuracy, surpassing alternative methods by 8 to 14 percent. It's important to acknowledge that while this performance level may not meet the benchmarks for standalone image classification tasks, it represents a significant advancement in our ability to learn approximate representations of class labels without relying on pre-existing semantic understanding of those labels. This capability is critical for our approach, as it enables effective hierarchical classification in scenarios where semantic knowledge of labels is either limited or entirely absent.

Table 5.2. Performance comparison of NeuroHD with SoA architectures on CIFAR datasets

Arch	Rel. # (x smaller)	Speedup (x faster)	Rel. size (x smaller)	CIFAR-10 Acc. (%)	CIFAR-100 Acc. (%)
Transformers	16.8	5.4	17.9	98.1	91.8
	53.8	58	204.8	**99.1**	90.8
CNNs	13.4	122.7	3.3	98.7	91.5
	13.1	13.2	3.3	90.1	81.5
Neuro-Symbolic	**1**	**301.5**	**1**	**99.1**	**97.7**

5.4.3 Performance of Hierarchical Classification

Performance of Hierarchical Classification

In this section, we delve into the empirical performance of our proposed neuro-symbolic architecture, emphasizing its efficacy in classification tasks. Our analysis is anchored in comparative evaluations with state-of-the-art (SoA) architectures, focusing on both accuracy metrics and computational efficiency. Table 5.2 presents a comprehensive comparison of metrics on the CIFAR-10 and CIFAR-100 datasets.

Accuracy

Our neuro-symbolic model, denoted (NeuroHD), showcases superior performance on the CIFAR-10 and CIFAR-100 datasets, achieving state-of-the-art accuracy while maintaining a significantly smaller footprint than conventional deep learning architectures. As illustrated in Table 5.2, transformer-based networks, despite their larger size attributable to multi-headed attention layers, generally outperform CNN counterparts in terms of accuracy. However, they are also the most resource-intensive. In contrast, CNNs exhibit a balance of efficiency and performance, with exceptions like EfficientNet-V2S [192]. Notably, all evaluated architectures, with the exception of the smaller variant of RegNetY [167], surpass the 98% accuracy threshold on CIFAR-10, with our model (NeuroHD) and CaiT-XS24 [197] reaching SoA accuracy of 99.1%.

For the CIFAR-100 dataset, our approach achieves 97.7% surpassing the previous bench-

Table 5.3. Accuracy of NeuroHD across datasets

Dataset	Accuracy (%)
Fashion-MNIST [211]	97.2
SVHN [150]	95.4
CIFAR-10 [108]	99.1
CIFAR-100 [108]	97.7

mark of 96.08% set by EfficientNet-L2 [191], the largest model in the EfficientNet family. This underscores the effectiveness of our model in leveraging neuro-symbolic principles to enhance classification accuracy. Table 5.3 shows the accuracy of NeuroHD on additional datasets. We see that on Fashion-MNIST [211] and SVHN [150], NeuroHD obtains slightly lower but comparable accuracy to the SoA.

It's also worth noting that NeuroHD beats previous benchmarks of other HDC based neuro-symbolic architectures, like NSHD [116]. On CIFAR-10, NSHD achieves just over 90% and about 80% on CIFAR-100.

5.4.4 Compute Cost

The computational efficiency of our neuro-symbolic architecture is a highlight of our experimental findings, as delineated in Table 5.2. Notably, our model (NeuroHD), outperforms the benchmark in terms of both size and speed. Specifically, it is 3.3× smaller and 13.2× faster than the most compact model in our comparison, RegNetY [167]. Moreover, when juxtaposed with the largest compared model, EfficientNet-V2S [192], NeuroHD demonstrates an extraordinary reduction in size—204.8× smaller—and an enhancement in processing speed—58× faster. This remarkable efficiency is largely attributable to the model's lean design, which necessitates merely 1.6 million parameters.

The profound improvements in computational efficiency can be ascribed to the novel structuring of the classification challenge as a hierarchical sequence of symbolic problems. Consider, for instance, the CIFAR-100 dataset, which, under our model, is segmented into 50 group classes. The "Coarse Classifier" within our architecture is tasked with discerning the

broader category to which an image belongs—effectively a 1 out of 50 classification problem, as opposed to the original 1 out of 100. This initial classification significantly reduces the complexity of the task at hand. Subsequently, the ”Fine Classifier” engages in a binary classification, conditioned on the group delineated by the ”Coarse Classifier.” It is this methodical simplification of the problem space that underpins the exceptional efficiency of our NeuroHD.

Our findings underscore the inherent advantage of adopting a hierarchical approach to classification, particularly in terms of computational resources and processing time. This strategy not only enhances model performance but also opens avenues for deploying advanced machine learning models on resource-constrained platforms.

5.5 Conclusion

This study presented NeuroHD, a novel architecture that fuses the strengths of deep learning and hyperdimensional computing (HDC) to address the limitations of each approach individually. NeuroHD effectively bridges the gap between HDC's symbolic processing and deep learning's feature extraction capabilities, resulting in a model that is both compact and efficient. Our experiments demonstrate that NeuroHD significantly outperforms state-of-the-art methods in size and speed—being up to 204 times smaller and 58 times faster—while achieving SoA accuracies on the CIFAR datasets.

The success of NeuroHD not only showcases the potential of integrating connectionist and symbolic approaches but also opens new avenues for developing efficient and powerful machine learning models. This hybrid approach marks a step forward in the pursuit of on-device ML for complex applications.

5.6 Acknowledgements

Chapter 5 in full, is a reprint of the material as it appears in NeuroHD: Neuro-Symbolic Classification with Hyperdimensional Computing, submitted to ISLPED 2024. The dissertation

author was the primary investigator and author of this paper.

Chapter 6

Summary and Future Directions

This thesis provides a comprehensive analysis and enhancement of machine learning-anchored computation efficiency, with a particular spotlight on federated learning, image and text classification. The ever-increasing ubiquity of IoT applications is unparalleled, a reality that necessitates robust, efficient, and incisive frameworks to handle the information associated with these applications. From our investigations and results, it has become apparent that the development and application of neuro-symbolic architectures significantly rectify computational inadequacies and inefficiencies. These include the challenge of handling distributed data, and computational and energy constraints of on-device learning—plugging at the inadequacies of traditional machine learning methodology.

We evaluated novel architectures that effectively merge the strengths of deep learning and HDC to achieve desired improvements in compute and communication efficiency. This synthesis particularly caters to the processing of large-scale datasets, which often present with intricate information like images and text.

The work on federated learning has outlined a novel algorithm, FedHDC, alongside its optimized version, FHDnn, which leverages the distinct attributes of hyperdimensional computing to offer faster and more robust computation. Proving most valuable was this method's capacity to handle network errors gracefully. Besides these improvements, the computational and energy expenses were considerably reduced, justifying the architectures' effectiveness and

efficiency in IoT settings.

Our research also made a captivating headway in multi-modal learning, inclusive of text and image classification. We believe resolving accuracy-compute trade-off will accelerate advances in deep learning, and further boost the adaptability and usefulness of HDC. Our proposed models, featuring neuro-symbolic architectures, proved that achieving competitive accuracy does not have to translate into larger model sizes or reduced computation speed.

In text classification, our research introduced neuro-symbolic processes that demonstrated potential for efficient learning in tasks like multi-label classification. Our process, TinyXML HD, specifically, exhibited considerable improvements in both speed and size of the learned models. This manifestly argues for the effectiveness of our proposed architecture in handling text-based information.

Notably, in the realm of image classification, our exploration clearly bridged the divide between the accuracy of DL and the computational simplicity of HDC. Our innovative method for hierarchical classification, paired with our neuro-symbolic architecture, increases processing efficiency while maintaining competitive accuracy. Consequently, it also efficiently reduced the model sizes—thereby addressing a previously identified challenge of HDC in handling complex image data.

These advances hold significant potential to inform the ever-evolving IoT landscape, revolutionizing federated learning and image and text classification processes. Our results provide a firm foundation upon which further research can be built, hinting at new possibilities for combining deep learning and HDC in a valuable, complementary manner.

In conclusion, our findings demonstrate that neuro-symbolic architectures—efficiently blending the properties of deep learning and HDC—offer a promising pathway for efficient and robust machine learning mechanisms. We trust that these advancements will serve as a bedrock for future systems and applications espousing the principles and practices of efficient machine learning. As we further unlock the perimeters of this space, it is conceivable that we stand to reap even greater dividends in the efficient processing and classification of different modalities data.

6.1 Future Directions

The work presented in this dissertation elucidates the merits of a hybrid architectural approach, combining Hyperdimensional Computing (HDC) with Deep Learning (DL) to harness their complementary strengths. In light of these promising initial findings, we foresee several avenues for expanding upon the foundational research laid out herein. This chapter delineates potential future directions aimed at further refining and enhancing the capabilities of the hybrid model, thereby broadening its applicability and efficiency within the realm of Internet of Things (IoT) applications.

6.1.1 Enhancing Multi-Level Hierarchical classification

The initial exploration of encoding a simplistic, single-tier hierarchy for image classification represents only the beginning of investigating hierarchical representations within a hybrid HDC-DL framework. The logical next step involves developing more intricate, multi-tier hierarchical structures that capture a higher degree of complexity in data representation. By decomposing the problem into multiple hierarchical levels, each managed effectively by Convolutional Neural Networks (CNNs), there is significant potential to improve recognition and classification accuracies.

Future research will focus on designing and implementing these complex hierarchical architectures, emphasizing their applicability across a wide range of IoT use cases. This approach is particularly relevant for datasets with a large number of classes, such as ImageNet, where a simple two-tier hierarchy might not simplify the problem sufficiently and could result in lower accuracy. These advanced hierarchical structures would also be beneficial for applications where coarse classification is more critical than fine-grained classification results.

Furthermore, this hierarchical pipeline will enhance interpretability by allowing for tracking the classifier's decisions at each level of the hierarchy. This feature will enable us to pinpoint the exact stage where the classifier made an error, providing valuable insights for

refining and improving the model.

6.1.2 Exploring Hierarchical Navigable Small Worlds for Enhanced Search

The Hierarchical Navigable Small Worlds (HNSW) model, renowned for its efficacious approximation of the Nearest Neighbor search algorithm through graph-theoretic constructs, presents an intriguing area of exploration in conjunction with HDC's capabilities. Given HDC's demonstrated proficiency in representing graph data structures, albeit in a nascent stage, it poses an interesting proposition to investigate what novel dimensions HDC could introduce to HNSW models. By capitalizing on the statistical and probabilistic properties inherent in HDC arithmetic, there exists a promising avenue to pioneer more compact, yet potent representations for graph data. Such representations, when further processed through a deep learning lens, could unveil novel methodologies to navigate and search within these high-dimensional spaces efficiently. This synergistic exploration could yield improvements in search performance, especially pertinent to vector databases prevalent in Large Language Models (LLMs).

The trajectory delineated by these proposed directions not only underlines the potential for substantial advancements in hybrid HDC-DL models but also accentuates the unexplored territories within this domain. As we venture forth, the implications of these advancements herald a new era for on-device intelligence in IoT applications, paving the way for more autonomous, efficient, and intelligent systems.

Bibliography

[1] Uci machine learning repository. http://archive.ics.uci.edu/ml/datasets/ISOLET.

[2] Ala Al-Fuqaha, Mohsen Guizani, Mehdi Mohammadi, Mohamad Aledhari, and Moussa Ayyash. Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE Communications Surveys & Tutorials*, 17(4):2347–2376, 2015.

[3] Hassan Abu Alhaija, Zhengming Ding, Niyun Cha, Tai-Yin Chen, and Yi Ren. Computational complexity of deep learning: A survey. *arXiv preprint arXiv:2011.05248*, 2020.

[4] Ishtiaq Ali, Muhammad Javed Khan, and Muhammad Mahbubur Rahman. Xt: An xgboost-based extreme multi-label text classification system. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, pages 1355–1358. ACM, 2018.

[5] Ganesh Ananthanarayanan, Paramvir Bahl, Peter Bodík, Krishna Chintalapudi, Matthai Philipose, Lenin Ravindranath, and Sudipta Sinha. Real-time video analytics: The killer app for edge computing. *computer*, 50(10):58–67, 2017.

[6] Austin P Arechiga and Alan J Michaels. The robustness of modern deep learning architectures against single event upset errors. In *2018 IEEE High Performance extreme Computing Conference (HPEC)*, pages 1–6. IEEE, 2018.

[7] Fatemeh Asgarinejad, Anthony Thomas, and Tajana Rosing. Detection of epileptic seizures from surface eeg using hyperdimensional computing. In *2020 42nd Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC)*, pages 536–540. IEEE, 2020.

[8] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Computer networks*, 54(15):2787–2805, 2010.

[9] Rohit Babbar and Bernhard Schölkopf. Efficient multi-label classification with many labels. *International Conference on Machine Learning*, 2013.

[10] Sebastian Bader and Pascal Hitzler. Dimensions of neural-symbolic integration-a structured survey. *arXiv preprint cs/0511042*, 2005.

[11] Ravikumar Balakrishnan, Mustafa Akdeniz, Sagar Dhakal, and Nageen Himayat. Resource management and fairness for federated learning over wireless edge networks. In *2020 IEEE 21st International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, pages 1–5. IEEE, 2020.

[12] Ravikumar Balakrishnan, Tian Li, Tianyi Zhou, Nageen Himayat, Virginia Smith, and Jeff Bilmes. Diverse client selection for federated learning via submodular maximization. In *International Conference on Learning Representations*, 2021.

[13] Yoshua Bengio. Learning deep architectures for ai. *Foundations and Trends® in Machine Learning*, 2(1):1–127, 2009.

[14] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

[15] Jeremy Bernstein, Yu-Xiang Wang, Kamyar Azizzadenesheli, and Animashree Anandkumar. signsgd: Compressed optimisation for non-convex problems. In *International Conference on Machine Learning*, pages 560–569. PMLR, 2018.

[16] Kush Bhatia, Kunal Dahiya, Himanshu Jain, Purushottam Kar, Anurag Mittal, Yashoteja Prabhu, and Manik Varma. Extreme multi-label text classification: A benchmark. *arXiv preprint arXiv:1503.02099*, 2015.

[17] Kush Bhatia, Himanshu Jain, Purushottam Kar, Manik Varma, and Prateek Jain. Sparse local embeddings for extreme multi-label classification. In *Advances in neural information processing systems*, pages 730–738, 2015.

[18] Kush Bhatia, Prateek Jain, Purushottam Kar, Manik Varma, and Anshumali Jain. Sparse local embeddings for extreme multi-label classification. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1155–1164. ACM, 2016.

[19] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konečný, Stefano Mazzocchi, H Brendan McMahan, et al. Towards federated learning at scale: System design. *arXiv preprint arXiv:1902.01046*, 2019.

[20] Nader Bouacida, Jiahui Hou, Hui Zang, and Xin Liu. Adaptive federated dropout: Improving communication efficiency and generalization for federated learning. In *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, page 1–6. IEEE, May 2021.

[21] Sebastian Caldas, Jakub Konečny, H Brendan McMahan, and Ameet Talwalkar. Expanding the reach of federated learning by reducing client resource requirements. *arXiv preprint arXiv:1812.07210*, 2018.

[22] Sebastian Caldas, Jakub Konečny, H. Brendan McMahan, and Ameet Talwalkar. Expanding the reach of federated learning by reducing client resource requirements. *arXiv:1812.07210 [cs, stat]*, Jan 2019. arXiv: 1812.07210.

[23] Rishikanth Chandrasekaran, Fatemeh Asgareinjad, Justin Morris, and Tajana Rosing. Multi-label classification with hyperdimensional representations. *IEEE Access*, 11:108458 –108474, 2023.

[24] Wei-Cheng Chang, Hsiang-Fu Yu, Kai Zhong, Yiming Yang, and Inderjit S. Dhillon. A modular deep learning approach for extreme multi-label text classification. *CoRR*, abs/1905.02331, 2019.

[25] Minmin Chen, Zhixing Xu, Kilian Q Weinberger, and Fei Sha. Bonsai: An efficient framework for learning-based hierarchical document classification. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1567–1576, 2019.

[26] Tianyi Chen, Georgios Giannakis, Tao Sun, and Wotao Yin. Lag: Lazily aggregated gradient for communication-efficient distributed learning. *Advances in Neural Information Processing Systems*, 31, 2018.

[27] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020.

[28] Weiwei Chen, Zeyuan Song, and Tie-Yan Liu. Pfastrexml: Parallelizing fastxml for extreme multi-label text classification. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, pages 1941–1944. ACM, 2018.

[29] Wenlin Chen, Samuel Horvath, and Peter Richtarik. Optimal client sampling for federated learning. *arXiv preprint arXiv:2010.13723*, 2020.

[30] Zhaokang Chen and Bertram E. Shi. Appearance-based gaze estimation using dilated-convolutions. *CoRR*, abs/1903.07296, 2019.

[31] Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. A survey of model compression and acceleration for deep neural networks. *arXiv preprint arXiv:1710.09282*, 2017.

[32] James W Cooley and John W Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of computation*, 19(90):297–301, 1965.

[33] NVIDIA Corporation. Nvidia jetson. https://developer.nvidia.com/embedded/jetson-nano-developer-kit.

[34] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. *Advances in neural information processing systems*, 28, 2015.

[35] I. Danihelka, G. Wayne, B. Uria, N. Kalchbrenner, and A. Graves. Associative long short-term memory. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 1986–1994, 2016.

[36] Jia Deng, Hao Zhang, Xianming Liu, Xingang Li, and Dacheng Tao. A comprehensive survey on model compression and acceleration. *Artificial Intelligence Review*, 2022.

[37] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[38] Li Deng, Jingyuan Li, and Junyi Chen. Deep learning in iot: A review. *Future Generation Computer Systems*, 2021.

[39] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[40] Arpan Dutta, Saransh Gupta, Behnam Khaleghi, Rishikanth Chandrasekaran, Weihong Xu, and Tajana Rosing. Hdnn-pim: Efficient in memory design of hyperdimensional computing with feature extraction. In *Proceedings of the Great Lakes Symposium on VLSI 2022*, pages 281–286, 2022.

[41] Arpan Dutta, Saransh Gupta, Behnam Khaleghi, Rishikanth Chandrasekaran, Weihong Xu, and Tajana Rosing. Hdnn-pim: Efficient in memory design of hyperdimensional computing with feature extraction. In *Proceedings of the Great Lakes Symposium on VLSI 2022*, GLSVLSI '22, page 281–286, New York, NY, USA, 2022. Association for Computing Machinery.

[42] Andre Elisseeff and Jason Weston. Kernel methods for multi-labelled classification and categorical regression problems. In *Proceedings of the 2001 International Conference on Machine Learning*, pages 179–186. Morgan Kaufmann Publishers Inc., 2001.

[43] Andre Elisseeff and Jason Weston. Kernel methods for multi-labelled classification and categorical regression problems. In *Advances in neural information processing systems*, pages 681–687. 2002.

[44] Ronald A Fisher. The use of multiple measurements in taxonomic problems. *Annals of eugenics*, 7(2):179–188, 1936.

[45] Jerry A. Fodor and Zenon W. Pylyshyn. Connectionism and cognitive architecture: A critical analysis. *Cognition*, 28(1):3–71, 1988.

[46] E Paxon Frady and Friedrich T Sommer. Extending hyperdimensional computing with gradient information. In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE, 2018.

[47] E Paxon Frady and Friedrich T Sommer. Gradient-based learning in high-dimensional random neural networks. *Neural Networks*, 124:304–311, 2020.

[48] Stephen I Gallant, John A Rodriguez, and Brian J Gallant. Toward a human-like open-domain chatbot. *Brain Sciences*, 3(1):1–49, 2013.

[49] Ashwinkumar Ganesan, Hang Gao, Sunil Gandhi, Edward Raff, Tim Oates, James Holt, and Mark McLean. Learning with holographic reduced representations. *Advances in Neural Information Processing Systems*, 34:25606–25620, 2021.

[50] Ashwinkumar Ganesan, Hang Gao, Sunil Gandhi, Edward Raff, Tim Oates, James Holt, and Mark McLean. Learning with holographic reduced representations. *CoRR*, abs/2109.02157, 2021.

[51] Jun Gao, Kun Liu, Shizhong He, and Huan Deng. Large-scale multi-label learning with missing labels. *IEEE Transactions on Knowledge and Data Engineering*, 30(10):1967–1979, 2017.

[52] Michael Gastpar. Uncoded transmission is exactly optimal for a simple gaussian "sensor" network. *IEEE Transactions on Information Theory*, 54(11):5247–5251, 2008.

[53] R W Gayler. Multiplicative binding, representation operators & analogy. In *Advances in Analogy Research: Integration of Theory and Data from the Cognitive, Computational, and Neural Sciences*, pages 1–4, 1998.

[54] R W Gayler. A vector symbolic architecture for composing and retrieving predicate-argument relations. *Connection Science*, 15(2):119–140, 2003.

[55] Ross W Gayler. Vector symbolic architectures: a new direction in artificial intelligence. In *AAAI/IAAI*, pages 959–966. AAAI Press/The MIT Press, 2002.

[56] Andrea Goldsmith. *Wireless communications*. Cambridge university press, 2005.

[57] Yunchao Gong and Svetlana Lazebnik. Iterative quantization: A procrustean approach to learning binary codes. In *CVPR 2011*, pages 817–824. IEEE, 2011.

[58] Alireza Goudarzi, Mohsen Imani, and Tajana S Rosing. Hyperdimensional computing for reinforcement learning. In *Proceedings of the 38th International Conference on Machine Learning*, pages 3832–3841. PMLR, 2021.

[59] Saurabh Goyal, Sumit Pandey, and Hema A. Murthy. Fastxml: A fast, accurate and stable tree-classifier for extreme multi-label learning. In *Proceedings of the 31st International Conference on Machine Learning (ICML)*, pages 193–201. JMLR Workshop and Conference Proceedings, 2014.

[60] Klaus Greff, Sjoerd van Steenkiste, and Jürgen Schmidhuber. Binding problem in artificial neural networks. *Neural Networks*, 124:177–193, 2020.

[61] Yunhui Guo, Mohsen Imani, Jaeyoung Kang, Sahand Salamat, Justin Morris, Baris Aksanli, Yeseong Kim, and Tajana Rosing. Hyperrec: Efficient recommender systems with hyperdimensional computing. In *2021 26th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 118–123. IEEE, 2021.

[62] Akhil Gupta and Rakesh Kumar Jha. A survey of 5g network: Architecture and emerging technologies. *IEEE access*, 3:1206–1232, 2015.

[63] Ankur Gupta, Rahul Gupta, Sandeep Verma, Saket Agarwal, and Manik Varma. X-transformer: Transformer with expert-level explanations. *arXiv preprint arXiv:2012.07023*, 2020.

[64] Zellig S Harris. Distributional structure. *Word*, 10(2-3):146–162, 1954.

[65] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[66] Joel Hestness, Sharan Narang, Newsha Ardalani, Gregory Diamos, Heewoo Jun, Hassan Kianinejad, Md Mostofa Ali Patwary, Yang Yang, and Yanqi Zhou. The computational limits of deep learning. *arXiv preprint arXiv:2007.05558*, 2020.

[67] Geoffrey Hinton. Learning multiple layers of representation. *Trends in cognitive sciences*, 11(10):428–434, 2007.

[68] Samuel Horváth, Chen-Yu Ho, Ludovit Horvath, Atal Narayan Sahu, Marco Canini, and Peter Richtárik. Natural compression for distributed deep learning. *arXiv preprint arXiv:1905.10988*, 2019.

[69] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[70] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[71] Pan Hu, Junha Im, Zain Asgar, and Sachin Katti. Starfish: Resilient image compression for aiot cameras. page 14, 2018.

[72] IDC. The Growth in Connected IoT Devices, 2019. [Online].

[73] Mohammad Sadegh Imani, Abbas Rahimi, and Tajana Rosing. Exploring hyperdimensional computing as a lightweight alternative to neural networks. In *Proceedings of the 2017 IEEE International Conference on Rebooting Computing (ICRC)*, pages 1–8. IEEE, 2017.

[74] Mohsen Imani, Samuel Bosch, Mojan Javaheripi, Bita Rouhani, Xinyu Wu, Farinaz Koushanfar, and Tajana Rosing. Semihd: Semi-supervised learning using hyperdimensional computing. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8. IEEE, 2019.

[75] Mohsen Imani, Chenyu Huang, Deqian Kong, and Tajana Rosing. Hierarchical hyperdimensional computing for energy efficient classification. In *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2018.

[76] Mohsen Imani, Yeseong Kim, Sadegh Riazi, John Messerly, Patric Liu, Farinaz Koushanfar, and Tajana Rosing. A framework for collaborative learning in secure high-dimensional space. In *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, pages 435–446, 2019.

[77] Mohsen Imani, Yeseong Kim, Thomas Worley, Sarangh Gupta, and Tajana S. Rosing. Hdcluster: An accurate clustering using brain-inspired high-dimensional computing. In *IEEE/ACM Design Automation and Test in Europe Conference (DATE)*, pages 838–841, 2019.

[78] Mohsen Imani, Deqian Kong, Abbas Rahimi, and Tajana Rosing. Voicehd: Hyperdimensional computing for efficient speech recognition. In *2017 IEEE International Conference on Rebooting Computing (ICRC)*, pages 1–8. IEEE, 2017.

[79] Mohsen Imani, Justin Morris, John Messerly, Helen Shu, Yaobang Deng, and Tajana Rosing. Bric: Locality-based encoding for energy-efficient brain-inspired hyperdimensional computing. In *Proceedings of the 56th Annual Design Automation Conference 2019*.

[80] Mohsen Imani and Tajana S Rosing. Hdc: A high-dimensional computing framework for approximate regression. *IEEE Transactions on Computers*, 68(12):1799–1813, 2019.

[81] Mohsen Imani, Zhuowen Zou, Samuel Bosch, Sanjay Anantha Rao, Sahand Salamat, Venkatesh Kumar, Yeseong Kim, and Tajana Rosing. Revisiting hyperdimensional learning for fpga and low-power architectures. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2021.

[82] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[83] Himanshu Jain, Yashoteja Prabhu, and Manik Varma. Extreme multi-label loss functions for recommendation, tagging, ranking & other missing label applications. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 935–944. ACM, 2016.

[84] Ting Jiang, Deqing Wang, Leilei Sun, Huayi Yang, Zhengyang Zhao, and Fuzhen Zhuang. Lightxml: Transformer with dynamic negative sampling for high-performance extreme multi-label text classification. *CoRR*, abs/2101.03305, 2021.

[85] Yuang Jiang, Shiqiang Wang, Victor Valls, Bong Jun Ko, Wei-Han Lee, Kin K Leung, and Leandros Tassiulas. Model pruning enables efficient federated learning on edge devices. *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[86] Aditya Joshi, Malka N Halgamuge, and Shivkumar Kalyanaraman. Language geometry using random indexing. *Knowledge-Based Systems*, 118:75–85, 2017.

[87] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Fastxml: A fast and accurate multi-class classifier for xml data. *Machine Learning*, 102(2):243–272, 2016.

[88] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Keith Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and open problems in federated learning. *arXiv preprint arXiv:1912.04977*, 2019.

[89] Peter Kairouz *et al.* Advances and open problems in federated learning. *Foundations and Trends in Machine Learning*, 2021.

[90] Pentii Kanerva, Jan Kristoferson, and Anders Holst. Random indexing of text samples for latent semantic analysis. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, volume 22, 2000.

[91] Pentti Kanerva. *Sparse Distributed Memory*. MIT Press, 1988.

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

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

[94] Pentti Kanerva. What we mean when we say" what's the dollar of mexico?": Prototypes and mapping in concept space. In *2010 AAAI fall symposium series*, 2010.

[95] Pentti Kanerva, Jan Kristofersson, and Anders Holst. Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors. *Cognitive Computation*, 1(2):139–159, 2008.

[96] Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank Reddi, Sebastian Stich, and Ananda Theertha Suresh. Scaffold: Stochastic controlled averaging for federated learning. In *International Conference on Machine Learning*, pages 5132–5143. PMLR, 2020.

[97] Ioannis Katakis, Grigorios Tsoumakas, and Ioannis Vlahavas. Multilabel text classification for automated tag suggestion. In *ECML/PKDD Discovery Challenge*, 2008.

[98] Behnam Khaleghi, Mohsen Imani, and Tajana Rosing. Prive-hd: Privacy-preserved hyperdimensional computing. In *IEEE/ACM Design Automation Conference (DAC)*. IEEE, 2020.

[99] Behnam Khaleghi, Jaeyoung Kang, Hanyang Xu, Justin Morris, and Tajana Rosing. Generic: highly efficient learning engine on edge using hyperdimensional computing. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, pages 1117–1122, 2022.

[100] Behnam Khaleghi, Hanyang Xu, Justin Morris, and Tajana Rosing. tiny-hd: Ultra-efficient hyperdimensional computing engine for iot applications. In *Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 1–6. IEEE, 2021.

[101] Behnam Khaleghi, Hanyang Xu, Justin Morris, and Tajana Šimunić Rosing. tiny-hd: Ultra-efficient hyperdimensional computing engine for iot applications. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 408–413. IEEE, 2021.

[102] Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

[103] Yeseong Kim *et al.* Geniehd: Efficient dna pattern matching accelerator using hyperdimensional computing. In *DATE*, 2020.

[104] Brent Komer, Terrence C. Stewart, Aaron R. Voelker, and Chris Eliasmith. A neural representation of continuous space using fractional binding. In *Annual Meeting of the Cognitive Science Society*, 2019.

[105] Jakub Konečnỳ, H Brendan McMahan, Daniel Ramage, and Peter Richtárik. Federated optimization: Distributed machine learning for on-device intelligence. *arXiv preprint arXiv:1610.02527*, 2016.

[106] Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.

[107] Jakub Konečnỳ and Peter Richtárik. Randomized distributed mean estimation: Accuracy vs. communication. *Frontiers in Applied Mathematics and Statistics*, page 62, 2018.

[108] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

[109] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[110] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 2012.

[111] James F Kurose. *Computer networking: A top-down approach featuring the internet, 3/E*. Pearson Education India, 2005.

[112] Lidan Lai and Malathi K Sundareshan. Hdc: A high-dimensional classifier. *Pattern Recognition*, 60:25–39, 2016.

[113] Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR06)*, volume 2, pages 2169–2178. IEEE, 2006.

[114] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[115] Yann LeCun *et al.* Deep learning. *nature*, 2015.

[116] Hyunsei Lee, Jiseung Kim, Hanning Chen, Ariela Zeira, Narayan Srinivasa, Mohsen Imani, and Yeseong Kim. Comprehensive integration of hyperdimensional computing with deep learning towards neuro-symbolic ai. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2023.

[117] Myeongjun Lee, Dongjin Kim, and Jungkyu Park. Light-xml: A memory and computation-efficient approach for extreme multi-label text classification. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4120–4129, 2018.

[118] D. D. Lewis, Y. Yang, T. Rose, and F. Li. Rcv1: A new benchmark collection for text categorization research. *JMLR*, 2004.

[119] Fei-Fei Li, Marco Andreeto, Marc'Aurelio Ranzato, and Pietro Perona. Caltech 101, Apr 2022.

[120] Ming Li, Dingding Tang, Lei Yang, Xin Liu, Shaohai Wang, and Xizhao Wang. Multi-label text classification: A survey. *IEEE Access*, 2021.

[121] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine*, 2020.

[122] Xiang Li, Kaixuan Huang, Wenhao Yang, Shusen Wang, and Zhihua Zhang. On the convergence of fedavg on non-iid data. *arXiv preprint arXiv:1907.02189*, 2019.

[123] Yuanqing Li and Ross Gayler. A comparison of vector symbolic architectures. In *International Conference on Cognitive Computing*, pages 98–107. Springer, 2018.

[124] Tian Li *et al.* Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine*, 2020.

[125] Qi Liao and Xingdi Yuan. Replace or retrieve keywords in document for information retrieval. *arXiv preprint arXiv:1905.01823*, 2019.

[126] S. Liao and B. Yuan. Circconv: A structured convolution with low complexity. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4287–4294, 2019.

[127] Geert Litjens, Thijs Kooi, Babak Ehteshami Bejnordi, Arnaud Arindra Adiyoso Setio, Francesco Ciompi, Mohsen Ghafoorian, Jeroen AWM Van Der Laak, Bram Van Ginneken, and Clara I Sánchez. A survey on deep learning in medical image analysis. *Medical image analysis*, 2017.

[128] Bingyu Liu, Xue Jin, and Chuanping Li. Deep learning for extreme multi-label text classification. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1155–1158. ACM, 2017.

[129] Jingzhou Liu, Wei-Cheng Chang, Yuexin Wu, and Yiming Yang. Deep learning for extreme multi-label text classification. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '17, page 115–124, New York, NY, USA, 2017. Association for Computing Machinery.

[130] Pengfei Liu, Xipeng Qiu, and Xuanjing Huang. Recurrent neural network for text classification with multi-task learning. *arXiv preprint arXiv:1605.05101*, 2016.

[131] Franz Loewenherz, Victor Bahl, and Yinhai Wang. Video analytics towards vision zero. *Institute of Transportation Engineers. ITE Journal*, 87(3):25, 2017.

[132] Liqiang Lu and Yun Liang. Spwa: An efficient sparse winograd convolutional neural networks accelerator on fpgas. In *Proceedings of the 55th Annual Design Automation Conference*, pages 1–6, 2018.

[133] George Madjarov, Dragi Kocev, Dejan Gjorgjevikj, and Sašo Džeroski. An extensive experimental comparison of methods for multi-label learning. In *Proceedings of the 2012 European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 317–332. Springer, 2012.

[134] Gjorgji Madjarov, Dragi Kocev, Dejan Gjorgjevikj, and Saso Dzeroski. A literature survey on algorithms for multi-label learning. *arXiv preprint arXiv:1502.05988*, 2015.

[135] P Manivasakan and S Arumugam. A comparative analysis of classification techniques for genbase dataset. *International Journal of Applied Engineering Research*, 11(3):1834–1839, 2016.

[136] Paul J. Marcelis, Vijay S. Rao, and R. Venkatesha Prasad. Dare: Data recovery through application layer coding for lorawan. In *2017 IEEE/ACM Second International Conference on Internet-of-Things Design and Implementation (IoTDI)*, pages 97–108, 2017.

[137] Julian McAuley and Jure Leskovec. Hidden factors and hidden topics: Understanding rating dimensions with review text. In *Proceedings of the 7th ACM conference on Recommender systems*. ACM, 2013.

[138] H Brendan McMahan, Eider Moore, Daniel Ramage, and Seth Hampson. Communication-efficient learning of deep networks from decentralized data. page 10.

[139] H Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. *Artificial intelligence and statistics*, 2017.

[140] Brendan McMahan *et al.* Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, 2017.

[141] Enrique López Mencia and María N Moreno. Multi-label decision trees. *Pattern Analysis and Applications*, 11(1):43–52, 2008.

[142] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[143] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[144] Jeff Mitchell, Mirella Lapata, and Vera Demberg. Composition in distributional models of semantics. *Cognitive Science*, 34(8):1388–1429, 2010.

[145] Mehdi Mohammadi, Ala Al-Fuqaha, Sameh Sorour, and Mohsen Guizani. Deep learning for iot big data and streaming analytics: A survey. *IEEE Communications Surveys Tutorials*, 2018.

[146] Justin Morris, Kazim Ergun, Behnam Khaleghi, Mohsen Imani, Baris Aksanli, and Tajana Rosing. Hydrea: Utilizing hyperdimensional computing for a more robust and efficient machine learning system. *ACM Transactions on Embedded Computing Systems (TECS)*, 2022.

[147] Justin Morris, Yilun Hao, Saransh Gupta, Ranganathan Ramkumar, Jeffrey Yu, Mohsen Imani, Baris Aksanli, and Tajana Rosing. Multi-label hd classification in 3d flash. In *2020 IFIP/IEEE 28th International Conference on Very Large Scale Integration (VLSI-SOC)*, pages 10–15, 2020.

[148] Sara Mostafavi, Debajyoti Ray, David Warde-Farley, Chris Grouios, and Quaid Morris. Genemania: a real-time multiple association network integration algorithm for predicting gene function. *Genome biology*, 9(1):S4, 2008.

[149] Jinseok Nam, Jungi Kim, Enrico Menci, Iryna Gurevych, and Min Zhang. Large-scale multi-label text classification-revisiting neural networks. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 437–452. Springer, 2014.

[150] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, volume 2011, page 5, 2011.

[151] Maximilian Nickel, Lorenzo Rosasco, and Tomaso Poggio. Holographic embeddings of knowledge graphs. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[152] Shadi A Noghabi, Landon Cox, Sharad Agarwal, and Ganesh Ananthanarayanan. The emerging landscape of edge computing. *GetMobile: Mobile Computing and Communications*, 23(4):11–20, 2020.

[153] Deniz Oktay, Johannes Ballé, Saurabh Singh, and Abhinav Shrivastava. Scalable model compression by entropy penalized reparameterization. *arXiv preprint arXiv:1906.06624*, 2019.

[154] Anthony Paleo, Kristofor D Carlson, and Larry S Davis. Hyperdimensional object representation for scene context characterization. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 79–86. IEEE, 2009.

[155] Jihong Park, Sumudu Samarakoon, Mehdi Bennis, and M'erouane Debbah. Enabling ai in future wireless networks: A data life cycle perspective. *IEEE Communications Magazine*, 2020.

[156] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, 32, 2019.

[157] David Patterson, Joseph Gonzalez, Quoc Le, Chen Liang, Lluis Munguia, Daniel Rothchild, David So, Matthieu Texier, and Jeffrey Dean. The carbon footprint of machine learning training will plateau, then shrink. *IEEE Spectrum*, 58(10):28–35, 2021.

[158] Juha Petäjäjärvi, Konstantin Mikhaylov, Matti Hämäläinen, and Jari Iinatti. Evaluation of lora lpwan technology for remote health and wellbeing monitoring. In *2016 10th International Symposium on Medical Information and Communication Technology (ISMICT)*.

[159] Raspberry Pi. Raspberry pi 4. https://www.raspberrypi.com/products/raspberry-pi-4-model-b/.

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

[161] John Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. Technical Report MSR-TR-98-14, Microsoft Research, 1998.

[162] Yashoteja Prabhu, Anil Kag, Shrutendra Harsola, Rahul Agrawal, and Manik Varma. Parabel: Partitioned label trees for extreme classification with application to dynamic search advertising. In *The Web Conference 2018*, pages 993–1002, 2018.

[163] Yashoteja Prabhu and Manik Varma. Fastxml: A fast, accurate and stable tree-classifier for extreme multi-label learning. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 263–272. ACM, 2014.

[164] Yashoteja Prabhu and Manik Varma. Fastxml: A fast, accurate and stable tree-classifier for extreme multi-label learning. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 263–272. ACM, 2018.

[165] J Ross Quinlan. *C4.5: Programs for machine learning*. Elsevier, 1993.

[166] Dmitri A Rachkovskij. Representation and processing of structures with binary sparse distributed codes. *IEEE Transactions on Neural Networks and Learning Systems*, 27(12):2691–2706, 2016.

[167] Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. Designing network design spaces. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10428–10436, 2020.

[168] Abbas Rahimi, Pentti Kanerva, and Jan M Rabaey. A robust and energy-efficient classifier using brain-inspired hyperdimensional computing. In *Proceedings of the 2016 international symposium on low power electronics and design*, pages 64–69, 2016.

[169] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems*, volume 20. Curran Associates, Inc., 2007.

[170] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *Advances in neural information processing systems*, pages 1177–1184, 2008.

[171] Okko J Räsänen and Jukka P Saarinen. Sequence prediction with sparse distributed hyperdimensional coding applied to the analysis of mobile phone use patterns. *IEEE transactions on neural networks and learning systems*, 27(9):1878–1889, 2015.

[172] Jesse Read, Bernhard Pfahringer, Geoff Holmes, and Eibe Frank. Multi-label classification using ensembles of pruned sets. In *Proceedings of the 2008 European conference on machine learning and knowledge discovery in databases*, pages 50–65. Springer, 2008.

[173] Jesse Read, Bernhard Pfahringer, Geoff Holmes, and Eibe Frank. Classifier chains for multi-label classification. In *Proceedings of the 2010 Conference on ECAI 2010: 19th European Conference on Artificial Intelligence*, pages 1007–1012. IOS Press, 2010.

[174] Jesse Read, Bernhard Pfahringer, Geoff Holmes, and Eibe Frank. Classifier chains for multi-label classification. *Machine learning*, 85(3):333–359, 2011.

[175] Jesse Read, Bernhard Pfahringer, Geoff Holmes, and Eibe Frank. Efficient multi-label classification for evolving data streams. *Machine learning*, 2011.

[176] Amirhossein Reisizadeh, Aryan Mokhtari, Hamed Hassani, Ali Jadbabaie, and Ramtin Pedarsani. Fedpaq: A communication-efficient federated learning method with periodic averaging and quantization. In *International Conference on Artificial Intelligence and Statistics*, pages 2021–2031. PMLR, 2020.

[177] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[178] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[179] Sahand Salamat, Jaeyoung Kang, Yeseong Kim, Mohsen Imani, Niema Moshiri, and Tajana Rosing. Fpga acceleration of protein back-translation and alignment. In *Design, Automation and Test in Europe Conference and Exhibition (DATE)*. IEEE, 2021.

[180] Vincent Y Shen, Nobuo Saito, Michael R Lyu, Mary Ellen Zurko, Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. *Proceedings of the 10th international conference on World Wide Web*, pages 285–295, 2001.

[181] Paul Smolensky. On the proper treatment of connectionism. *Behavioral and Brain Sciences*, 11(1):1–23, 1988.

[182] Paul Smolensky. Tensor product variable binding and the representation of symbolic structures in connectionist systems. *Artificial intelligence*, 46(1-2):159–216, 1990.

[183] Cees G. M. Snoek, Marcel Worring, Jan C. van Gemert, Jan-Mark Geusebroek, and Arnold W. M. Smeulders. The challenge problem for automated detection of 101 semantic concepts in multimedia. In *ACM Multimedia*, pages 421–430, 2006.

[184] Richard Socher, Alex Perelygin, Jean Y Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the conference on empirical methods in natural language processing (EMNLP)*, pages 1631–1642, 2013.

[185] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[186] Sebastian U Stich. Local sgd converges fast and communicates little. *arXiv preprint arXiv:1805.09767*, 2018.

[187] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in nlp. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3645–3650, 2019.

[188] Qing Su, Jing Yang, and Xiaojun Wu. Gradient-based hyperdimensional computing for deep learning. *IEEE Access*, 9:36736–36746, 2021.

[189] Ananda Theertha Suresh, X Yu Felix, Sanjiv Kumar, and H Brendan McMahan. Distributed mean estimation with limited communication. In *International Conference on Machine Learning*, pages 3329–3337. PMLR, 2017.

[190] Yasutoshi Tagami. Annexml: Approximate nearest neighbor search for extreme multi-label classification. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 455–464. ACM, 2017.

[191] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *ICML*, 2019.

[192] Mingxing Tan and Quoc Le. Efficientnetv2: Smaller models and faster training. *arXiv preprint arXiv:2104.00298*, 2021.

[193] Chandra Thapa, Mahawaga Arachchige Pathum Chamikara, Seyit Camtepe, and Lichao Sun. Splitfed: When federated learning meets split learning. *arXiv preprint arXiv:2004.12088*, 2020.

[194] Anthony Thomas, Sanjoy Dasgupta, and Tajana Rosing. Theoretical foundations of hyperdimensional computing. *Foundations and Trends in Electronic Design Automation*, 11(1):1–117, 2017.

[195] Anthony Thomas, Sanjoy Dasgupta, and Tajana Rosing. Theoretical foundations of hyperdimensional computing. *Journal of Artificial Intelligence Research*, 72:215–249, 2021.

[196] Shuai Tong, Jiliang Wang, and Yunhao Liu. Combating packet collisions using non-stationary signal scaling in lpwans. In *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services*, pages 234–246, 2020.

[197] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Herve Jegou. Going deeper with image transformers. *arXiv preprint arXiv:2103.17239*, 2021.

[198] Grigorios Tsoumakas and Ioannis Katakis. Multi-label classification: An overview. *International Journal of Data Warehousing and Mining*, 3(3):1–13, 2007.

[199] Grigorios Tsoumakas, Ioannis Katakis, and Ioannis Vlahavas. Effective and efficient multilabel classification in domains with large number of labels. In *ECML/PKDD 2008 Workshop on Mining Multidimensional Data*, 2008.

[200] Grigorios Tsoumakas, Electra Spyromitros-Xioufis, Jozef Vilcek, and Ioannis Vlahavas. Mulan: A java library for multi-label learning. *Journal of Machine Learning Research*, 12:2411–2414, 2011.

[201] Grigorios Tsoumakas and Ioannis Vlahavas. An introduction to multi-label classification. *International Journal of Data Warehousing and Mining*, 3(3):1–13, 2007.

[202] Grigorios Tsoumakas and Ioannis Vlahavas. Random k-labelsets: An ensemble method for multilabel classification. *IEEE transactions on knowledge and data engineering*, 23(7):1079–1089, 2011.

[203] Vladimir Vapnik and Alexey Chervonenkis. A new approach to the problem of pattern recognition based on learning and decision functions of many variables. *Automation and Remote Control*, 24(5):774–780, 1963.

[204] Christoph von der Malsburg. The what and why of binding: The modeler's perspective. *Neuron*, 24(1):95–104, 1999.

[205] Hui-Po Wang, Sebastian U Stich, Yang He, and Mario Fritz. Progfed: Effective, communication, and computation efficient federated learning by progressive training. *arXiv preprint arXiv:2110.05323*, 2021.

[206] Yao Wang, Stephan Wenger, Jiantao Wen, and Aggelos K Katsaggelos. Error resilient video coding techniques. *IEEE signal processing magazine*, 17(4):61–82, 2000.

[207] Yunchi Wang, Min Lin, Jiahong Wang, Jinyu Zhang, and Jiantao Zhou. Gradient-based hyperdimensional computing for clustering and classification. *IEEE Access*, 8:39277–39287, 2020.

[208] Jianqiao Wangni, Jialei Wang, Ji Liu, and Tong Zhang. Gradient sparsification for communication-efficient distributed optimization. *Advances in Neural Information Processing Systems*, 31, 2018.

[209] Xizixiang Wei and Cong Shen. Federated learning over noisy channels: Convergence analysis and design examples. *IEEE Transactions on Cognitive Communications and Networking*, 2022.

[210] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

[211] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

[212] Zirui Xu, Fuxun Yu, Jinjun Xiong, and Xiang Chen. Helios: heterogeneity-aware federated learning with dynamically balanced collaboration. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pages 997–1002. IEEE, 2021.

[213] Yang You, Jing Luo, Hongxia Jin, and Jiawei Yang. Attentionxml: Attention-based multi-label text classification with relative attention loss. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 619–629, 2019.

[214] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. In *International Conference on Learning Representations*, 2016.

[215] Zhiwei Zhan, Junhu Ren, Xiaoming Zhang, Sihai Han, and Weisong Shi. Resource management for internet of things: A survey. *IEEE Internet of Things Journal*, 2020.

[216] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 2016.

[217] Min-Ling Zhang and Zhi-Hua Zhou. Ml-knn: A lazy learning approach to multi-label learning. *Pattern Recognition*, 40(7):2038–2048, 2007.

[218] Min-Ling Zhang and Zhi-Hua Zhou. A survey on multi-label learning. *IEEE transactions on knowledge and data engineering*, 2013.

[219] Min-Ling Zhang and Zhi-Hua Zhou. A review on multi-label learning algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 26(8):1819–1837, 2014.

[220] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018.

[221] Xiaoxuan Zhang, Jie Zhao, Jianxun Lian, Han Zhang, and Xia Hu. Parabel: Partitioned label trees for extreme classification with incomplete labels. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6617–6628, 2020.

[222] Jiantao Zhou, Yunchi Wang, Min Lin, Jinyu Zhang, and Jing Yang. Unsupervised gradient-based hyperdimensional computing. *IEEE Access*, 9:16555–16565, 2021.

[223] Zhi Zhou, Xu Chen, En Li, Liekang Zeng, Ke Luo, and Junshan Zhang. Edge intelligence: Paving the last mile of artificial intelligence with edge computing. *Proceedings of the IEEE*, 2019.

[224] Zhi-Hua Zhou. Ensemble methods for multi-label classification. *Morgan & Claypool Publishers*, 27(3):309–319, 2012.

[225] Michael Zhu and Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*, 2017.

[226] Zhuowen Zou, Yeseong Kim, M Hassan Najafi, and Mohsen Imani. Manihd: Efficient hyper-dimensional learning using manifold trainable encoder. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 850–855. IEEE, 2021.

[227] Arkaitz Zubiaga. Enhancing navigation on wikipedia with social tags. *Preprint*, 2009.