Hyperbolic Representations of Source Code

Raiyan Khan[†*], **Thanh V. Nguyen**[‡], **Sengamedu H. Srinivasan**[‡]

[†] Columbia University, [‡]AWS AI

Abstract

Learning effective representations of data is an important task in machine learning. Existing methods typically compute representations or embeddings in Euclidean space, which has shortcomings in representing hierarchical structures of the underlying data. Alternatively, hyperbolic geometry offers a representation scheme that is suited for robust, high-fidelity representations of tree-structured data. In this paper, we explore hyperbolic graph convolutional models for learning hyperbolic representations of source code, which exhibit natural hierarchies. We leverage the abstract syntax tree (AST) of source code and learn its graph-based representation to predict the function name from its body. We compare Lorentz and Poincaré Disk models of hyperbolic geometry with Euclidean geometry. We also propose several readout schemes to compute the graph-level representations and apply them to the method name prediction task. Using a Lorentz hyperbolic model, we establish a new state-of-the-art result on the `ogbg-code2` benchmark for the task.

Introduction

Representation learning is an important building block of deep learning. For program understanding, learning effective representations or embeddings of source code is crucial to be successful in code-related tasks such as bug detection, code generation or code summarization. Existing work typically focuses on two underlying representations of programs: the textual surface of source code as a sequence of tokens and its abstract syntax representations (Allamanis, Brockschmidt, and Khademi 2018; Alon et al. 2020).

In contrast to the textual representation, an AST encodes rich syntactic structures of source code and hierarchies of program elements. Recently, graph neural networks have become more and more effective in representing graph-structured data. As such, a series of work leverages the structural information in ASTs for learning vector representations of the nodes and the AST by recursively incorporating information from neighboring nodes and capturing the graph structure.

Conventional source code embeddings are learned in Euclidean spaces, which face challenges in representing hierarchical data with high fidelity. Exponential growth in branching structures of hierarchical graphs leads to exhibit crowding effects, and true distances between nodes are difficult to preserve. Hyperbolic space offers a potential solution to overcome these drawbacks. Properties such as constant negative curvature and exponential scaling allow hyperbolic space to map tree structures with low distortion. Inspired by the recent development of hyperbolic neural network architectures (Ganea, Becigneul, and Hofmann 2018; Liu, Nickel, and Kiela 2019; Chami et al. 2019), we propose to learn hyperbolic representations of source code atop the abstract syntax tree. We explore a number of model architecture components, including readout layers for the task of concise yet expressive graph compression for method name prediction.

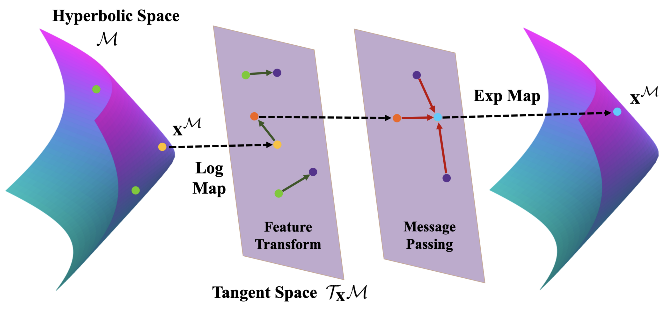
Hyperbolic Space
$\mathcal{M}$
$\mathbf{x}^{\mathcal{M}}$
Log Map
Feature Transform
Message Passing
Exp Map
$\mathbf{x}^{\mathcal{M}}$
Tangent Space $\mathcal{T}_{\mathbf{x}}\mathcal{M}$

Figure 1: An overview of hyperbolic graph neural network models. Points in hyperbolic space are mapped to an Euclidean tangent space, after which the feature transformation and neighborhood aggregation steps take place, and points are mapped back to hyperbolic space.

In this work, we make three main contributions.

1. We show that hyperbolic graph convolutional networks provide improvements for the method name prediction task compared to their Euclidean counterparts. More specifically, we show that Lorentz model achieves a new state-of-the-art performance on the `ogbg-code2` task.
2. We provide a comparison of a number of graph pooling methods for non-Euclidean space, in addition to proposing a novel pooling method.
3. We propose an alternate approach to introducing nonlinearities into our graph neural network by simply relying on the tangent space projections, which reduces the com-

[*]This work was done when Raiyan Khan interned at AWS AI.
Copyright © 2022, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

putational and model complexity.

Background

We adopt the framework in (Chami et al. 2019) for learning hyperbolic representations on graphs and propose a new component to compute graph embeddings. At a high level, the problem of learning graph representations is to find a mapping f from the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with the vertex set $\mathcal{V}$ and edge set $\mathcal{E}$ to a set of node and graph embedding vectors:

$$f : (\mathcal{V}, \mathcal{E}, \{\mathbf{x}_i^{0,E}\}_{i\in\mathcal{V}}) \to Z \in \mathbb{R}^{(|\mathcal{V}|+1)\times d},$$

where $\{\mathbf{x}_i^{0,E}\}_{i\in\mathcal{V}}$ are d_0-dimensional initial node features in an Euclidean space. We are interested in an inductive, graph-level prediction task where f is expected to generalize to unseen graphs. Note that the extra dimension in $|\mathcal{V}|+1$ is due to the embedding of the whole graph based on a $\mathrm{READOUT}$ function described below.

Graph Convolutional Networks (GCNs) are a popular neural architecture for this purpose. A GCN can be thought of as a message passing algorithm that computes the following for node $i \in \mathcal{V}$ and layer ℓ:

$$\begin{aligned}\mathbf{h}_i^{\ell,E} &= \mathbf{W}^\ell \mathbf{x}_i^{\ell-1,E} + \mathbf{b}^\ell \\ \mathbf{x}_i^{\ell,E} &= \sigma(\mathbf{h}_i^{\ell,E} + \sum_{j\in\mathcal{N}(i)} w_{ij}\mathbf{h}_j^{\ell,E})\end{aligned}$$

where $\mathcal{N}(i) = \{j : (i,j) \in \mathcal{E}\}$, $\{\mathbf{W}^\ell, \mathbf{b}^\ell\}_{\ell\in[L]}$ denote weight and bias parameters, and $\sigma(\cdot)$ is a non-linear activation function. The weight w_{ij} can be computed in different ways such as attention (Kipf and Welling 2016; Veličković et al. 2018). Finally, the graph embedding is computed by $\mathbf{g}^{L,E} = \mathrm{READOUT}(\{\mathbf{x}_i^{L,E}\})$ for some readout function of the last layer node embeddings, such as sum or max.

To encode hierarchical graph structures and benefit from expressive hyperbolic embeddings, (Chami et al. 2019) generalizes GCNs to hyperbolic geometry, leading to Hyperbolic Graph Convolutional Networks (HGCNs). The main difficulty of such a generalization is that natural operations such as vector addition and matrix-vector multiplication in Euclidean vector space do not carry over into hyperbolic space. For a hyperbolic manifold with curvature $-1/K$ $(K > 1)$, HGCN forward propagation is defined as

$$\begin{aligned}\mathbf{h}_i^{\ell,H} &= (\mathbf{W}^\ell \otimes \mathbf{x}_i^{\ell-1,H}) \oplus \mathbf{b}^\ell \\ \mathbf{y}_i^{\ell,H} &= \mathrm{AGG}(\mathbf{h}^{\ell,H})_i \\ \mathbf{x}_i^{\ell,H} &= \sigma^\otimes(\mathbf{y}_i^{\ell,H})\end{aligned}$$

where $\oplus$ denotes Mobius gyro vector addition and $\otimes$ denotes Mobius gyro matrix multiplication. (Note that $\oplus, \otimes, \sigma(\cdot)$, AGG depend on K. We ignore this to simplify the notation.) The exact formulation of each hyperbolic operation depends on the hyperbolic model on which the neural network operates. The authors in (Ganea, Becigneul, and Hofmann 2018) lays out the mathematical foundations, which represent corresponding operations in hyperbolic space, for building hyperbolic neural networks with the Poincaré Ball model, whereas (Chami et al. 2019) extends that work to the hyperboloid or Lorentz model. In essence, the underlying intuition is to perform fundamental operations of neural networks in tangent spaces, which are first-order, local Euclidean approximations of the hyperbolic manifold. We illustrate the idea in Figure 1. Formally, these operations are realized by the exponential map $\exp_{\mathbf{x}}(\cdot)$ that projects a vector in the tangent space of $\mathbf{x}$ back on the manifold and the reverse logarithmic map $\log_{\mathbf{x}}(\cdot)$ such that

$$\begin{aligned}\mathbf{W} \otimes \mathbf{x}^H &:= \exp_{\mathbf{o}}(\mathbf{W}\log_{\mathbf{o}}(\mathbf{x}^H)) \\ \mathbf{x}^H \oplus \mathbf{b} &:= \exp_{\mathbf{x}^\mathbf{H}}(P_{\mathbf{o}\to\mathbf{x}^H}(\mathbf{b}))\end{aligned}$$

where $P_{\mathbf{o}\to\mathbf{x}^H}(\cdot)$ is the parallel transport from the tangent space of the origin $\mathbf{o}$ to another of $\mathbf{x}^H$ (here again the mappings depend on K, which we omit for notational simplicity.) We define the maps and the parallel transport explicitly in the appendix for Poincaré Ball and Lorentz models. Similarly, one can define hyperbolic non-linear activation as

$$\sigma^\otimes(\mathbf{x}^H) = \exp_{\mathbf{o}}^{K_\ell}(\sigma(\log_{\mathbf{o}}^{K_{\ell-1}}(\mathbf{x}^H))). \quad (1)$$

A hyperbolic aggregation to average the nodes' representations:

$$\mathrm{AGG}^K(\mathbf{x}^H)_i = \exp_{\mathbf{x}_i^H}^K\left(\sum_{j\in\mathcal{N}(i)} w_{ij}\log_{\mathbf{x}_i^H}^K(\mathbf{x}_j^H)\right). \quad (2)$$

Lorentz, Poincaré Ball and Klein Models of Hyperbolic Geometry. There are multiple models for hyperbolic space, see (Peng et al. 2021). We primarily experiment with Lorentz Hyperboloid and Poincaré Ball models, and the logarithmic and exponential maps are derived independently for each of them (Ganea, Becigneul, and Hofmann 2018; Chami et al. 2019). Along with the aggregation via the tangent space formulation defined in Eq. (2), we explore a direct computation of weighted sum using the Einstein midpoint approach. Since Einstein midpoint is defined in a Klein model, we transform coordinates in a certain hyperbolic model (e.g., Lorentz model) to the corresponding Klein model, compute the Einstein midpoint and transform the midpoint to the original space again. See Appendix for more details.

Hyperbolic Representation for Code

We build our model on the HGCN architecture to learn hyperbolic embeddings of source code with several modifications. First, the above non-linear activation scheme often causes the hyperbolic points to fall out of the manifold and leads to the numerical instability. Therefore, we omit the nonlinearity activation in our model since the exponential and logarithmic maps already involve nonlinear mappings, as noted by Ganea, Becigneul, and Hofmann (2018).

Previous hyperbolic models (Chami et al. 2019) have focused on node or link prediction tasks and not on graph-level learning, with the exception of (Liu, Nickel, and Kiela 2019) who implement a centroid-based approach for graph level predictions. This method involves learning a list of hyperbolic centroids during training and is computationally expensive, especially when performed on many graphs. We therefore explore a number of different readout layers for our hyperbolic models. The readout layer pools the final node embeddings across the graph into a summarizing representation that is used for graph-level prediction tasks. Additionally, we introduce a hyperbolic readout step in our model in order to compute the graph-level embedding. We leverage a wide variety of general graph-based readout strategies and certain hyperbolic variants thereof described below. Note that we attempted to adapt a differentiable Fréchet mean implementation (Lou et al. 2020), however we were unable to run the code on our data due to its runtime instability.

READOUT functions

We propose a number of readout schemes for hyperbolic features. Specifically, after mapping node representations to the Euclidean tangent space, we apply the pooling step and then map points back to hyperbolic space:

$$\mathbf{g}^{L,H} = \exp_{\mathbf{o}}(\mathrm{READOUT}(\log_{\mathbf{o}}(\{\mathbf{x}_i^{L,H}\}))), \qquad (3)$$

with one of the standard sum, mean or max pooling functions. The sum and mean operations follow the principle laid out in Eq. (2) while the max pooling can be seen as performing a non-linear transformation on hyperbolic objects.

We also implement the *Einstein midpoint* according to a previous approach (Dai et al. 2021; Li, Cai, and He 2017). This allows us to perform mean graph aggregation using the Lorentz (also called Hyperboloid) model of hyperbolic space. Node representations are mapped to and from the Lorentz manifold. Einstein midpoint can also be used to aggregate messages across the neighbors of a node. Therefore, we use this technique for message passing in addition to readout.

Finally, we add a simple graph augmentation technique in which a *"virtual node"* is added to the graph that shares a connection with all nodes in the original graph, to bypass the need of a readout layer (Gilmer et al. 2017). At each message-passing layer, messages are passed between the virtual node and all graph nodes. Instead of reducing the original graph, we use the virtual node to perform method name prediction in our downstream task.

Experimental Results

To demonstrate the utility of hyperbolic representations for source code, we evaluate our approach on a downstream task of predicting the function name given a function body. The task is widely known as *code summarization* (Alon et al. 2019; Allamanis, Peng, and Sutton 2016; Zügner et al. 2021) where the name of a given function can be considered the semantic label or summarization of the function logic in the body. The semantic labeling of code snippets involves understanding the content of the method body and usually requires aggregation of a variety of different expressions and statements, so it is important to learn a succinct code embedding. We focus on the abstract syntax representations of code snippets to leverage the hierarchical structures and construct bi-directional edges between parent and child nodes.

Dataset. We use the `ogbg-code2` dataset available in the Open Graph Benchmark (Hu et al. 2020). `ogbg-code2` consists of 452,741 Python functions, which were extracted from GitHub (Husain et al. 2019). Since this is a graph dataset, the functions are represented as abstract syntax trees. For code summarization, the name node of each input AST is masked before being used to predict the target function name. The whole dataset is split into train/validation/test sets to avoid duplication of code and labels, as well as to prevent trivial memorization of naming conventions. The hyperbolicity of a graph can be quantified using Gromov's δ, (Chami et al. 2019) a measurement in which tree represent the graph structure with maximum hyperbolicity. Since our data consists of all trees, each graph in our dataset has maximum hyperbolicity ($\delta = 0$).

Model. We compute the graph embedding of the AST using GNN variants (GIN, vanilla GCN, and hyperbolic GCN) and various choices of the readout function. The final prediction layer consists of a decoder comprised of five linear classifiers used to predict the sub-token at the first five positions of the method name. We use cross-entropy loss between the set of predictions and the set of reference sub-tokens.

Training details. To ensure a fair comparison, we fix the training procedures and model architecture components that are not being investigated across runs of each subsequent experiment. Since dropout cannot be directly applied in hyperbolic space, we do not use dropout in any of our analyses. Since our model's parameters lie in Euclidean space, we use the standard Adam optimizer for training.

Metrics. We conduct a series of experiments to establish the utility of hyperbolic representations using two hyperbolic geometries (Poincaré Ball and Lorentz), in comparison to conventional Euclidean models. For each experiment, we report precision, recall and F1 score on the test set. We also report the average Jaccard similarity at the sub-token level as well as the number of exact matches between predictions and references. The metrics are computed using the set union of predicted sub-tokens and reference sub-tokens.

Variations. In conjunction with these experiments, we explore several downsampling approaches in our comparison with various readout layer formulations, as well as different methods for message passing.

Comparing Euclidean and hyperbolic representations. Table 1 shows the performance of top-performing HGCN models based on Lorentz, Poincaré Ball, and Euclidean frameworks. It can be seen that hyperbolic frameworks outperform the Euclidean framework for the method name prediction task, with the Lorentz model outperforming the Poincaré Ball model. The following are some specific observations.

Vocabulary. Vocabulary size of 5K has a coverage of 90.3% while vocabulary size of 25K has coverage of 98.0%. Larger vocabulary size tends to boost the performance of hyperbolic models while the performance of the Euclidean model does not improve.

Embedding dimension. The performance of hyperbolic models increase as the embedding dimension increases from 300 to 550 while that of Euclidean model decreases.

Curvature. The best results for the Lorentz model are obtained when K is greater than 1. Note that curvature is given by $-1/K$ where K is typically in the range 3–5. Figure 2 capture the variation of F1 score with curva-

Geo	$\|V\|$	d	L	C	F1	AJ	#EM	M
L	25K	550	4	4	**0.1814**	**0.140**	**1273**	303M
L	25K	550	4	3	0.1802	0.139	1251	303M
L	25K	600	3	4	0.1795	0.138	1234	329M
L	25K	600	3	3	0.1795	0.139	1241	329M
L	25K	500	4	3	0.1774	0.136	1204	275M
L	25K	400	4	4	0.1767	0.138	1227	219M
L	25K	300	4	4	0.1737	0.135	1236	164M
P	5K	600	5	1	0.1627	0.118	1097	92M
P	25K	300	5	1	0.1609	0.125	1081	165M
P	5K	500	5	1	0.1599	0.118	1109	75M
P	5K	400	5	1	0.1572	0.117	1069	60M
P	5K	300	5	1	0.1557	0.116	1075	44M
L	5K	300	5	1	0.1517	0.113	1026	44M
E	5K	300	5	1	0.1291	0.095	812	44M
E	25K	300	5	1	0.1290	0.101	818	165M
E	5K	400	5	1	0.1240	0.092	802	60M
E	5K	500	5	1	0.1230	0.091	808	75M
E	5K	600	5	1	0.1150	0.086	728	92M

Table 1: Performance of models under different settings of the vocabulary size and the latent dimension. The results are ranked by F1 score. The total number of examples in the test set is 21948. "Geo" is the geometry (**L**orentz, **P**oincaré, **E**uclidean) $|V|$ is the vocabulary size, d is the embedding dimension, L is the number of layers, C is the curvature, AJ is the average Jaccard score, #EM is the number of exact matches, and $|M|$ is the model size (in megabytes).

ture for Lorentz and Poincaré Ball. We can see that the curvature greater than 1 helps.

Layers. The best results for the Lorentz model are obtained for with 3 to 4 layers, in which the network propagates information between nodes from 3 to 4 hops away.

Metrics correlation. The metrics F1, average Jaccard similarity, and the number of exact matches are highly correlated. The correlation between F1 score and average Jaccard similarity of entries shown in Table 1 is 0.9927 and that between F1 score and number of exact matches is 0.9959.

Some examples of method names generated from each model are compared to the reference method name in Table 2, with samples selected to be representative of each model's predictive strengths and weaknesses.

In particular, the Open Graph Benchmark (Hu et al. 2020) has a public leaderboard for model performance on the `ogbg-code2` dataset[1]. We observe that the Lorentz model outperforms the best performing model using `GMAN+bag of tricks` with F1 score of 0.1770 ± 0.0012 (as of November 10th, 2021). Note that the leaderboard models benefit from standard regularization and stability techniques such as dropout and batch normalization. In contrast, these operations are not hyperbolic-friendly, so we do not include them in our hyperbolic implementation.

Additional Experiments

Readout layer. Using the Lorentz model with fixed hyperparameters, we sought to investigate the impact of the readout

[1]https://ogb.stanford.edu/docs/leader_graphprop/#ogbg-code2

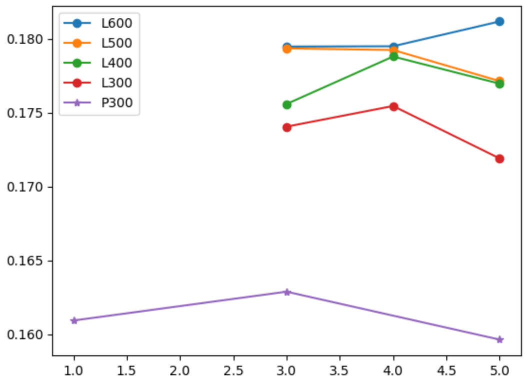

Figure 2: F1 vs Curvature for Lorentz ('L') and Poincaré Ball ('P') models for the embedding dimensions indicated. The vocabulary size is 25K.

layer on the model performance. The results of our analyses are summarized in Table 4. Variance across different graph pooling methods was relatively small. Tangent space pooling methods and the Einstein mean pooling approach (Einstein midpoint calculated across all nodes in a single graph) achieved the highest performances. While max pooling reached performed relatively well, we observed training instabilities with the presence of the max operator in the model, both in max pooling and in using ReLU in the hyperbolic activation layer. Note that since the centroid-based model is extremely memory-consuming and computationally more expensive than other approaches, we used a relatively simpler model for this experiment (Lorentz model with activation and dimensionality of 50) across all runs to enable a fair comparison. Our exploration of readout layers suggests that max pooling and Einstein mean pooling offer the greatest increase in model performance and training stability.

Tangent Space versus Einstein midpoint for message passing. We also tested the performance of the Einstein midpoint approach for message passing, since it is originally used to circumvent the tangent space projection step for hyperbolic message passing (Dai et al. 2021). The results of this approach are benchmarked against the tangent space aggregation used in the other models in our study. The Einstein approach is more computationally efficient than tangent space message passing, since it does not need to learn parameter weights during the message passing step. We applied the approaches to the best performing models in Lorentz space (vocabulary of 25K and embedding dimension of 550) and Poincaré Ball (vocabulary of 5K and embedding dimension of 600)[2]. From Table 3, we observe that *Einstein midpoint is helpful for Lorentz model and tangent space aggregation works better for Poincaré Ball model.*

Removing activation layer. Finally, we observed the effect of removing the activation layer in our hyperbolic models, instead using the nonlinearities present in projection steps

[2]These are the best settings for these models from Table 1.

Table 2: Representative method name predictions for each model.

Reference	Lorentz	Poincaré Ball	Euclidean
run_migrations_offline	**run_migrations_offline**	main_migrations_offline	main
render_to_response	**render_to_response**	**render_to_response**	render_email
process_result_value	process_bind_value	convert_value	get
query_yes_no	confirm_yes_no	yes_yes_no	**query_yes_no**
remove_temporary_source	delete_temporary	**remove_temporary_source**	reset

Table 3: The effect of activation layers and message passing approaches in hyperbolic models

			Manifold	
Activation	Message Passing	Metric	Lorentz-V25K-D550	Poincaré-V5K-D600
No	Tangent Space	F1	0.161	0.163
No	Einstein Midpoint	F1	0.180	0.148
Yes	Tangent Space	F1	0.132	0.163
Yes	Einstein Midpoint	F1	**0.181**	0.147

Table 4: Hyperbolic model performance with various readout functions.

Metric	Max	Mean	Sum	Centroid	Einstein	Virtual
Precision	**0.165**	0.161	0.162	0.148	0.163	0.149
Recall	**0.106**	0.104	0.104	0.096	0.105	0.088
F1	**0.122**	0.119	0.120	0.112	0.121	0.105

from hyperbolic space to tangent space. Ultimately, as is shown in Table 3, *the inclusion of the activation layer did not appear to change the outcome of the model performance, for either Lorentz or Poincaré Ball implementations*. This finding suggests that the projection step does indeed play a role of non-linear activation layers via non-linear mappings to the model. Additionally, we observed that removing the activation from our hyperbolic models improves the stability during training time. While we only used ReLU activation in our models, exploration of different activation functions in a hyperbolic setting may be useful.

Conclusion

We demonstrate the utility of HGCNs on the graph-level method name prediction task for source code. Additionally, we explore a number of architectural improvements for training hyperbolic neural networks involving message passing, activation, and readout layers. We establish a new state-of-the-art result on the `ogbg-code2` benchmark for the task using a graph convolutional neural network and a Lorentz hyperbolic model.

References

Allamanis, M.; Brockschmidt, M.; and Khademi, M. 2018. Learning to represent programs with graphs. *International Conference on Learning Representations*.

Allamanis, M.; Peng, H.; and Sutton, C. 2016. A convolutional attention network for extreme summarization of source code. In *International conference on machine learning*, 2091–2100. PMLR.

Alon, U.; Sadaka, R.; Levy, O.; and Yahav, E. 2020. Structural Language Models of Code. In *International Conference on Machine Learning*.

Alon, U.; Zilberstein, M.; Levy, O.; and Yahav, E. 2019. Code2vec: Learning Distributed Representations of Code. *Proc. ACM Program. Lang.*, 3(POPL).

Chami, I.; Ying, Z.; Ré, C.; and Leskovec, J. 2019. Hyperbolic Graph Convolutional Neural Networks. In Wallach, H.; Larochelle, H.; Beygelzimer, A.; d'Alché-Buc, F.; Fox, E.; and Garnett, R., eds., *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Dai, J.; Wu, Y.; Gao, Z.; and Jia, Y. 2021. A Hyperbolic-to-Hyperbolic Graph Convolutional Network. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 154–163.

Ganea, O.; Becigneul, G.; and Hofmann, T. 2018. Hyperbolic Neural Networks. In Bengio, S.; Wallach, H.; Larochelle, H.; Grauman, K.; Cesa-Bianchi, N.; and Garnett, R., eds., *Advances in Neural Information Processing Systems*, volume 31.

Gilmer, J.; Schoenholz, S. S.; Riley, P. F.; Vinyals, O.; and Dahl, G. E. 2017. Neural Message Passing for Quantum Chemistry. *Proceedings of the 34th International Conference on Machine Learning*, 1263–1272.

Hu, W.; Fey, M.; Zitnik, M.; Dong, Y.; Ren, H.; Liu, B.;

Catasta, M.; and Leskovec, J. 2020. Open Graph Benchmark: Datasets for Machine Learning on Graphs. In *Advances in Neural Information Processing Systems, 2020*.

Husain, H.; Wu, H.-H.; Gazit, T.; Allamanis, M.; and Brockschmidt, M. 2019. Codesearchnet challenge: Evaluating the state of semantic code search. *arXiv preprint arXiv:1909.09436*.

Kipf, T. N.; and Welling, M. 2016. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*.

Li, J.; Cai, D.; and He, X. 2017. Learning Graph-Level Representation for Drug Discovery. *CoRR*.

Liu, Q.; Nickel, M.; and Kiela, D. 2019. Hyperbolic Graph Neural Networks. In Wallach, H.; Larochelle, H.; Beygelzimer, A.; d'Alché-Buc, F.; Fox, E.; and Garnett, R., eds., *Advances in Neural Information Processing Systems*, volume 32.

Lou, A.; Katsman, I.; Jiang, Q.; Belongie, S.; Lim, S.-N.; and De Sa, C. 2020. Differentiating through the Fréchet Mean. In III, H. D.; and Singh, A., eds., *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, 6393–6403. PMLR.

Peng, W.; Varanka, T.; Mostafa, A.; Shi, H.; and Zhao, G. 2021. Hyperbolic Deep Neural Networks: A Survey. arXiv:2101.04562.

Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Lio, P.; and Bengio, Y. 2018. Graph attention networks. In *International Conference on Learning Representations*.

Zügner, D.; Kirschstein, T.; Catasta, M.; Leskovec, J.; and Günnemann, S. 2021. Language-agnostic representation learning of source code from structure and context. *arXiv preprint arXiv:2103.11318*.

Appendix

Hyperbolic Geometry

Riemannian Metric

In order to understand the tangent space operations for HGCN, we start by defining a Riemannian metric, which is a set of smoothly varying inner products on tangent spaces $g_{\mathbf{x}} : \mathcal{T}_{\mathbf{x}}\mathcal{M} \times \mathcal{T}_{\mathbf{x}}\mathcal{M} \to \mathbb{R}$. Riemannian metrics have been used to measure distances on Riemannian manifolds, which are defined by a smooth manifold and Riemannian metric pair.

Lorentz (Hyperboloid) Model

First, the Minkowski inner product between $(d+1)$-dimensional vectors $\mathbf{x}$ and $\mathbf{y}$ is:

$$\langle \mathbf{x}, \mathbf{y} \rangle_{\mathcal{L}} := -x_0 y_0 + x_1 y_1 + ... + y_d$$

Then, we define the d-dimensional Lorentz model of hyperbolic space with unit imaginary radius and constant negative curvature of -1 as a Riemannian manifold in which:

$$\mathbb{H}^{d,1} := \{\mathbf{x} \in \mathbb{R}^{d+1} : \langle \mathbf{x}, \mathbf{x} \rangle_{\mathcal{L}} = -1, x_0 > 0\}$$

and

$$g_{\mathbf{x}} := \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{bmatrix}$$

Next, we establish projections to and from the Lorentz manifold and its tangent spaces. For a point $\mathbf{x} = (x_0, \mathbf{x}_{1:D}) \in \mathbb{R}^{d+1}$, we can transport it to the Lorentz manifold with the following projection:

$$\Pi_{\mathbb{R}^{d+1} \to \mathbb{H}^{d,1}}(\mathbf{x} := (\sqrt{1 + ||\mathbf{x}_{1:d}||_2^2}, \mathbf{x}_{1:d})$$

For the reverse projection, a point $\mathbf{y} \in \mathbb{R}^{d+1}$ can be projected onto its corresponding Lorentz manifold with the following transformation:

$$\Pi_{\mathbb{R}^{d+1} \to \mathcal{T}_{\mathbf{x}}\mathbb{H}^{d,1}}(\mathbf{y}) := \mathbf{y} + \langle \mathbf{x}, \mathbf{y} \rangle_{\mathcal{L}\mathbf{x}}$$

From here, we can define the exponential and logarithmic maps as follows. Given points $\mathbf{x}$ and $\mathbf{y}$ in our hyperbolic space $\mathbb{H}^{d,K}$ with dimension d and of constant negative curvature $1/K$, the Euclidean tangent space centered at $\mathbf{x}$ is denoted as $\mathcal{T}_{\mathbf{x}}\mathbb{H}^{d,K}$. Then, we can have a tangent vector $\mathbf{v} \in \mathcal{T}_{\mathbf{x}}\mathbb{H}^{d,K}$ such that $\mathbf{v} \neq \mathbf{0}$. Assuming that $\mathbf{x} \neq \mathbf{y}$, we can define the exponential and logarithmic maps under the hyperboloid model as:

$$\exp_{\mathbf{x}}(\mathbf{v}) = \cosh\Big(\frac{||\mathbf{v}||_{\mathcal{L}}}{\sqrt{K}}\Big)\mathbf{x} + \sqrt{K}\sinh\Big(\frac{||\mathbf{v}||_{\mathcal{L}}}{\sqrt{K}}\Big)\frac{\mathbf{v}}{||\mathbf{v}||_{\mathcal{L}}}$$

$$\log_{\mathbf{x}}^{K}(\mathbf{y}) = d_{\mathcal{L}}^{K}(\mathbf{x}, \mathbf{y})\frac{\mathbf{y} + \frac{1}{K}\langle \mathbf{x}, \mathbf{y} \rangle_{\mathcal{L}}\mathbf{x}}{||\mathbf{y} + \frac{1}{K}\langle \mathbf{x}, \mathbf{y} \rangle_{\mathcal{L}}\mathbf{x}||_{\mathcal{L}}}$$

Further details can be found in (Chami et al. 2019).

Poincaré Ball Model

The Poincaré ball model with unit radius and constant negative curvature -1 in d dimensions is defined as the Riemannian manifold:

$$\mathbb{D}^{d,1} := \{\mathbf{x} \in \mathbb{R}^d : ||\mathbf{x}||^2 < 1\}$$

$$g_{\mathbf{x}} = \lambda_{\mathbf{x}}^2 I_d$$

where $\lambda_{\mathbf{x}} := \frac{2}{1 - ||\mathbf{x}||_2^2}$ and I_d is the identity matrix.

The exponential and logarithmic maps for the Poincaré Ball model can be defined in Poincaré Ball space $\mathcal{B}$ for $\mathbf{x} \in \mathcal{B}$, the tangent vector $\mathbf{v} \neq \mathbf{0}$ and the point $\mathbf{y} \neq \mathbf{0}$:

$$\exp_{\mathbf{x}}(\mathbf{v}) = \mathbf{x} \oplus \left(\tanh(\frac{\lambda_{\mathbf{x}}||\mathbf{v}||}{2})\frac{\mathbf{v}}{||\mathbf{v}||}\right)$$

$$\log_{\mathbf{x}}(\mathbf{y}) = \frac{2}{\lambda_{\mathbf{x}}}\operatorname{arctanh}(||-\mathbf{x} \oplus \mathbf{y}||)\frac{-\mathbf{x} \oplus \mathbf{y}}{||-\mathbf{x} \oplus \mathbf{y}||}$$

where $\oplus$ represents Möbius addition for any $\mathbf{x}, \mathbf{y} \in \mathcal{B}$:

$$\mathbf{x} \oplus \mathbf{y} = \frac{(1 + 2\langle \mathbf{x}, \mathbf{y} \rangle + ||\mathbf{y}||^2)\mathbf{x} + (1 - ||\mathbf{x}||^2)\mathbf{y}}{1 + 2\langle \mathbf{x}, \mathbf{y} \rangle + ||\mathbf{x}||^2||\mathbf{y}||^2}$$

Hyperbolic Message Passing

Einstein midpoint is a method of calculating the midpoint (or mean) of points in hyperbolic space. It can be used as a message passing scheme and it is defined in the Lorentz model of hyperbolic space, therefore we will start by defining the projections from Lorentz space to Lorentz space.

Klein Model

The projection from Lorentz space, $\mathcal{L}$ to Klein space $\mathcal{L}$ are defined as follows. Given a point $\mathbf{x} = [x_0, x_1, ..., x_n] \in \mathcal{L}$ and its corresponding point $\mathbf{k} = [k_0, k_1, ..., k_{n-1}] \in \mathcal{K}$, we can define the projection from Lorentz space to Klein space as:

$$p_{\mathcal{L}\to\mathcal{K}}(\mathbf{x}) = \frac{[x_1, ..., x_n]}{x_0}$$

Next, we can define the projection from Klein space back to Lorentz space as:

$$p_{\mathcal{K}\to\mathcal{L}}(\mathbf{k}) = \frac{1}{\sqrt{1 - ||\mathbf{k}||^2}}[1, \mathbf{k}]$$

Einstein Midpoint

Given a set of node representations $\mathbf{h}_i$ and the indices of its neighbors $j \in \mathcal{N}(i)$, we can now define the Einstein midpoint aggregation as the following set of transformations:

$$\begin{aligned}
\mathbf{h}_j^{\mathcal{K}} &= p_{\mathcal{L}\to\mathcal{K}}(\mathbf{h}_j^{\mathcal{L}}) \\
\mathbf{m}_i^{\mathcal{K}} &= \sum_{j\in\mathcal{N}(i)} \gamma_j \mathbf{h}_j^{\mathcal{K}} / \sum_{j\in\mathcal{N}(i)} \gamma_j \\
\mathbf{m}_i^{\mathcal{L}} &= p_{\mathcal{K}\to\mathcal{L}}(\mathbf{m}_i^{\mathcal{K}})
\end{aligned}$$

where $\gamma_j = \frac{1}{\sqrt{1-||\mathbf{h}_j^{\mathcal{K}}||^2}}$ is the Lorentz factor. Further details can be found in (Dai et al. 2021).