Hyperdimensional Representation Learning for Node Classification and Link Prediction

Abhishek Dalvi
The Pennsylvania State University
University Park, Pennsylvania, USA
abd5811@psu.edu

Vasant Honavar
The Pennsylvania State University
University Park, Pennsylvania, USA
vuh14@psu.edu

Abstract

We introduce Hyperdimensional Graph Learner (HDGL), a novel method for node classification and link prediction in graphs. HDGL maps node features into a very high-dimensional space (*hyperdimensional* or HD space for short) using the *injectivity* property of node representations in a family of Graph Neural Networks (GNNs) and then uses HD operators such as *bundling* and *binding* to aggregate information from the local neighborhood of each node yielding latent node representations that can support both node classification and link prediction tasks. HDGL, unlike GNNs that rely on computationally expensive iterative optimization and hyperparameter tuning, requires only a single pass through the data set. We report results of experiments using widely used benchmark datasets which demonstrate that, on the node classification task, HDGL achieves accuracy that is competitive with that of the state-of-the-art GNN methods at substantially reduced computational cost; and on the link prediction task, HDGL matches the performance of DeepWalk and related methods, although it falls short of computationally demanding state-of-the-art GNNs.

CCS Concepts

• **Computing methodologies → Learning latent representations**; *Semi-supervised learning settings.*

Keywords

Hyperdimensional Computing, Graph Neural Networks, Transductive Learning, Representation Learning

1 Introduction

Social networks, citation networks, molecular, e.g., protein-protein interaction networks, etc. are naturally represented as graphs where nodes represent the entities that make up the network and edges between pairs of nodes encode relationships between entities. For example, in a social network, nodes represent individuals and links denote social connections; in a citation network, nodes represent articles, and (directed) links denote citations; In a protein-protein interaction network, the nodes represent proteins and links represent their pairwise interactions. Two of the key problems presented by such data include node classification and link prediction [27, 28]. The state-of-the-art methods for solving such problems rely on graph representation learning methods [5, 51, 56, 59].

The state-of-the-art performance achieved by such methods comes at the expense of high computational costs – and the large carbon footprints and the attendant adverse environmental impacts [7, 8, 43, 46] – of data-hungry deep learning methods that rely on iterative parameter optimization and hyperparameter tuning. Against this background, we explore a computationally efficient, one-pass learning algorithm in which the model needs to see each data sample only once, as an alternative to state-of-the-art graph neural networks for node classification and link prediction.

Key Contributions: We introduce a Hyperdimensional Graph Learner (HDGL), a novel method for node classification and link prediction in graphs. HDGL maps node features into a very high-dimensional space (*hyperdimensional* or HD space for short) using the *injectivity* property of node representations in a family of Graph Neural Networks (GNNs) and then uses HD operators such as *bundling* and *binding* to aggregate information from the local neighborhood of each node yielding latent node representations that can support both node classification and link prediction tasks. HDGL provides a one-pass learning alternative to GNNs trained using computationally expensive iterative optimization methods. We report results of extensive experiments using widely used benchmark datasets which demonstrate that, on the node classification task, HDGL achieves accuracy that is competitive with that of the state-of-the-art GNN methods at substantially reduced computational cost; and on the link prediction task, HDGL matches the performance of DeepWalk and related methods, although it falls short of computationally demanding state-of-the-art GNN methods. We further show that HDGL, which does not require expensive iterative learning procedures, is well-suited for data/class-incremental learning, making it an attractive alternative to GNNs.

2 Related Work

Graph Neural Networks (GNNs), introduced by Frasconi et al. [11], Gori et al. [15], Scarselli et al. [40], Sperduti and Starita [42], which learn embeddings of nodes in a graph by aggregating information from local neighborhoods of the nodes, have recently emerged as the dominant approach to graph representation learning [16, 24, 49], because of their superior performance across a broad

arXiv:2402.17073v3 [cs.LG] 27 Feb 2025

spectrum of applications [54, 58, 59]. However, as noted above, their state-of-the-art performance comes at a high computational cost and adverse environmental impact. In contrast, HDGL needs only a single pass through the training data, and hence, offers a computationally efficient, low energy footprint alternative to GNN for node classification and link prediction.

Hyperdimensional computing [20, 21, 34, 45] offers a brain-inspired alternative to deep neural networks for artificial intelligence and machine learning applications, that maps each object using a sufficiently high-dimensional random encoding (HD encoding for short) to produce a binary or bipolar vector [20, 21]. Simple operations, like element-wise additions and dot products [20, 34, 45], are used to perform computations on the encoded objects. Computations on HD vectors, because they yield binary or bipolar vectors, can be realized using low-precision, fast, low-power, energy-efficient hardware [25, 26]. Hence, there is a growing interest in the use of HD computing in machine learning [13, 26, 31, 34].

Hyperdimensional Graph Encodings. Poduval et al. [39] and Nunes et al. [35] have investigated hyperdimensional representations of graphs for similarity-based retrieval and graph classification. Nunes et al. [35] constructs the Graph HD vector using node features derived using PageRank [4]. Poduval et al. [39] assign random HD-vectors to nodes and considers them as node features to build a Graph HD vector. Thus, both methods ignore node features, whereas our focus is on methods that learn node-level representations for node classification and link prediction.

Hyperdimensional Node Embeddings. Recent work has explored hyperdimensional embeddings of nodes in a graph Kang et al. [22], Li et al. [29]. Specifically, Kang et al. [22] proposed RelHD, a novel processing-in-memory (PIM) hardware architecture [32] based on FeFET technology [3]. Li et al. [29] introduced the Hyper-Node method for learning HD embeddings of nodes. However, both RelHD and HyperNode encode node features as binary indicator variables before mapping them to HD space and hence are unsuited for graphs with continuous or integer-valued node attributes e.g., word counts. Furthermore, none of the aforementioned methods [22, 29, 35, 39] support link prediction on graphs.

In contrast, HDGL offers an effective and efficient method for constructing more expressive node embeddings that can accommodate not only binary or multi-valued, but also integer-valued or continuous node features. The resulting HD encodings of nodes support both node classification and link prediction.

3 Preliminaries

We proceed to briefly summarize the key concepts needed to set the stage for introducing HDGL.

3.1 Node Classification and Link Prediction on Graphs

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ represents set of nodes, $\mathcal{E}$ represents the set of undirected edges $\mathcal{E} \subseteq \{(u, v) | u, v, \in \mathcal{V}\}$. Let $N = |\mathcal{V}|$ denote the number of nodes in the graph. Supposed each node $v \in \mathcal{V}$ is described by a tuple of d features $\mathbf{x}_v \in \mathbb{R}^d$. The k-hop neighbors of node v in graph $\mathcal{G}$ are represented using multi-set $\mathcal{N}^k(v)$. The node classification problem can be described as follows: Given labels $y_w \in \{1, \cdots, L\}$ for each node $w \in \mathcal{W}$, where $\mathcal{W} \subset \mathcal{V}$; the task

is to predict the labels of unlabeled nodes, i.e., nodes in $\mathcal{V} \setminus \mathcal{W}$. Similarly, the link prediction problem entails predicting the missing edges/new edges in $\mathcal{G}$, e.g., predicting new links/recommendation in a social network. We aim to solve both problems by learning a latent representation $\phi(v)$ for each node v, utilizing the structure of $\mathcal{G}$, i.e., the connectivity between nodes, the feature vector $\mathbf{x}_v$, and the node labels y_w.

3.2 Hyperdimensional Computing

Hyper-dimensional computing, originally introduced by Kanerva [20] is a brain-inspired approach to representing information that encodes each object $x \in \mathcal{X}$ using a high dimensional mapping $\phi : \mathcal{X} \rightarrow \mathcal{H}$, where $\mathcal{H}$ is typically $\{-1, 1\}^n$ or $\{0, 1\}^n$ where n is sufficiently large [20, 21]. All computations are performed in $\mathcal{H}$, using simple operations, e.g., element-wise additions and dot products [20, 34]. The mapping ϕ is often random, and low precision which lends itself to fast, low-power, energy-efficient hardware realizations [25, 26]. In this paper, we assume $\mathcal{H} = \{0, 1\}^n$. We proceed to summarize some of the key properties of HD representation.

Near Orthogonality of Random vectors in HD Space. HD encodings of objects are produced by sampling each dimension independently and uniformly from a distribution. Let $\mathbf{a} \in \{0, 1\}^n$ and $\mathbf{b} \in \{0, 1\}^n$ be two such HD vectors where $a_i \sim Ber(p = 0.5); \forall i \in \{1, \cdots, n\}$ and $b_i \sim Ber(p = 0.5); \forall i \in \{1, \cdots, n\}$. We can then say that $\mathbf{a}$ and $\mathbf{b}$ are near orthogonal or $d_H[\mathbf{a}, \mathbf{b}]$, the normalized hamming distance between $\mathbf{a}$ and $\mathbf{b}$, ≈ 0.5 [20]. This is also known as a blessing of HD spaces.

High Memory Capacity of HD Representation. As noted by Neubert et al. [34], in the case of an n-dimensional binary HD space where each of the n-dimensional vectors is f-sparse, e.g., the number of nonzero entries is fn, the resulting HD space has capacity given by $\binom{n}{\lfloor f \cdot n \rfloor}$. For example, when $n = 1000$, and $f = 0.05$, the resulting HD space can store approximately 10^{80} or almost as many patterns as the number of atoms in the observable universe (which is estimated to be between 10^{78} and 10^{82}).

Noise or Error Tolerance. Due to the distributed representation of data across numerous dimensions in HD spaces, errors in HD computing are limited to only a fraction of bits [1, 34, 45]. Thus, HD computing offers noise tolerance without the need for expensive error correction mechanisms [34].

We next briefly describe some of the key operations used to compute with HD representations:-

Bundling, denoted by $\oplus$, is a bitwise majority operation represented by $\oplus : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}^n$. Bundling randomly sampled HD vectors results in a new vector which is similar to the input vectors. Bundling $\oplus$ is associative and commutative i.e $\mathbf{a} \oplus (\mathbf{b} \oplus \mathbf{c}) = (\mathbf{a} \oplus \mathbf{b}) \oplus \mathbf{c} = (\mathbf{c} \oplus \mathbf{a}) \oplus \mathbf{b}$. Note that in case of Bundling an even number of HD vectors, there is a possibility of ties since Bundling is a bitwise majority operation. To break ties, a tie-breaking policy needs to be used, e.g., random choice of 0 or 1 on dimensions where ties occur. Bundling can be thought of as the HD space analog of the mean operation euclidean spaces. Bundling, because it commutes, can be used to represent sets as well as multisets of objects encoded using HD vectors.

Binding, denoted by $\otimes$, is a bitwise Exclusive OR operation represented by $\otimes : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}^n$, is an operation which

takes two input vectors and maps them to a new vector. An important property of Binding is that Binding two HD vectors results in a vector that is dissimilar to both. Formally, let $\mathbf{a} \in \{0,1\}^n$ and $\mathbf{b} \in \{0,1\}^n$ be two sampled HD vectors and $\mathbf{c} = \mathbf{a} \otimes \mathbf{b}$. Due to the near-orthogonality of HD vectors, we can then say that $d_H[\mathbf{c}, \mathbf{a}] \approx 0.5$ and $d_H[\mathbf{c}, \mathbf{b}] \approx 0.5$. Binding is invertible, associative and commutative i.e $\mathbf{b} \otimes (\mathbf{a} \otimes \mathbf{b}) = (\mathbf{b} \otimes \mathbf{a}) \otimes \mathbf{b} = \mathbf{a} \otimes \mathbf{b} \otimes \mathbf{b} = \mathbf{a}$. Moreover, Binding is distributive over bundling i.e $\mathbf{a} \otimes (\mathbf{b} \oplus \mathbf{c}) = (\mathbf{a} \otimes \mathbf{b}) \oplus (\mathbf{a} \otimes \mathbf{c})$. Note that Binding also has a reflective property i.e the distance between two vectors doesn't change when both vectors are bound to the same vector. Formally, $d_H[\mathbf{a}, \mathbf{b}] = d_H[\mathbf{c} \otimes \mathbf{a}, \mathbf{c} \otimes \mathbf{b}]$. Note that this also works even vectors are slightly different i.e $d_H[\mathbf{a}, \mathbf{b}] \approx d_H[\mathbf{c} \otimes \mathbf{a}', \mathbf{c} \otimes \mathbf{b}]$ if $\mathbf{a} \approx \mathbf{a}'$.

Rotation, also called permutation, denoted by Π, is a bitwise left shift operation $\Pi : \{0,1\}^n \rightarrow \{0,1\}^n$. A key property of Rotation is that rotation results in dissimilar/orthogonal vectors i.e $d_H[\mathbf{a}, \Pi(\mathbf{a})] \approx 0.5$ and it is distributive over both Bundling and Binding and like binding, rotation is also reflective i.e $d_H[\mathbf{a}, \mathbf{b}] \approx d_H[\Pi(\mathbf{a}), \Pi(\mathbf{b})]$. Rotation was introduced by Gayler [12] in order to protect or preserve information because information can be lost/cannot be distinguished because Binding is associative and commutative i.e:- $(\mathbf{a} \otimes \mathbf{b}) \otimes (\mathbf{c} \otimes \mathbf{d}) = (\mathbf{a} \otimes \mathbf{c}) \otimes (\mathbf{b} \otimes \mathbf{d})$. In order to make the make a distinction between $(\mathbf{a} \otimes \mathbf{b}) \otimes (\mathbf{c} \otimes \mathbf{d})$ and $(\mathbf{a} \otimes \mathbf{c}) \otimes (\mathbf{b} \otimes \mathbf{d})$, rotation is used :- $(\mathbf{a} \otimes \mathbf{b}) \otimes \Pi(\mathbf{c} \otimes \mathbf{d})$.

Note that the operations referenced earlier have equivalents in the bipolar space $\{-1, 1\}^n$. Specifically, the Bundle operation can be represented as signed addition in the bipolar space, while the binding operation aligns with multiplication in the bipolar space.

3.3 Graph Neural Networks

GNNs use the node features, together with the connectivity between the nodes information to learn the embedding $\mathbf{h}_v$ of each node $v \in \mathcal{V}$. Most GNN use a neighborhood aggregation scheme where each node recursively aggregates the features of its neighbors to compute its feature vector [54, 55, 58, 59]. After k iterations of such aggregation, the node's feature vector yields its embedding. More precisely, the latent representation of node v at the kth iteration is given by: $\mathbf{h}_v^{(k)} = \text{Agg}(\{\mathbf{W}^{(k)}\mathbf{h}_j^{(k-1)}; j \in \mathcal{N}^1(v) \cup \{v\}\})$; where $\{\cdot\}$ denotes a multiset and $\mathcal{N}^1(v)$ denotes the 1-hop neighbors of v. Different GNN models, e.g., spectral methods [24], attention-based methods [49], and LSTM-based methods [16] differ primarily with respect to the aggregation function they use.

Representational Power of GNN. The representational power of GNNs can be characterized in terms of their ability to distinguish between different graph structures [55]. The greater the representational power, the more minute the differences between graphs that can be detected by the GNN. GNN have been shown to be at most as powerful as the Weisfeiler-Lehman (WL) graph isomorphism test, which iteratively updates node features by aggregating neighbors' features using an injective aggregation function [17, 52]. The injectivity of the aggregation function ensures that the differences in node neighborhoods result in differences in the resulting aggregations [17]. Consequently, GNN are at most as powerful as the WL test in distinguishing graph structures if the mapping $\mathcal{A} : \mathcal{G} \rightarrow \mathbb{R}^d$ produced by the iterative aggregation

scheme: $\mathbf{h}_v^{(k)} = g(\mathbf{h}_v^{(k-1)}, f(\{\mathbf{h}_j^{(k-1)}; j \in \mathcal{N}^1(v)\}))$ where $g(\cdot)$ and $f(\cdot)$ and $\mathcal{A}$'s graph-level readout are injective.

4 Hyper-Dimensional Graph Learner

HDGL constructs a HD representation of graphs that matches the expressive power of GNN. HDGL exploits the properties of HD operations, namely, Bundling ($\oplus$), Binding ($\otimes$), and Rotation (Π) to ensure that graphs with similar node features and topologies map to similar HD representations. Specifically, HDGL: (i). Maps node features $\mathbf{x}_v \in \mathbb{R}^d$ to a intermediate HD representation $\mathbf{r}_v \in \{0,1\}^\beta$ $\forall v \in \mathcal{V}$. (ii). Constructs a latent HD representation $\mathbf{z}_v \in \{0,1\}^\beta$ using $\mathbf{r}_v$ and $\mathcal{N}^k(v)$. (iii). Uses $\mathbf{z}_v$ to predict node labels and links in the graph. We proceed to describe each of these steps in detail.
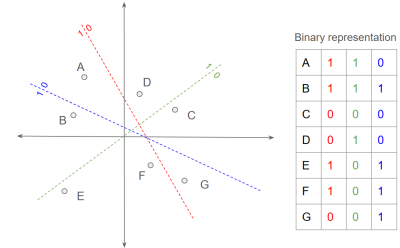

Figure 1: Mapping node features to a binary space using Random Hyperplanes: points are assigned 1 or 0 based on their position relative to each hyperplane.

4.1 Mapping Node Features to HD-space

The first crucial step is to map node features from $\mathbb{R}^d$ space to $\{0,1\}^\beta$ HD space such that $\beta \gg d$. We do this using random hyperplane tessellations [10], a type of Locality Sensitive Hashing [6, 19]. The key idea is to generate β random hyperplanes in $\mathbb{R}^d$ and set the corresponding co-ordinate of the HD representation of a node based on whether its feature vector in $\mathbb{R}^d$ lies on the positive or the negative side of the hyperplane. With β such hyper-planes in hand, we can generate an n dimensional binary sketch for each node. More precisely, let $\mathbf{Q} : \mathbb{R}^d \rightarrow \mathbb{R}^\beta$ be a matrix with each row $\mathbf{q}_1^T, \cdots \mathbf{q}_\beta^T$ drawn from $\mathcal{N}(0, I_d)$ and γ be uniformly distributed on $[-\lambda, \lambda]$. For conciseness, we represent β samples of γ as $\Gamma \sim [-\lambda, \lambda]^\beta$. Using $\mathbf{Q}$ and Γ, a randomized binary sketch $\mathbf{r}_v$ for a node $v \in \mathcal{V}$ is given by $\mathbf{r}_v = \text{sign}(\mathbf{Q}\mathbf{x}_v + \Gamma)$. The offset Γ when it is non-zero ensures that the sampled hyperplane does not pass through the origin. Note that the resulting binary sketch of each node is in fact a β-dimensional binary HD vector. Nodes with similar features will have similar HD representations; and nodes with dissimilar features will have dissimilar HD representations.

4.2 Neighborhood Aggregation Scheme

To simplify the mapping of nodes to their HD encodings, we assume that nodes that have identical multi-sets of features over their k-hop neighborhoods are likely to also have identical local topologies. Under this assumption, we can approximate injectivity of the embedding $\mathbf{z}_v$ of each node v to its HD representation by aggregating features of nodes over a k-hop neighborhood of v as follows: $\mathbf{z}_v = \phi\Big(\mathbf{r}_v, \varphi_1(\{\mathbf{r}_j; j \in \mathcal{N}^1(v)\}), \ldots, \varphi_K(\{\mathbf{r}_m; m \in \mathcal{N}^K(v)\})\Big)$ where $\mathbf{z}_v$ denotes the HD embedding of node v and $\phi(\cdot)$ and $\varphi_k(\cdot); k \in \{1, \cdots, K\}$ are injective functions.

Abhishek Dalvi and Vasant Honavar

Ensuring that similar nodes have similar HD representations. Suppose for a moment that the Binding operation $\otimes$ is used to realize $\varphi_k(\cdot)$. Because binding of two similar vectors results in dissimilar vectors, it would adversely impact the utility of the HD representation for predicting node labels based on node features and local node topologies.

In contrast, we use Bundling operation because it is permutation-invariant and is well-suited for implementing the functions $\varphi_k(\cdot)$ under the assumption that nodes that have identical multi-sets of features over their k-hop neighborhoods are likely to also have identical local topologies. More precisely, $\varphi_k\big(\{\mathbf{r}_j; j \in \mathcal{N}^k(v)\}\big) = \oplus_{j \in \mathcal{N}^k(v)} \mathbf{r}_j$ where $\oplus_{j \in \mathcal{N}^k(v)} \mathbf{r}_j$ denotes bundling the HD representations of all of the k-hop neighbors of v. This operation is analogous to finding a high-dimensional vector that is most similar to all of the input high-dimensional vectors, similar to finding the mean in Euclidean spaces and like the mean, it is not necessarily injective. However, as noted by [55], GNNs that use the mean to aggregate features over node neighborhoods [16, 24] perform quite well in practice because the mean operation becomes increasingly injective when graphs consist of diverse node features, which is typically the case in real-world graphs.

Binding for linking node with its neighbors. Having chosen Bundling to realize $\varphi_k(\cdot)$, we proceed to consider how to realize $\phi(\cdot)$, which has to couple the representation of a node obtained using bundling (see above) with the representations of its neighbors.

As we will see shortly, if we choose to realize $\phi(\cdot)$ using the Binding operation $\otimes$, we obtain a representation that strikes a reasonable balance between discriminating between dissimilar graphs and generalizing over similar graphs. Let $v \in \mathcal{V}$ and $w \in \mathcal{V}$ be two nodes in a graph. Suppose $\phi(\cdot)$ is realized using the Binding Operation $\otimes$ and $\psi_k(\cdot)$ is realized using the Bundling Operation. Considering only 1-hop and 2-hop neighbors of v, the latent representation of v and w will be

$$\mathbf{z}_v = \mathbf{r}_v \otimes \Big(\underset{j \in \mathcal{N}^1(v)}{\oplus} \mathbf{r}_j \Big) \otimes \Big(\underset{m \in \mathcal{N}^2(v)}{\oplus} \mathbf{r}_m \Big)$$
$$\mathbf{z}_w = \mathbf{r}_w \otimes \Big(\underset{j \in \mathcal{N}^1(w)}{\oplus} \mathbf{r}_j \Big) \otimes \Big(\underset{m \in \mathcal{N}^2(w)}{\oplus} \mathbf{r}_m \Big) \tag{1}$$

Suppose we have

$$\mathbf{r}_w \approx \mathbf{r}_v; \quad \Big(\underset{j \in \mathcal{N}^1(v)}{\oplus} \mathbf{r}_j \Big) \approx \Big(\underset{j \in \mathcal{N}^1(w)}{\oplus} \mathbf{r}_j \Big)$$

and $\underset{j \in \mathcal{N}^2(v)}{\oplus} \mathbf{r}_j$ is dissimilar to $\underset{j \in \mathcal{N}^2(w)}{\oplus} \mathbf{r}_j$. Then $\mathbf{z}_v$ and $\mathbf{z}_w$ will be dissimilar since binding is reflecting i.e

$$d_H\Big[\Big(\underset{j \in \mathcal{N}^2(v)}{\oplus} \mathbf{r}_j \Big), \Big(\underset{j \in \mathcal{N}^2(w)}{\oplus} \mathbf{r}_j \Big)\Big] \approx d_H\Big[\mathbf{z}_v, \mathbf{z}_w\Big]$$

It is easy to see that if all the terms in $\mathbf{z}_v$ and $\mathbf{z}_w$ are similar, i.e., v and w have similar node features, and similar 1-hop and 2-hop neighborhoods, then $\mathbf{z}_v \approx \mathbf{z}_w$ as desired. On the other hand, if the encodings of the features of the 2-hop neighbors of both v and w differ, $\mathbf{z}_v$ and $\mathbf{z}_w$ will be dissimilar as desired. When not only the 2-hop neighbors but also the 1-hop neighbors of both v and w are dissimilar, the dissimilarity between $\mathbf{z}_v$ and $\mathbf{z}_w$ becomes even greater. The key property that makes binding attractive for realizing the function $\phi(\cdot)$ is its reflectivity, i.e., the distance between two

HD vectors remains unchanged when both of them are bound to the same third vector.

On the other hand, it is easy to see that Bundling is not a good choice for realizing $\phi(\cdot)$. Because bundling is a bitwise majority operation, bundling the node vector $\mathbf{r}_v$ with neighborhood information vector $\oplus_{j \in \mathcal{N}^k(v)} \mathbf{r}_j$; results in a vector where the information from a node's neighborhood suppresses the information about the node's features, which is an undesirable property.

Improving injectivity via Rotation. Because Binding is associative and commutative, it can adversely impact the injectivity of $\varphi_k(\cdot)$ and hence the representational power of our HD model. For example, suppose $v \in \mathcal{V}$ and $w \in \mathcal{V}$ are two nodes in a graph with encoding obtained using HD mapping that uses Bundling to realize φ_k and binding to realize $\phi(\cdot)$ and $\mathbf{z}_v$ and $\mathbf{z}_w$ are computed as per Eq. (1). Consider the scenario in which $\mathbf{x}_v = \mathbf{x}_w$, $\mathcal{N}^1(v) = \mathcal{N}^2(w)$ and $\mathcal{N}^2(v) = \mathcal{N}^1(w)$. Now, assuming $\mathcal{N}^1(v) \neq \mathcal{N}^2(v)$, the representation of w can be rewritten as $\mathbf{z}_w = \mathbf{r}_v \otimes \Big(\oplus_{j \in \mathcal{N}^2(v)} \mathbf{r}_j \Big) \otimes \Big(\oplus_{m \in \mathcal{N}^1(v)} \mathbf{r}_m \Big)$. Since, Binding is both associative and commutative, the latent representations $\mathbf{z}_v$ and $\mathbf{z}_w$ become equal, despite their 1-hop and 2-hop neighbors being different. This adversely impacts the representation's ability to discriminate between dissimilar graphs. To address this issue, we utilize the Rotation Π operator. Now, the latent representation for nodes $v \in \mathcal{V}$ and $w \in \mathcal{V}$ changes:

$$\mathbf{z}_v = \mathbf{r}_v \otimes \Pi\Big(\underset{j \in \mathcal{N}^1(v)}{\oplus} \mathbf{r}_j \Big) \otimes \Pi\,\Pi\Big(\underset{m \in \mathcal{N}^2(v)}{\oplus} \mathbf{r}_m \Big)$$

$$\mathbf{z}_w = \mathbf{r}_v \otimes \Pi\Big(\underset{j \in \mathcal{N}^2(v)}{\oplus} \mathbf{r}_j \Big) \otimes \Pi\,\Pi\Big(\underset{m \in \mathcal{N}^1(v)}{\oplus} \mathbf{r}_m \Big)$$

where $\Pi(\cdot)$ and $\Pi\,\Pi(\cdot)$ denote single and double rotation operations respectively. Because the rotation of HD vector yields a HD vector that is dissimilar or orthogonal to the original, now $\mathbf{z}_v$ and $\mathbf{z}_w$ will be dissimilar. That is,

$$d_H\Big[\Pi\Big(\underset{j \in \mathcal{N}^1(v)}{\oplus} \mathbf{r}_j \Big), \Pi\,\Pi\Big(\underset{m \in \mathcal{N}^2(v)}{\oplus} \mathbf{r}_m \Big)\Big] \approx 0.5$$

$$d_H\Big[\Pi\Big(\underset{j \in \mathcal{N}^2(v)}{\oplus} \mathbf{r}_j \Big), \Pi\,\Pi\Big(\underset{m \in \mathcal{N}^1(v)}{\oplus} \mathbf{r}_m \Big)\Big] \approx 0.5$$

Hence, considering only 1-hop and 2-hop neighbors, the proposed aggregation scheme to be used by HDGL for computing the HD representation of a node $v \in \mathcal{V}$ in the graph is given by

$$\mathbf{z}_v = \mathbf{r}_v \otimes \Pi\Big(\underset{j \in \mathcal{N}^1(v)}{\oplus} \mathbf{r}_j \Big) \otimes \Pi\,\Pi\Big(\underset{m \in \mathcal{N}^2(v)}{\oplus} \mathbf{r}_m \Big) \tag{2}$$

4.3 Node Label Prediction using HDGL

With the procedure for constructing a HD representation of nodes and their local topologies in place, we turn to the task of using the resulting HD node representation to predict the node labels using $\mathbf{z}_v$. Here, we follow the standard approach to predicting class labels of HD encoded objects [13]. Specifically, during the learning phase, we Bundle the collection of HD encodings of nodes belonging to each class yielding a HD Class Hyper-Vector for each of the classes. During the inference phase, given the HD vectors for each of the classes, we predict the label of an unlabeled node u as simply the label associated with the class hyper-vector that is, among all class HD class vectors is closest to the HD vector $\mathbf{z}_u$, the encoding of

---

Algorithm 1: Pseudo-code for HDGL

Input: $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, Node Features $\mathbf{x}_v$, Train Nodes $\mathcal{T}_\ell \subset \mathcal{V}$ for
each class $\forall \ell \in \{1, \cdots, L\}$ and Unlabeled Test Nodes $\mathcal{V}_{\text{test}}$
Sample $\mathbf{Q} : [\mathbf{q}_1^T \sim \mathcal{N}(0, I_d), \cdots, \mathbf{q}_\beta^T \sim \mathcal{N}(0, I_d)]$;
Sample $\Gamma \sim [-\lambda, \lambda]^\beta$;
for $v \in \mathcal{V}$ **do**
$\quad \mathbf{r}_v \leftarrow \text{sign}(\mathbf{Q}\mathbf{x}_v + \Gamma)$;
for $v \in \mathcal{V}$ **do**
$\quad \mathbf{z}_v \leftarrow \mathbf{r}_v \otimes \Pi\left(\oplus_{j \in \mathcal{N}^1(v)} \mathbf{r}_j\right) \otimes \Pi\,\Pi\left(\oplus_{m \in \mathcal{N}^2(v)} \mathbf{r}_m\right)$;
if *Node Label Prediction Task* **then**
$\quad$ **for** $\ell \in \{1, \cdots, L\}$ **do**
$\quad\quad \mathbf{c}_\ell \leftarrow \oplus_{i \in \mathcal{T}_\ell} \mathbf{z}_i$;   #Computing Label Hyper Vectors;
$\quad$ **for** $u \in \mathcal{V}_{test}$ **do**
$\quad\quad \hat{y}_u \leftarrow \text{argmin}_{\ell \in \{1, \cdots, L\}}\, d_H[\mathbf{z}_u, \mathbf{c}_\ell]$; #Test-set Inference;
if *Link Prediction Task* **then**
$\quad \mathbf{e}^+ \leftarrow \oplus_{(u,v) \in \mathcal{E}}(\mathbf{z}_u \otimes \mathbf{z}_v)$ and $\mathbf{e}^- \leftarrow \oplus_{(u,v) \notin \mathcal{E}}(\mathbf{z}_u \otimes \mathbf{z}_v)$;
$\quad$ Compute $\mathbf{Z}^+$ and $\mathbf{Z}^-$ using Eqn. (3);
$\quad \mathbf{D}^+ \leftarrow dist(\mathbf{Z}^+, \mathbf{Z})$ and $\mathbf{D}^- \leftarrow dist(\mathbf{Z}^-, \mathbf{Z})$;
$\quad$ **for** $i = 1$ **to** N **do**
$\quad\quad$ **for** $j = 1$ **to** N **do**
$\quad\quad\quad$ **if** $D_{ij}^+ < D_{ij}^-$ **then**
$\quad\quad\quad\quad \hat{A}_{ij} \leftarrow \sigma\left((1 - D_{ij}^+) + D_{ij}^-\right)$;
$\quad\quad\quad$ **else**
$\quad\quad\quad\quad \hat{A}_{ij} \leftarrow \sigma\left(D_{ij}^+ - (1 - D_{ij}^-)\right)$;

---

u. More precisely let $\forall \ell \in \{1, \cdots, L\}$, $\mathcal{T}_\ell \subset \mathcal{V}$ be the subset of nodes that make up the training set with label ℓ and $\mathcal{V}_{\text{test}} \subset \mathcal{V}$ be the subset of unlabeled nodes in the graph. The class HD vectors are constructed for each class as follows: $\mathbf{c}_\ell = \oplus_{i \in \mathcal{T}_\ell} \mathbf{z}_i$; $\forall \ell \in \{1, \cdots, L\}$, resulting in a set of class HD vectors $\{\mathbf{c}_1, \cdots, \mathbf{c}_L\}$. Let $u \in \mathcal{V}_{\text{test}}$ be a node for which the class label is to be predicted. We predict the class label for u using its HD representation and the HD representations of each of the classes as follows: $\hat{y}_u = \text{argmin}_{\ell \in \{1, \cdots, L\}}\, d_H[\mathbf{z}_u, \mathbf{c}_\ell]$.

4.4 Link Prediction Using HDGL

Our approach takes advantage of a technique introduced by Kanerva [20, 21]'s for representing and retrieving data using HD representation. Lets consider a graph with 7 nodes and there are 3 edges in the graph betweem node $(1, 2)$, $(3, 6)$ and $(5, 7)$. By binding latent representation between pairs of nodes which have edges between them and bundling the resulting vectors, we obtain a sum-vector that encapsulates the edge information of the graph: $\mathbf{e}^+ = (\mathbf{z}_1 \otimes \mathbf{z}_2) \oplus (\mathbf{z}_3 \otimes \mathbf{z}_6) \oplus (\mathbf{z}_5 \otimes \mathbf{z}_7)$.
Edge information can be recovered solely using hypervectors $\mathbf{e}^+$ and $\mathbf{z}_i; \forall i \in \mathcal{V}$. For eg. $\mathbf{z}_2 \approx \mathbf{e}^+ \otimes \mathbf{z}_1$ and $\mathbf{z}_7 \approx \mathbf{e}^+ \otimes \mathbf{z}_5$.

For the transductive link prediction problem, we leverage on the assumption that the underlying semantics governing existing links, which are based on latent representations of nodes in an edge, will also apply to new edges. For instance, if nodes $(1, 2)$ are connected and we are predicting edges involving node 4 and if $\mathbf{z}_1 \approx \mathbf{z}_4$, then we would expect an edge $(4, 2)$ to be predicted. Note that it's also crucial to accurately predict non-edges to minimize

false positives. Therefore, we also make use of hypervector $\mathbf{e}^-$ to represent information about edges that are *absent* in the graph.

Following this overview, we can now delve into a detailed explanation of our model. Formally: $\mathbf{e}^+ = \oplus_{(u,v) \in \mathcal{E}}(\mathbf{z}_u \otimes \mathbf{z}_v)$; $\mathbf{e}^- = \oplus_{(u,v) \notin \mathcal{E}}(\mathbf{z}_u \otimes \mathbf{z}_v)$. Given this encoding of edges that are present and the edges that are absent in the graph, we can proceed to predict links as follows:

To find putative new links we Bind $\otimes$ the edge information hypervector $\mathbf{e}^+$ and the latent representation of all the nodes in the graph. This results in a new matrix $\mathbf{Z}^+$ where the ith row is $\mathbf{z}_i \otimes \mathbf{e}^+$; representing the possible node representation of the target node in a link, with the corresponding source node representation being $\mathbf{z}_i$. We follow the same procedure for the hypervector encoding information about the edges that are absent, resulting in $\mathbf{Z}^-$. Precisely:

$$\mathbf{Z}^+ = \begin{bmatrix} \mathbf{z}_1 \otimes \mathbf{e}^+ \\ \vdots \\ \mathbf{z}_N \otimes \mathbf{e}^+ \end{bmatrix} \quad \mathbf{Z}^- = \begin{bmatrix} \mathbf{z}_1 \otimes \mathbf{e}^- \\ \vdots \\ \mathbf{z}_N \otimes \mathbf{e}^- \end{bmatrix} \tag{3}$$

Now, we find pairwise normalized distances between rows of $\mathbf{Z}^+$ and rows of $\mathbf{Z}$. This results in $\mathbf{D}^+ = dist(\mathbf{Z}^+, \mathbf{Z}) \in [0, 1]^{N \times N}$, a matrix representing distances. If entry (i, j) is close to zero, it indicates that the "distance" between row i of $\mathbf{Z}^+$ is close to row j in $\mathbf{Z}$, suggesting the possibility of a link i and j. Similarly, we find pairwise normalized distances between rows of $\mathbf{Z}^-$ and rows of $\mathbf{Z}$ yielding $\mathbf{D}^- = dist(\mathbf{Z}^-, \mathbf{Z})$. An entry close to zero in $\mathbf{D}^-$ suggests the absence of an edge between the corresponding nodes.

We now generate a prediction for the adjacency matrix, denoted by $\hat{\mathbf{A}}$, where $\hat{A}_{ij} \in [0, 1]$. We predict values that fall within this interval because the evaluation metrics used are AUC-ROC and Average Precision. For real-world applications, an appropriate threshold can be selected depending on the specific use case to effectively interpret these predictions. To obtain the final predicted adjacency matrix of probabilities, which reflects the likelihood of an edge being present based on $\mathbf{D}^+$ and $\mathbf{D}^-$, we perform the following:

$$\hat{A}_{ij} = \begin{cases} \sigma\left((1 - D_{ij}^+) + D_{ij}^-\right) & \text{if } D_{ij}^+ < D_{ij}^- \\ \sigma\left(D_{ij}^+ - (1 - D_{ij}^-)\right) & \text{else} \end{cases}$$

where $\sigma(\cdot)$ is the sigmoid function. The rationale behind this operation lies in the comparison between D_{ij}^+ and D_{ij}^-. When $D_{ij}^+ < D_{ij}^-$, it indicates a higher likelihood of a connection between nodes i and j according to the distance metric. Hence, we apply the operation $\sigma((1 - D_{ij}^+) + D_{ij}^-)$ to obtain a value closer to one. We use the same logic when $D_{ij}^+ \geq D_{ij}^-$ to obtain a value close to zero.

5 Experiments

We proceed to report results of extensive experiments comparing HDGL with a set of state-of-the-art node classification methods and link prediction methods, including several GNN models, in terms of both predictive performance and runtime. The code for HDGL can be found at: https://github.com/Abhishek-Dalvi410/HDGL.

5.1 Data sets

Since HDGL is designed to ensure that nodes with similar features and neighborhoods have comparable representations—often resulting in identical node labels—we focus exclusively on graphs with these characteristics as benchmark datasets for this study.

For our experiments, we utilize commonly used node labeling graph benchmarks: the CORA graph dataset [57], sourced from McCallum et al. [30]; the CiteSeer graph dataset [57], sourced from Giles et al. [14]; and the PubMed dataset [33]. Additionally, we include three more datasets from the Microsoft Academic Graph: Coauthor CS and Coauthor Physics from Shchur et al. [41], and the BlogCatalog Graph dataset from Huang et al. [18]. Additionally, we introduce a new version of the DBLP dataset, built upon the works of Pan et al. [36]. We remove nodes with no neighbors in the graph obtained from Pan et al. [36]. The original node features from Pan et al. [36] consist of manuscript titles. We utilize BERT [9] to obtain embeddings for the titles and employ them as node features. The data sets and their statistics are given in Table 1.

We adhere to the Train/Validation/Test splits described in Yang et al. [57], Kipf and Welling [24], and Veličković et al. [49] for the CORA, CiteSeer, and PubMed datasets, respectively. For the Coauthor CS, Coauthor Physics, BlogCatalog, and DBLP datasets, we employ the data splitting strategy outlined in Shchur et al. [41].

We test transductive link prediction models on CORA, CiteSeer, and Pubmed datasets and with train/val/test splits from Kipf and Welling [23] work i.e validation and test sets each contain 5% and 10% of links.

Table 1: Dataset Summary.

Dataset	Nodes	Edges	Features	Feature Type
CORA	2708	5429	1433	$\{0, 1\}$
CiteSeer	3327	4732	3703	$\{0, 1\}$
PubMed	19717	44338	500	$(0, 1)$
BlogCatalog	5196	171743	8189	$\mathbb{Z}^+$
Coauthor CS	18333	81894	6805	$\mathbb{Z}^+$
Coauthor Physics	34493	247962	8415	$\mathbb{Z}^+$
DBLP	17725	105781	768	$\mathbb{R}$

5.2 Models

For semi-supervised node classifcation task, we compare the performance of HDGL with that of several strong baselines Logistic Regression (LogReg), DeepWalk [38] and Label Propagation [60]. We also directly compare HDGL with state-of-the-art models: Graph Convolutional Network (GCN) [24], Graph Attention Network (GAT) [49], and lastly, a fast GNN model, namely, Simplified Graph Convolutional Network (SGC) [53] which is essentially a logistic regression model trained using features extracted from k-hop neighborhoods of nodes as input. We also assess our model against RelHD [22], an existing HD algorithm used for node classification

For the transductive link prediction task, we evaluate our model against standard link prediction baselines:- DeepWalk [38] and Spectral clustering (SC) [44]. We also compare our model against graph autoencoder models:- Graph Autoencoder (GAE) and Variational Graph Autoencoder (VGAE) from Kipf and Welling [23].

5.3 HDGL Model Specifications

To ensure that HDGL are directly comparable with most GNN architectures which typically use 2 layers, and hence aggregate information over 2-hop neighbors, we limit HDGL to aggregate information over at most 2-hops i.e HD node representations are calculated as per Eq.(2).

Because Bundling of an even number of HD vectors can result in ties that would need to be broken using tie-breaking policy (See Section 3.2), we opt to randomly sample odd number of neighbors, specifically, 11 of the 1-hop neighbors and 21 of the 2-hop neighbors during the construction of HD node representations by HDGL for datasets except BlogCatalog. For BlogCatalog we sample 21 of the 1-hop neighbors and 51 of the 2-hop neighbors since Blogcatalog graph is denser than the other datasets.

For node classification in CORA, CiteSeer, and BlogCatalog datasets, we map node features to $50,000$-dimensional HD vectors through random projections. Similarly, for the PubMed, Coauthor CS, Coauthor Physics, and DBLP datasets, we map node features to $20,000$-dimensional HD vectors using the same method. While these choices may seem arbitrary, it's crucial to note that selecting dimensions between $20,000$ and $50,000$ typically suffices to maintain the desired properties of HD-computing. This assertion finds extensive support in the research from Kanerva [20, 21].

5.4 Training and Evaluation

Iterative machine learning methods like GNN use separate validation data to stop training and tune hyperparameters, alongside training data. In contrast, methods such as HDGL which involve no iterative training do not need validation data. Hence, to ensure fair comparison, HDGL uses training and validation sets together for learning, whereas GNN use training data for training the model, validation data for early stopping and hyperparameter tuning.

In the case of GNN methods that rely on iterative training, we choose the models with hyperparameters tuned to obtain optimal performance on validation data. We used grid search over 12 different configurations for tuning these hyperparameters. For GCN and GAT, we optimized the following hyperparameters: the number of hidden units in the first layer, learning rate, and weight decay. Specifically, GCN models were tested with 32 and 64 hidden units, while GAT models were tested with 4 and 8 attention heads, each having 8 hidden units. Both GCN and GAT models were evaluated with weight decays of 1e-3, 1e-4, and 5e-4. SGC, which does not have hidden layers, had its hyperparameters tuned for learning rate and weight decay. The weight decay tuning of SGC was over the following values 1e-3, 5e-4, 1e-4, 5e-5, 1e-5, and 5e-6.

GCN, GAT, and SGC models were tuned with learning rates of 1e-2 and 1e-3, using early stopping criteria set to 25 epochs for the 1e-2 learning rate and 50 epochs for the 1e-3 learning rate.

As LogReg, DeepWalk, and Label Prop represent well-established baselines, we do not conduct a hyperparameter search. For Label Propagation, we use 20 iterations of propagation (except for blogcatalog dataset where we use 10) and set the alpha factor [1] to 0.5. For DeepWalk, we utilize the DGL library, setting the embedding dimension to 32 and the number of walks to 20, with a batch size of 64. We sample such walks for all nodes in the graph and train DeepWalk model for 10 epochs and then employ logistic regression for classification, incorporating early stopping based on the validation set.

Following Kang et al. [22], the RelHD algorithm was employed with 10,000 dimensional random basis HD vectors, as the performance of RelHD plateaus around this dimensionality.

---

[1]Alpha factor in Label Propa determines the balance between existing and propagated labels. Higher alpha prioritizes existing labels; lower alpha favors propagated ones.

Table 2: Average Test Set Accuracy and Standard Deviation (in Percentage) Over 10 Random Weight Initializations for Deep Learning Models and 10 Randomized Instantiations of RelHD and HDGL Across Datasets (* denotes datasets with $\{0, 1\}$ features. In these datasets, RelHD performs comparably to HDGL)

Method	Cora*	Citeseer*	Pubmed	BlogCat	CompSci	Physics	DBLP
LogReg	52.2 ± 0.5	46.3 ± 0.4	67.9 ± 0.5	65.3 ± 1.0	86.8 ± 0.6	88.2 ± 0.9	56.2 ± 3.0
DeepWalk	65.3 ± 1.2	46.2 ± 0.9	69.3 ± 0.9	58.6 ± 1.3	78.1 ± 0.9	86.4 ± 0.9	66.1 ± 2.2
Label Prop	70.9 ± 0.0	47.7 ± 0.0	71.3 ± 0.0	47.8 ± 3.8	76.6 ± 0.9	85.6 ± 1.1	66.4 ± 1.6
GCN	81.9 ± 0.6	70.3 ± 0.5	79.0 ± 0.4	69.5 ± 0.4	91.0 ± 0.7	92.3 ± 0.6	72.1 ± 0.8
GAT	82.8 ± 0.5	72.0 ± 0.6	79.1 ± 0.3	65.3 ± 1.2	90.9 ± 0.6	92.5 ± 0.9	75.4 ± 1.0
SGC	79.8 ± 0.5	68.5 ± 0.4	75.4 ± 0.5	70.8 ± 0.3	89.1 ± 0.6	92.1 ± 0.7	76.3 ± 0.9
RelHD	78.9 ± 0.5	69.3 ± 0.4	69.1 ± 0.4	60.9 ± 1.2	85.5 ± 0.8	85.3 ± 0.9	44.9 ± 0.2
HDGL	79.5 ± 0.7	70.0 ± 0.6	76.8 ± 1.1	69.3 ± 0.5	88.9 ± 0.4	91.0 ± 0.9	68.7 ± 1.2

We report measured performance on the test data, averaged across 10 random selections of pertinent parameters. These parameters include the random hyperplanes employed for constructing high-dimensional representations used by HDGL, random basis vector initializations for RelHD, weight initializations of the deep learning models, and data splits for datasets such as Coauthor Physics, Coauthor CS, BlogCatalog, and DBLP. For Cora, Citeseer, and Pubmed, we use predefined splits given in Yang et al. [57].

For the link prediction task, we rely on the results directly obtained from the paper for GVAE from Kipf and Welling [23].
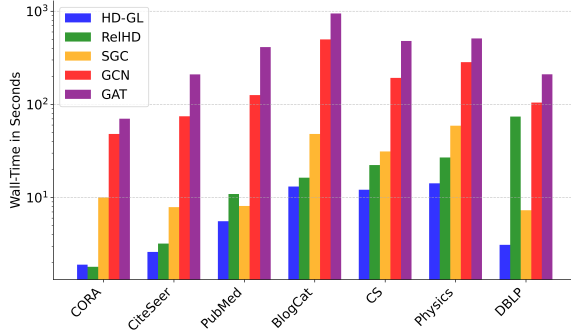

Figure 2: Comparison of Learning Times (in seconds) i.e Across Datasets for HDGL and RelHD with GCN, GAT and SGC with Hyperparameter Tuning with Exploration of Search Space with 12 configurations. Runtime for HDGL and RelHD exclusively includes only time Elapsed for learning label representations using train/validation data.

5.5 Implementation Details

All experiments were conducted on Google Colab Pro with High-memory CPU. To ensure fair comparison, although not ideal for HDGL, we used the PyTorch framework [37], complemented by the Deep Graph Library (DGL) [50] for importing datasets and implementing deep learning models. Due to the lack of support for bitwise majority operations in PyTorch, we convert our vectors to the bipolar space and utilize signed addition for compatibility. As mentioned in Section 3.2, HD computing works in both binary and bipolar spaces, allowing us to switch between them as necessary. None of our experiments used parallel execution or GPUs. We have also implemented PyTorch version of RelHD, as the experiments detailed in Kang et al. [22] were reliant on hardware architecture implementations. We undertake this effort to compare with our model and use RelHD as a baseline in this study.

5.6 Results

Node Classification Performance. The results of our comparison of HDGL with the other methods are shown in Table 2. We observe that HDGL without the need for computationally expensive iterative training or hyperparameter optimization achieves node prediction accuracy that is competitive with those of state-of-the-art GNN models (GCN, GAT, SGC) across most of the benchmark data sets, in each case reaching performance that is within 1-2% of the best performing model. HDGL also consistently outperforming basic deep learning baseline methods (LogReg, DeepWalk, Label-Prop). The notable advantage of HDGL over RelHD lies in its ability to accommodate features beyond the binary $\{0, 1\}$ range, as evident from the results presented in Table 2, where HDGL significantly outperforms. However, when dealing with datasets featuring $\{0, 1\}$ attributes like CORA and Citeseer, the performance of HDGL and RelHD appears nearly identical.

Node Classification Run-times. Figure 2 compares the run-time for the node classification task between HDGL and RelHD (both of which requires no hyperparameter tuning) with those of GCN, GAT, and SGC (with hyperparameter tuning), alongside the run-time of HDGL. We observe that, the run-time of HDGL is substantially less than that of all the state-of-the-art GNNs. While in most cases, RelHD exhibits slightly or significantly worse run times compared to HDGL, it's crucial to note that HDGL consistently delivers superior performance across various datasets when compared to RelHD.

It is important to note that although we used grid search over 12 configurations for hyperparameter optimization, typically deep learning models explore a much larger parameter space. This expanded search naturally comes with increased runtime, especially when GPUs are utilized, further widening the energy efficiency gap between Graph Neural Networks (GNN) and traditional deep learning models like HDGL.

Link Prediction Performance. Even though HDGL representations are primarily tailored for transductive node classification, their performance for link prediction is comparable to that of Deep-Walk and Spectral clustering; as seen from Table 4. However, they do not reach the same level of effectiveness as GAE and VGAE, which leverage GNN methods. Despite lacking iterative training like deep learning methods, HDGL still manages to achieve respectable results in link prediction. Moreover, unlike deep learning approaches that typically require distinct models for link and node classification, HDGL tackles both tasks within a unified framework.

Table 3: Class Incremental Node Labeling Results. No hyperparameter grid search is used for GCN and SGC algorithms. Wall-time (in seconds) includes both training and inference of the model. HDGL does not require retraining from $t = 2$.

Model		BlogCatalog					Coauthor Physics			
		t = 1	t = 2	t = 3	t = 4	t = 5	t=1	t=2	t=3	t=4
GCN	Accuracy	94.94	85.81	75.16	74.82	69.61	98.95	97.33	95.57	92.58
	Runtime	51.2s	18.7s	16.5s	33.9s	23.9s	65.9s	62.3s	65.8s	51.2s
SGC	Accuracy	94.56	82.31	73.23	72.72	69.81	98.94	97.39	95.36	91.81
	Runtime	6.3s	3.72s	3.43s	3.73s	3.6s	9.1s	8.5s	8.8s	9.7s
HDGL	Accuracy	89.11	88.49	75.56	71.73	69.91	98.56	94.87	92.53	90.8
	Runtime	20.2s	0.3s	0.4s	0.4s	0.6s	38.3s	1.5s	1.7s	1.7s

We also perform link prediction experiments by varying the dimensionality of the HDGL model. This exploration arises from observing a slight yet discernible variation in performance for link prediction tasks, unlike the more consistent outcomes observed in node classification experiments. As mentioned earlier, typically, dimensions ranging from 20,000 to 50,000 are sufficient for HD properties to hold. However, higher dimensionality generally leads to better performance. Nevertheless, beyond a certain threshold, the performance tends to plateau, as illustrated in Figure 3.

Table 4: Link prediction results averaged over 10 Randomized Instantiations of HDGL and datasplits. Results for other models are from Kipf and Welling [23] paper.

Methods	Cora		Citeseer		PubMed	
	AUC	AP	AUC	AP	AUC	AP
DeepWalk	0.831	0.850	0.805	0.836	0.844	0.841
SC	0.846	0.885	0.791	0.826	0.849	0.888
GAE	0.910	0.920	0.895	0.899	0.964	0.965
VGAE	0.914	0.926	0.908	0.920	0.944	0.947
HDGL	0.849	0.880	0.768	0.842	0.853	0.884
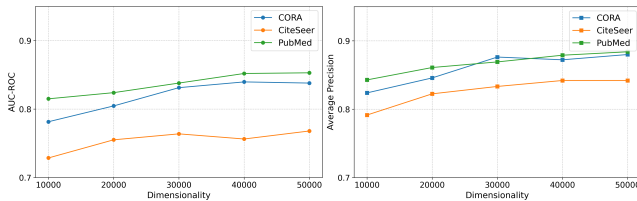
Figure 3: Link Prediction Performance of HDGL under Various Dimensionality Configurations.

HDGL in the Class-Incremental Learning Setting Consider a scenario with a dynamic graph where nodes and edges remain static, but the new node labels appear over time i.e initially, only a subset of nodes is labeled, belonging to two classes. Over time, new nodes receive labels; thus, introducing new classes. This is a node labeling instance of the class-incremental learning problem [2, 47, 48]. In this setting, deep learning methods typically require retraining on the data for previously learned classes along with the data for the new classes to avoid catastrophic forgetting of the previously learned classes [47]. This retraining incurs significant additional runtime and computation.

In contrast to deep neural networks, which necessitate retraining, Hyperdimensional Graph Learner (HDGL) operates without an iterative training phase. Instead, it constructs label hypervectors using predefined latent node representations. This approach

eliminates the need for retraining, resulting in substantial savings in both runtime and computation costs.

To simulate incremental node classification on a fully labeled graph, we use BlogCatalog and Coauthor Physics Dataset. We first split the nodes into train/val/test as in Shchur et al. [41]. We mask the train/val/test such that at time $t = 1$ only the labels $\{1, 2\}$ are available; and at $t = 2$ only the labels $\{1, 2, 3\}$ are available; and so on such that at the final time step, all labels $\{1, \cdots, L\}$ are available.

To simplify matters for GNN models, we assume that the models are provided the list of L possible labels although only a subset have been encountered in the training data. This allows the last layer of the GNN to have L outputs. Also, we forego hyperpameter search for the GNN models to minimize the computational overhead.

In comparing HDGL against GCN and SGC in our experimental setup, we opted not to include the GAT model due to its longer runtime, as depicted in Figure 2. As seen from Table 3, we see that HDGL incurs a significant computational overhead only at timestep 1, necessitating the computation of node representations followed by inference. However, for subsequent timesteps, HDGL leverages previously computed node representations, requiring only inference. In contrast, deep learning models like GCN and SGC mandate retraining for each timestep to accommodate new labels, resulting in comparatively slower runtimes.

6 Conclusion

We introduced HDGL, a novel transductive learning algorithm for graphs using hyperdimensional representations. HDGL offers a computationally efficient alternative to graph neural networks by leveraging the *injectivity* property of node representations from Graph Neural Networks (GNNs). It uses HD operators such as bundling and binding to aggregate information from the local neighborhood of each node. The resulting latent node representations support both node classification and link prediction tasks. Furthermore, HDGL eliminates the need for iterative training, making it ideal for class-incremental learning and applications requiring high accuracy models at lower computational cost and learning time compared to traditional graph neural network methods. The advantages and efficacy of HDGL are validated by comprehensive experiments comparing it with state-of-the-art GNN methods.

Acknowledgments

References

[1] Subutai Ahmad and Jeff Hawkins. 2015. Properties of Sparse Distributed Representations and their Application to Hierarchical Temporal Memory. arXiv:1503.07469

[2] Eden Belouadah, Adrian Popescu, and Ioannis Kanellos. 2021. A comprehensive study of class incremental learning algorithms for visual tasks. *Neural Networks* 135 (2021), 38–54.

[3] Sven Beyer, Stefan Dünkel, Martin Trentzsch, Johannes Müller, Andreas Hellmich, Dirk Utess, Jan Paul, Dominik Kleimaier, John Pellerin, Stefan Müller, Johannes Ocker, Antoine Benoist, Haidi Zhou, Menno Mennenga, Martin Schuster, Fabio Tassan, Marko Noack, Ali Pourkeramati, Franz Müller, Maximilian Lederer, Tarek Ali, Raik Hoffmann, Thomas Kämpfe, Konrad Seidel, Halid Mulaosmanovic, Evelyn T. Breyer, Thomas Mikolajick, and Stefan Slesazeck. 2020. FeFET: A versatile CMOS compatible device with game-changing potential. *2020 IEEE International Memory Workshop (IMW)* (2020), 1–4.

[4] Sergey Brin and Lawrence Page. 1998. The Anatomy of a Large-Scale Hypertextual Web Search Engine. *Computer Networks* 30 (1998), 107–117. http://www-db.stanford.edu/~backrub/google.html

[5] Hongyun Cai, Vincent W Zheng, and Kevin Chen-Chuan Chang. 2018. A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE transactions on knowledge and data engineering* 30, 9 (2018), 1616–1637.

[6] Moses S. Charikar. 2002. Similarity Estimation Techniques from Rounding Algorithms. In *Proceedings of the Thiry-Fourth Annual ACM Symposium on Theory of Computing* (Montreal, Quebec, Canada) *(STOC '02)*. Association for Computing Machinery, New York, NY, USA, 380–388. https://doi.org/10.1145/509907.509965

[7] Alex de Vries. 2023. The growing energy footprint of artificial intelligence. *Joule* 7, 10 (2023), 2191–2194.

[8] Radosvet Desislavov, Fernando Martínez-Plumed, and José Hernández-Orallo. 2023. Trends in AI inference energy consumption: Beyond the performance-vs-parameter laws of deep learning. *Sustainable Computing: Informatics and Systems* 38 (2023), 100857.

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:1810.04805 [cs.CL] https://arxiv.org/abs/1810.04805

[10] Sjoerd Dirksen, Shahar Mendelson, and Alexander Stollenwerk. 2022. Sharp Estimates on Random Hyperplane Tessellations. *SIAM Journal on Mathematics of Data Science* 4, 4 (2022), 1396–1419. https://doi.org/10.1137/22M1485826

[11] P. Frasconi, M. Gori, and A. Sperduti. 1998. A general framework for adaptive processing of data structures. *IEEE Transactions on Neural Networks* (1998).

[12] Ross W. Gayler. 1998. Multiplicative Binding, Representation Operators and Analogy (Workshop Poster). (01 1998).

[13] Lulu Ge and Keshab K Parhi. 2020. Classification using hyperdimensional computing: A review. *IEEE Circuits and Systems Magazine* 20, 2 (2020), 30–47.

[14] C. Lee Giles, Kurt D. Bollacker, and Steve Lawrence. 1998. CiteSeer: An Automatic Citation Indexing System. In *Proceedings of the Third ACM Conference on Digital Libraries* (Pittsburgh, Pennsylvania, USA) *(DL '98)*. Association for Computing Machinery, New York, NY, USA, 89–98. https://doi.org/10.1145/276675.276685

[15] Marco Gori, Gabriele Monfardini, and Franco Scarselli. 2005. A new model for learning in graph domains. In *IEEE International Joint Conference on Neural Networks*.

[16] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. *Advances in neural information processing systems* (2017).

[17] Ningyuan Teresa Huang and Soledad Villar. 2021. A short tutorial on the weisfeiler-lehman test and its variants. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 8533–8537.

[18] Xiao Huang, Jundong Li, and Xia Hu. 2017. Label Informed Attributed Network Embedding. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining* (Cambridge, United Kingdom) *(WSDM '17)*. Association for Computing Machinery, New York, NY, USA, 731–739. https://doi.org/10.1145/3018661.3018667

[19] Piotr Indyk and Rajeev Motwani. 1998. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing* (Dallas, Texas, USA) *(STOC '98)*. Association for Computing Machinery, New York, NY, USA, 604–613. https://doi.org/10.1145/276698.276876

[20] Pentti Kanerva. 1988. *Sparse Distributed Memory*. The MIT Press.

[21] Pentti Kanerva. 2009. Hyperdimensional Computing: An Introduction to Computing in Distributed Representation with High-Dimensional Random Vectors. *Cognitive Computation* (2009).

[22] Jaeyoung Kang, Minxuan Zhou, Abhinav Bhansali, Weihong Xu, Anthony Thomas, and Tajana Rosing. 2022. RelHD: A Graph-based Learning on FeFET with Hyperdimensional Computing. In *2022 IEEE 40th International Conference on Computer Design (ICCD)*. 553–560. https://doi.org/10.1109/ICCD56317.2022.00087

[23] Thomas N Kipf and Max Welling. 2016. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308* (2016).

[24] Thomas N Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*.

[25] Denis Kleyko, Dmitri A. Rachkovskij, Evgeny Osipov, and Abbas Rahimi. 2023. A Survey on Hyperdimensional Computing aka Vector Symbolic Architectures, Part I: Models and Data Transformations. *ACM Comput. Surv.* 55, 6 (2023), 130:1–130:40. https://doi.org/10.1145/3538531

[26] Denis Kleyko, Dmitri A. Rachkovskij, Evgeny Osipov, and Abbas Rahimi. 2023. A Survey on Hyperdimensional Computing aka Vector Symbolic Architectures, Part II: Applications, Cognitive Models, and Challenges. *ACM Comput. Surv.* 55, 9 (2023), 175:1–175:52. https://doi.org/10.1145/3558000

[27] Ajay Kumar, Shashank Sheshar Singh, Kuldeep Singh, and Bhaskar Biswas. 2020. Link prediction techniques, applications, and performance: A survey. *Physica A: Statistical Mechanics and its Applications* 553 (2020), 124289.

[28] Bentian Li and Dechang Pi. 2020. Network representation learning: a systematic literature review. *Neural Computing and Applications* 32, 21 (2020), 16647–16679.

[29] Haomin Li, Fangxin Liu, Yichi Chen, and Li Jiang. 2023. HyperNode: An Efficient Node Classification Framework Using HyperDimensional Computing. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. 1–9. https://doi.org/10.1109/ICCAD57390.2023.10323813

[30] Andrew Kachites McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. 2000. Automating the construction of internet portals with machine learning. *Information Retrieval* 3 (2000), 127–163.

[31] Justin Morris, Kazim Ergun, Behnam Khaleghi, Mohen Imani, Baris Aksanli, and Tajana Simunic. 2022. HyDREA: Utilizing Hyperdimensional Computing for a More Robust and Efficient Machine Learning System. *ACM Transactions on Embedded Computing Systems* 21, 6 (2022), 1–25.

[32] Onur Mutlu, Saugata Ghose, Juan Gómez-Luna, and Rachata Ausavarungnirun. 2022. A modern primer on processing in memory. In *Emerging Computing: From Devices to Systems: Looking Beyond Moore and Von Neumann*. Springer, 171–243.

[33] Galileo Namata, Ben London, Lise Getoor, and Bert Huang. 2012. Query-driven Active Surveying for Collective Classification.

[34] Peer Neubert, Stefan Schubert, and Peter Protzel. 2019. An introduction to hyperdimensional computing for robotics. *KI-Künstliche Intelligenz* 33 (2019), 319–330.

[35] Igor Nunes, Mike Heddes, Tony Givargis, Alexandru Nicolau, and Alexander V. Veidenbaum. 2022. GraphHD: Efficient graph classification using hyperdimensional computing. *2022 Design, Automation & Test in Europe Conference & Exhibition* (2022), 1485–1490.

[36] Shirui Pan, Jia Wu, Xingquan Zhu, Chengqi Zhang, and Yang Wang. 2016. Tri-Party Deep Network Representation. In *International Joint Conference on Artificial Intelligence*. https://api.semanticscholar.org/CorpusID:943660

[37] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* 32 (2019).

[38] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 701–710.

[39] Prathyush Poduval, Haleh Alimohamadi, Ali Zakeri, Farhad Imani, M. Hassan Najafi, Tony Givargis, and Mohsen Imani. 2022. GrapHD: Graph-Based Hyperdimensional Memorization for Brain-Like Cognitive Learning. *Frontiers in Neuroscience* 16 (2022). https://doi.org/10.3389/fnins.2022.757125

[40] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. 2009. The Graph Neural Network Model. *IEEE Transactions on Neural Networks* (2009).

[41] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. 2018. Pitfalls of Graph Neural Network Evaluation. *Relational Representation Learning Workshop, NeurIPS* (2018).

[42] A. Sperduti and A. Starita. 1997. Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks* (1997).

[43] Emma Strubell, Ananya Ganesh, and Andrew McCallum. 2020. Energy and policy considerations for modern deep learning research. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 34. 13693–13696.

[44] Lei Tang and Huan Liu. 2011. Leveraging social media networks for classification. *Data Mining and Knowledge Discovery* 23 (2011), 447–478.

[45] Anthony Thomas, Sanjoy Dasgupta, and Tajana Rosing. 2021. A theoretical perspective on hyperdimensional computing. *Journal of Artificial Intelligence Research* 72 (2021), 215–249.

[46] Neil C. Thompson, Kristjan Greenewald, Keeheon Lee, and Gabriel F. Manso. 2021. Deep Learning's Diminishing Returns: The Cost of Improvement is Becoming Unsustainable. *IEEE Spectrum* 58, 10 (2021), 50–55. https://doi.org/10.1109/MSPEC.2021.9563954

[47] Songsong Tian, Lusi Li, Weijun Li, Hang Ran, Xin Ning, and Prayag Tiwari. 2024. A survey on few-shot class-incremental learning. *Neural Networks* 169 (2024), 307–324.

[48] Gido M Van de Ven, Tinne Tuytelaars, and Andreas S Tolias. 2022. Three types of incremental learning. *Nature Machine Intelligence* 4, 12 (2022), 1185–1197.

[49] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *International Conference on Learning Representations*.

[50] Minjie Wang, Lingfan Yu, Da Zheng, Quan Gan, Yujie Gai, Zihao Ye, Mufei Li, Jinjing Zhou, Qi Huang, Chao Ma, Ziyue Huang, Qipeng Guo, Haotong Zhang, Haibin Lin, Junbo Jake Zhao, Jinyang Li, Alex Smola, and Zheng Zhang. 2019. Deep Graph Library: Towards Efficient and Scalable Deep Learning on Graphs. *ArXiv* (2019).

[51] Xiao Wang, Deyu Bo, Chuan Shi, Shaohua Fan, Yanfang Ye, and S Yu Philip. 2022. A survey on heterogeneous graph embedding: methods, techniques, applications and sources. *IEEE Transactions on Big Data* 9, 2 (2022), 415–436.

[52] Boris Weisfeiler and Andrei Leman. 1968. The reduction of a graph to canonical form and the algebra which appears therein. *nti, Series* 2, 9 (1968), 12–16.

[53] Felix Wu, Tianyi Zhang, Amauri H. de Souza, Christopher Fifty, Tao Yu, and Kilian Q. Weinberger. 2019. Simplifying Graph Convolutional Networks. In *International Conference on Machine Learning*.

[54] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. 2020. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems* 32, 1 (2020), 4–24.

[55] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How Powerful are Graph Neural Networks?. In *International Conference on Learning Representations*. https://openreview.net/forum?id=ryGs6iA5Km

[56] Mengjia Xu. 2021. Understanding graph embedding methods and their applications. *SIAM Rev.* 63, 4 (2021), 825–853.

[57] Zhilin Yang, William W. Cohen, and Ruslan Salakhutdinov. 2016. Revisiting Semi-Supervised Learning with Graph Embeddings. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48* (New York, NY, USA) *(ICML'16)*. JMLR.org, 40–48.

[58] Daokun Zhang, Jie Yin, Xingquan Zhu, and Chengqi Zhang. 2018. Network representation learning: A survey. *IEEE transactions on Big Data* 6, 1 (2018), 3–28.

[59] Jingya Zhou, Ling Liu, Wenqi Wei, and Jianxi Fan. 2022. Network representation learning: from preprocessing, feature extraction to node embedding. *ACM Computing Surveys (CSUR)* 55, 2 (2022), 1–35.

[60] Xiaojin Zhu and Zoubin Ghahramani. 2002. Learning from labeled and unlabeled data with label propagation. https://api.semanticscholar.org/CorpusID:15008961