Product Space Embeddings in Hyperbolic Graph Neural Networks

Christopher Healy
Stanford University
cjhealy@stanford.edu

Eli Pugh
Stanford University
epugh@stanford.edu

Abstract

Graph Convolutional Networks (GCN's) map network nodes to embeddings in some geometry. Throughout most of their development, these embeddings were strictly Euclidean, mainly for ease of use. Recently, work has been done adapting these GCN's to embed nodes in curved geometry, most notably spherical manifolds and hyperbolic manifolds. These methods generalize notions of linear feature transformation and aggregation to instead map input features to a curved Riemannian manifold, and to iteratively update these embeddings to satisfy relevant criteria. We present a scheme for mixed embedding spaces; we embed nodes onto a Cartesian product space composed of different manifolds, following the notion that heterogeneous dimensions allow for increased expressiveness. We investigate the efficacy of these methods in a series of experiments.

1 Introduction

GCN's and the corresponding geometric embeddings are used for a variety of standard tasks on large network datasets, including node classification, the grouping of nodes into categories, and link prediction, inferring the existence of edges between nodes based on prior data. Accurate frameworks for these tasks are crucial for a variety of applications. In retail applications, customers and products can be modelled as networks, and accurate node classification can be used to optimize recommendation frameworks [12]. Successful link prediction on networks of pharmaceuticals can be used to predict otherwise unexpected side effects from multiple drugs [11]. Expressive embeddings can help image classifiers perform few-shot learning on categories that they were not trained on [14], and permit significantly faster parsing between and within documents when combined with Natural Language techniques [15].

In order to perform these tasks, embedding methods must be learned to represent objects in a way that their distance in some metric space reflects their similarity. In the vast majority of these tasks, the representations are learned in Euclidean space. Euclidean embeddings lend themselves very well to machine learning applications because calculating distances, inner products, and geodesics, and manifold operations are most simple and efficient in Euclidean space. However, Euclidean space does have fundamental limitations. Hierarchical or tree-like data is not well represented in Euclidean space, and even simple structures can require an intractable number of dimensions to accurately represent [10]. Intuitively, this is because the Euclidean distance metric imposes more constraints on the transitivity of proximity than is required by the triangle inequality; the space of points within some proximity of another point form a hyper-sphere, and that space grows as a polynomial of distance, whereas in a tree with a constant branching factor, the number of nodes grows exponentially with distance.

To learn representations of hierarchical data, hyperbolic embeddings are much more fit for the task. Hyperbolic embeddings can be thought of as continuous versions of trees. The space of points some distance away from a given point grows exponentially instead of as a polynomial, better leveraging

Preprint. Under review.

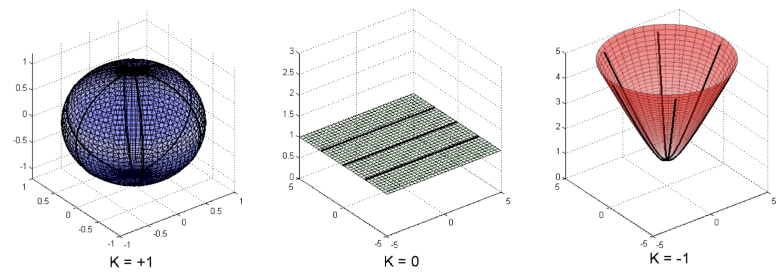

Figure 1: 2 Dimensional versions of the various manifolds discussed. K denotes curvature. From an intuitive perspective, can be understood as the fashion in which dimensions interact. the lines are geodesics, a generalization of straight lines for curved spaces. Note that on a sphere, with positive curvature, all geodesics intersect twice, whereas in flat space, they may remain parallel and equidistant. With negative curvature, geodesics wind up diverging from each other at an exponential rate. Picture from [6]

the triangle inequality to allow for more flexible and less transitive proximities and creating a natural space for tree embeddings.

Since graph structures are often much more accurately embedded in low-dimensional hyperbolic spaces Nickel and Kiela [10], some teams have shown that using hyperbolic embeddings can significantly improve performance for link prediction, node classification, and reconstruction on real-world graph-structured data Chami et al. [2]. Despite this, it's also often possible that hierarchical entities also vary in degree or expressiveness in some dimensions, which is most easily and accurately represented by Euclidean embeddings. To learn more accurate low-dimensional representations for graph-structured data, we explore using a Cartesian product of both hyperbolic and Euclidean manifolds for embedding data. We then compare performance on link prediction on the Cora Sen et al. [13] and Airport [2] datasets, as well as node classification tasks on the PubMed Namata et al. [9] dataset.

2 Prior Work

Gu et al. [6] pioneered the use of mixed manifolds for embeddings. They employed Riemannian optimization methods to create shallow mixed embeddings of data, including network data, and improved performance on multiple benchmarks. Shallow embeddings constitute a mapping of nodes to embeddings; this allows for significantly more flexible embedding methods, as any set of embeddings is viable. GCNs, by contrast, optimize a fixed method of embedding any given node to optimize outcomes. Unlike shallow embeddings, GCNs can leverage node features, can transfer to unseen graphs, and scale more efficiently.

Ganea et al. [5] pioneered hyperbolic neural networks, a modified mathematical framework for neural network layers that lends itself specifically to hyperbolic embeddings by adapting all of the linear operations in neural networks to their closest hyperbolic analogue. This allowed for increased stability in training and increased the efficacy of hyperbolic embedding networks to generally surpass euclidean embeddings across a wide variety of tasks

In 2019, Chami et al. [2], Liu et al. [8], and Bachmann et al. [1] all contributed to adaptations of Gal and Ghahramani [4]'s work for Graph Neural Networks specifically. [2] and Liu et al. [8] both focused on hyperbolic space, while Bachmann et al. [1] adapted this notion to arbitrary manifolds of constant curvature. We focus specifically on the approach of Chami et al. [2], a framework for hyperbolic GCN embeddings that increases stability by adapting all of the conventional operations used in Euclidean space for hyperbolic space. In addition, they use the Poincaré Ball model in order to increase stability.

Contribution: We, in turn adapt Chami et al. [2]'s method for mixed hyperbolic and euclidean graphs. We adapt their method to accommodate mixed embeddings hyperbolic and Euclidean geometries, and evaluate the efficacy of the resulting embeddings.

3 Background

Much of the background on HGCNs needed for our work is described in more depth in Chami et al. [2]. We will briefly summarize the necessary background, and then give a basis for our approach.

3.1 Problem

The input to our task is a graph $(\mathcal{V}, \mathcal{E}, (\mathbf{x}_i^{\mathcal{M}})_{i \in \mathcal{V}})$ where $\mathcal{V}$ is the vertex set, $\mathcal{E}$ is the edge set, $\mathbf{x}_i$ are the embeddings, and $\mathcal{M}$ is the manifold used as the representation space. We'll denote $\mathbb{R}$ as Euclidean space, $\mathbb{H}$ the Lorentz model of hyperbolic space, and $\mathbb{P}$ the Poincaré Ball model of hyperbolic space. We aim to learn a function to map a graph to a set of d dimensional embeddings in $\mathbb{R}^{|V| \times d}$ to be used for link prediction or node classification.

3.2 Euclidean Graph Convolutional Networks

Let $\mathcal{I}(u)$ denote the set of neighbors of u, $\mathbf{W}^l, \mathbf{b}^l$ be learned weights and biases of layer l, and σ be a non-linear function (namely ReLu in our work). GCNs pass messages through layers where message $\mathbf{m}_v$ from node v to its neighbors is computed through a neural network. Nodes update their representations $\mathbf{x}$ by aggregating the messages.

$$
\begin{aligned}
\text{Transform:} \qquad \mathbf{m}_v^l &= \mathbf{W}^l \mathbf{x}_i^{l-1} + \mathbf{b}^l \\
\text{Aggregation:} \qquad \mathbf{x}_u^l &= \sigma(\mathbf{m}_i^l + \sum_{j \in \mathcal{I}(u)} w_{ij} \mathbf{m}_j^l)
\end{aligned}
$$

Message propagation and aggregation is performed over many layers and time steps to propagate messages over neighborhoods. The weights w_{ij} are calculated using a neural network of their own, and are normalized for any given node i with a softmax across all of its neighbors j.

3.3 Manifolds and Models

The **Lorentz** model of hyperbolic space with constant negative curvature K embeds the hyperbolic manifold into a higher dimensional euclidean space. We define the Minkowski inner product $\langle \mathbf{x}, \mathbf{y} \rangle_{\mathcal{L}} = -x_0 y_0 + x_1 y_1 + \cdots x_n y_n$, then define $\mathbb{H}^{n,K}$ and the tangent space $\mathcal{T}_{\mathbf{x}} \mathbb{H}^{n,K}$ at $\mathbf{x}$ as:

$$
\begin{aligned}
\mathbb{H}^{n,K} &:= \left\{ \mathbf{x} \in \mathbb{R}^{n+1} \ : \ \langle \mathbf{x}, \mathbf{x} \rangle_{\mathcal{L}} = -K, \ x_0 > 0 \right\} \\
\mathcal{T}_{\mathbf{x}} \mathbb{H}^{n,K} &:= \left\{ \mathbf{v} \in \mathbb{R}^{n+1} \ : \ \langle \mathbf{v}, \mathbf{x} \rangle_{\mathcal{L}} = 0 \right\}
\end{aligned}
$$

Let $\mathbf{x} \in \mathbb{H}$ and $\mathbf{u} \in \mathcal{T}_{\mathbf{x}} \mathbb{H}$ be unit speed, then this induces geodesics γ and a distance function d given by:

$$
\begin{aligned}
\gamma_{\mathbf{u} \to \mathbf{v}}^K(t) &= \cosh\left(\frac{t}{\sqrt{K}}\right) \mathbf{u} + \sqrt{K} \sinh\left(\frac{t}{\sqrt{K}}\right) \mathbf{v} \\
d_{\mathcal{L}}^K(\mathbf{x}, \mathbf{y}) &= \sqrt{K} \operatorname{arcosh}(-\langle \mathbf{x}, \mathbf{y} \rangle_{\mathcal{L}} / K)
\end{aligned}
$$

We can then project vectors in $\mathcal{T}_{\mathbf{x}} \mathbb{H}^{n,K}$ to $\mathbb{H}^{n,K}$ using the exponential map $\exp_{\mathbf{x}}^K(\mathbf{v}) = \gamma(1)$ where $\gamma(0) = \mathbf{x}$ and $\dot{\gamma}(0) = \mathbf{v}$. We can map back using the logarithmic map $\log_{\mathbf{x}}^K$, the inverse of $\exp_{\mathbf{x}}^K(\mathbf{v})$. This is an analogue for subtraction. These are given analytically:

$$
\begin{aligned}
\exp_{\mathbf{x}}^K(\mathbf{v}) &= \cosh\left(\frac{\|\mathbf{v}\|_{\mathcal{L}}}{\sqrt{K}}\right) \mathbf{x} + \sqrt{K} \sinh\left(\frac{\|\mathbf{v}\|_{\mathcal{L}}}{\sqrt{K}}\right) \frac{\mathbf{v}}{\|\mathbf{v}\|_{\mathcal{L}}} \\
\log_{\mathbf{x}}^K &= d_{\mathcal{L}}^K(\mathbf{x}, \mathbf{y}) \frac{\mathbf{y} + \frac{1}{K} \langle \mathbf{x}, \mathbf{y} \rangle_{\mathcal{L}} \mathbf{x}}{\|\mathbf{y} + \frac{1}{K} \langle \mathbf{x}, \mathbf{y} \rangle_{\mathcal{L}} \mathbf{x}\|_{\mathcal{L}}}
\end{aligned}
$$

As exponential maps forfeit addition's commutativity (taking a point and a vector as inputs), it is useful to be able to map the tangent space of one point on the manifold to another. We can do this with parallel transport, which constitutes moving the vector along a path such that the vector has its stable shifting directions in the local Euclidean frame at any point. On a hyperbolic manifold, this is given by

$$P_{\mathbf{x}\to\mathbf{y}}(\mathbf{v}) = \mathbf{v} - \frac{\langle log_{\mathbf{x}}(\mathbf{y}), \mathbf{v}\rangle_{\mathcal{L}}}{d_{\mathcal{L}}(\mathbf{x},\mathbf{y})^2}(log_{\mathbf{x}}(\mathbf{y}) + log_{\mathbf{y}}(\mathbf{x}))$$

Lastly, let us denote the north pole of the Lorentz model as the point

$$\mathbf{o} = [\sqrt{K}, 0, 0, 0, ...]$$

This point is notable on the manifold as the sole point that maps to itself in all rotational symmetries. We will use logarithmic mappings to its tangent space $\mathcal{T}_{\mathbf{o}}\mathbb{H}^{n,K}$ map points on our hyperboloid to euclidean space for matrix multiplication.

The **Poincaré Ball** model of hyperbolic space with constant negative K curvature is an alternate representation, which has no redundancy in terms of dimensions. As it is just a different coordinate system on the same space, it suffices mathematically to simply provide the coordinate transformation back to Lorentz– any or property can be computed by simply transforming to Lorentz and back:

$$\mathbf{x}_P = \frac{\mathbf{x}_H[1:d]}{\mathbf{x}_H[0]+1}$$

$$\mathbf{x}_H = \frac{[1+||\mathbf{x}_P||^2, \mathbf{x}_P]}{1-||\mathbf{x}_P||^2}$$

There is no theoretical difference between Lorentz and Poincaré space other than parameterization. However, optimizations on Lorentz embeddings tend to have higher numerical stability.

Euclidean manifolds have zero curvature with points in $\mathbf{R}^n$. They have straight geodesics, and distances, exponential map, and logarithmic map are given by:

$$d(\mathbf{x},\mathbf{y}) = \sqrt{\sum_i (x_i - y_i)^2} \qquad \exp_x(v) = x + v \qquad \log_x(y) = y - x \qquad P_{x\to y}(v) = v$$

While there is no point in euclidean space that is intrinsically an analogue to the north pole, for convenience we can use $\mathbf{o} = 0$.

Cartesian Product spaces are a product of smooth manifolds $\mathbb{M} = M_1 \times \cdots \times M_k$. For our case we examine $\mathbb{M} = \mathbb{H}^{d_H,K_H} \times \mathbb{R}^{d_E} \times \mathbb{P}^{d_P,K_P}$. In this space, we compute geodesics, exponential maps, and logarithmic maps by section, and aggregate distances by the euclidean method. Let $\mathbf{x} = [\mathbf{x}_H, \mathbf{x}_E, \mathbf{x}_P]$ where $\mathbf{x}_H \in \mathbb{H}^{d_H,K_H}$, $\mathbf{x}_E \in \mathbb{R}^{d_E}$, and $\mathbf{x}_P \in \mathbb{P}^{d_p,K_p}$, and thus $\mathbf{x} \in \mathbb{M}$. Define $\mathbf{y} \in \mathbb{M}$ similarly, and $\mathbf{v} \in \mathcal{T}_{\mathbf{x}}\mathbb{M}$. We then have:

$$\begin{aligned}
\exp_{\mathbf{x}}(\mathbf{v}) &= [\, \exp_{\mathbf{x}_H}(\mathbf{v}_H),\ \exp_{\mathbf{x}_E}(\mathbf{v}_E),\ \exp_{\mathbf{x}_P}(\mathbf{v}_P) \,] \\
\log_{\mathbf{x}}(\mathbf{y}) &= [\, \log_{\mathbf{x}_H}(\mathbf{y}_H),\ \log_{\mathbf{x}_E}(\mathbf{y}_E),\ \log_{\mathbf{x}_P}(\mathbf{y}_P) \,] \\
d(\mathbf{x},\mathbf{y}) &= \sqrt{d(\mathbf{x}_H,\mathbf{y}_H)^2 + d(\mathbf{x}_E,\mathbf{y}_E)^2 + d(\mathbf{x}_P,\mathbf{y}_P)^2} \\
P_{\mathbf{x}\to\mathbf{y}}(\mathbf{v}) &= [\, P_{\mathbf{x}_H\to\mathbf{y}_H}(\mathbf{v}_H), P_{\mathbf{x}_E\to\mathbf{y}_E}(\mathbf{v}_E), P_{\mathbf{x}_P\to\mathbf{y}_P}(\mathbf{v}_P) \,]
\end{aligned}$$

4 Methodology

To train embeddings for downstream tasks, we use a variant of the Hyperbolic Graph Convolutional Network (HGCN) as proposed Chami et al. [2]. This is only a slight variant on the standard GCN method as described in 3.2. Transformations must be learned in hyperbolic space, which is notably not a vector space in general, and thus Chami et al. [2] generalize $\mathbf{Wx}$ to manifolds. For the hyperboloid. The most straightforward method is to exploit the linearity of tangent spaces, and the invertibility of the logarithmic map. For matrix multiplication, any point on the manfold can be uniquely mapped to first map $\mathbf{x}$ to the tangent space $\mathcal{T}_{\mathbf{o}}\mathbb{M}^{d,K}$, undergo any amount of linear operations, and then be projected back onto the manifold with an exponential map. Notably, in their implementation, the bias term is added with a second exponential map, after the vector x is projected back onto the manifold; the bias term is parallel transported from $\mathbf{o}$ to $\mathbf{x}$, and then a second exponential mapping is performed.

$$\mathbf{W} \otimes^K \mathbf{x} := \exp_{\mathbf{o}}^K(\mathbf{W} \log_{\mathbf{o}}^K(\mathbf{x})) \qquad \mathbf{x} \oplus^K \mathbf{b} := \exp_{\mathbf{x}}^K(P_{\mathbf{o}\to\mathbf{x}}^K(\mathbf{b}))$$

Thus the first step in any layer is as follows:

$$\mathbf{h}_i^l = (\mathbf{W}^l \otimes^K \mathbf{x}_i^{l-1}) \oplus^K \mathbf{b}^l$$

where l denotes the layer.

This is not the only difficulty, as message aggregation also involves a weighted mean. The most common generalization of the mean to hyperbolic spaces, the Frechet Mean Fréchet [3], is too expensive to compute, while a series of exponential maps on parallel transported vectors (an iterative version of the bias method outlined above) is dependent on the order with which the neighbors are added, as parallel transport is notoriously order dependent. Chami et al. [2] instead once again exploit the linearity on the tangent spaces. Weights are calculated in the tangent space of the north pole, while the actual aggregation occurs as a single exponential map, with the input vector as a weighted average of the logarithmic mappings from the new neighborhood embeddings to our present embedding. Mathematically we can write this as:

$$\mathbf{w}_{ij}^l = \text{Softmax}_{j\in\mathcal{I}(i)}\mathbf{AN}^l(\log_{\mathbf{o}}(\mathbf{h}_i)||\log_{\mathbf{o}}(\mathbf{h}_j))$$
$$\mathbf{y}_i^l = \exp_{\mathbf{o}}(\sum_{\mathcal{I}(i)} \mathbf{w}_{ij}^l \log_{h_i}(\mathbf{h}_j))$$

Where $\mathbf{AN}^l$ is a seperate, standard feedforward neural network. The aggregation in the local tangent space, as opposed to the north pole, has been shown to improve outcomes.

Lastly, we have a non linearity. In keeping with the theme, we project into the vector space of the north pole, perform our nonlinearity, and then project back (in our experiments, we use a relu).

$$\mathbf{x}_i^l = \exp_{\mathbf{o}}(\sigma(\log_{\mathbf{o}}(\mathbf{y}_i^l)))$$

Thus we can write the entire scheme for a layer as

$$\mathbf{h}_i^l = (\mathbf{W}^l \otimes^K \mathbf{x}_i^{l-1}) \oplus^K \mathbf{b}^l$$
$$\mathbf{w}_{ij}^l = \text{Softmax}_{j\in\mathcal{I}(i)}\mathbf{AN}^l(\log_{\mathbf{o}}(\mathbf{h}_i)||\log_{\mathbf{o}}(\mathbf{h}_j))$$
$$\mathbf{y}_i^l = \exp_{\mathbf{o}}(\sum_{\mathcal{I}(i)} \mathbf{w}_{ij}^l \log_{h_i}(\mathbf{h}_j))$$
$$\mathbf{x}_i^l = \exp_{\mathbf{o}}(\sigma(\log_{\mathbf{o}}(\mathbf{y}_i^l)$$

While Chami et al. [2] construct this for $\mathbb{H}^{d,K}$, this works generally, namely for Cartesian product manifolds $\mathbb{M} = \mathbb{H}^{d_H,K_H} \times \mathbb{R}^{d_E} \times \mathbb{P}^{d_P,K_P}$ as described in the Cartesian Product section of 3.3. Aside from inter-manifold interactions, the only time dimensions from the different manifolds in the product space will interact is during the matrix multplication, at which point they have all been projected to a linear space.

5 Experiments

5.1 Experimental Setup

We benchmarked the product space embeddings across both node classification and link prediction, with two separate datasets for each. Following the method used by Krioukov et al. [7] and Nickel and Kiela [10], for link prediction, we utilize the Fermi-Dirac decoder to calculate edge probability

$$p(e_{\mathbf{u},\mathbf{v}}) = \left(e^{\frac{d(\mathbf{u},\mathbf{v})-r}{t}} + 1\right)^{-1}$$

where r, t are hyperparameters. For node classification, as in Chami et al. [2], we map the final embeddings to the tangent space of the north pole with a logistic map and perform a standard logistic regression.

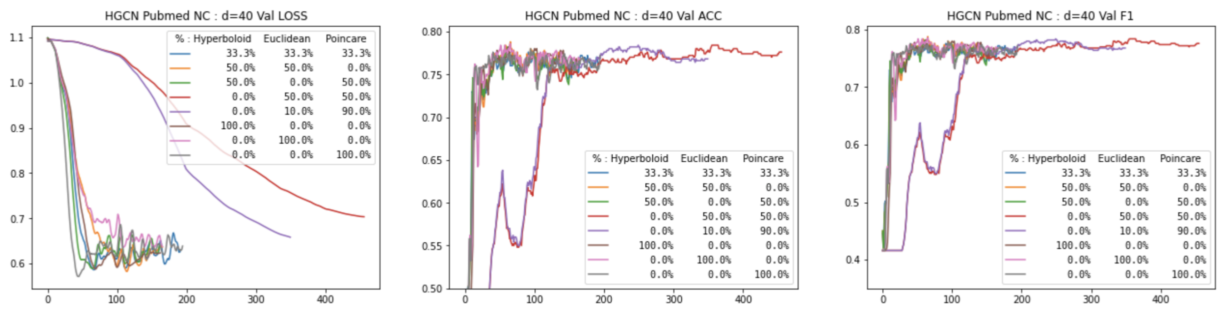

Figure 2: Training Loss and Evaluation Scores for the PubMed node classification task. Note in particular the anomalous training paths of Euclidean-Poincaré mixtures

For node classification, we use the Pubmed dataset. Pubmed [9] contains 19,717 medical papers grouped into 3 categories (with edges corresponding to citations).

For link prediction, we used the Cora and Airport datasets. Cora [13] consists of 2708 machine learning papers divided into 7 classes (with edges corresponding to citations), while Airport has nodes as airports and edges as flights with 2236 nodes (also gathered by [2]).

We test each dataset-task combination with two dimensionalities, 20 and 40, with two layers each. We fine tune learning rates specifically for each run, in order to minimize overflow and maximize scores.

5.2 Results

Our full results are shown in the Appendix. In Figure 2, we show the evolution of scores on the validation set over the course of training. For node classification, we use pure accuracy as our metric. For link prediction, we use average precision.

5.3 Discussion

Perhaps our most interesting result is the exaggerated divergence between Euclidean-Poincaré mixtures and Euclidean-hyperbolic mixtures. While the results tend to vary between pure hyperbolic and Poincaré embeddings due to numerical stability, these effects are greatly exaggerated when mixed with Euclidean manifolds. Euclidean-Poincaré manifolds in particular had notably different training trajectories, as shown in Figure 2.

The net results between the best performing mixture and the best performing pure manifold on any given task are very small, with mixtures out competing pure manifolds approximately half the time. Further investigation is warranted onto the relative benefits of separate matrices **W** for each manifold, as opposed to sharing a matrix. If projections to the tangent space tend to have wildly different magnitudes based on manifold, the gradient along the smaller magnitude manifold could easily get lost in noise.

6 Conclusion

We adapted Chami et al. [2]'s method of hyperbolic graph embeddings for Cartesian product manifolds. We demonstrate small improvements in general over pure manifolds in some cases, but largely mixtures appear to offer limited upside at the expense of increased complexity and numerical instability.

References

[1] G. Bachmann, G. Bécigneul, and O.-E. Ganea. Constant curvature graph convolutional networks. *arXiv preprint arXiv:1911.05076*, 2019.

[2] I. Chami, Z. Ying, C. Ré, and J. Leskovec. Hyperbolic graph convolutional neural networks. In *Advances in neural information processing systems*, pages 4868–4879, 2019.

[3] M. R. Fréchet. Les éléments aléatoires de nature quelconque dans un espace distancié. *Annales de l'institut Henri Poincaré*, 10(4):215–310, 1948. URL `http://www.numdam.org/item/AIHP_1948__10_4_215_0`.

[4] Y. Gal and Z. Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. *arXiv preprint arXiv:1506.02142*, 2015.

[5] O. Ganea, G. Bécigneul, and T. Hofmann. Hyperbolic neural networks. In *Advances in neural information processing systems*, pages 5345–5355, 2018.

[6] A. Gu, F. Sala, B. Gunel, and C. Ré. Learning mixed-curvature representations in product spaces. In *International Conference on Learning Representations*, 2018.

[7] D. Krioukov, F. Papadopoulos, M. Kitsak, A. Vahdat, and M. Boguná. Hyperbolic geometry of complex networks. *Physical Review E*, 82(3):036106, 2010.

[8] Q. Liu, M. Nickel, and D. Kiela. Hyperbolic graph neural networks. In *Advances in Neural Information Processing Systems*, pages 8230–8241, 2019.

[9] G. Namata, B. London, L. Getoor, B. Huang, and U. EDU. Query-driven active surveying for collective classification. In *10th International Workshop on Mining and Learning with Graphs*, volume 8, 2012.

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

[11] V. Nováček and S. K. Mohamed. Predicting polypharmacy side-effects using knowledge graph embeddings. *AMIA Summits on Translational Science Proceedings*, 2020:449, 2020.

[12] E. Palumbo, G. Rizzo, R. Troncy, E. Baralis, M. Osella, and E. Ferro. Knowledge graph embeddings with node2vec for item recommendation. In *European Semantic Web Conference*, pages 117–120. Springer, 2018.

[13] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

[14] G.-S. Xie, L. Liu, F. Zhu, F. Zhao, Z. Zhang, Y. Yao, J. Qin, and L. Shao. Region graph embedding network for zero-shot learning. In *European Conference on Computer Vision*, pages 562–580. Springer, 2020.

[15] C. Xiong, R. Power, and J. Callan. Explicit semantic ranking for academic search via knowledge graph embedding. In *Proceedings of the 26th international conference on world wide web*, pages 1271–1279, 2017.

Appendix

TASK	DATASET	DIMS	% HYPER	% EUC	% POIN	LOSS	ACC/AP
NC	PUBMED	20	0.333	0.333	0.333	0.652	0.76
NC	PUBMED	40	0.333	0.333	0.333	0.634	0.752
NC	PUBMED	20	0.5	0.5	0	0.68	0.761
NC	PUBMED	40	0.5	0.5	0	0.692	0.775
NC	PUBMED	20	0.5	0	0.5	0.616	0.759
NC	PUBMED	40	0.5	0	0.5	0.615	0.758
NC	PUBMED	20	0	0.5	0.5	1.1	0.456
NC	PUBMED	40	0	0.5	0.5	1.097	0.407
NC	PUBMED	20	0.9	0.1	0	0.773	0.764
NC	PUBMED	40	0.9	0.1	0	0.763	0.761
NC	PUBMED	20	0	0.1	0.9	1.1	0.455
NC	PUBMED	40	0	0.1	0.9	1.097	0.407
NC	PUBMED	20	1	0	0	0.629	0.746
NC	PUBMED	40	1	0	0	0.628	0.767
NC	PUBMED	20	0	1	0	0.691	0.775
NC	PUBMED	40	0	1	0	0.708	0.77
NC	PUBMED	20	0	0	1	0.627	0.763
NC	PUBMED	40	0	0	1	0.622	0.765
LP	CORA	20	0.333	0.333	0.333	2.253	0.849
LP	CORA	40	0.333	0.333	0.333	0.745	0.935
LP	CORA	20	0.5	0.5	0	0.755	0.93
LP	CORA	40	0.5	0.5	0	0.719	0.934
LP	CORA	20	0.5	0	0.5	0.794	0.92
LP	CORA	40	0.5	0	0.5	0.744	0.937
LP	CORA	20	0	0.5	0.5	2.253	0.792
LP	CORA	40	0	0.5	0.5	2.252	0.812
LP	CORA	20	0.9	0.1	0	2.244	0.877
LP	CORA	40	0.9	0.1	0	2.236	0.884
LP	CORA	20	0	0.1	0.9	2.254	0.614
LP	CORA	40	0	0.1	0.9	2.254	0.702
LP	CORA	20	1	0	0	0.753	0.929
LP	CORA	40	1	0	0	0.79	0.921
LP	CORA	20	0	1	0	2.248	0.861
LP	CORA	40	0	1	0	2.223	0.883
LP	CORA	20	0	0	1	0.793	0.94
LP	CORA	40	0	0	1	0.833	0.938
LP	AIRPORT	20	0.333	0.333	0.333	1.023	0.914
LP	AIRPORT	40	0.333	0.333	0.333	0.867	0.929
LP	AIRPORT	20	0.5	0.5	0	0.981	0.914
LP	AIRPORT	40	0.5	0.5	0	0.871	0.925
LP	AIRPORT	20	0.5	0	0.5	0.839	0.926
LP	AIRPORT	40	0.5	0	0.5	0.853	0.927
LP	AIRPORT	20	0	0.5	0.5	2.128	0.895
LP	AIRPORT	40	0	0.5	0.5	2.088	0.904
LP	AIRPORT	20	0.9	0.1	0	1.429	0.911
LP	AIRPORT	40	0.9	0.1	0	1.668	0.916
LP	AIRPORT	20	0	0.1	0.9	2.244	0.594
LP	AIRPORT	40	0	0.1	0.9	2.232	0.798
LP	AIRPORT	20	1	0	0	1.598	0.91
LP	AIRPORT	40	1	0	0	1.473	0.918
LP	AIRPORT	20	0	1	0	1.691	0.886
LP	AIRPORT	40	0	1	0	1.723	0.903
LP	AIRPORT	20	0	0	1	1.184	0.918
LP	AIRPORT	40	0	0	1	1.258	0.924

Table 1: Comparative Performance