Learning Hyperbolic Representations in Real-World Networks

Manan Shah
manans@stanford.edu

Sagar Maheshwari
msagar@stanford.edu

Abstract

Graphs are inherently complex structures, and learning suitable representations from these dynamic networks for downstream prediction tasks is a canonical problem with great practical implications. Indeed, the generation of node embeddings that accurately capture the structure and high dimensionality of a graph may prove incredibly useful in the development of models for traditional prediction and classification tasks across nodes and edges. In this work, we implement and analyze numerous benchmark frameworks to learn representations on graphs. We further develop intuition for the use of hyperbolic embeddings, frameworks that claim to improve node-level embeddings with little verification in the status quo. In particular, our work aims to advance understanding of representation learning in hyperbolic spaces alongside its benefits and deficits when compared to traditional methods on large-scale graphs. We provide three significant contributions to the field:

1. The generation of baselines for hyperbolic embeddings against standardized embedding frameworks on networks with varying hyperbolicity as well as real-world networks, bolstering limited empirical findings in current literature.
2. The development and empirical evaluation of a novel algorithm premised on node2vec called HYPERWALK that incorporates intuition from Poincaré hyperbolic embeddings to generate and embed random walks in hyperbolic space.
3. The development and empirical evaluation of a novel algorithm premised on graph convolutional networks called HYPERCONV that performs node-level feature aggregation and encoding in hyperbolic space.

We provide an open-source implementation of our findings, embedding frameworks, and visualization methods at `www.github.com/mananshah99/hyperbolic`. Although our repository contains code shared between project reports produced for CS 229 and CS 224W, all results and methods presented in this work are solely for CS 224W.

1 Introduction

Graphs are incredibly powerful tools to encode relationships between diverse objects and process unstructured real-world data. However, while there are numerous benefits to such complexity, the diverse nature of graphs has resulted in difficulty analyzing networks with varying size, shape, and structure. Furthermore, conducting downstream prediction tasks (e.g. regression or hierarchy generation) from arbitrary nodes, edges, or substructures in a graph requires fixed-dimensional representations of these structures, a task which is made nontrivial due to the arbitrarily complex nature of networks.

The generation of embeddings for nodes in a graph provides an effective and efficient way to approach the analysis of graphical structures while retaining the diversity encoded in the graph itself. Precisely, given graph $G(V, E)$, the canonical formalization of the node embedding task is to learn a function $f|_{v \in V}$ such that $f : v \to X^d$ for some metric space (X, s) equipped with suitable metric s, and dimension d. A suitably learned embedding function f has a desired property of mapping similar nodes close to one another in the embedding space X^d. The resulting fixed-dimensional embeddings for nodes can be readily used as input feature matrices for downstream tasks, emphasizing the importance of properly capturing graph structure in embedding generation.

Numerous methods have been previously proposed to learn $f : v \rightarrow \mathbb{R}^d$ in both supervised and unsupervised manners. Inspired by the critical insight of DeepWalk [10] to treat random walks on graphs as sentences similar to those used in language models, node2vec [3] generalized the concept of random walks to include neighborhood structures in embedding generation. More recently, semi-supervised graph convolutional networks (GCNs) [5] have been developed to embed nodes using spectral graph convolutions. In stark contrast with these methods, however, are node embedding frameworks premised on Poincaré [9] and Lorentz [8] embeddings. Arguing that state-of-the-art embedding methods do not account for latent hierarchical structures in complex networks, the authors of these recent papers suggest that learning $f : v \rightarrow \mathcal{H}^d$ is more suitable for ensuring these structures are represented in the embedding vectors.

In this work, we analyze the efficacy of hyperbolic embeddings on standardized datasets as well as real-world data to enhance the current lack of empirical findings on these fronts. We further develop two novel methods (HYPERWALK and HYPERCONV) for embedding networks in hyperbolic space that serve as extensions to approaches based on random walks and graph convolutions. Both HYPERWALK and HYPERCONV outperform their Euclidean counterparts on numerous evaluation tasks, and embedding visualizations via t-SNE and PCA confirm the benefits of extending classical methods to hyperbolic space.

We begin in Section 2 by detailing the prior work with embedding graphs in both Euclidean and hyperbolic space. In particular, we emphasize Euclidean embedding methods including DeepWalk, node2vec, and graph convolutional networks (Section 2.1), and we emphasize hyperbolic embedding methods that embed nodes by utilizing the structure of Poincaré balls (Section 2.2). We further develop intuition for the hyperbolicity of graphs in Section 2.2, providing insight into the benefits of hyperbolic embeddings and paving the way for experimental evaluations of the efficacy of hyperbolic-based frameworks. We detail the implementation of baseline embedding frameworks alongside our proposed HYPERWALK and HYPERCONV frameworks, developing intuition where necessary. We provide background for the datasets used in Section 4, and we present our main results (including empirical evaluations of Poincaré embeddings and evaluations of HYPERWALK and HYPERCONV) in Section 5. We conclude with a foray into potential future work in Section 6.

2 Prior Work

2.1 Euclidean Embeddings

We first detail methods that generate node embeddings in Euclidean space. While DeepWalk and node2vec share a common paradigm in treating random walks as sentences with nodes as words, the graph convolutional network (GCN) framework employs a significantly different approach to embedding generation.

Perozzi et al. [10] develops a method to generate node embeddings by treating random walks as sentences in linguistic models. Inspired by the widespread success of language models in learning latent distributed representations from words, Perozzi et al. replicated a similar modeling framework for networks titled DeepWalk. In particular, due to the empirical observation that words in a sentence and random walks both follow similar power-law distributions, DeepWalk relied on the intuition that random walks could be modeled as sentences in a particular language. Analogous to natural language modeling frameworks that estimate the probability that a word will appear in a sentence, DeepWalk estimates the probability that a vertex appears in a random walk by learning feature vectors $f_v \mid v \in V$. For scalability purposes, the authors limit prediction tasks to predicting the nearest $2w$ neighbors of vertex v_i (where w is some constant) and use hierarchical softmax to approximate the conditional probabilities during gradient descent.

Grover et al. [3] generalizes DeepWalk by utilizing both the current node and its predecessor to identify the next node in a random walk. By tuning two parameters $p, q \in \mathbb{R}$ where the former loosely controls depth-first transitions and the latter breadth-first behavior, Grover et al's approach—titled node2vec—is able to interpolate between different kinds of neighborhoods to better incorporate graph structure in embedding generation. node2vec conducts a similar optimization procedure as DeepWalk, sampling vertices $v \in V$, conducting random walks, and updating feature vectors in gradient descent. Due to its ability to smoothly characterize networks according to both the homophily and structural hypotheses for neighborhood formation, node2vec successfully improves performance over DeepWalk across numerous varied benchmarks.

DeepWalk and node2vec—methods similar in their intuition and methodology involving using random walks to learn network structure—perform well at generating fixed-dimensional feature vectors from graphs. However, they suffer from numerous drawbacks due to their inability to utilize node attributes, assumptions regarding conditional independence and symmetry in feature space, and failure to encode highly non-linear properties of networks. More specifically, in the generation of feature vectors by considering random walks across graphs, node2vec and

DeepWalk ignore vertex attributes, instead using vertex IDs to track the progress of walks on the graph. While walks provide information valuable regarding graph structure, node-level features may provide critical information critical to discovering communities related by attributes independent of graph connections. In order to incorporate such information while preserving local node structure in graphs, graph convolutional networks were motivated.

Kipf et al. [5] employs a recently developed approach to graph embedding, deviating significantly from the random walk-based approaches of Perozzi et al. and Grover et al. to provide an end-to-end approach to representation learning from graphs. As canonical formulations of convolutional networks rely on assumptions regarding the neighborhood structure of input data (e.g. pixels), alterations must be made to the traditional convolutional paradigm to account for the dynamic structure of graphs. Precisely, Kipf et al. use layers of spectral filters that, in aggregate, are able to capture high order features from graphs. Demarcating their model as a graph convolutional network (GCN), Kipf et al. train their framework in a semi-supervised manner by assuming labels are provided for a small subset of nodes in an input graph. By developing a well-behaved layer propagation rule utilizing spectral filters, the GCN encodes graph structure explicitly by learning a function $f(X, A)$, where X is a matrix of learned node features and A is the adjacency matrix representation of G. A starkly distinguished approach from DeepWalk and node2vec, the representations learned with GCNs outperform both earlier models on numerous benchmark datasets. A recent extension of stochastic graph convolutions for large-scale inductive representation learning was developed in [4]; our methods were built to extend the framework presented therein.

Graph convolutional networks therefore allow for the inductive learning of features from graphs while leveraging node-level information as opposed to the transductive feature learning paradigm of node2vec and DeepWalk. However, both graph convolutional methods as well as random-walk based approaches currently generate node-level embeddings in Euclidean space, thereby failing to leverage the structural benefits obtained from the representation of graphs in a Poincaré ball (as elucidated in [9] and [8]).

2.2 Hyperbolic Embeddings and Graph Hyperbolicity

The development of node-level embeddings in hyperbolic space has recently yielded promising results ([9], [8]). Hyperbolic space, which uses non-Euclidean geometries, is characterized by its constant negative curvature. Due to this curvature, distances increase exponentially away from the origin and give hyperbolic spaces the name of continuous analogues of trees. Specifically, the distance between nodes in a particular level of a tree increases exponentially as the level increases. In hyperbolic space, such a tree can be represented in simply two dimensions: nodes of a certain level all lie on sphere of a certain radius, within which lie nodes of lower levels. Moreover, unlike Euclidean space which is uniquely characterized by its namesake model, hyperbolic space can be characterized by multiple models, including the Beltrami-Klein and hyperboloid models. Regardless of the model, hyperbolic spaces are critically relevant to model hierarchical data. Due to its mathematical underpinnings, distance in hyperbolic space can be used to infer both node hierarchy and similarity, making hyperbolic embeddings especially useful in unsupervised learning settings.

Nickel et al. [9] examines the task of learning hierarchical representations of data by embedding in hyperbolic space; more specifically, the n-dimensional Poincaré ball. Through empirical studies, Nickel et al. discover that Euclidean embeddings are limited in their ability to model intricate patterns in network structures due to the need for high dimensions. Motivated by this finding, the authors formulate a model for generating node embeddings in an n-dimensional Poincaré ball. Of the many hyperbolic space models, the authors use the Poincare model due to its ease-of-use in gradient-based optimization: nodes are initially given random embeddings, which are then updated by minimizing a loss function using Riemannian Stochastic Gradient Descent. The authors use this procedure to generate node embeddings for the WordNet dataset and a collaboration network to compare performance to Euclidean embeddings on the tasks of lexical entailment and link prediction, respectively. In both tasks, Poincaré embeddings were able to outperform Euclidean embeddings, even with significantly lower dimensions.

Hyperbolic embeddings excel in their encoding of both node similarity and node hierarchy, which allows for unsupervised identification of latent hierarchical structures. This feature is not present in semi-supervised GCN embeddings, which require a training dataset to be able to properly encode node similarity. Furthermore, hyperbolic embeddings are more scalable and parallelizable than GCN embeddings due to the nature of Riemannian optimization. In spite of these benefits, however, the field of hyperbolic embeddings is relatively new and as such has significant room for improvement. In particular, all datasets used in [9] and [8] have explicit hierarchical structure, making for promising node embedding structures on the Poincaré disk. In particular, the results of Nickel et al. suggest that hyperbolic embeddings result in better predictive performance for tasks such as lexical entailment and link prediction when generated for graph with hierarchical structures. However, while hyperbolic embeddings demonstrate improved performance for explicitly hierarchical datasets, it is important to understand their performance on datasets with varying hierarchial structures. Hence, a quantitative metric for the hierarchical nature of a graph would prove useful.

In this work, we use the graph hyperbolicity metric developed in [2]. Given a graph G, consider all possible 4-tuples of nodes (a, b, c, d). Define S_1, S_2, S_3 as

$$S_1 = d(a, b) + d(c, d)$$
$$S_2 = d(a, c) + d(b, d)$$
$$S_3 = d(a, d) + d(b, c)$$

where $d(x, y)$ denotes the shortest-path distance between nodes x and y in graph G. If M_1 and M_2 denote the two largest values among S_1, S_2, and S_3, then we denote the hyperbolicity of tuple (a, b, c, d) as $\text{hyp}(a, b, c, d) = M_1 - M_2$. We define the hyperbolicity δ of graph G as:

$$\delta(G) = \max_{a,b,c,d \in V(G)} \text{hyp}(a, b, c, d)$$

If G has hyperbolicity δ, we say it is δ-hyperbolic. This definition of hyperbolicity provides a tight bound on the worst additive distortion of distances in a graph when its vertices are embedded into a weighted tree. For example, trees are 0-hyperbolic and $n \times n$ grids are $(n - 1)$-hyperbolic. Therefore, the δ hyperbolicity metric is informative of the intrinsic hierarchical nature of a graph.

3 Methodology

Having described critical components in the intuition behind numerous Euclidean embedding frameworks and having introduced the concept of hyperbolic embeddings, we next turn to the development of baseline methods that utilize such embedding methods to embed nodes on arbitrary graphs $G(V, E)$ with vertex set G and edge set E. We subsequently formalize our proposed frameworks HyperWalk and HyperConv, and we provide details regarding their implementation.

3.1 Traditional Embedding Frameworks

Graph factorization [1] is a canonical baseline method used to generate embeddings for $G(V, E)$ with $O(|E|)$ complexity. In particular, define the embedding matrix $B \in \mathbb{R}^{|V| \times d}$ for embedding dimensionality d, and define the embedding of node i as $B_i \in \mathbb{R}^{1 \times d}$. For a graph with symmetric adjacency matrix $A \in \mathbb{R}^{|V| \times |V|}$, graph factorization factors A by minimizing the loss function

$$\mathcal{L}(B, \lambda) = \sum_{(i,j) \in E} (A_{ij} - \langle B_i, B_j \rangle)^2 + \lambda \sum_i \|B_i\|^2$$

Graph factorization is therefore a simple and efficient method often used to generate initial representations for nodes in graphs, but does not provide performance near the level of the aforementioned Euclidean embedding frameworks in Section 2.1. We use such a system as a baseline implementation across varying graphical structures.

DeepWalk employs a random-walk based approach to embedding generation by maximizing the probability of observing the subsequent and previous k nodes for a random walk centered at node i. In particular, it seeks to maximize

$$\mathcal{L}(B) = \sum_{i \in \tilde{V}} \log P(v_{i-k} \dots v_i \dots v_{i+k} \mid B_i)$$

where $v \in V$, $\tilde{V}$ represents a random sample of nodes in V, and B_i is defined as in the graph factorization methodology. In a similar manner to DeepWalk, node2vec performs random walks across graphs to generate embeddings, but introduces parameters p and q that bias towards breadth-first walks and depth-first walks respectively. By choosing the correct balance between these two parameters, node2vec is able to learn both community structure and general graph structure, improving the quality of node embeddings.

Graph convolutional networks represent a distinct paradigm in node embedding generation than the adjacency-matrix focus of graph factorization and the random-walk emphasis of DeepWalk and node2vec. In particular, GCNs define a convolution operation on graphs, iteratively aggregating the embeddings of node neighbors to update the embeddings of each node. Pooling such local embeddings through multiple iterations thus preserve the local and global structure of graphs. Note that while both spatial filters that operate on G and spectral filters that operate on the graph Laplacian L_G have been utilized, we analyze spectral filters in this work.

Hyperbolic embeddings on arbitrary graphs are generated using the Poincaré ball model. Define $\mathcal{B}^d$ as the open d-dimensional unit ball. The Poincaré ball corresponds to the Riemmanian manifold $(\mathcal{B}^d, g_x)$, where we define

$$g_x = \left(\frac{2}{1 - \|x\|^2} \right)^2 g^E$$

with $x \in \mathcal{B}^d$ and g^E the Euclidean metric tensor. Using this notion of hyperbolic space, note that the distance function d_H is defined as

$$d_H(u,v) = \operatorname{arccosh}\left(1 + 2\frac{\|u-v\|^2}{(1-\|u\|^2)(1-\|v\|^2)}\right)$$

In order to calculate the Poincaré embeddings $\Theta = \{\theta_i\}_{i=1}^n$, we solve the optimization problem given by

$$\Theta' = \arg\min_{\Theta} \mathcal{L}(\Theta)$$

where each θ_i lies in the unit ball $\mathcal{B}^d$ and $\mathcal{L}$ is a loss function dependent on hyperbolic distance d. Due to the manifold nature of the Poincaré ball, we use Riemannian Stochastic Gradient Descent (RGSD) as described in [9] in order to find the optimal embedding θ_i for all nodes. Furthermore, we implement a burn in procedure during RSGD that marginally improves the angular orientation of the initial node embeddings determined from a normal distribution centered at the origin.

3.2 Hyperbolic Random Walks

Inspired by the aforementioned Poincaré embeddings, we develop a novel node embedding algorithm HYPERWALK leveraging random walks in hyperbolic space. As in Section 3.1, we seek to learn mapping function $f : V \to \mathbb{R}^d$ that maps nodes to their feature vectors in dimension d. For a given node $u \in V$, we define $N_S(u)$ as the neighborhood of u using neighborhood sampling strategy S. As in node2vec [3], we seek to maximize the objective function

$$\max_f \sum_{u \in V} \log P(N_S(u) \mid f(u))$$

In particular, the neighborhood sampling strategy S^* is a second order biased random walk. To construct $N_{S^*}(u)$ for a source node u, we simulate a random walk of length l. Let n_i denotes the i-th node in the walk; subsequent nodes in the random walk are generated according to

$$P(n_i = m \mid n_{i-1} = n) = \begin{cases} \frac{\pi_{nm}}{Z} & (n,m) \in E \\ 0 & \text{otherwise} \end{cases}$$

where Z is some normalizing constant. Rather than performing an unbiased walk ($\pi_{nm} = w_{nm}$ where w_{nm} represents edge weight), we perform a biased random walk according to graph hyperbolicity. Suppose our random walk just traversed edge (u,n) and is now at node n. We set the unnormalized transition probability π_{nm} for all edges (n,m) using $\pi_{nm} = \alpha_{pq}(u,m) \cdot w_{nm}$, where

$$\alpha_{pq}(u,m) = \begin{cases} \frac{1}{p} + r_{nm} & \text{if } d(u,m) = 0 \\ 1 + r_{nm} & \text{if } d(u,m) = 1 \\ \frac{1}{q} + r_{nm} & \text{if } d(u,m) = 2 \end{cases}$$

In this definition of bias α_{pq}, $d(u,m)$ denotes the shortest-path distance between nodes u and m in graph G and $r_{nm} = d_H(P_n, P_m)$, where P_x denotes the Poincaré embedding for node x generated using the aformentioneid procedure (recall d_H denotes hyperbolic distance). As in node2vec, parameters p and q control the BFS-like and DFS-like nature of the random walk. The inclusion of the additional r_{nm} term ensures that similarity between nodes n and m as represented by their Poincaré embeddings is also incorporated in the random walk, allowing for hyperbolic structure to be reflected in the learned node embeddings.

3.3 Hyperbolic Graph Convolutions

In order to augment current state-of-the-art graph convolutional frameworks with insights obtained from graph hyperbolicity, we develop a nodel emebdding framework HYPERCONV that leverages node distance in hyperbolic space. We again seek to learn a feature mapping function $f : V \to \mathbb{R}^d$, and we do so by building upon the GraphSAGE framework introduced in [4]. In particular, we introduce a hyperbolic node-level feature aggregator which weights neighbor node features according to the hyperbolic distance between source and neighbor nodes. As in [4], define $\mathbf{h}^{k-1}_{\mathcal{N}(v)}$ as the aggregated feature vectors of all neighbors $\mathcal{N}(v)$ of vertex v. In the case that sampling of such neighbors is not required, all neighbors of v are considered; if sampling is performed, nodes with the closest hyperbolic distance to source v are selected (as opposed to random sampling in GraphSAGE). We now compute

$$\mathbf{h}^k_v \leftarrow \sigma(\mathbf{W} \cdot \text{HMEAN}(\{\mathbf{h}^{k-1}_v\} \cup \{\mathbf{h}^{k-1}_u, \forall u \in \mathcal{N}(u)\})$$

where we define HMEAN as the weighted mean of feature vectors with weights determined by the hyperbolic distance between u and v, $d_H(u,v)$. In particular, the feature vector $\mathbf{h}^{k-1}_u$ of each node u is scalar multiplied by $1/d_H(u,v)$, thereby upweighting the feature vectors of neighbors closer to the source node v in hyperbolic space As noted in [4], such an aggregator is a rough approximation of spectral convolutions presented in [5], and the concatenation of v's previous feature vector with its aggregated neighborhood vector can be viewed as a "skip connection" between different layers in the graph convolutional network. The remainder of network training proceeds as in [4].

4 Datasets

As the generation of hyperbolic embeddings is a relatively novel field in representation learning on graphs, a critical issue with the hyperbolic embedding work in [4] (as discussed in Section 2.2) was its evaluation of such embeddings on small datasets with clearly defined explicit hierarchies. To remedy this issue and develop a baseline comparison of the areas where hyperbolic embeddings provide the most representational power, large real-world datasets that contain both explicit and latent hierarchies are required for careful analysis.

ChG-Miner. [7] represents a drug-target interaction network with information on which proteins encoded by genes are targeted by drugs on the United States market. Nodes represent drugs and genes, and edges correspond to a drug-target interaction. The network has 12,358 nodes and 15,424 edges, rendering it significantly larger than other networks employed to evaluate hyperbolic embeddings. Furthermore, with no explicit hierarchy in drug-target interactions and no explicit hyperbolicity encoded in the graph, ChG-Miner represents a dataset that does not have intuitively strong reasons to prefer hyperbolic embeddings over Euclidean ones.

email-Eu-core. [6] represents an email interaction network between members of a large European research institution. Edges represent communications between institution members, and additional information is provided regarding the departments of each institution member for ground-truth evaluation on community detection algorithms. Furthermore, the presence of a hierarchy is explicitly encoded into this network due to the different titles and roles of individuals sending one another emails as well as the directed nature of the graph. Therefore, the assumptions made in the generation of hyperbolic embeddings are assuredly present in this network, providing intuitively strong reasons to prefer hyperbolic embeddings over Euclidean ones.

In aggregate, these two datasets represent graphs at either end of the spectrum of latent hierarchy presence: while ChG-Miner has no encoded hierarchies, email-Eu-core explicitly encodes hierarchies in its construction. Both datasets additionally represent vastly different numbers of prediction classes: while email-Eu-core has 42 classes of nodes, ChG-Miner only has 2 classes.

5 Experiments

In order to systematically construct and evaluate hyperbolic embeddings against Euclidean embeddings, the aforementioned frameworks in Section 3 were implemented to ingest graph input data and produce node-level embeddings for each input node. These embeddings were evaluated on the node classification task using the aformentioned datasets ChG-Miner and email-Eu-core.

Overview. For each dataset, we evaluated Euclidean embeddings, Poincaré embeddings, HYPERWALK, and HYPERCONV on the task of node classification. In particular, we analyzed the performance of graph factorization, DeepWalk, node2vec, Poincaré embeddings without burn in, and Poincaré embeddings with burn in alongside both our novel methodologies. Nodes were classified by training a logistic regression classifier on 80 percent of the node embeddings and evaluating the predictions of the classifier on the remaining 20 percent. Note that three metrics are used to evaluate the power of each model on node classification tasks: accuracy, weighted F_1, and macro F_1. Accuracy is defined as expected, and we have that

$$F_{1,\text{weighted}} = \frac{1}{|\mathcal{C}|} \sum_{c \in \mathcal{C}} \left(2|c| \cdot \frac{\text{precision}_c \cdot \text{recall}_c}{\text{precision}_c + \text{recall}_c} \right)$$

which represents the traditional F_1 score weighted by the support of each class, while $F_{1,\text{macro}}$ does not weight by support and therefore necessarily produces a value between precision and recall. Results for both datasets are reported in Table 1, with best results according to each metric reported in bold font.

Dataset Hyperbolicity. As in Section 2.2, we compute the hyperbolicity of both the ChG-Miner and email-Eu-core graphs, and we obtain that the ChG-Miner graph is 2-hyperbolic while the email-Eu-core graph is 0.5-hyperbolic. Recall that the hyperbolicity of a graph roughly measures the distance between the shortest path metric of a given graph and the tree metric; therefore, a smaller δ value for hyperbolicity indicates that a graph represents a stronger tree-like structure. Computation of hyperbolicity in this manner therefore conforms with our intuitions

	email-Eu-core (42 classes, $\delta = 0.5$)			ChG-miner (2 classes, $\delta = 2$)		
	Accuracy	Weighted F_1	Macro F_1	Accuracy	Weighted F_1	Macro F_1
Graph factorization	0.2147	0.0872	0.0202	0.7056	0.6453	0.5496
DeepWalk	0.7413	0.7183	0.5016	0.7005	0.6690	0.5983
node2vec (p, q = 0.25)	0.7612	0.7334	0.5342	0.6937	0.6552	0.5780
Poincaré	**0.9104**	**0.8963**	**0.6299**	0.6862	0.6382	0.5529
Poincaré + burn-in	0.9005	0.8859	0.6055	0.6991	0.6594	0.5822
HYPERWALK (p, q = 0.25)	0.7861	0.7596	0.5267	0.6896	0.6548	0.5797
HYPERCONV	0.8059	0.7721	0.5281	**0.7603**	**0.7479**	**0.7018**

Table 1: *Node Classification on email-Eu-core and ChG-miner graphs.* Graph factorization, Deepwalk, node2vec, Poincaré embeddings, HYPERWALK, and HYPERCONV were implemented as in Section 3, with accuracy, weighted F_1, and macro F_1 statistics reported for both datasets. Note the significant improvements of Poincaré embeddings on the hierarchially structured email dataset, while the lack of structure in the ChG-miner dataset inhibits any meaningful improvement by Poincaré frameworks. Best results in each category are reported in bold.

that email-Eu-core represents a structured network while ChG-Miner represents an amorphous one, and as a result hyperbolic embeddings are expected to perform superior to traditional Euclidean embeddings in the email dataset.

5.1 Euclidean and Hyperbolic Embedding Evaluation

We first present analysis detailing performance of hyperbolic and Euclidean embeddings on both datasets, with all evaluation metrics presented in Table 1. In particular, we detail the performance of each algorithm across both datasets, highlighting irregularities and noting where network structure influenced embedding performance.

The graph factorization algorithm was trained over 1000 epochs on both datasets. In particular, graph factorization was trained with weight decay 0.05 and learning rate $\alpha = 0.01$ on the email dataset and with weight decay 0.03 and learning rate $\alpha = 0.01$ on the ChG-miner dataset. Note that we use a stronger weight decay factor on the email-Eu-core dataset due to its smaller size. Of all embeddings, graph factorization has the worst performance on email-Eu-core with 21.47% accuracy, but best performance on ChG-miner with 70.56% accuracy. These results may be intuitively explained by noting that factorization methods are unable to learn network connectivity unless connectivity is explicitly encoded in the loss function, and so they tend to perform better in unstructured networks (like ChG-miner). However, when required to embed highly hierarchial networks, factorization methods produce nearly random results, motivating the necessity of other embedding frameworks.

The DeepWalk algorithm was trained on both datasets with 10 walk iterations. Note that DeepWalk acheived an F_1 score of 0.6690, highest amongst all embedding methods for the ChG-miner dataset. By utilizing parameters $p = q = 0.25$, the node2vec algorithm altered the random walk bias from the uniform sampling of DeepWalk for the same number of walk iterations. On both the email-Eu-core and ChG-miner dataset, the node2vec performance metrics closely relate to those of DeepWalk, indicating that while minor improvements may be achieved with altering random walk structure, radical restructuring is required for significant gains in accuracy.

Poincaré embeddings were computed including and excluding the burn-in procedure. Without burn-in, after calculating initial embedding values using a normal distribution centered at the origin, we train over the course of 500 iterations beginning with learning rate 0.003 and finishing with learning rate 0.001. Furthermore, we calculate embeddings in a Poincaré ball with dimension 128. Notice that this hyperbolic embedding procedure without burn-in has the highest performance metrics for the email-Eu-core dataset. This is a significant as email-Eu-core has well-defined hierarchical structures in addition to 42 unique classes. This performance reinforces prior findings that hyperbolic embeddings are optimal for identifying hierarchical structures, but also indicates that such results can be extended to networks with numerous node classes.

When calculating the Poincaré embeddings using burn-in, we burn in the initial embeddings over the course of 50 epochs, and then train with 450 epochs. As implemented without burn-in, we begin training with learning rate 0.003, finish with learning rate 0.001, and calculate embeddings in a Poincaré ball with dimension 128. Notice that the performance metrics of Poincaré with burn-in, though high, are slighly lower than those of Poincaré without burn-in. This is noteworthy since prior research suggests that implementing burn-in significantly improves performance metrics, a property that may not hold true with larger, real-world networks.

Visualizations. Alongside computing quantitative metrics detailing the performance of each embedding framework in node classification, we additionally created PCA and t-SNE visualizations of the learned embeddings on

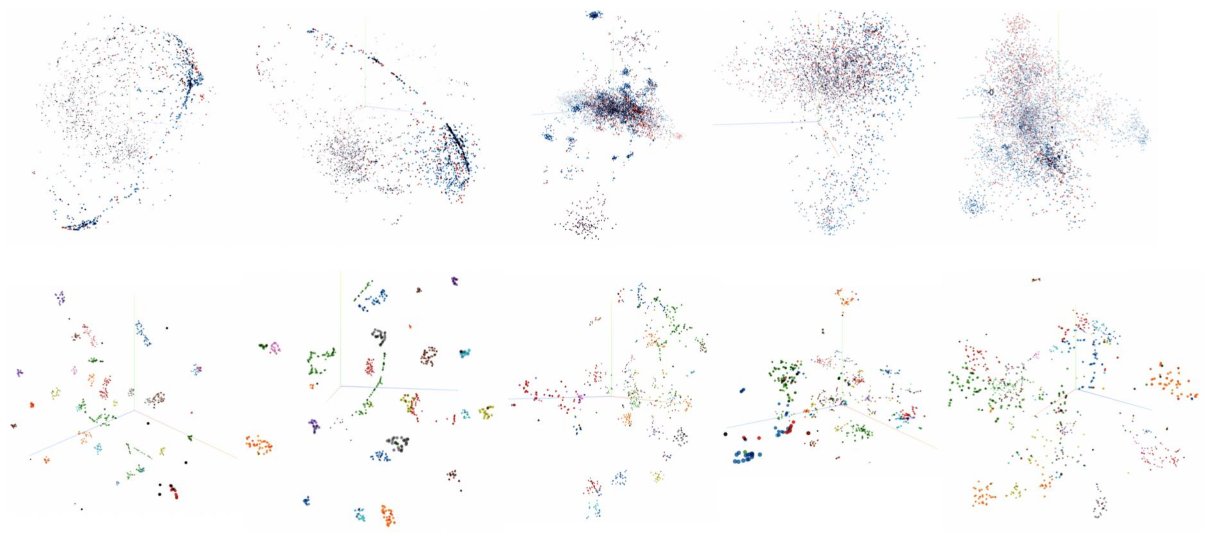

Figure 1: *Representation of Learned Embeddings on ChG-miner (top) and email-Eu-core (bottom) graphs*. Due to the binary nature of the ChG-miner dataset, PCA was employed to visualize its class distribution (with 0 in blue and 1 in red) for varying embedding frameworks. In contrast, the 42 classes in the email-Eu-core graph were represented with t-SNE trained with a perplexity of 22 for 1,000 steps. (a) represents Poincaré with burn-in, (b) represents Poincaré without burn-in, (c) represents DeepWalk, (d) represents graph factorization, and (e) represents node2vec.

each dataset to analyze the distinctions between the resulting representations. In Figure 1, the first row of PCA visualizations represents the embeddings computed on the ChG-miner two-class dataset, with the different classes represented in blue and red. The second row of t-SNE visualizations represents the learned embeddings on the email-Eu-core dataset, with the 42 classes colored differently.

Note immediately the spherical nature of the Poincaré embeddings in Figure 1 (a) and (b); this structure is expected as Poincaré balls form spheres in Euclidean space. The Poincaré model's failure to produce meaningful distinctions between red and blue points on the ChG-miner dataset further explains its inability to excel at node classification on the dataset, while the increasingly distinct clusters forming in DeepWalk (c), graph factorization (d), and node2vec (e) embeddings indicate their superiority at distinguishing the ChG-miner classes. In stark contrast to these results, the immediately evident and distinguished clusters found in the Poincaré t-SNE visualizations in (a) and (b) represent the power of Poincaré embeddings at embedding hierarchial datasets. The strongly distinguished clusters, unlike the more sparse and intermixed clusters in (c)-(e), indicate the unique value of Poincaré embeddings on datasets such as the email dataset, with the visualizations supporting the metrics observed in Table 1.

5.2 HyperWalk and HyperConv Evaluation

We next present analysis detailing performance of HYPERWALK and HYPERCONV on both datasets; note that the metrics for both models are included in Table 1. We further visualize embeddings generated by both novel embedding frameworks on each dataset, noting benefits induced by working within hyperbolic space.

The HYPERWALK algorithm was trained on both datasets with 10 walk iterations using the Poincaré embeddings generated without burn in as described above. Recall that HYPERWALK is based on the node2vec algorithm but incorporates hyperbolic distances in its random walk bias. Hence, it is worthy to note the difference in performance metrics for HYPERWALK and node2vec. On the email-Eu-core dataset, HYPERWALK achieves an accuracy of 0.7861, higher than the 0.7612 accuracy of node2vec. This performance difference suggests that incorporating hyperbolic distances in the random walk bias produces better embeddings than those of regular node2vec, particularly on datasets with non-trivial latent hierarchies. On the ChG-Miner dataset, HYPERWALK achieves an accuracy of 0.6896, while node2vec achieves an accuracy of 0.6937. This similar performance by both algorithms suggests that incorporating hyperbolic distances in the random walk bias yields no additional benefits when implemented on graphs with no known hierarchies. These results mimic those of [9] in that the inclusion of hyperbolic information only produces visible benefits for graphs with non-trivial hierarchies.

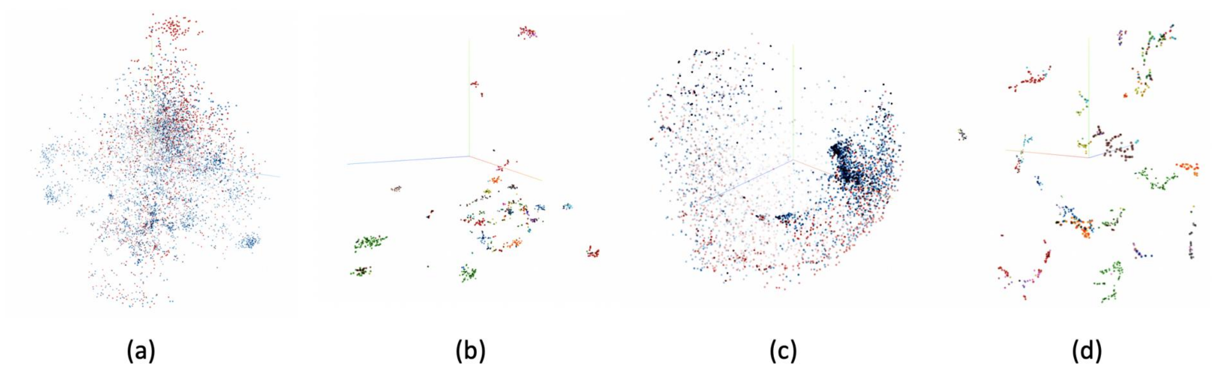

Figure 2: *Representation of Learned Embeddings for HyperWalk and HyperConv*. PCA visualizations (a and c) on ChG-miner and t-SNE visualizations (b and d) on email-Eu-core are depicted for both HYPERWALK and HYPERCONV, with Figures (a) and (b) representing embeddings produced by HYPERWALK and Figures (c) and (d) representing embeddings produced by HYPERCONV.

The HYPERCONV algorithm was trained on both datasets with a learning rate of 1.0 on email-Eu-core and a learning rate of 0.6 on ChG-miner in order to ensure smooth gradient descent. Furthermore, node sampling was not performed across either dataset due to the lack of necessity of such performance enhancements on the relatively small sizes of email-Eu-core and ChG-miner. Note that the HYPERCONV framework acheives superior results to all Euclidean-based frameworks as well as HYPERWALK on the email-Eu-core dataset but is unable to match the performance metrics of the Poincaré methods. However, on the ChG-miner dataset, HYPERCONV achieves significant improvements over all other frameworks considered (including the Poincaré method and HYPERWALK), indicating that the unique combination of convolutional operations and hyperbolic distance results in a more stable embedding outcome even in graphs with little to no hyperbolic structure.

All told, these results evince that graph convolutions do indeed perform superior to random-walk based approaches even within hyperbolic space, but a convolutional training paradigm leaves much to be desired in comparison with a Riemannian SGD update rule on hyperbolic graphs (with low δ). However, HYPERCONV was shown to vastly outperform all other approaches on graphs with high δ, indicating that it was more robust to the structure of the network. Furthermore, the results indicate that convolutional frameworks retain their superiority over traditional walk-based approaches when hyperbolic weighting is applied, indicating that the representational power derived from graph convolutions is better able to represent local node structure.

Visualizations. Figure 2 represents a visualization schematic for HYPERWALK and HYPERCONV similar to the schematic presented in Section 5.1, where we have that (a) represents a PCA visualization of HYPERWALK embeddings on ChG-miner, (b) represents a t-SNE visualization of HYPERWALK embeddings on email-Eu-core, and (c) and (d) are analogous to (a) and (b) for HYPERCONV. Note immediately the structure of the embeddings produced by HYPERCONV in (c); the spherical structure is obtained from the influence of hyperbolicity in the convolution operation, and the resulting clustering of blue points away from red points is visual confirmation of the outstanding results of HYPERCONV on ChG-miner. Furthermore, the t-SNE plots for (b) and (d) more readily correspond to localized clusters than the respective graphs in Figure 1 (c)-(e), corroborating the performance metrics observed in Table 1. However, the t-SNE plots both fall short of the Poincaré t-SNE visualizations in Figure 1 (a) and (b), indicating that both HYPERWALK and HYPERCONV have room for improvement on email-Eu-core.

6 Conclusions and Future Work

Our work presents two primary contributions: (1) the implementation of hyperbolic graph embeddings and their evaluation on real-world datasets with varying hyperbolicity, and (2) the development of two novel algorithms, HYPERWALK and HYPERCONV, that both outperform their counterparts based on Euclidean embeddings with HYPERCONV providing significantly improved robust results on both benchmark datasets. Potential future applications of our work include the extension of our proposed models to link predictions on graphs of varying hyperbolicity, the further enhancement of HYPERWALK to better represent hyperbolic embeddings in PCA visualization, and the evaluation of our models on larger datasets to better evaluate performance. Manan Shah implemented the development of baseline models, the development of HYPERCONV, the visualization pipeline, and conducted experiments. Sagar Maheshwari implemented node2vec, HYPERWALK, and conducted experiments.

References

[1] Deng Cai, Xiaofei He, Jiawei Han, and Thomas S Huang. Graph regularized nonnegative matrix factorization for data representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(8):1548–1560, 2011.

[2] Mikhael Gromov. Hyperbolic groups. In *Essays in group theory*, pages 75–263. Springer, 1987.

[3] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864. ACM, 2016.

[4] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NIPS*, 2017.

[5] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[6] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graph evolution: Densification and shrinking diameters. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1(1):2, 2007.

[7] Jure Leskovec and Andrej Krevl. {SNAP Datasets}:{Stanford} large network dataset collection. 2015.

[8] Maximilian Nickel and Douwe Kiela. Learning continuous hierarchies in the lorentz model of hyperbolic geometry. *arXiv preprint arXiv:1806.03417*, 2018.

[9] Maximillian Nickel and Douwe Kiela. Poincaré embeddings for learning hierarchical representations. In *Advances in neural information processing systems*, pages 6338–6347, 2017.

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