Residual Hyperbolic Graph Convolution Networks

Yangkai Xue[1], Jindou Dai[1], Zhipeng Lu[2*], Yuwei Wu[1,2*], Yunde Jia[2,1]

[1]Beijing Key Laboratory of Intelligent Information Technology, School of Computer Science & Technology, Beijing Institute of Technology, China

[2]Guangdong Laboratory of Machine Perception and Intelligent Computing, Shenzhen MSU-BIT University, China

xue.yangkai@bit.edu.cn, daijindou@foxmail.com, zhipeng.lu@hotmail.com, wuyuwei@bit.edu.cn, jiayunde@smbu.edu.cn

Abstract

Hyperbolic graph convolutional networks (HGCNs) have demonstrated representational capabilities of modeling hierarchical-structured graphs. However, as in general GCNs, over-smoothing may occur as the number of model layers increases, limiting the representation capabilities of most current HGCN models. In this paper, we propose residual hyperbolic graph convolutional networks ($\mathcal{R}$-HGCNs) to address the over-smoothing problem. We introduce a hyperbolic residual connection function to overcome the over-smoothing problem, and also theoretically prove the effectiveness of the hyperbolic residual function. Moreover, we use product manifolds and HyperDrop to facilitate the $\mathcal{R}$-HGCNs. The distinctive features of the $\mathcal{R}$-HGCNs are as follows: (1) The hyperbolic residual connection preserves the initial node information in each layer and adds a hyperbolic identity mapping to prevent node features from being indistinguishable. (2) Product manifolds in $\mathcal{R}$-HGCNs have been set up with different origin points in different components to facilitate the extraction of feature information from a wider range of perspectives, which enhances the representing capability of $\mathcal{R}$-HGCNs. (3) HyperDrop adds multiplicative Gaussian noise into hyperbolic representations, such that perturbations can be added to alleviate the over-fitting problem without deconstructing the hyperbolic geometry. Experiment results demonstrate the effectiveness of $\mathcal{R}$-HGCNs under various graph convolution layers and different structures of product manifolds.

Introduction

Hyperbolic graph convolutional networks (HGCNs) have been an emerging research topic due to its superior representation capabilities of modeling hierarchical graphs [Chami et al. 2019, Dai et al. 2021]. Different from Euclidean spaces with a polynomial expanding space volume, hyperbolic spaces increase exponentially growth of space volume with radius, which is well-suited to the geometry of hierarchical data. Benefiting from this property, great progress has been made by generalizing Euclidean methods to hyperbolic spaces such as hyperbolic graph convolutional networks [Chami et al. 2019, Dai et al. 2021], hyperbolic image embeddings [Khrulkov et al. 2020], and hyperbolic word embeddings [Nickel and Kiela 2017, 2018]. However, the over-smoothing issue impeding the development of deep HGCNs, over-smoothing means node features becomes indistinguishable after passing through a large number of graph convolution layers. It proved that graph convolution is a special form of Laplacian smoothing [Li, Han, and Wu 2018] . Smoothing on nodes can reduce intra-class differences, while over-smoothing makes model less discriminative with indistinguishable node features.

In this paper, we propose residual hyperbolic graph convolutional networks ($\mathcal{R}$-HGCNs) to address the over-smoothing problem. Specifically, we introduce a hyperbolic residual connection function and use product manifolds and HyperDrop into HGCNs. The details are as follows: (1) The hyperbolic residual connection transmits the initial node information to each layer to prevent node features from being indistinguishable, and the hyperbolic identity mapping prevents performance degradation caused by deepening the models. (2) Product manifolds pick different origin points on different components, which makes the same input have different embedding results in different components, giving them the ability to view the graph structure from different perspectives. This enhances the representational ability of $\mathcal{R}$-HGCNs. (3) HyperDrop adds multiplicative Gaussian noise on hyperbolic neurons to alleviate the over-fitting issue, inheriting the insight of training with noise and preserving the hyperbolic geometry, Extensive experiments demonstrate the effectiveness of $\mathcal{R}$-HGCNs under various graph convolution layers and different structures of product manifolds. The contributions of this paper are summarized as follows:

- We propose $\mathcal{R}$-HGCN, a product manifold based deep hyperbolic graph convolutional network that makes up for the deficiency of existing HGCNs for capturing long-range relationships in graphs.
- We design hyperbolic residual connection that addresses the over-smoothing issue while deepening HGCNs and theoretically prove the effectiveness of the hyperbolic residual connection.
- We propose to use product manifolds with different origin points in different components, which enables $\mathcal{R}$-HGCNs to extract more comprehensive features from the data.
- We develop HyperDrop, a regularization method tailored for hyperbolic representations. It can improve $\mathcal{R}$-HGCNs' generalization ability, alleviating the over-fitting issue.

[*]Corresponding authors.

Related Work

Graph convolutional networks typically embed graphs in Euclidean spaces since Euclidean spaces are the most commonly used and easy to calculate. Many researchers have noticed that Euclidean spaces have limitations while modeling data with hierarchical structure. Sa *et al.*[De Sa et al. 2018] claimed that it is not possible to embed trees into Euclidean spaces with arbitrarily low distortion, even in an infinite-dimensional Euclidean space. Meanwhile, trees can be embedded into a two-dimensional hyperbolic space with arbitrarily low distortion. Such a surprising fact benefits from the pretty property of hyperbolic spaces: unlike the volume of a ball in Euclidean space, which expands polynomially with radius, the volume of space in hyperbolic space grows exponentially with radius. [Liu, Nickel, and Kiela 2019]. Thus, hyperbolic spaces are commonly viewed as a smooth version of tree and more suitable to model hierarchical data.

Several works discovered that graphs, *e.g.*, biological networks and social networks, exhibit a highly hierarchical structure [Krioukov et al. 2010, Papadopoulos et al. 2012]. Krioukov *et al.*[Krioukov et al. 2010] proved that the typical properties such as power-law degree distribution and strong clustering in such graphs are closely related to the curvature of hyperbolic spaces. Based on the above observations, generalizing GCNs from Euclidean spaces to hyperbolic spaces have been an emerging research topic. Liu *et al.*[Liu, Nickel, and Kiela 2019] and Chami *et al.*[Chami et al. 2019] first bridged the research gap and concurrently proposed HGCNs. Following the above works, many advanced techniques are proposed to improve HGCNs. Dai *et al.*[Dai et al. 2021] discovered that performimg graph convolution in tangent space will distort the global structure of hyperbolic spaces because tangent space is only a local approximation of hyperbolic manifolds. Yao *et al.*[Yao, Pi, and Chen 2022] designs a Hyperbolic Skipped Knowledge Graph Convolutional Network to capture the network structure characteristics in hyperbolic knowledge embeddings. Liu *et al.*[Liu and Lang 2023] propose a Multi-curvature Hyperbolic Heterogeneous Graph Convolutional Network (McH-HGCN) based on type triplets for heterogeneous graphs.

Preliminaries

Hyperbolic Manifold. A Riemannian manifold or Riemannian space $(\mathcal{M}, g)$ is a real and smooth manifold $\mathcal{M}$ equipped with a positive-definite metric tensor g. It is a topological space that is locally homeomorphic to an Euclidean space at each point $\vec{x} \in \mathcal{M}$, and the local Euclidean space is termed the tangent space $\mathcal{T}_{\vec{x}}\mathcal{M}$.

Lorentz Model. A d-dimensional Lorentz model ($\mathcal{L}^d$, g) is defined by the manifold $\mathcal{L}^d = \{\vec{x} = [x_0, x_1, \cdots, x_d] \in \mathbb{R}^{d+1} : \langle \vec{x}, \vec{x} \rangle_{\mathcal{L}} = -1, \vec{x}_0 > 0\}$ where the Lorentz inner product is defined as

$$\langle \vec{x}, \vec{y} \rangle_{\mathcal{L}} = \vec{x}^\top g \vec{y} = -x_0 y_0 + \sum_{i=1}^{d} x_i y_i, \tag{1}$$

and the metric tensor $g = \mathrm{diag}([-1, 1, \cdots, 1])$ where $\mathrm{diag}(\cdot)$ denotes a diagonal matrix.

Exponential and logarithmic maps. Mappings between Riemannian manifold and their tangent spaces are termed exponential and logarithmic maps. Let $\vec{x}$ be a point on the Lorentz manifold $\mathcal{L}$, $\mathcal{T}_{\vec{x}}\mathcal{L}$ be the tangent space at $\vec{x}$, and $\vec{v}$ be a vector on the tangent space $\mathcal{T}_{\vec{x}}\mathcal{L}$. The exponential map $\exp_{\vec{x}}(\vec{v})$ that projects $\vec{v}$ onto the manifold $\mathcal{L}$ is defined as

$$\exp_{\vec{x}}(\vec{v}) = \cosh(\|\vec{v}\|_{\mathcal{L}})\vec{x} + \sinh(\|\vec{v}\|_{\mathcal{L}})\frac{\vec{v}}{\|\vec{v}\|_{\mathcal{L}}}, \tag{2}$$

where $\|\vec{v}\|_{\mathcal{L}} = \sqrt{\langle \vec{v}, \vec{v} \rangle_{\mathcal{L}}}$ is the norm of $\vec{v}$. The logarithmic map, inverse to the exponential map at $\vec{x}$, is given by

$$\log_{\vec{x}}(\vec{y}) = \frac{\mathrm{arcosh}(-\langle \vec{x}, \vec{y} \rangle_{\mathcal{L}})}{\sqrt{\langle \vec{x}, \vec{y} \rangle_{\mathcal{L}}^2 - 1}}(\vec{y} + \langle \vec{x}, \vec{y} \rangle_{\mathcal{L}}\vec{x}). \tag{3}$$

Parallel Transport. The generalization of parallel translation to non-Euclidean geometry is termed parallel transport. For two points $\vec{x}, \vec{y} \in \mathcal{L}$ on the Lorentz model, the parallel transport of a tangent vector $\vec{v} \in \mathcal{T}_{\vec{x}}\mathcal{L}$ on the tangent space at $\vec{x}$ to the tangent space $\mathcal{T}_{\vec{y}}\mathcal{L}$ at $\vec{y}$, along a smooth curve on the Lorentz model, is defined as

$$P_{\vec{x}\to\vec{y}}(\vec{v}) = \vec{v} - \frac{\langle \log_{\vec{x}}(\vec{y}), \vec{v} \rangle_{\mathcal{L}}}{d_{\mathcal{L}}(\vec{x}, \vec{y})^2}\big(\log_{\vec{x}}(\vec{y}) + \log_{\vec{y}}(\vec{x})\big). \tag{4}$$

Method

We propose residual hyperbolic graph convolutional networks ($\mathcal{R}$-HGCNs) to address the over-smoothing problem and enhance the representational ability of HGCNs.

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ denote a graph with a vertex set $\mathcal{V}$ and an edge set $\mathcal{E}$. $\boldsymbol{X} \in \mathbb{R}^{n \times d}$ denotes the node features that typically lie in Euclidean spaces. n is the number of nodes, and d is the dimension of the node features.

Residual Hyperbolic Graph Convolution

Lorentz Operations. Due to the strict manifold constraint of the Lorentz model, basic operations (matrix multiplication, vector addition, *etc*) are non-trivial to be generalized to Lorentz representations. Based on the maps mentioned above, we define the following operations.

Definition 1 *(Lorentz matrix-vector multiplication). Let* $\boldsymbol{W}$ *be a* $(d+1) \times (d+1)$ *real matrix and* $\vec{x} \in \mathcal{L}$ *be an input in the Lorentz model. Then we define the Lorentz matrix-multiplication as*

$$\boldsymbol{W} \otimes \vec{x} := \exp_{\vec{o}}\big(\boldsymbol{W}\log_{\vec{o}}(\vec{x})\big), \tag{5}$$

where $\exp(\cdot)$ *and* $\log(\cdot)$ *are defined as Eqs* (2) *and* (3).

Definition 2 *(Lorentz scalar multiplication). The Lorentz scalar multiplication of a scale* ξ *and* $\vec{x} \in \mathcal{L}$ *on the Lorentz model is defined as*

$$\xi \odot \vec{x} := \exp_{\vec{o}}\big(\xi\log_{\vec{o}}(\vec{x})\big). \tag{6}$$

Definition 3 *(Lorentz vector addition). The Lorentz vector addition of* $\vec{x}, \vec{y} \in \mathcal{L}$ *on the Lorentz model is defined as*

$$\vec{x} \oplus \vec{y} := \exp_{\vec{x}}(P_{\vec{o}\to\vec{x}}(\log_{\vec{o}}(\vec{y}))), \tag{7}$$

where $P_{\vec{o}\to\vec{x}}(\cdot)$ *is the parallel transport operator defined in Eq* (4).

Definition 4 *(Lorentz activation function). For $\vec{x} \in \mathcal{L}$, the Lorentz activation function on the Lorentz model is defined as*

$$\sigma_{\mathcal{L}}(\vec{x}) := \exp_{\vec{o}}\big(\sigma(\log_{\vec{o}}(\vec{x}))\big), \tag{8}$$

where $\sigma(\cdot)$ can be any activation function such as $\mathrm{ReLU}(\cdot)$.

Residual Hyperbolic Graph Convolution. Since the input of a hyperbolic graph convolutional network is required to be hyperbolic, we construct the initial Lorentz node features $\boldsymbol{H}^{(0)} \in \mathbb{R}^{n\times(d+1)}$ whose i-th row $\boldsymbol{H}_i^{(0)}$ being the Lorentz feature of the i-th node is generated by

$$\begin{aligned}\boldsymbol{H}_i^{(0)} &= \exp_{\vec{o}}\big([0, \boldsymbol{X}_i]\big)\\ &= \Big[\cosh\big(\|\boldsymbol{X}_i\|_2\big), \sinh\big(\|\boldsymbol{X}_i\|_2\big)\frac{\boldsymbol{X}_i}{\|\boldsymbol{X}_i\|_2}\Big],\end{aligned} \tag{9}$$

where $\boldsymbol{X}_i$ denotes the i-row of $\boldsymbol{X}$. Such a construction is based on the fact that $[0, \boldsymbol{X}_i] \in \mathbb{R}^{d+1}$ can be viewed as a tangent vector on the tangent space at the origin point, satisfying $\langle \vec{o}, [0, \boldsymbol{X}_i]\rangle_{\mathcal{L}} = 0$ where $\vec{o} = [1, 0, \cdots, 0]$ is the origin point of the Lorentz model.

The performance of a hyperbolic graph convolutional network declines as the number of graph convolution layers increases, that is called *over-smoothing* issue [Li, Han, and Wu 2018]. This is because the graph convolution is proved to be a special form of Laplacian smoothing, making node features tend to be indistinguishable after extensive graph convolutions[Li, Han, and Wu 2018]. Inspired by [Chen et al. 2020], we design hyperbolic residual connection and hyperbolic identity mapping to tackle this issue. The residual hyperbolic graph convolution operator $\mathrm{hgc}(\cdot)$ is defined as

$$\begin{aligned}\mathrm{hgc}(\boldsymbol{H}) &= \sigma_{\mathcal{L}}\Big(\big((1-\beta)\boldsymbol{I} + \beta\boldsymbol{W}\big)\otimes\bar{\boldsymbol{H}}\Big),\\ \bar{\boldsymbol{H}} &= \big((1-\alpha)\odot(\tilde{\boldsymbol{A}}\otimes\boldsymbol{H})\big)\oplus\big(\alpha\odot\boldsymbol{H}_0\big),\end{aligned} \tag{10}$$

where $\otimes$, $\odot$, and $\oplus$ are the Lorentz matrix-vector multiplication (Definition 1), the Lorentz scalar multiplication (Definition 2), and the Lorentz vector addition (Definition 3). Here $\sigma_{\mathcal{L}}(\cdot)$ is the Lorentz activation function (Definition 4). α and β are hyper-parameters to control the weight of hyperbolic residual connection and hyperbolic identity mapping.

Formally, at the ℓ-th layer, residual hyperbolic graph convolution performs like

$$\begin{aligned}\boldsymbol{H}^{(\ell)} &= \mathrm{hgc}(\boldsymbol{H}^{(\ell-1)})\\ &= \sigma_{\mathcal{L}}\Big(\big((1-\beta_\ell)\boldsymbol{I} + \beta_\ell\boldsymbol{W}^{(\ell)}\big)\otimes\bar{\boldsymbol{H}}\Big),\\ \bar{\boldsymbol{H}} &= \big((1-\alpha_\ell)\odot(\tilde{\boldsymbol{A}}\otimes\boldsymbol{H}^{(\ell-1)})\big)\oplus\big(\alpha_\ell\odot\boldsymbol{H}^{(0)}\big),\end{aligned} \tag{11}$$

where $\mathrm{hgc}(\cdot)$ takes the node features $\boldsymbol{H}^{(\ell-1)}$ from the previous layer, and outputs the node features $\boldsymbol{H}^{(\ell)}$ at the ℓ-th layer.

Compared to the convolution operator in vanilla GCNs, *i.e.* $\boldsymbol{H}^{(\ell)} = \sigma(\tilde{\boldsymbol{A}}\boldsymbol{H}^{(\ell-1)}\boldsymbol{W})$, $\mathrm{hgc}(\cdot)$ relieves over-smoothing issue through two modifications: (1) hyperbolic residual connection adds information paths from initial node features to each graph convolution layer, such that no matter how deep a hyperbolic graph convolutional network is, the node features at the top layer still combine initial node features avoid becoming indistinguishable; (2) hyperbolic identity mapping ensures that a deep hyperbolic graph convolutional network is not worse than a shallow model. In the extreme case where the values of β are set to be zero after the i-th layer, the model degenerates to an i-layer GCN no matter how deep it is.

Effectiveness of Hyperbolic Residual Connection

This section is meant to theoretically explain the efficiency of our network architecture. Inspired by [Cai and Wang 2020], we first define a hyperbolic version of “Dirichlet energy" for tracking node embeddings as follows. The Dirichlet energy of a function measures the "smoothness" of a unit norm function. The indistinguishable parameters leading to the over-smoothing issue result in small Dirichlet energy. For details of formulas and proofs in this section, see the supplementary material.

Definition 5 *Dirichlet energy $E(f)$ of a scalar function $f \in \mathcal{L}^d \subset \mathbb{R}^{d+1}$ on the graph G is defined as*

$$E(f) = \log_o(f)^T\tilde{\Delta}\log_o(f),$$

where $\tilde{\Delta} = I_{d+1} - \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}, \tilde{A} = A + I_{d+1}, \tilde{D} = D + I_{d+1}$, while A and D are the adjacency and degree matrices of G. For a vector field $F_{(d+1)\times c} = (f_1, \ldots, f_c)$, its Dirichlet energy is

$$E(F) = \mathrm{tr}(\log_o(F)^T\tilde{\Delta}\log_o(F)).$$

Note that the defined Dirichlet energy always pulls back the node embedding in Lorentz space to its tangent space at the origin point.

If the hyperbolic graph convolution operator $hgc(\cdot)$ as in (10) has no original input, i.e. $\bar{H} = \tilde{P}\otimes H$ therein, then

$$E(H^{(l)}) \le (1-\lambda)^2\|(1-\beta_l)I + \beta_l\boldsymbol{W}^{(l)}\|_2^2E(H^{(l-1)}), \tag{12}$$

where $0 < \lambda < 2$ is the smallest non-zero eigenvalue of $\tilde{\Delta}$ and $\|X\|_2$ denotes the maximal singular value of X. Note that $\|X\|_2 = \max_{\|u\|_2=1}\|Xu\|_2$ for any matrix X. Hence by the triangle inequality,

$$\|((1-\beta_l)I + \beta_l\boldsymbol{W}^{(l)})u\|_2 \le 1 - \beta_l + \beta_l\|\boldsymbol{W}^{(l)}u\|_2. \tag{13}$$

Actually $\|Xu\|_2$ for any weight matrix X may be estimated by

Lemma 1 *If $X = (X_{ij})$ is a $n\times n$ weight matrix, i.e. $\sum_{j=1}^n X_{ij} = 1, X_{ij} \ge 0$, then for any $u \in \mathbb{R}^n$ with $\|u\|_2 = 1$, $\|Xu\|_2 \le \sqrt{n}$.*

Hence combined with (12) and (13), we easily prove that in HGCNs without initial input, we have $E(H^{(l)}) \le d(1-\lambda)^2E(H^{(l-1)})$. Thus if the graph G does not have enough expansion, say $(1-\lambda) < \frac{1}{\sqrt{d}}$, then HGCNs would be exponentially over-smoothing as the number of layers increases.

The above result suggests us add an initial input as in $hgc(\cdot)$,

$$\bar{H} = \Big(\Big((1-\alpha_l)\odot(\tilde{P}\otimes H^{(l-1)})\Big)\oplus\Big(\alpha_l\odot H^{(0)}\Big)\Big),$$

seeking to interfere the decreasing of Dirichlet energy, which is the motivation of $\mathcal{R}$-HGCNs.

We investigate in details the interference of initial input. For simplicity we assume that the features in process all have positive entries so that ReLU does not affect the evaluation of Dirichlet energy. Thus utilizing the same argument as in the proof of (12) in the case with initial input, we have

$$E(H^{(l)}) = (y_l \log_o(z))^T \tilde{\Delta}(y_l \log_o(z)), \tag{14}$$

with $y_l = (1-\beta_l)I + \beta_l W^{(l)}$ and $z = \exp_{z_1}(\alpha_l P_{o\to z_1}(\log_o(H^{(0)})))$ the last equality of which is due to linearity of parallel transport, and $z_1 = \exp_o((1-\alpha_l)\log_o(\tilde{P}\otimes H^{(l-1)})) = \exp_o(z_2)$.

By definition of parallel transport, we have

$$\begin{aligned} &P_{o\to z_1}(\log_o(H^{(0)})) \\ &= \log_o(H^{(0)}) - \frac{\langle \log_o(z_1), \log_o(H^{(0)})\rangle_{\mathcal{L}}}{d_{\mathcal{L}}(o, z_1)}(z_2 + \log_{z_1}(o)) \end{aligned} \tag{15}$$

Further by definition,

$$\begin{aligned} z_1 &= \cosh((1-\alpha_l)\|\tilde{P}\log_o(H^{(l-1)}))\|_{\mathcal{L}})o \\ &+ \frac{\sinh((1-\alpha_l)\|\tilde{P}\log_o(H^{(l-1)})\|_{\mathcal{L}})}{\|\tilde{P}\log_o(H^{(l-1)})\|_{\mathcal{L}}}\tilde{P}\log_o(H^{(l-1)}) \end{aligned} \tag{16}$$

Also by definition, $\exp_{z_1}(x) = \cosh(\|x\|_{\mathcal{L}})z_1 + \frac{\sinh(\|x\|_{\mathcal{L}})}{\|x\|_{\mathcal{L}}}x$. Then again by the property of parallel transport, we have

$$z = \theta_l \log_o(H^{(0)}) + \phi_l \tilde{P}H^{(l-1)} + \psi_l o, \tag{17}$$

where θ_l, ϕ_l, ψ_l are coefficients depending on α_l, β_l and the Lorentzian norm of the above 3 vectors. Noting that θ_l only depends on α_l and $H^{(0)}$, the effect of $\log_o(H^{(0)})$ is always not negligible. Also ($z = (z_0, \dots)$)

$$\log_o(z) = \frac{arcosh(z_0)}{\sqrt{z_0^2 - 1}}(z - z_0 o),$$

which removes z_0 from z and re-scale it by a large factor. Then altogether we have

$$\begin{aligned} E(H^{(l)}) &= (\tilde{\theta}_l \log_o(H^{(0)}) + \tilde{\phi}_l \tilde{P}H^{(l-1)} + \tilde{\psi}_l o)^T \tilde{\Delta}(\cdots) \\ &= \tilde{\theta}_l^2 E(H^{(0)}) + \cdots, \end{aligned} \tag{18}$$

where $\tilde{\theta}_l$ is negligible similarly. Thus we prove that in $\mathcal{R}$-HGCNs with initial input, the Dirichlet energy $E(H^{(l)})$ will be bounded away from zero even if the Dirichlet energy in the corresponding HGCNs without initial input decrease to zero.

Product Manifold

We use the product manifold of the Lorentz models as embedding space. The Lorentz components of the product manifold are independent of each other.

The product manifold is the Cartesian product of a sequence of Riemannian manifolds, each of which is called a *component*. Given a sequence of the Lorentz models $\mathcal{L}_1^{d_1}, \cdots, \mathcal{L}_j^{d_j}, \cdots, \mathcal{L}_k^{d_k}$ where d_j denotes the dimension of the j-th component, the product manifold is defined as $\mathbb{L} = \mathcal{L}_1^{d_1} \times \cdots \times \mathcal{L}_j^{d_j} \times \cdots \times \mathcal{L}_k^{d_k}$. The coordinate of a point $\vec{x}$ on $\mathbb{L}$ is written as $\vec{x} = [\vec{x}_1, \cdots, \vec{x}_j, \cdots \vec{x}_k]$ where $\vec{x}_j \in \mathcal{L}_j^{d_j}$. Similarly, the coordinate of a tangent vector $\vec{v} \in \mathcal{T}_{\vec{x}}\mathbb{L}$ is written as $\vec{v} = [\vec{v}_1, \cdots, \vec{v}_j, \cdots, \vec{v}_k]$ where $\vec{v}_j \in \mathcal{T}_{\vec{x}_j}\mathcal{L}_j^{d_j}$. For $\vec{x}, \vec{y} \in \mathbb{L}$ and $\vec{v} \in \mathcal{T}_{\vec{x}}\mathbb{L}$, the exponential and logarithmic maps on $\mathbb{L}$ are defined as

$$\exp_{\vec{x}}(\vec{v}) = [\exp_{\vec{x}_1}(\vec{v}_1), \cdots, \exp_{\vec{x}_j}(\vec{v}_j), \cdots, \exp_{\vec{x}_k}(\vec{v}_k)], \tag{19}$$

$$\log_{\vec{x}}(\vec{y}) = [\log_{\vec{x}_1}(\vec{y}_1), \cdots, \log_{\vec{x}_j}(\vec{y}_j), \cdots, \log_{\vec{x}_k}(\vec{y}_k)]. \tag{20}$$

Different from ordinary product manifolds, we use $\mathbb{L} = (\mathcal{L}^d)_{o_1} \times (\mathcal{L}^d)_{o_2} \times \cdots \times (\mathcal{L}^d)_{o_k}$, where $(\mathcal{L}^d)_{o_i}$ are copies of d-dimensional Lorentz spaces with randomly prescribed origin points $o_i \in \mathcal{L}^d$. This gives $\mathcal{R}$-HGCNs the ability to extract node features from a wider range of different perspectives. Mathematically, such construction of product manifolds is inspired by the general construction of manifolds using Euclidean strata with different coordinates.

Hyperbolic Dropout

Hyperbolic Dropout(HyperDrop) adds multiplicative Gaussian noise on Lorentz components to regularize the HGCNs and alleviate the over-fitting issue. Concretely, let $\mathbb{L} = \mathcal{L}_1^{d_1} \times \cdots \times \mathcal{L}_j^{d_j} \times \cdots \times \mathcal{L}_k^{d_k}$ denote a product manifold of k Lorentz models where d_j denotes the dimension of the j-th Lorentz component. Given an input $\vec{l} = [\vec{l}_1, \cdots, \vec{l}_j, \cdots \vec{l}_k] \in \mathbb{L}$ on the product manifold where $\vec{l}_j \in \mathcal{L}_j^{d_j}$, HyperDrop is formulated as

$$\begin{aligned} \vec{y} &= [\vec{y}_1, \cdots, \vec{y}_j, \cdots, \vec{y}_k], \\ \vec{y}_j &= \xi_j \odot f_{\theta_j}(\vec{l}_j), \\ \xi_j &\sim \mathcal{N}(1, \sigma^2), \end{aligned} \tag{21}$$

where ξ_j is the multiplicative Gaussian noise drawn from the Gaussian distribution $\mathcal{N}(1, \sigma^2)$. Following [Srivastava et al. 2014], we set $\sigma^2 = \eta/(1-\eta)$ where η denotes drop rate. $\odot$ denotes the Lorentz scalar multiplication that is the generalization of scalar multiplication to the Lorentz representations, defined in Definition 2. $f_{\theta_j}(\cdot)$ could be any realization of a desirable function, such as a neural network with parameters θ_j.

It is noted that we sample ξ_j from the Gaussian distribution instead of the Bernoulli distribution used in the standard dropout for the following reason. If ξ_j is drawn from the Bernoulli distribution and happens to be 0 (with a probability of η) at the ℓ-th neural network layer, the information flow of the j-th Lorentz component will be interrupted, leading to the deactivation of the j-th Lorentz component after the ℓ-th neural network layer. In contrast, ξ_j drawn from a Gaussian distribution with mean value 1 is exactly equal to 0 is a small probability event. Thus, the j-th Lorentz component always works.

We may interpret HyperDrop from a Bayesian perspective. For convenience, we take a single Lorentz model and the Lorentz linear transformation as an example, *i.e.* $\vec{y} = \xi \odot f_\theta(\vec{l})$ and $f_\theta(\vec{l}) = \vec{l} \otimes \theta$. We have

$$\vec{y} = \xi \odot (\vec{l} \otimes \theta), \ \ \xi \sim \mathcal{N}(1, \sigma^2) \tag{22}$$

equal to

$$\begin{gathered}\vec{y} = \vec{l} \otimes \boldsymbol{M}, \\ \text{with } m_{r,c} = \xi \theta_{r,c}, \text{ and } \xi \sim \mathcal{N}(1, \sigma^2),\end{gathered} \tag{23}$$

where $\otimes$ denotes the Lorentz matrix-vector multiplication as defined in Definition 1, $\boldsymbol{M}$ is the matrix with $m_{r,c}$ as entries, and θ is the matrix with $\theta_{r,c}$ as entries. Eq (23) can be interpreted as a Bayesian treatment that the posterior distribution of the weight is given by a Gaussian distribution *i.e.* $q_\phi(m_{r,c}) = \mathcal{N}(\theta_{r,c}, \sigma^2\theta_{r,c}^2)$. The HyperDrop sampling procedure Eq (22) can be interpreted as rising from a reparameterization of the posterior on the parameter $\boldsymbol{M}$ as shown in Eq (23).

Residual Hyperbolic Graph Convolutional Network

We then investigate our architecture of $\mathcal{R}$-HGCN and its effect of preventing over-smoothing. In fact, we always combine $\mathcal{R}$-HGCN with initial input. Note that our model is of the form $\mathbb{L} = (\mathcal{L}^d)_{o_1} \times (\mathcal{L}^d)_{o_2} \times \cdots \times (\mathcal{L}^d)_{o_k}$, where $(\mathcal{L}^d)_{o_i}$ are copies of d-dimensional Lorentz spaces with random prescribed origin points $o_i \in \mathcal{L}^d$. Then for any $x = (x_1, \ldots, x_k) \in \mathbb{L}$, its Dirichlet energy is defined as

$$E(x) = \max_{1 \le i \le k} (\log_{o_i}(x_i)^T \tilde{\Delta} \log_{o_i}(x_i)) := \max_{1 \le i \le k} E_i(x_i). \tag{24}$$

Then by the similar argument with the proof of (12), we can estimate $E_i(H_i^{(l)})$ separately and then opt for the maximal among them, which may be better behaved than any single component due to possible fluctuation.

Network Architecture. Let $\mathbb{L} = (\mathcal{L}^d)_{o_1} \times (\mathcal{L}^d)_{o_2} \times \cdots \times (\mathcal{L}^d)_{o_k}$ denote the product manifold of k Lorentz models where d_j is the dimension of the j-th Lorentz model. The initial node features $\boldsymbol{H}^{(0)}$ on the product manifold of Lorentz models are given by ($\vec{o} = [\vec{o}_1, \cdots, \vec{o}_j, \cdots, \vec{o}_k]$)

$$\begin{gathered}\boldsymbol{H}^{(0)} = [\boldsymbol{H}^{1,(0)}, \cdots, \boldsymbol{H}^{j,(0)}, \cdots, \boldsymbol{H}^{k,(0)}], \\ \boldsymbol{H}_i^{j,(0)} = \exp_{\vec{o}}\big([0, \boldsymbol{X}_i]\big),\end{gathered} \tag{25}$$

where $\boldsymbol{H}_i^{j,(0)} \in \mathbb{R}^{(d_j+1)}$ denotes the i-th row of the node features $\boldsymbol{H}^{j,(0)} \in \mathbb{R}^{n \times (d_j+1)}$ on the j-th Lorentz component.

The graph convolution on the product manifold of the Lorentz models combined with HyperDrop is realized by instantiating $f(\cdot)$ in Eq (21) as the hyperbolic graph convolution operator $\mathrm{hgc}(\cdot)$, *i.e.* Eq (11) becomes

$$\begin{gathered}\boldsymbol{H}^{(\ell)} = [\boldsymbol{H}^{1,(\ell)}, \cdots, \boldsymbol{H}^{j,(\ell)}, \cdots, \boldsymbol{H}^{k,(\ell)}], \\ \boldsymbol{H}_i^{j,(\ell)} = \xi_i^j \odot \mathrm{hgc}(\boldsymbol{H}^{j,(\ell-1)}), \\ \xi_i^j \sim \mathcal{N}(1, \sigma^2),\end{gathered} \tag{26}$$

Datasets	PUBMED	CITESEER	CORA	AIRPORT
Classes	3	6	7	4
Nodes	19,717	3,327	2,708	3,188
Edges	44,338	4,732	5,429	18,631
Features	500	3,703	1,433	4

Table 1: Dataset statistics.

where $\boldsymbol{H}^{j,(\ell)} \in \mathbb{R}^{n \times (d_j+1)}$ is the node features on the j-th Lorentz component at the ℓ-th layer. $\boldsymbol{H}_i^{j,(\ell)} \in \mathbb{R}^{(d_j+1)}$ is the i-th row (*i.e.*, the i-th node) of $\boldsymbol{H}^{j,(\ell)}$. $\odot$ denotes the Lorentz scalar multiplication defined in Definition 2. ξ_i^j is the random multiplicative noise drawn from the Gaussian distribution $\mathcal{N}(1, \sigma^2)$. We set $\sigma = \eta/(1-\eta)$ where η denotes the drop rate. The node features at the last layer can be used for downstream tasks. Taking the node classification task as an example, we map node features to the tangent spaces of the product manifolds, and send tangent representations to a fully-connected layer followed by a softmax for classification.

Experiments

Experiments are performed on the semi-supervised node classification task. We first evaluate the performance of $\mathcal{R}$-HGCN under different configurations of models, including various graph convolution layers and different structures of product manifolds. Then, we compare with several state-of-the-art Euclidean GCNs and HGCNs, showing that $\mathcal{R}$-HGCN achieves competitive results. Further, we compare with DropConnect[Wan et al. 2013], a related regularization mathod for deep GCNs.

Datasets and Baselines

We use four standard commonly-used citation network graph datasets : PUBMED, CITESEER, CORA and AIRPORT [Sen et al. 2008]. Dataset statistics are summarized in Table 1. Experiment details see in the supplementary material.

Validation Experiments

Here we demonstrate the effectiveness of the $\mathcal{R}$-HGCN and our regularization method under different model configurations. For $\mathcal{R}$-HGCN, increasing the number of hyperbolic graph convolution layers almost always brings improvements on three datasets.

The criterion for judging the effectiveness of HyperDrop is to test whether the performance of $\mathcal{R}$-HGCN is improved with the help of HyperDrop. In Table **??**, we report the performance of HyperDrop with various graph convolution layers and structures of product manifolds on PUBMED, CITESEER CORA and AIRPORT. We observe that in most experiment, HyperDrop improves the performance of $\mathcal{R}$-HGCN. For example, on CORA, $\mathcal{R}$-HGCN$_{[2\times8]}$ obtains 1.7%, 0.7%, 0.3% and 0.2% gains with 4, 8, 16 and 32 layers. The stable improvements demonstrate that HyperDrop can effectively improve the generalization ability of $\mathcal{R}$-HGCN.

We also compare $\mathcal{R}$-HGCN with a deep Euclidean method, GCNII. The hyperbolic residual connection and hyperbolic

Datasets	Methods	4 layers		8 layers		16 layers		32 layers	
		Original	HyperDrop	Original	HyperDrop	Original	HyperDrop	Original	HyperDrop
PUBMED	GCNII	79.3 ± 0.3		79.9 ± 0.3		79.9 ± 1.7		80.0 ± 1.9	
	$\mathcal{P}$-HGCN$_{[2\times8]}$	79.1 ± 0.2	**79.8** ± 0.3	80.0 ± 0.1	**80.3** ± 0.1	79.1 ± 0.2	79.2 ± 0.3	79.2 ± 0.3	79.5 ± 0.4
	$\mathcal{P}$-HGCN$_{[4\times4]}$	79.0 ± 0.2	79.5 ± 0.2	79.8 ± 0.1	80.1 ± 0.3	79.9 ± 0.3	80.0 ± 0.2	79.8 ± 0.4	**80.3** ± 0.3
	$\mathcal{P}$-HGCN$_{[8\times2]}$	79.0 ± 0.2	79.4 ± 0.2	79.7 ± 0.2	79.9 ± 0.2	80.0 ± 0.3	**80.1** ± 0.3	80.1 ± 0.2	80.3 ± 0.4
	$\mathcal{P}$-HGCN$_{[16\times1]}$	78.7 ± 0.3	79.2 ± 0.3	79.4 ± 0.1	79.9 ± 0.2	79.3 ± 0.3	79.5 ± 0.4	79.2 ± 0.2	80.1 ± 0.4
CITESEER	GCNII	69.3 ± 2		70.6 ± 1.4		70.5 ± 1.3		70.8 ± 1.6	
	$\mathcal{P}$-HGCN$_{[2\times8]}$	71.4 ± 0.2	72.0 ± 0.4	72.1 ± 0.2	72.3 ± 0.3	71.9 ± 0.5	71.9 ± 0.7	72.1 ± 0.6	72.3 ± 0.7
	$\mathcal{P}$-HGCN$_{[4\times4]}$	71.4 ± 0.2	72.0 ± 0.4	71.9 ± 0.3	72.2 ± 0.3	71.5 ± 0.5	71.7 ± 1.0	71.4 ± 0.2	71.9 ± 0.7
	$\mathcal{P}$-HGCN$_{[8\times2]}$	71.2 ± 0.8	70.4 ± 0.9	68.6 ± 0.8	71.1 ± 0.7	72.0 ± 0.2	71.9 ± 0.9	72.0 ± 0.1	72.0 ± 0.5
	$\mathcal{P}$-HGCN$_{[16\times1]}$	71.3 ± 0.3	**72.4** ± 0.3	71.9 ± 0.3	**72.5** ± 0.5	72.2 ± 0.4	**72.3** ± 0.4	72.3 ± 0.6	**72.5** ± 0.9
CORA	GCNII	76.6 ± 2.4		79.4 ± 1.4		81.3 ± 1.0		81.5 ± 1.4	
	$\mathcal{P}$-HGCN$_{[2\times8]}$	80.3 ± 0.7	**82.0** ± 0.5	81.5 ± 0.2	**82.2** ± 0.4	81.8 ± 0.2	**82.1** ± 0.2	81.9 ± 0.3	82.1 ± 0.1
	$\mathcal{P}$-HGCN$_{[4\times4]}$	80.3 ± 0.8	81.7 ± 0.5	81.3 ± 0.2	81.9 ± 0.4	81.7 ± 0.2	81.9 ± 0.2	81.6 ± 0.5	82.1 ± 0.7
	$\mathcal{P}$-HGCN$_{[8\times2]}$	77.9 ± 0.8	80.8 ± 0.8	80.2 ± 0.9	81.5 ± 0.3	81.7 ± 0.2	81.8 ± 0.2	81.9 ± 0.5	**82.3** ± 0.8
	$\mathcal{P}$-HGCN$_{[16\times1]}$	79.4 ± 0.8	81.9 ± 0.7	80.4 ± 0.4	82.5 ± 0.4	81.6 ± 0.6	81.5 ± 0.5	81.6 ± 0.7	81.4 ± 0.6
AIRPORT	GCNII	88.9 ± 0.5		89.1 ± 0.3		89.2 ± 0.4		89.6 ± 0.7	
	$\mathcal{P}$-HGCN$_{[2\times8]}$	88.9 ± 0.4	**89.1** ± 0.2	89.2 ± 1.0	89.4 ± 0.9	89.1 ± 0.3	89.3 ± 0.5	89.7 ± 0.2	90.0 ± 0.4
	$\mathcal{P}$-HGCN$_{[4\times4]}$	88.6 ± 0.6	89.0 ± 0.2	89.3 ± 0.4	89.4 ± 0.3	89.6 ± 0.1	89.6 ± 0.2	89.7 ± 0.1	**90.2** ± 0.7
	$\mathcal{P}$-HGCN$_{[8\times2]}$	88.7 ± 0.2	88.7 ± 0.4	89.3 ± 0.5	**89.6** ± 0.7	89.4 ± 0.2	**89.8** ± 0.5	89.6 ± 0.4	89.7 ± 0.7
	$\mathcal{P}$-HGCN$_{[16\times1]}$	88.5 ± 0.7	88.6 ± 0.4	88.6 ± 0.3	88.5 ± 0.5	88.7 ± 0.2	88.9 ± 0.5	89.0 ± 0.3	89.2 ± 0.5

Table 2: Comparisons on various graph convolution layers and different structures of 16-dimensional product manifolds w and w/o HyperDrop. We also compare with a related work GCNII using 16-dimensional embedding space. Mean accuracy (%) and standard deviation are reported. $\mathcal{P}$-HGCN$_{[d\times m]}$ denotes the $\mathcal{P}$-HGCN with a product manifold of m d-dimensional Lorentz models.

	Methods	PUBMED	CITESEER	CORA
Euclidean	GCN[Kipf and Welling 2017]	79.1 ± 0.3	71.2 ± 0.6	81.3 ± 0.5
	GAT[Veličković et al. 2017]	77.7 ± 0.2	70.9 ± 0.4	82.4 ± 0.6
	GraphSage[Hamilton, Ying, and Leskovec 2017]	77.3 ± 0.3	67.8 ± 1.1	77.3 ± 0.8
	SGC[Wu et al. 2019]	69.3 ± 0.0	78.9 ± 0.0	80.9 ± 0.0
	APPNP[Klicpera, Bojchevski, and Günnemann 2019]	80.1 ± 0.2	71.6 ± 0.4	**83.7** ± 0.5
	GCNII(8)[Chen et al. 2020]	79.9 ± 0.3	72.4 ± 0.9	83.5 ± 0.7
Hyperbolic	HGCN [Chami et al. 2019]	**80.3** ± 0.3	-	79.9 ± 0.2
	H2HGCN[Dai et al. 2021]	79.9 ± 0.5	-	82.8 ± 0.4
	κ GCN [Bachmann, Bécigneul, and Ganea 2020]	78.3 ± 0.6	70.7 ± 0.5	80.0 ± 0.6
	LGCN [Zhang et al. 2021]	78.6 ± 0.7	71.9 ± 0.7	83.3 ± 0.7
	$\mathcal{P}$-HGCN$_{[16\times1]}$(8)	79.4 ± 0.1	71.9 ± 0.3	80.4 ± 0.4
	$\mathcal{P}$-HGCN$_{[16\times1]}$(8)+HyperDrop	79.9 ± 0.2	**72.5** ± 0.5	82.5 ± 0.4
	$\mathcal{P}$-HGCN$_{[2\times8]}$(8)	80.0 ± 0.1	72.1 ± 0.2	81.5 ± 0.2
	$\mathcal{P}$-HGCN$_{[2\times8]}$(8)+HyperDrop	**80.3** ± 0.1	72.3 ± 0.3	82.2 ± 0.4

Table 3: Mean accuracy (%) and standard deviation on PUBMED, CITESEER, and CORA. We set the dimensions of embedding spaces to 16 for all methods and the number of graph convolution layers to 8 (number in parentheses) for deep models, *i.e.*, GCNII and $\mathcal{P}$-HGCN. $\mathcal{P}$-HGCN$_{[d\times m]}$ denotes the $\mathcal{P}$-HGCN with a product manifold of m d-dimensional Lorentz models.

identity mapping in $\mathcal{R}$-HGCN are inspired by GCNII. The main difference between $\mathcal{R}$-HGCN and GCNII is, $\mathcal{R}$-HGCN performs graph representation learning in hyperbolic spaces while GCNII is in Euclidean spaces. As shown in Table **??**, $\mathcal{R}$-HGCN shows superiority compared to GCNII. Actually, $\mathcal{R}$-HGCN is only baseline model we developed for evaluating the effectiveness of HyperDrop, and we give up extra training tricks for clear evaluations. For example, in Section , $\mathcal{R}$-HGCN obtains the same mean accuracy 83.5% as GCNII with 8 layers on CORA while using HyperDrop and DropConnect[Wan et al. 2013] together. DropConnect is used in other hyperbolic graph convolutional networks, such HGCN [Chami et al. 2019] and LGCN [Zhang et al. 2021]. We claim that the superior performance of $\mathcal{R}$-HGCN compared to GCNII benefits from the representing capabilities of hyperbolic spaces while dealing with hierarchical-structure data. It confirms the significance of hyperbolic representation learning.

Layers	$\mathcal{P}$-HGCN$_{[2\times8]}$		$\mathcal{P}$-HGCN$_{[4\times4]}$		$\mathcal{P}$-HGCN$_{[8\times2]}$		$\mathcal{P}$-HGCN$_{[16\times1]}$	
	with IRC	w/o IRC	with IRC	w/o IRC	with IRC	w/o IRC	with IRC	w/o IRC
2	78.9 ± 0.4	78.9 ± 0.2	79.0 ± 0.3	78.8 ± 0.3	78.8 ± 0.3	78.9 ± 0.4	78.7 ± 0.2	78.6 ± 0.4
4	79.1 ± 0.4	79.3 ± 0.2	79.0 ± 0.2	79.0 ± 0.6	79.2 ± 0.3	78.7 ± 0.4	79.2 ± 0.3	78.6 ± 0.4
8	80.3 ± 0.0	78.6 ± 0.2	80.2 ± 0.2	79.1 ± 0.2	80.1 ± 0.1	78.5 ± 0.6	79.8 ± 0.2	76.1 ± 3.5
16	79.2 ± 0.3	29.8 ± 11.1	80.0 ± 0.2	60.1 ± 7.8	80.1 ± 0.3	60.1 ± 7.8	79.5 ± 0.4	49.9 ± 2.1

Table 4: Testing accuracy (%) comparisons on different layers and model structures w and w/o hyperbolic residual connection(HRC). $\mathcal{P}$-HGCN$_{[d\times m]}$ denotes the $\mathcal{P}$-HGCN with a product manifold of m d-dimensional Lorentz models.

Methods	PUBMED	CITESEER	CORA
w/o dropout	79.4 ± 0.1	71.9 ± 0.3	80.3 ± 0.4
DropConnect	79.7 ± 0.3	71.9 ± 0.3	83.0 ± 0.6
HyperDrop	79.9 ± 0.2	72.5 ± 0.5	82.5 ± 0.4
Both	79.9 ± 0.3	72.7 ± 0.5	83.5 ± 0.9

Table 5: Comparisons of HyperDrop and DropConnect.

Ablation Experiments

We conducted ablation experiments on the PUBMED dataset to observe the effect of our proposed hyperbolic residual connection on $\mathcal{R}$-HGCN performance in different dimension selection approaches, respectively. As can be seen from Tabel 4, without the hyperbolic residual connection, the fortunate performance of the model shows a different degree of decrease respectively. Moreover, as the number of model layers increases, the decline in model performance is more pronounced. The experimental results prove that hyperbolic residual connection has a great helpful effect on the model performance.

Performance Comparisons

The comparisons with several state-of-the-art Euclidean GCNs and HGCNs are shown in Table 3. We have three observations. First and most importantly, compared with other HGCNs that are typically shallow models, $\mathcal{R}$-HGCN shows better results on PUBMED and CITESEER. Through, on PUBMED, HGCN also achieves the best accuracy, 80.3%. Note that HGCN uses extra link prediction task as pre-training model while $\mathcal{R}$-HGCN does not use this training trick for a clear evaluation of HyperDrop; and the performance of HGCN decreases when the link prediction pre-training is not used. On CORA, LGCN achieves the highest mean accuracy 83.3% among HGCNs. Note that both HGCN and LGCN utilize DropConnect [Wan et al. 2013] technique for training. As shown in Section , $\mathcal{R}$-HGCN obtains 83.5% mean accuracy on CORA while also using DropConnect, that is 0.2% higher than that of LGCN. Second, both $\mathcal{R}$-HGCN$_{[16\times1]}$ and $\mathcal{R}$-HGCN$_{[2\times8]}$ benefit from HyperDrop on three datasets. It proves HyperDrop alleviates over-fitting issue in hyperbolic graph convolutional network and improves the generalization ability of $\mathcal{R}$-HGCN on the test set. Third, compared with Euclidean GCNs, $\mathcal{R}$-HGCN combined with HyperDrop achieves the best results on PUBMED and CITESEER. It confirms the superiority of hyperbolic representation learning while modeling graph data.

Comparisons with DropConnect

Table 5 shows the performance of HyperDrop and DropConnect[Wan et al. 2013] on PUBMED, CITESEER, and CORA using a 16-dimensional $\mathcal{R}$-HGCN. Since there is no dropout method tailed for hyperbolic representations before HyperDrop, some works [Chami et al. 2019, Zhang et al. 2021] use DropConnect as a regularization. DropConnect is one of variants of dropout methods that randomly zeros out elements of the Euclidean parameters in model, and it can be used in hyperbolic graph convolutional network as the parameters in hyperbolic graph convolutional network are Euclidean.

For DropConnect, we search the drop rate from 0.1 to 0.9 and report the best results. DropConnect obtains improvements on PUBMED and CORA but not on CITESEER. In contrast, HyperDrop achieves stable improvements on three datasets, and higher mean accuracy on PUBMED and CITESEER compared to DropConnect. HyperDrop and DropConnect are two dropout methods that the former works on hyperbolic representations and the latter works on Euclidean parameters. They can work together effectively for a better generalization of $\mathcal{R}$-HGCN. As the results on CITESEER and CORA show, using HyperDrop and DropConnect together has better performance than using only HyperDrop or DropConnect individually.

Conclusion

In this paper, we have proposed $\mathcal{R}$-HGCN, a product manifold based residual hyperbolic graph convolutional network for overcoming the over-smoothing problem. The residual connections can prevent node representations from being indistinguishable by hyperbolic residual connection and hyperbolic identity mapping. The product manifold with different origin points also provides a wider range of perspectives of data. A novel hyperbolic dropout method, HyperDrop, is proposed to alleviate the over-fitting issue while deepening models. Experiments have demonstrated the effectiveness of $\mathcal{R}$-HGCNs under various graph convolution layers and different structures of product manifolds.

Acknowledgements

This work was supported by the Natural Science Foundation of China (NSFC) under Grants No. 62172041 and No. 62176021, Natural Science Foundation of Shenzhen under Grant No. JCYJ20230807142703006, and 2023 Shenzhen National Science Foundation(No. 20231128220938001).

References

Bachmann, G.; Bécigneul, G.; and Ganea, O. 2020. Constant curvature graph convolutional networks. In *International Conference on Machine Learning (ICML)*, 486–496.

Cai, C.; and Wang, Y. 2020. A note on over-smoothing for graph neural networks. *arXiv preprint arXiv:2006.13318*.

Chami, I.; Ying, Z.; Ré, C.; and Leskovec, J. 2019. Hyperbolic graph convolutional neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 4868–4879.

Chen, M.; Wei, Z.; Huang, Z.; Ding, B.; and Li, Y. 2020. Simple and deep graph convolutional networks. In *International Conference on Machine Learning (ICML)*, 1725–1735.

Dai, J.; Wu, Y.; Gao, Z.; and Jia, Y. 2021. A hyperbolic-to-hyperbolic graph convolutional network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 154–163.

De Sa, C.; Gu, A.; Ré, C.; and Sala, F. 2018. Representation tradeoffs for hyperbolic embeddings. *Proceedings of Machine Learning Research*, 80: 4460.

Hamilton, W.; Ying, Z.; and Leskovec, J. 2017. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 1024–1034.

Khrulkov, V.; Mirvakhabova, L.; Ustinova, E.; Oseledets, I.; and Lempitsky, V. 2020. Hyperbolic image embeddings. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 6418–6428.

Kipf, T. N.; and Welling, M. 2017. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*.

Klicpera, J.; Bojchevski, A.; and Günnemann, S. 2019. Predict then propagate: Graph neural networks meet personalized pagerank. In *International Conference on Learning Representations (ICLR)*.

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

Li, Q.; Han, Z.; and Wu, X.-M. 2018. Deeper insights into graph convolutional networks for semi-supervised learning. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 3538–3545.

Liu, Q.; Nickel, M.; and Kiela, D. 2019. Hyperbolic graph neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 8230–8241.

Liu, Y.; and Lang, B. 2023. McH-HGCN: multi-curvature hyperbolic heterogeneous graph convolutional network with type triplets. *Neural Computing and Applications*, 35(20): 15033–15049.

Nickel, M.; and Kiela, D. 2017. Poincaré embeddings for learning hierarchical representations. In *Advances in Neural Information Processing Systems (NeurIPS)*, 6338–6347.

Nickel, M.; and Kiela, D. 2018. Learning continuous hierarchies in the lorentz model of hyperbolic geometry. In *International Conference on Machine Learning (ICML)*, 3776–3785.

Papadopoulos, F.; Kitsak, M.; Serrano, M. Á.; Boguná, M.; and Krioukov, D. 2012. Popularity versus similarity in growing networks. *Nature*, 489(7417): 537–540.

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

Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; and Salakhutdinov, R. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research (JMLR)*, 15(1): 1929–1958.

Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Lio, P.; and Bengio, Y. 2017. Graph attention networks. In *International Conference on Learning Representations (ICLR)*.

Wan, L.; Zeiler, M.; Zhang, S.; Le Cun, Y.; and Fergus, R. 2013. Regularization of neural networks using dropconnect. In *International Conference on Machine Learning (ICML)*, 1058–1066.

Wu, F.; Zhang, T.; Souza Jr, A. H. d.; Fifty, C.; Yu, T.; and Weinberger, K. Q. 2019. Simplifying graph convolutional networks. In *International Conference on Machine Learning (ICML)*, 6861–6871.

Yao, S.; Pi, D.; and Chen, J. 2022. Knowledge embedding via hyperbolic skipped graph convolutional networks. *Neurocomputing*, 480: 119–130.

Zhang, Y.; Wang, X.; Shi, C.; Liu, N.; and Song, G. 2021. Lorentzian graph convolutional networks. In *Proceedings of the Web Conference (WWW)*, 1249–1261.