Hyperbolic Kernel Convolution: A Generic Framework

Eric Qu[*]
University of California, Berkeley
ericqu@berkeley.edu

Lige Zhang[*]
Duke Kunshan University
lz245@duke.edu

Habib Debaya
Duke Kunshan University
md433@duke.edu

Yue Wu
Duke Kunshan University
yw501@duke.edu

Dongmian Zou[†]
Duke Kunshan University
dongmian.zou@duke.edu

Abstract

The past sexennium has witnessed rapid advancements of hyperbolic neural networks. However, it is challenging to learn good hyperbolic representations since common Euclidean neural operations, such as convolution, do not extend to the hyperbolic space. Most hyperbolic neural networks omit the convolution operation and cannot effectively extract local patterns. Others either only use non-hyperbolic convolution, or miss essential properties such as equivariance to permutation. We propose HKConv, a novel trainable hyperbolic convolution which first correlates trainable local hyperbolic features with fixed kernel points placed in the hyperbolic space, then aggregates the output features within a local neighborhood. HKConv is a generic framework where any coordinate model of the hyperbolic space can be flexibly used. We show that neural networks with HKConv layers advance state-of-the-art in various tasks. The code of our implementation is available at `https://github.com/BruceZhangReve/Hyperbolic-Kernel-Convolution`

1 Introduction

In geometric deep learning, hyperbolic neural networks (HNNs) have achieved remarkable successes thanks to the fact that hyperbolic spaces have large capacity and facilitate embedding of hierarchical structures [1]. HNNs have found wide applications such as word embedding [2], knowledge graph completion [3], and image segmentation [4].

The core neural operation in HNNs is the linear layers, known as fully-connected layers, used in the seminal work of Ganea [5] and later works [6–8]. However, linear layers cannot embed strong inductive bias such as local patterns. To better capture local patterns, convolution operations are often used in the Euclidean domain, which played a crucial role in the early success of deep learning [9]. Convolutional layers extract local patterns from the input by measuring and aggregating correlations between the filter and different regions of the input. Unfortunately, the convolution operation is largely overlooked in HNNs. Some HNNs [4, 10–12] utilize standard convolutional layers, which operate in the Euclidean space rather than the hyperbolic space. The only existing work that explicitly defines a hyperbolic convolution is HNN++ [7], where convolution is done by first concatenating features in the reception field and then applying a hyperbolic fully-connected layer. Although the convolution in HNN++ effectively incorporates neighborhood information, it could only apply to ordered and structured data (e.g., sequences or images). This is because it is not equivariant to permutation of the order of input features. Simply switching the order of concatenation will completely change the output. Moreover, it only applies to the case where each feature has the same number of neighbors and does not work for heterogeneous relations between data points. Therefore, the convolution in HNN++ does not generalize to, e.g., graph data. For graph data, many works [6, 13–16] have

[*]Equal contribution.
[†]Correspondence to: Dongmian Zou.

generalized graph convolutional network (GCN) to the hyperbolic space. The main idea is to use message passing to extract local patterns defined by a graph. However, graph convolution relies only on the topology of the underlying graph, neglecting the rich geometric structures in the hyperbolic space itself.

One difficulty in designing hyperbolic convolution stems from the fact that the principles used in designing CNN convolutions do not generalize to the hyperbolic space. First, the hyperbolic space is typically used as the embedding space of features, not a domain where the features are defined as functions. Second, the hyperbolic space lacks the structure of a regular grid and cannot facilitate alignment of convolutional kernels and input features. To address the first issue, the convolutional filter has to also be defined in the same hyperbolic space; to address the second issue, the convolutional filter should not be required to swipe over the space. In view of these requirements, we draw inspiration from point cloud analysis [17–19] and introduce fixed point kernels in hyperbolic space to develop a hyperbolic convolutional layer, which we call the **H**yperbolic **K**ernel **Conv**olution (HKConv). HKConv draws learnable transformations from relative positions of input features, whose correlation with the point kernels are then processed to produce the output features. HKConv relies on the intrinsic geometry of the hyperbolic space and can be defined for any coordinate model of the hyperbolic space. We outline our contributions as follows.

- We formulate HKConv, a hyperbolic convolution method based on point kernels. To the best of our knowledge, no previous work has constructed hyperbolic neural layers that focus on extracting local hyperbolic patterns.
- HKConv is a generic framework that operates under any model of the hyperbolic space. We provide concrete implementation for both the Lorentz model and the Poincaré ball model.
- HKConv is equivariant to permutation of ordering of the input hyperbolic features. It is also invariant to parallel translation of a local neighborhood. Moreover, its expressivity is invariant with respect to isometric transformation of the point kernels.
- We demonstrate the effectiveness of HKConv by showing its competitive performance in hyperbolic neural networks.

2 Related Works

Hyperbolic Neural Networks. The earliest hyperbolic network was HNN [5], where linear layers and recurrent layers were defined according to the Poincaré ball model. HNN was recently generalized to HNN++ [7], which introduced concatenation, convolution, and attention mechanisms in the Poincaré ball. Nickel and Kiela [20] pointed out the numerical instability of operations in the Poincaré ball and promoted the use of the Lorentz model. More recently, Chen et al. [8] proposed to perform operations directly in the hyperbolic space without relying on the tangent spaces and thus derived the fully hyperbolic network. Fan et al. [21] took a similar approach, but constrained the maps to be Lorentz transformations. It is also possible to use hybrid models. For instance, Gulcehre et al. [22] established a hyperbolic attention using both the Lorentz and the Klein models. For a more complete list of various hyperbolic networks with their underlying models, we refer to the recent survey by Peng et al. [23].

Non-Euclidean Convolution. Recently, CNNs have been generalized to different manifolds. Geometric convolutional models include the spherical CNN [24], the mesh CNN [25], and the icosahedral CNN [26]. In these works, the input features are defined as a function on the space, not embedded in the space, and thus their methods do not generalize to hyperbolic models as considered in our work.

Convolution has also been generalized to graph data, with a focus on effective processing of graph topology. Specifically, GCN convolution is implemented by local message passing and aggregation [27]. Due to the tree-like structure and power law distribution in many graph data, it is natural to use the hyperbolic space to represent geometric information missing from graph topology [28]. Many hyperbolic networks [6, 13–16] generalized GCN convolution. In these works, the input features are transformed directly by various hyperbolic fully-connected layers and the aggregation inherits from the graph edge weights. Unlike previous approaches, HKConv transforms relative features in each neighborhood and therefore encodes more local geometric information when applied to graph data.

We remark that some HNNs [4, 10–12] employed standard convolutional layers. We also remark that the title of a previous work [29] seems to suggest similar network construction. However, there is no direct relation between our works since in that work, “hyperbolic” describes the type of differential equation and “convolution” refers to the regular operation for images.

3 Background

3.1 Hyperbolic Geometry

Hyperbolic geometry is a non-Euclidean geometry with a constant negative curvature [30, 31]. In our paper, we use $\mathbb{H}^m$ to denote an m-dimensional hyperbolic space. In order to define operations in the hyperbolic space, it is convenient to work with an isometric Cartesian-like model (coordinate system) such as the Lorentz model $\mathbb{L}^m$, Poincaré ball $\mathbb{B}^m$, and the Klein disk model $\mathbb{K}^m$. Our HKConv is a generic framework that works with all these models. A detailed review for hyperbolic geometry is introduced in Appendix A.1.

3.2 Hyperbolic Neural Operations

Linear Layers and Activation. Linear and activation layers are basic components of a deep learning model, and the hyperbolic version of which is actually a simple mimic of the Euclidean linear and activation layers.

Different linear transformation designs have been introduced to hyperbolic neural networks. A widely used method is to use exponential and logarithmic maps to take linear transformations back to the tangent space of the origin [6]. An advantage of such method is that it is compatible with every hyperbolic model. $\mathrm{HLinear}(\mathbf{x})$ can be summarized as the following:

$$\begin{aligned} \mathbf{x}' &= \exp_{\mathbf{o}}^{\mathcal{H}}(\mathbf{W}\log_o^{\mathcal{H}}(\mathbf{x})); \\ \mathrm{HLinear}(x) &= \exp_{\mathbf{x}'}^{\mathcal{H}}(\mathrm{PT}_{\mathbf{o}\to\mathbf{x}'}\log_o^{\mathcal{H}}(\mathbf{b})), \end{aligned} \tag{1}$$

where b is a hyperbolic bias vector and W is an arbitrary matrix.

For Poincaré, the above operations can be represented via Mobiüs operations:

$$\mathrm{HLinear}(\mathbf{x}) = (\mathbf{W}\otimes\mathbf{x})\oplus\exp_o^{\mathcal{B}}(\mathbf{b}), \tag{2}$$

where b is also a hyperbolic bias vector and c is the curvature. Following such regime, HAct can be easily defined:

$$\mathrm{HAct}(\mathbf{x}) = \exp_o^{\mathcal{H}}(\sigma\log_o^{\mathcal{H}}(\mathbf{x})), \tag{3}$$

where σ is the activation function.

However, extensive use of exponential or logarithmic operations may lead to a numerical instability [6] [8]. Therefore, by making use of $\langle\mathbf{x},\mathbf{y}\rangle_{\mathbf{L}}$, the linear transformation on the Lorentz model can be designed in a "fully hyperbolic" approach:

$$\mathrm{HLinear}(\mathbf{x}) = \begin{bmatrix} \sqrt{\|\mathbf{W}\mathbf{x}\|^2+1} \\ \mathbf{W}\mathbf{x} \end{bmatrix}, \text{where } \mathbf{x}\in\mathbb{L}^m, \mathbf{W}\in\mathbb{R}^{n\times(m+1)}, \text{and } \mathrm{HLinear}(\mathbf{x})\in\mathbb{L}^n.$$

Another advantage of this method is that $\mathrm{HLinear}$ can be directly equipped with additional activation, bias and normalization, which leads to a general expressive transformation summarized as the following:

$$\mathrm{HLinear}(\mathbf{x}) = \begin{bmatrix} \sqrt{\|\mathbf{h}(\mathbf{x})\|^2+1} \\ \mathbf{h}(\mathbf{x}) \end{bmatrix} \in L^n, \text{ where } \mathbf{h}(\mathbf{x}) = \frac{\lambda\sigma(\mathbf{v}^\top\mathbf{x}+b')}{\|\mathbf{W}\tau(\mathbf{x})+\mathbf{b}\|}(\mathbf{W}\tau(\mathbf{x})+\mathbf{b}). \tag{4}$$

Specifically, $\mathbf{v}\in\mathbb{R}^{n+1}$ and $\mathbf{W}\in\mathbb{R}^{n\times(m+1)}$ are trainable weights, $\mathbf{b}\in\mathbb{R}^n$ and $b'\in\mathbb{R}$ are trainable biases, σ is the sigmoid function, τ is the activation function, and the trainable parameter $\lambda>0$ scales the range. We may also write $h(\mathbf{x};\mathbf{W},\mathbf{b})$ to emphasize its dependence on $\mathbf{W}$ and $\mathbf{b}$.

Additional Operations. Neural network operations in the hyperbolic space also involve other basic components: embedding layers, centralization layers, and output layers. An embedding layer transforms Euclidean feature vectors to hyperbolic vectors, whereas an output layer maps final hyperbolic representations into Euclidean vectors. A centralization layer performs averaging/aggregation in the hyperbolic space. A more detailed review for those neural operations is in Appendix A.2.

4 Methodology

4.1 Hyperbolic Kernel Points

To construct an HKConv layer, the first step is to fix K kernel points $\{\tilde{\mathbf{x}}_k\}_{k=1}^K$ in $\mathbb{H}^m$, where m is the dimension of the input features. The kernel points can be regarded as located in the neighborhood of the hyperbolic origin $\mathbf{o} \in \mathbb{H}^m$, taken to be $\mathbf{o} = [1, \mathbf{0}_m^\top]^\top \in \mathbb{L}^m$ in the Lorentz model or $\mathbf{o} = \mathbf{0}_m \in \mathbb{B}^m$ in the Poincaré model, where $\mathbf{0}_m$ is the m-dimensional zero vector.

The locations of $\{\tilde{\mathbf{x}}_k\}_{k=1}^K$ are determined according to the principle of Thomas et al. [18]: the kernel points should be as far as possible from each other, but also not too distant from the origin. Specifically, let $d_{\mathcal{H}}$ denote the geodesic distance function. The kernel points are determined by solving an optimization problem where the following loss function is minimized:

$$\mathfrak{L}(\{\tilde{\mathbf{x}}_k\}_{k=1}^K) = \sum_{k=1}^{K} \sum_{l \neq k} \frac{1}{d_{\mathcal{H}}(\tilde{\mathbf{x}}_l, \tilde{\mathbf{x}}_k)} + \sum_{k=1}^{K} d_{\mathcal{H}}(\mathbf{o}, \tilde{\mathbf{x}}_k). \tag{5}$$

In our context, the first term in (5) guarantees expressivity as it associates distinct features with each kernel point. If the kernel points are too close to each other, the output from the kernel points will be too similar. The second term ensures the solution does not diverge to infinity. Another reason for having the second term is to prevent vanishing gradients. When the kernel points are too distant, gradients may vanish easily because of the inverse hyperbolic trigonometric function used in $d_{\mathcal{H}}$ (shown in Appendix A). We further illustrate a case of gradient vanishment in Appendix C.

The kernel points are pre-determined since it would be numerically unstable to train them in the hyperbolic space together with other parameters. We further illustrate this situation in the ablation study in Appendix E. This does not restrict the expressivity of HKConv, since learnable transformations are still applied to the input features.

In general, it is difficult to determine an analytical solution to the minimization of (5). We use the Riemannian gradient descent method [32] to approximate a proper value. Figure 1 shows examples of 5 and 10 kernel points in $\mathbb{L}^2$ and $\mathbb{B}^2$. We observe that the kernel points are around the origin; meanwhile, no pair of points are too close to each other. This agrees with our intuition in the formulation of (5).

4.2 Hyperbolic Kernel Convolution

Analogous to convolution in the Euclidean space, HKConv applies the kernel points to local neighborhoods of input hyperbolic features. It is flexible to choose the neighborhoods, such as nearest neighbors according to geodesic distances or, when the hyperbolic features are an embedding of graph features, 1-hop neighbors according to graph topology.

Let $\mathbb{X} \subset \mathbb{H}^m$ denote the collection of input hyperbolic features. For each $\mathbf{x} \in \mathbb{X}$, we use $\mathcal{N}(\mathbf{x}) \subset \mathbb{X} - \{\mathbf{x}\}$ to denote the neighbors of $\mathbf{x}$. We describe the HKConv layer in the following three steps of operations: first, transformation of relative input features; second, aggregation based on correlation with kernels; third, final aggregation for the neighborhood. We describe the details as follows.

Step 1. To ensure the locality of the extracted information, we need to use relative features between an input $\mathbf{x}$ and its neighbors. More specifically, we need to perform a hyperbolic "translation". To this end, we first transform each $\mathbf{x}_i \in \mathcal{N}(\mathbf{x})$ into the tangent space by applying the logarithmic map, then move them towards $\mathbf{o}$ by applying the parallel transport from the tangent space at $\mathbf{x}$ to the tangent space at $\mathbf{o}$ (denoted as $\mathrm{PT}_{\mathbf{x}\to\mathbf{o}}$), and lastly bring them back to $\mathbb{H}^m$ by applying the exponential map. For convenience, we extend the parallel transport $\mathrm{PT}_{\mathbf{x}\to\mathbf{y}} : \mathcal{T}_{\mathbf{x}}\mathbb{H}^m \to \mathcal{T}_{\mathbf{y}}\mathbb{H}^m$ to $\mathrm{T}_{\mathbf{x}\to\mathbf{y}} : \mathbb{H}^m \to \mathbb{H}^m$, which can be regarded as a translation operator defined on the hyperbolic space. Specifically, given $\mathbf{x}, \mathbf{y}, \mathbf{u} \in \mathbb{H}^m$,

$$\mathrm{T}_{\mathbf{x}\to\mathbf{y}}(\mathbf{u}) := \exp_{\mathbf{y}}\left(\mathrm{PT}_{\mathbf{x}\to\mathbf{y}}(\log_{\mathbf{x}}(\mathbf{u}))\right). \tag{6}$$

Also, to analogize to "translation by $\mathbf{x}$", we denote

$$\mathbf{u} \ominus \mathbf{x} := \mathrm{T}_{\mathbf{x}\to\mathbf{o}}(\mathbf{u}). \tag{7}$$

Clearly, $\mathbf{x} \ominus \mathbf{x} = \mathbf{o}$. Moreover, because logarithmic and exponential maps are distance preserving, (7) preserves the relative distance between $\mathbf{x}$ and its neighbors. That is,

$$d_{\mathcal{H}}(\mathbf{x}, \mathbf{x}_i) = d_{\mathcal{H}}(\mathbf{o}, \mathbf{x}_i \ominus \mathbf{x}), \text{ for all } \mathbf{x}_i \in \mathcal{N}(\mathbf{x}).$$

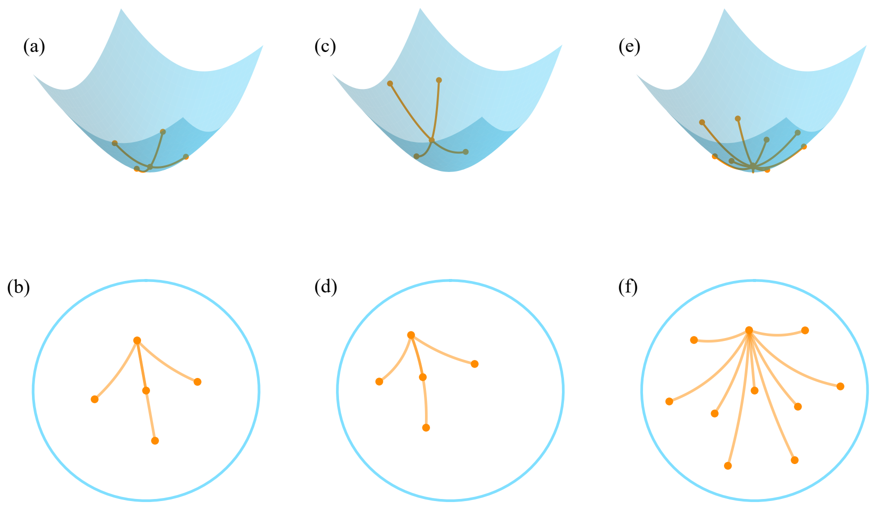


Figure 1: Illustration of kernel points in $\mathbb{L}^2$ and $\mathbb{B}^2$. Figs. (a) and (b) illustrate $K = 5$ points in the Lorentz and Poincaré models, respectively; Figs. (c) and (d) apply a parallel transport to all the points in (a) and (b); Figs. (e) and (f) show $K = 10$ points in the Lorentz and Poincaré models, respectively. In each subfigure, we show the geodesics from one starting point to all the others. To show different patterns, we take two different choices of starting points in the Lorentz and Poincaré models, respectively.

We remark that (7) is similar to "adding bias" $\oplus$ in some designs of hyperbolic linear layers [5, 6], but in general $\mathbf{u} \ominus \mathbf{x}$ need not equal $\mathbf{u} \oplus (-\mathbf{x})$.

For each $\mathbf{x}_i \in \mathcal{N}(\mathbf{x})$, the translation $\mathbf{x}_i \ominus \mathbf{x}$ gives the relative input feature of $\mathbf{x}_i$ with respect to $\mathbf{x}$. The first step of HKConv is to extract information using a learnable hyperbolic linear layer for each kernel point $\tilde{\mathbf{x}}_k$, $k = 1, \cdots, K$. We denote it by $\mathrm{HLinear}_k : \mathbb{H}^m \to \mathbb{H}^n$ and extract

$$\mathbf{x}_{ik} = \mathrm{HLinear}_k(\mathbf{x}_i \ominus \mathbf{x}), \quad k = 1, \cdots, K. \tag{8}$$

Step 2. Similar to Euclidean convolution, we take the correlation between the input features and the kernel points $\tilde{\mathbf{x}}_k$. However, unlike the Euclidean domain where the kernels are moved to the neighborhood of the input, when we perform convolution, we move $\mathcal{N}(\mathbf{x})$ to the neighborhood of $\mathbf{o}$. On one hand, this approach prevents varying degrees of distortion of the kernel points across different locations, while on the other hand, it ensures that the expressions for both the exponential and logarithmic maps remain simple, regardless of the chosen hyperbolic model. A simple way of achieving this is to move each neighbor $\mathbf{x}_i \in \mathcal{N}(\mathbf{x})$ via the translation $\mathbf{x}_i \ominus \mathbf{x}$, and then perform the correlation with the kernel point $\tilde{\mathbf{x}}_k$ by $d_{\mathcal{H}}(\mathbf{x}_i \ominus \mathbf{x}, \tilde{\mathbf{x}}_k)$ or $d_{\mathcal{H}}(\mathbf{x}_{ik}, \tilde{\mathbf{x}}_k)$. While the former is simple to implement and more flexible, the latter enjoys an expressivity property that we will illustrate in Theorem 2.

Unlike Euclidean convolution, in our design, the above correlation is used as the weights for aggregating the $\mathbf{x}_{ik}$ obtained in (8). In the hyperbolic space, one way of aggregating features is to find their weighted mean (denoted as the "MEAN" operation), which is not well-defined: one can use the hyperbolic centroid [33] or the Klein midpoint [22], both of whose output features remain in the same hyperbolic space. Finally, aggregating the features yields an output feature for each neighbor of $\mathbf{x}$:

$$\mathbf{x}'_i = \mathrm{MEAN}(\{\mathbf{x}_{ik}\}_{k=1}^K, \{d_{\mathcal{L}}(\mathbf{x}_i \ominus \mathbf{x}, \tilde{\mathbf{x}}_k)\}_{k=1}^K); \tag{9}$$

or alternatively:

$$\mathbf{x}'_i = \mathrm{MEAN}(\{\mathbf{x}_{ik}\}_{k=1}^K, \{d_{\mathcal{H}}(\mathbf{x}_{ik}, \tilde{\mathbf{x}}_k)\}_{k=1}^K). \tag{10}$$

Both (9) and (10) endow $\mathbf{x}_{ik}$ with information of local geometry in the feature space. On the contrary, other hyperbolic message passing [6, 13] or attention [22] methods only use graph structural information, but do not take into account the geometry of the feature embedding space.

Step 3. The last step in HKconv is to aggregate all the $\mathbf{x}_i'$'s from the neighborhood. Many typical graph neighborhood aggregation methods can be adapted to hyperbolic space. When there is no other information, we simply apply equal weights to all the neighbors, but it it is also possible to apply the attention regime so that the weights are learnable. That is,

$$\mathbf{x}' = \mathrm{MEAN}(\{\mathbf{x}_i'\}_{i=1}^N, \{w_i\}_{i=1}^N), \tag{11}$$

where $\{w_i\}_{i=1}^N$ is either taken to be $\mathbf{1}_N$ or learned. In addition, more sophisticated neighbor aggregation is also feasible. For instance, one can employ the aggregation introduced in [34] as follows: $\mathrm{MLP}_\Phi\left((1+\epsilon)\cdot \mathrm{MLP}_f\left(c^{(l)}(v)\right) + \sum_{u\in N(v)} \mathrm{MLP}_f\left(c^{(l)}(u)\right)\right)$, where ϵ is a trainable parameter. However, since we have performed parallel transport to translate node v and its neighborhood to the origin in Step 1, we can ignore the first term with the coefficient $(1+\epsilon)$. Therefore, a hyperbolic version of such aggregation can be reduced to the following:

$$\mathbf{x}' = \mathrm{MLP}_\Phi\left(\mathrm{MEAN}(\{\mathrm{MLP}_f(\mathbf{x}_i')\}_{i=1}^N, \{w_i\}_{i=1}^N)\right), \tag{12}$$

where MLP is the linear layer defined under its respective hyperbolic model.

Altogether, for each input feature $\mathbf{x} \in \mathbb{H}^m$, our hyperbolic convolution produces a new hyperbolic feature $\mathbf{x}' \in \mathbb{H}^n$. For convenience, we call the chain of operations an HKConv layer, and denote $\mathbf{x}' = \mathrm{HKConv}(\mathbf{x}; \mathcal{N}(\mathbf{x}))$. When $\mathcal{N}(\mathbf{x})$ is clear, we also directly denote $\mathbf{x}' = \mathrm{HKConv}(\mathbf{x})$. Figure 2 presents an illustration of HKConv.

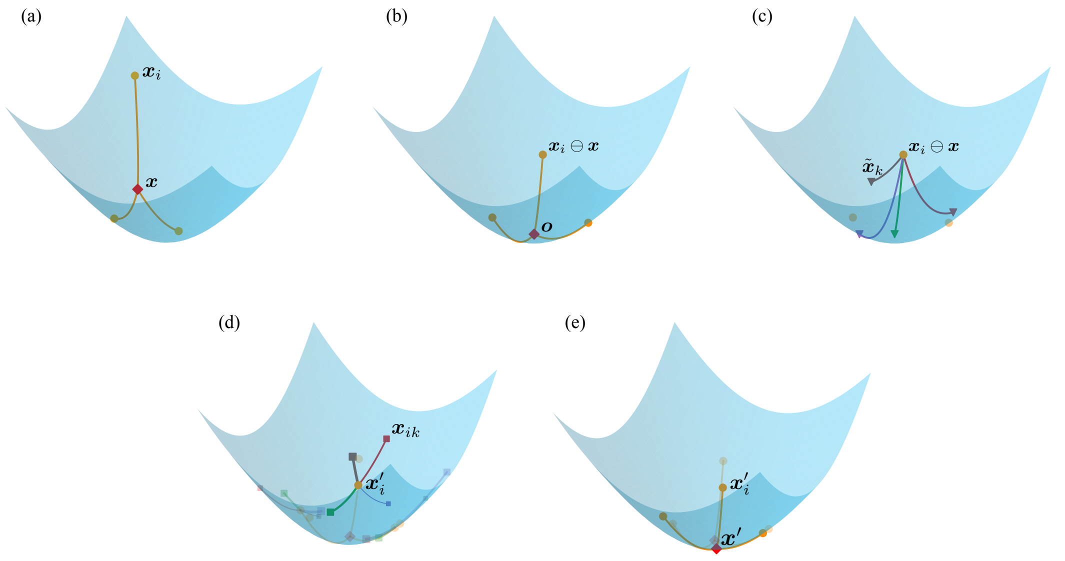


Figure 2: Illustration of HKConv. This illustration shows four kernel points and an input feature $\mathbf{x}$ with three neighbors in $\mathcal{N}(\mathbf{x})$. (a) The original positions of $\mathbf{x}$ (the diamond) and $\mathcal{N}(\mathbf{x})$ (the circle points). (b) $\{\mathbf{x}\} \cup \mathcal{N}(\mathbf{x})$ have been moved to the origin by the parallel transport. (c) The kernel points $\tilde{\mathbf{x}}_k$ (the down-triangles) and the geodesics between $\mathbf{x}_i \ominus \mathbf{x}$ and the kernel points; the correlations are the lengths of the geodesics. (d) The HLinear layer transformed $\mathbf{x}_{ik}$'s (the squares), whose weighted centroid is $\mathbf{x}_i'$; the thickness of the geodesic represents the weight. (e) $\mathbf{x}' = \mathrm{HKConv}(\mathbf{x})$, the centroid of $\mathbf{x}_i'$'s.

4.3 Properties of HKConv

Since the aggregation defined above depends solely on the neighborhood, which remains unaffected by any matrix representation, HKConv is equivariant to permutations in the order of data points. This property is not guaranteed if the convolution is done by applying hyperbolic linear layers to concatenated features, as done in HNN++, where the order of concatenation matters.

As to the hyperbolic features themselves, HKConv is also invariant under local translations in the hyperbolic space. Specifically, given an HKConv layer, if we perform a parallel translation from an input point $\mathbf{x}$ along the geodesic in the hyperbolic space, the output $\mathrm{HKConv}(\mathbf{x})$ will not change. This result holds locally, i.e., for a given $\mathbf{x}$ and its neighbors. We formulate it as follows.

Theorem 1 (Local translation invariance)**.** *Fix $\mathbf{x} \in \mathbb{X}$ and its neighborhood $\mathcal{N}(\mathbf{x}) \subset \mathbb{X}$. For any $\mathbf{y} \in \mathbb{L}^n$ in the geodesic from $\mathbf{o}$ to $\mathbf{x}$,*

$$\mathrm{HKConv}(\mathrm{T}_{\mathbf{x}\to\mathbf{y}}(\mathbf{x}); \mathrm{T}_{\mathbf{x}\to\mathbf{y}}(\mathcal{N}(\mathbf{x}))) = \mathrm{HKConv}(\mathbf{x}; \mathcal{N}(\mathbf{x})). \tag{13}$$

Theorem 1 indicates that HKConv is indeed an operation that depends on local geometry. Along the geodesics, as long as the relative positions between $\mathbf{x}$ and its neighbors stay constant, the output of HKConv will not change, regardless of the distance between $\mathbf{x}$ and the hyperbolic origin.

The next property concerns the choice of kernel points. Note that the solution to the optimization problem (5) is not unique. Essentially, the loss function $\mathfrak{L}$ in (5) is a function of distances between all pairs of $\tilde{\mathbb{X}} = \{\tilde{\mathbf{x}}_k\}_{k=1}^K \cup \{\mathbf{o}\}$, not the points themselves, and thus $\mathfrak{L}$ is invariant under isometric transformations of $\tilde{\mathbb{X}}$. It is important to guarantee that the search space of the parameters of HKConv do not depend on a specific isometric version of $\tilde{\mathbb{X}}$. In other words, the expressivity of HKConv should not be affected by the specific choice of minimizer of (5). This is achieved if we choose to use (10) in HKConv.

In our context, it is the easiest to understand isometries in the Lorentz model. Thanks to the inclusion of $\mathbf{o}$ as a fixed point, isometries reduce to those that operate for the spatial component, which can be regarded as a Euclidean domain, where isometries fixing the origin are necessarily linear. For linear layers that perform $\mathbf{W}\mathbf{x} + \mathbf{b}$ in the tangent space, it is easy to absorb such linear isometries in $\mathbf{W}$, and the expressivity remains unchanged. For the hyperbolic linear layer in [8], we formulate this result as the following theorem.

Theorem 2 (Expressivity)**.** *Let $\mathbf{W}_k \in \mathbb{R}^{n\times(m+1)}$, $\mathbf{b}_k \in \mathbb{R}^n$ be the weight matrices and bias vectors in $\mathrm{HLinear}_k$, $k = 1, \cdots, K$, of $\mathrm{HKConv}^{(0)}$, a HKConv layer with kernel points $\{\tilde{\mathbf{x}}_k\}_{k=1}^K \subset \mathbb{L}^n$. Suppose $\chi : \mathbb{L}^n \to \mathbb{L}^n$ is an isometry with $\mathbf{o}$ as a fixed point, i.e., $\chi(\mathbf{o}) = \mathbf{o}$. Let $\tilde{\mathbf{z}}_k = \chi(\tilde{\mathbf{x}}_k)$, $k = 1, \cdots, K$. Then there exist weight matrices $\mathbf{V}_k \in \mathbb{R}^{n\times(m+1)}$ and bias vectors $\mathbf{a}_k \in \mathbb{R}^n$, $k = 1, \cdots, K$, such that $\mathrm{HKConv}^{(1)}$, the HKConv layer produced from $\mathrm{HKConv}^{(0)}$ by replacing $\{\tilde{\mathbf{x}}_k\}_{k=1}^K$ with $\{\tilde{\mathbf{z}}_k\}_{k=1}^K$, $\{\mathbf{W}_k\}_{k=1}^K$ with $\{\mathbf{V}_k\}_{k=1}^K$ and $\{\mathbf{b}_k\}_{k=1}^K$ with $\{\mathbf{a}_k\}_{k=1}^K$, satisfies*

$$\mathrm{HKConv}^{(1)}(\mathbf{x}) = \mathrm{HKConv}^{(0)}(\mathbf{x}) \tag{14}$$

for any $\mathbf{x} \in \mathbb{X}$.

4.4 Connection with Traditional Convolution

For traditional convolution [35], assume $\mathbf{x}$ represents the input features on a grid. A convolution operator $\mathbf{C}(\theta)$, where θ is a filter/kernel, could be defined as $(\mathbf{C}\mathbf{x})_t = \sum_l \mathbf{x}_{t-l}\theta_l$, which is taking a weighted sum (aggregation) of the neighboring grids' input features. When it is implemented, the kernel is moved to specific region of the input to perform $\mathbf{x}_{t-l}\theta_l$. For our HKConv, step 1 (i.e., (8)) has a step of tranlation, which is similar to moving the kernel (for our implementation, we move the input instead of moving the kernel, but in the definition of convolution, it is equivalent whether which one is moved).

Step 2 of HKConv (i.e., (9) and (10)) is related to feature transformation and multi-channel kernels. Step 2 is similar to averaging hyperbolic transformations of one specific input but using all the different kernel points. Therefore, the aggregation with kernel points in step 2 can be viewed as a multi-channel variation of $\mathbf{x}_{t-l}\theta_l$. The reason for making feature transformation dependent on kernel points is that it provides richer geometric information about the local neighborhood.

Step 3 (i.e., (11) and (12)) is related to neighborhood aggregation $\sum_l$. It is also similar to the aggregation step that is commonly used in message passing graph neural networks.

4.5 Hyperbolic Kernel Networks

An Hyperbolic Kernel Network (HKN) can be constructed by cascading a number of HKConv layers. Depending on the data type, the first layer may be an embedding layer, where data points in the original domain are represented in the hyperbolic space. The output of the final HKConv layer may also need to go through further layers depending on the specific task. For instance, in graph classification where node features are embedded in the hyperbolic space, we apply a global pooling by taking a global centroid so that one single hyperbolic feature is obtained for each graph. For classification tasks, to represent the likelihood of the classes, the output needs to be either a scalar or a Euclidean vector. To this end, we can either apply the hyperbolic distance layer [13] that maps from $\mathbb{H}^n$ to $\mathbb{R}^\ell$ or simply take a logarithmic map to project the hyperbolic vector to $T_{\mathbf{o}}\mathbb{H}^n$. Details can be found in Appendix A.2.

While HKN is a generic framework within which one can choose various hyperbolic operations, we implement three variations of HKN in our experiment. Namely, we have SKN and LKN ("S" for Spatial and "L" for Lorentz), built upon the Lorentz model, and BKN built upon the Poincaré ball ("B" for Ball) model while using the Klein midpoint for aggregation. The difference between SKN and LKN is that SKN adopts a linear layer operating in the spatial component from [8], while LKN makes use of the tangent space $T_{\mathbf{o}}\mathbb{H}^n$ to perform $\mathrm{HLinear}$. In summary, we have two models: the Lorentz model and the Poincaré model; and two approaches: the tangent space approach and the spatial component approach.

5 Experiments

We evaluate the performance of HKNs (SKN, LKN, BKN) on node classification in §5.1. We also perform graph classification in Appendix E and link prediction in Appendix F. For all above tasks, each graph node has a hyperbolic representation. We present an additional experiment on a non-graph dataset in Appendix G. All experiments were executed on a GPU server with NVIDIA GeForce RTX 3090 GPUs (24GB memory).

5.1 Node Classification

When a graph has a small Gromov hyperbolicity δ, it can be well embedded in the hyperbolic space. In particular, a tree has $\delta = 0$; also, with $\kappa = -1$, $\delta = \tanh^{-1}(1/\sqrt{2}) \approx 0.88$ for $\mathbb{L}^n$ [1]. We expect hyperbolic models to work well for datasets where graphs have overall similar δ values. In node classification, the goal is to classify nodes within a graph based on the labels of some known nodes. The final output is a vector of distances for each node, representing the likelihood of all the classes.

Settings. We evaluate our HKNs on heterophilic graph datasets: the WebKB datasets [36] (Cornell, Texas, Wisconsin), with nodes as web pages and edges as hyperlinks; the Wikipedia Network datasets [37] (Chameleon, Squirrel), with nodes as Wikipedia pages linked by hyperlinks; and the Actor Co-occurrence Network [38] (Actor), where nodes are actors connected by co-appearance on the same Wikipedia page. The hyperbolicity δ is either 1 or 1.5, close to that of a hyperboloid. All models (HKNs) operate in hyperbolic space with curvature $\kappa = -1$. We use Riemannian Adam [32] as the optimizer; weight decay and dropout as regularizations; a learning rate scheduler that halves the learning rate after every determined epochs. For all models, hyper-parameters are optimized through a search on the validation set, considering the initial learning rate, weight decay, dropout, number of layers, hidden dimensions, and the number of kernel points.

Baselines. We compare HKNs with the following baseline methods: (1) multi-layer perceptron (MLP); (2) general graph neural networks including GCN [27], GAT [39], GraphSAGE [40], GCNII [41]; (3) methods for heterophilous graphs including Geom-GCN [42], WRGAT [43], LINKX [44], GloGNN and GloGNN++ [45]; (4) hyperbolic GCN methods: HGCN [6], H2H-GCN [15], HyboNet [8]. We report in Table 1 the mean F1 score (%) and standard deviation from three independent runs.

Models. Two variations of HKNs are implemented, where point-correlation stands for adopting (9) in Step 2 of HKConv, and self-correlation stands for adopting (10). While we implement all models using aggregation (11) for simplicity, we do test an additional variant, BKN-isoAgg, that implements (12) with a single layer MLP.

Results. In all the datasets, at least one HKN variant achieves statistically significant improvement over other methods. Compared with other hyperbolic methods, SKN consistently excels by a large margin (example visualizations for embeddings are also provided in Appendix D), and BKN-isoAgg (self-correlation) excels other hyperbolic methods by a large margin on four out of six benchmarks (Cornell, Texas, Wisconsin, Actor). BKN and LKN have also shown satisfying performance on

Table 1: Node classification results. Mean F1 scores (%) and standard deviations are reported. NaN indicates a numerically unstable result. The best score in each column is in **bold** font.

	Cornell	Texas	Wisconsin	Chameleon	Squirrel	Actor
# of Nodes	183	183	251	2,277	5,201	7,600
# of Edges	280	295	466	31,421	198,493	26,752
# of Features	1,703	1,703	1,703	2,325	2,089	931
# of Classes	5	5	5	5	5	5
Hyperbolicity	$\delta = 1$	$\delta = 1$	$\delta = 1$	$\delta = 1.5$	$\delta = 1.5$	$\delta = 1.5$
MLP	81.89±6.40	80.81±4.75	85.29±3.31	46.21±2.99	28.77±1.56	36.53±0.70
GCN [27]	60.54±5.30	55.14±5.16	51.76±3.06	64.82±2.24	53.43±2.01	27.32±1.10
GAT [39]	61.89±5.05	52.16±6.63	49.41±4.09	60.26±2.50	40.72±1.55	27.44±0.89
GraphSAGE [40]	75.95±5.01	82.43±6.14	81.18±5.56	58.73±1.68	41.61±0.74	34.23±0.99
GCNII [41]	77.86±3.79	77.57±3.83	80.39±3.40	63.86±3.04	38.47±1.58	37.44±1.30
Geom-GCN [42]	60.54±3.67	66.76±2.72	64.51±3.66	60.00±2.81	38.15±0.92	31.59±1.15
WRGAT [43]	81.62±3.90	83.62±5.50	86.98±3.78	65.24±0.87	48.85±0.78	36.53±0.77
LINKX [44]	77.84±5.81	74.60±8.37	75.49±5.72	68.42±1.38	61.81±1.80	36.10±1.55
GloGNN [45]	83.51±4.26	84.32±4.15	87.06±3.53	69.78±2.42	57.54±1.39	37.35±1.30
GloGNN++ [45]	85.95±5.10	84.05±4.90	88.04±3.22	71.21±1.84	57.88±1.76	37.70±1.40
HGCN [6]	79.43±0.47	70.13±0.32	83.26±0.51	NaN	62.31±0.57	36.58±0.79
H2H-GCN [15]	75.52±0.82	71.47±0.63	88.71±0.82	78.71±0.96	66.85±0.72	42.73±0.86
HyboNet [8]	77.27±0.71	72.23±0.94	86.52±0.51	74.91±0.58	69.07±0.64	45.74±0.82
WLHN [46]	77.29±4.66	75.41±5.98	78.62±3.44	-	55.76±0.92	36.42±1.42
Ours (self-correlation (10))						
SKN	87.12±2.62	**93.94**±6.56	**92.59**±1.85	82.60±1.14	**73.05**±1.80	**66.42**±1.42
LKN	75.76±3.21	92.42±3.47	78.40±1.07	77.53±2.75	57.59±7.41	50.66±4.91
BKN	78.03±3.47	77.27±3.94	82.09±2.83	68.25±3.32	63.25±1.50	65.46±2.54
BKN-isoAgg	**93.94**±1.31	85.61±6.94	90.74±4.90	72.59±2.49	66.59±0.96	65.12±0.52
Ours (point-correlation (9))						
SKN	86.36±2.62	92.42±7.31	91.36±2.14	**83.33**±0.73	72.44±1.70	66.12±1.92
LKN	76.52±3.47	93.18±3.93	77.78±3.20	77.66±2.71	64.61±1.65	44.45±3.75
BKN	63.64±2.27	63.64±2.27	61.11±5.55	56.41±10.43	53.02±1.93	45.51±3.67
BKN-isoAgg	76.52±1.31	83.33±4.73	74.07±1.85	64.90±2.21	63.78±0.56	62.21±1.60

certain tasks, respectively. To summarize, this shows the effectiveness of our general framework. We also note that self-correlation models generally outperform point-correlation models.

Analysis. Although the Lorentz model and the Poincaré model are isometric, the performance of corresponding HKNs can still be different due to numerical reasons. Comparing LKN and BKN, we notice that one model never consistently outperforms the other, despite the better numerical stability of the Lorentz model [6, 20]. We also notice that SKN consistently outperforms LKN, which is due to the expressive and numerically stable transformations from [8]. Further analysis on the number of kernel points is also discussed in Appendix D.

6 Conclusion and Limitation

We have introduced HKConv, a novel hyperbolic convolutional layer based on point kernels that extract information from local geometric relations between input features. The corresponding hyperbolic network, HKN, has achieved state-of-the-art performance in both node-level and graph-level tasks for tree-like datasets. We have particularly demonstrated the advantage of the important components in our convolution.

One possible limitation of this work is that the number of parameters in HKConv grows with the number of kernels, which may lead to overfitting for small datasets when the number of kernels is large. Additionally, it imposes a challenge for CUDA memory when performing on large graphs for node classification tasks. To address this issue, a future working direction is to explore sampling regimes when building the kernels for reducing model complexity.

Author Contributions

Eric Qu: Conceptualization; Methodology; Investigation (Lorentz model); Software; Writing - Original Draft. **Lige Zhang**: Methodology; Investigation (Poincaré model); Software; Writing - Original Draft **Habib Debaya**: Investigation (hyperboloid); Software; Writing - Review & Editing. **Yue Wu**: Investigation (hyperboloid). **Dongmian Zou**: Conceptualization; Methodology; Investigation; Formal analysis; Writing - Original Draft; Supervision.

Acknowledgements

This work is partially supported by National Natural Science Foundation of China (Grant No. 12301117) and the Summer Research Scholars (SRS) Program at Duke Kunshan University.

References

[1] Rishi Sonthalia and Anna Gilbert. Tree! i am no tree! i am a low dimensional hyperbolic embedding. *Advances in Neural Information Processing Systems*, 33:845–856, 2020. 1, 8

[2] Yoshihiro Nagano, Shoichiro Yamaguchi, Yasuhiro Fujita, and Masanori Koyama. A wrapped normal distribution on hyperbolic space for gradient-based learning. In *International Conference on Machine Learning*, pages 4693–4702. PMLR, 2019. 1, 15, 22

[3] Ines Chami, Adva Wolf, Da-Cheng Juan, Frederic Sala, Sujith Ravi, and Christopher Ré. Low-dimensional hyperbolic knowledge graph embeddings. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6901–6914, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.617. URL `https://aclanthology.org/2020.acl-main.617`. 1

[4] Mina Ghadimi Atigh, Julian Schoep, Erman Acar, Nanne van Noord, and Pascal Mettes. Hyperbolic image segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4453–4462, June 2022. 1, 2

[5] Octavian Ganea, Gary Bécigneul, and Thomas Hofmann. Hyperbolic neural networks. *Advances in neural information processing systems*, 31:5345–5355, 2018. 1, 2, 5, 14

[6] Ines Chami, Zhitao Ying, Christopher Ré, and Jure Leskovec. Hyperbolic graph convolutional neural networks. *Advances in neural information processing systems*, 32:4868–4879, 2019. 1, 2, 3, 5, 8, 9, 14, 21, 22, 23

[7] Ryohei Shimizu, YUSUKE Mukuta, and Tatsuya Harada. Hyperbolic neural networks++. In *International Conference on Learning Representations*, 2021. 1, 2, 24, 25

[8] Weize Chen, Xu Han, Yankai Lin, Hexu Zhao, Zhiyuan Liu, Peng Li, Maosong Sun, and Jie Zhou. Fully hyperbolic neural networks. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5672–5686, 2022. 1, 2, 3, 7, 8, 9, 14, 21, 22, 23, 24, 25

[9] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015. 1

[10] Valentin Khrulkov, Leyla Mirvakhabova, Evgeniya Ustinova, Ivan Oseledets, and Victor Lempitsky. Hyperbolic image embeddings. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6418–6428, 2020. 1, 2

[11] Ola Ahmad and Freddy Lecue. FisheyeHDK: Hyperbolic deformable kernel learning for ultra-wide field-of-view image recognition. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(6):5968–5975, Jun. 2022. doi: 10.1609/aaai.v36i6.20542. URL `https://ojs.aaai.org/index.php/AAAI/article/view/20542`.

[12] Ahmad Bdeir, Kristian Schwethelm, and Niels Landwehr. Fully hyperbolic convolutional neural networks for computer vision. *arXiv preprint arXiv:2303.15919*, 2023. 1, 2

[13] Qi Liu, Maximilian Nickel, and Douwe Kiela. Hyperbolic graph neural networks. *Advances in Neural Information Processing Systems*, 32:8230–8241, 2019. 1, 2, 5, 8

[14] Gregor Bachmann, Gary Bécigneul, and Octavian Ganea. Constant curvature graph convolutional networks. In *International Conference on Machine Learning*, pages 486–496. PMLR, 2020.

[15] Jindou Dai, Yuwei Wu, Zhi Gao, and Yunde Jia. A hyperbolic-to-hyperbolic graph convolutional network. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 154–163, 2021. 8, 9, 21, 23

[16] Yiding Zhang, Xiao Wang, Chuan Shi, Nian Liu, and Guojie Song. Lorentzian graph convolutional networks. In *Proceedings of the Web Conference 2021*, pages 1249–1261, 2021. 1, 2, 23

[17] Matan Atzmon, Haggai Maron, and Yaron Lipman. Point convolutional neural networks by extension operators. *ACM Trans. Graph.*, 37(4), jul 2018. ISSN 0730-0301. doi: 10.1145/3197517.3201301. URL `https://doi.org/10.1145/3197517.3201301`. 2

[18] Hugues Thomas, Charles R Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, François Goulette, and Leonidas J Guibas. Kpconv: Flexible and deformable convolution for point clouds. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 6411–6420, 2019. 4

[19] Zhi-Hao Lin, Sheng-Yu Huang, and Yu-Chiang Frank Wang. Convolution in the cloud: Learning deformable kernels in 3d graph convolution networks for point cloud analysis. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1800–1809, 2020. 2

[20] Maximillian Nickel and Douwe Kiela. Learning continuous hierarchies in the lorentz model of hyperbolic geometry. In *International Conference on Machine Learning*, pages 3779–3788. PMLR, 2018. 2, 9

[21] Xiran Fan, Chun-Hao Yang, and Baba C Vemuri. Nested hyperbolic spaces for dimensionality reduction and hyperbolic nn design. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 356–365, 2022. 2, 23

[22] Caglar Gulcehre, Misha Denil, Mateusz Malinowski, Ali Razavi, Razvan Pascanu, Karl Moritz Hermann, Peter Battaglia, Victor Bapst, David Raposo, Adam Santoro, and Nando de Freitas. Hyperbolic attention networks. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=rJxHsjRqFQ`. 2, 5, 24, 25

[23] W. Peng, T. Varanka, A. Mostafa, H. Shi, and G. Zhao. Hyperbolic deep neural networks: A survey. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, December 2021. doi: 10.1109/TPAMI.2021.3136921. 2

[24] Taco S. Cohen, Mario Geiger, Jonas Köhler, and Max Welling. Spherical CNNs. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=Hkbd5xZRb`. 2

[25] Rana Hanocka, Amir Hertz, Noa Fish, Raja Giryes, Shachar Fleishman, and Daniel Cohen-Or. Meshcnn: a network with an edge. *ACM Transactions on Graphics (TOG)*, 38(4):1–12, 2019. 2

[26] Taco Cohen, Maurice Weiler, Berkay Kicanaoglu, and Max Welling. Gauge equivariant convolutional networks and the icosahedral cnn. In *International conference on Machine learning*, pages 1321–1330. PMLR, 2019. 2

[27] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017. 2, 8, 9, 21, 22, 23

[28] Menglin Yang, Min Zhou, Zhihao Li, Jiahong Liu, Lujia Pan, Hui Xiong, and Irwin King. Hyperbolic graph neural networks: A review of methods and applications, 2022. 2

[29] Keegan Lensink, Bas Peters, and Eldad Haber. Fully hyperbolic convolutional neural networks. *Research in the Mathematical Sciences*, 9(4):60, 2022. 2

[30] James W Cannon, William J Floyd, Richard Kenyon, Walter R Parry, et al. Hyperbolic geometry. *Flavors of geometry*, 31(59-115):2, 1997. 3

[31] James W Anderson. *Hyperbolic geometry*. Springer Science & Business Media, 2006. 3

[32] Gary Becigneul and Octavian-Eugen Ganea. Riemannian adaptive optimization methods. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=r1eiqi09K7`. 4, 8, 21

[33] Marc Law, Renjie Liao, Jake Snell, and Richard Zemel. Lorentzian distance learning for hyperbolic representations. In *International Conference on Machine Learning*, pages 3672–3681. PMLR, 2019. 5, 15, 16

[34] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=ryGs6iA5Km. 6, 21, 22

[35] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995. 7

[36] Mark Craven, Dan DiPasquo, Dayne Freitag, Andrew McCallum, Tom Mitchell, Kamal Nigam, and Seán Slattery. Learning to construct knowledge bases from the world wide web. *Artificial intelligence*, 118(1-2):69–113, 2000. 8

[37] Benedek Rozemberczki, Carl Allen, and Rik Sarkar. Multi-scale attributed node embedding. *Journal of Complex Networks*, 9(2):cnab014, 2021. 8, 18

[38] Jie Tang, Jimeng Sun, Chi Wang, and Zi Yang. Social influence analysis in large-scale networks. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 807–816, 2009. 8, 18

[39] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=rJXMpikCZ. 8, 9, 23

[40] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017. 8, 9, 23

[41] Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. In *International Conference on Machine Learning*, pages 1725–1735. PMLR, 2020. 8, 9

[42] Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. Geom-gcn: Geometric graph convolutional networks. In *International Conference on Learning Representations*, 2019. 8, 9

[43] Susheel Suresh, Vinith Budde, Jennifer Neville, Pan Li, and Jianzhu Ma. Breaking the limit of graph neural networks by improving the assortativity of graphs with local mixing patterns. In *KDD*, 2021. 8, 9

[44] Derek Lim, Felix Hohne, Xiuyu Li, Sijia Linda Huang, Vaishnavi Gupta, Omkar Bhalerao, and Ser Nam Lim. Large scale learning on non-homophilous graphs: New benchmarks and strong simple methods. *Advances in Neural Information Processing Systems*, 34:20887–20902, 2021. 8, 9

[45] Xiang Li, Renyu Zhu, Yao Cheng, Caihua Shan, Siqiang Luo, Dongsheng Li, and Weining Qian. Finding global homophily in graph neural networks when meeting heterophily. In *International Conference on Machine Learning*, pages 13242–13256. PMLR, 2022. 8, 9

[46] Giannis Nikolentzos, Michail Chatzianastasis, and Michalis Vazirgiannis. Weisfeiler and leman go hyperbolic: Learning distance preserving node representations. In *International Conference on Artificial Intelligence and Statistics*, pages 1037–1054. PMLR, 2023. 9, 21, 22

[47] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008. 21

[48] Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018. 21

[49] Christoph Helma, Ross D. King, Stefan Kramer, and Ashwin Srinivasan. The predictive toxicology challenge 2000–2001. *Bioinformatics*, 17(1):107–108, 2001. 21

[50] Ida Schomburg, Antje Chang, Christian Ebeling, Marion Gremse, Christian Heldt, Gregor Huhn, and Dietmar Schomburg. Brenda, the enzyme database: updates and major new developments. *Nucleic acids research*, 32(suppl_1):D431–D433, 2004. 21

[51] Karsten M Borgwardt, Cheng Soon Ong, Stefan Schönauer, SVN Vishwanathan, Alex J Smola, and Hans-Peter Kriegel. Protein function prediction via graph kernels. *Bioinformatics*, 21 (suppl_1):i47–i56, 2005. 21

[52] Pinar Yanardag and SVN Vishwanathan. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1365–1374, 2015. 21

[53] Jinheon Baek, Minki Kang, and Sung Ju Hwang. Accurate learning of graph representations with graph multiset pooling. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=JHcqXGaqiGn. 21

[54] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *arXiv preprint arXiv:2005.00687*, 2020. 22

[55] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008. 23

[56] Richard C Wilson, Edwin R Hancock, Elżbieta Pekalska, and Robert PW Duin. Spherical and hyperbolic embeddings of data. *IEEE transactions on pattern analysis and machine intelligence*, 36(11):2255–2269, 2014. 23

[57] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. Convolutional sequence to sequence learning. In *International conference on machine learning*, pages 1243–1252. PMLR, 2017. 24, 25

[58] Myle Ott, Sergey Edunov, David Grangier, and Michael Auli. Scaling neural machine translation. In *Proceedings of the Third Conference on Machine Translation, Volume 1: Research Papers*, pages 1–9, Belgium, Brussels, October 2018. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/W18-6301. 24, 25

[59] Yinpeng Chen, Xiyang Dai, Mengchen Liu, Dongdong Chen, Lu Yuan, and Zicheng Liu. Dynamic convolution: Attention over convolution kernels. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11030–11039, 2020. 24, 25

[60] Haotong Qin, Yifu Ding, Mingyuan Zhang, Qinghua YAN, Aishan Liu, Qingqing Dang, Ziwei Liu, and Xianglong Liu. BiBERT: Accurate fully binarized BERT. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=5xEgrl_5FAJ. 24, 25

Appendix

The Appendix contain the following contents. §A reviews basic hyperbolic geometry and common hyperbolic layers used in the paper, §B proves Theorems in the main text, §C conducts experiments for kernel point generation, §D provides further analysis and visualizations for previous node classification results, §E conducts experiments for graph classification tasks, §F conducts experiments for link prediction tasks, §G conducts further experiments on non-graph tasks.

A Preliminaries

A.1 Hyperbolic Geometry

Definitions. A hyperbolic space $\mathbb{H}^n$ is an n-dimensional non-Euclidean manifold that has a constant negative curvature κ. It can be explicitly constructed using a coordinate model, where each point in the hyperbolic space is represented in a Euclidean fashion. In this paper we consider $\kappa = -1$, which is a common choice in literature. For clarity, this assumption will be maintained throughout all the formulas presented in this section. For the case of general curvature, we refer the reader to [5, 6, 8].

The Lorentz model $\mathbb{L}^n = (\mathcal{L}, \mathfrak{g}^{\mathcal{L}})$, known as the hyperboloid, is an n-dimensional manifold $\mathcal{L}$ embedded in the Minkowski space with coordinates taken from $\mathbb{R}^{n+1}$. Every point in $\mathbb{L}^n$ is represented by $x = \begin{bmatrix} x_t \\ \mathbf{x_s} \end{bmatrix}$, where $x_t > 0$ is the "time component" and $\mathbf{x_s} \in \mathbb{R}^n$ is the "spatial component." When the curvature is -1, they satisfy $\langle \mathbf{x}, \mathbf{x} \rangle_{\mathcal{L}} = -1$, where $\langle \cdot, \cdot \rangle_{\mathcal{L}}$ is the Lorentz inner product induced by the metric tensor $\mathfrak{g}^{\mathcal{L}} = \mathrm{diag}([-1, 1_n^\top])$ ($\mathbf{1}_n \in \mathbb{R}^n$ is the n-dimensional vector whose entries are all 1's):

$$\langle \mathbf{x}, \mathbf{y} \rangle_{\mathcal{L}} := \mathbf{x}^\top \mathfrak{g}^{\mathcal{L}} \mathbf{y} = -x_t y_t + \mathbf{x_s}^\top \mathbf{y_s}, \quad \mathbf{x}, \mathbf{y} \in \mathbb{L}^n.$$

The Poincaré ball model $\mathbb{B}^n = (\mathcal{B}, \mathfrak{g}^{\mathcal{B}})$ with curvature -1 is an n-dimensional manifold $\mathcal{B}$ represented using coordinates in $\{\mathbf{x} \in \mathbb{R}^n : \|\mathbf{x}\| < 1\}$ and equipped with a Riemannian metric $\mathfrak{g}^{\mathcal{B}}_{\mathbf{x}} = \lambda_{\mathbf{x}}^2 \mathfrak{g}^{\mathcal{E}}$, where $\lambda_{\mathbf{x}} := \frac{2}{1-\|\mathbf{x}\|^2}$, namely conformal factor, and $\mathfrak{g}^{\mathcal{E}} := \mathbf{I}_n$ being the Euclidean metric tensor.

The Klein model $\mathbb{K}^n = (\mathcal{K}, \mathfrak{g}^{\mathcal{K}})$ is also an n-dimensional manifold $\mathcal{K}$ represented using coordinates in $\{\mathbf{x} \in \mathbb{R}^n : \|\mathbf{x}\| < 1\}$. All the above models are isometric under the following mappings:

$$\pi_{\mathcal{L}\to\mathcal{B}}(x_0, x_1, \ldots, x_n) = \left(\frac{x_1}{1+x_0}, \ldots, \frac{x_n}{1+x_0}\right); \; \pi_{\mathcal{K}\to\mathcal{B}}(\mathbf{x}^{\mathcal{K}}) = \frac{1}{1+\sqrt{1-\|\mathbf{x}^{\mathcal{K}}\|^2}}\mathbf{x}^{\mathcal{K}}.$$

$$\pi_{\mathcal{B}\to\mathcal{L}}(y_1, \ldots, y_n) = \left(\frac{1+\sum_i y_i^2, 2y_1, \ldots, 2y_n}{1-\sum_i y_i^2}\right); \; \pi_{\mathcal{B}\to\mathcal{K}}(\mathbf{x}^{\mathcal{B}}) = \frac{2}{1+\|\mathbf{x}^{\mathcal{B}}\|^2}\mathbf{x}^{\mathcal{B}}.$$

Geodesics. Shortest paths in the hyperbolic space are called geodesics. The (geodesic) distance between two points in hyperbolic space is the length of the geodesic between them. For Lorentz and Poincaré model, the distance between two points $\mathbf{x}$ and $\mathbf{y}$ is given by:

$$d_{\mathcal{L}}(x, y) = \mathrm{arcosh}\left(-\langle x, y \rangle_{\mathcal{L}}\right) \; ; d_{\mathcal{B}}(x, y) = \mathrm{arcosh}\left(1 + \frac{2\|x-y\|^2}{(1-\|x\|^2)(1-\|y\|^2)}\right).$$

Tangent Space. The tangent space at $\mathbf{x} \in \mathbb{H}^n$, denoted as $\mathcal{T}_{\mathbf{x}}\mathbb{H}^n$, is the first order approximation of the manifold around $\mathbf{x}$. The canonical representation of the tangent space is given by $\mathcal{T}_{\mathbf{x}}\mathbb{L}^n = \{\mathbf{y} \in \mathbb{R}^{n+1} : \langle \mathbf{y}, \mathbf{x} \rangle_{\mathcal{L}} = 0\}$, and $\mathcal{T}_{\mathbf{x}}\mathbb{B}^n = \mathcal{T}_{\mathbf{x}}\mathbb{K}^n = \mathbb{R}^n$.

Exponential and Logarithmic Maps. For $\mathbf{x} \in \mathbb{H}^n$, the exponential map $\exp_{\mathbf{x}} : \mathcal{T}_{\mathbf{x}}\mathbb{H}^n \to \mathbb{H}^n$ projects a tangent vector into the hyperbolic space. Let γ be the geodesic satisfying $\gamma(0) = \mathbf{x}$ and $\gamma'(0) = \mathbf{v}$. Then $\exp_{\mathbf{x}}(\mathbf{v}) := \gamma(1)$.

For the Lorentz: $\exp_{\mathbf{x}}(\mathbf{v}) = \cosh(\|\mathbf{v}\|_{\mathcal{L}})\mathbf{x} + \|\mathbf{v}\|_{\mathcal{L}}^{-1}\sinh(\|\mathbf{v}\|_{\mathcal{L}})\mathbf{v}$, where $\|\cdot\|_{\mathcal{L}} := \sqrt{\langle \cdot, \cdot \rangle_{\mathcal{L}}}$;

For the Poincaré ball: $\exp_{\mathbf{x}}(\mathbf{v}) = \mathbf{x} \oplus \left(\tanh\left(\frac{\lambda_{\mathbf{x}}\|\mathbf{v}\|}{2}\right)\frac{\mathbf{v}}{\|\mathbf{v}\|}\right)$, where $\oplus$ is the Möbius addition.

In the above, Möbius operations are used for clarity. For Poincaré ball, given $\mathbf{x}$ and $\mathbf{y}$ in $\mathbb{B}^n$, the Möbius addition is expressed as:

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

Suppose a matrix $\mathbf{M} \in \mathbb{R}^{m \times n}$. Given $\mathbf{x} \in \mathbb{B}^n$, if $\mathbf{M}\mathbf{x} \neq \mathbf{0}$, the Möbius matrix-vector multiplication is expressed as:

$$\mathbf{M} \otimes \mathbf{x} = \tanh\left(\frac{\|\mathbf{M}\mathbf{x}\|}{\|\mathbf{x}\|} \tanh^{-1}(\|\mathbf{x}\|)\right) \frac{\mathbf{M}\mathbf{x}}{\|\mathbf{M}\mathbf{x}\|}.$$

The logarithmic map $\log_{\mathbf{x}} : \mathbb{H}^n \to \mathcal{T}_{\mathbf{x}}\mathbb{H}^n$ projects hyperbolic vectors into the tangent space. It is the inverse map of $\exp_{\mathbf{x}}$ in the sense that $\log_{\mathbf{x}}(\exp_{\mathbf{x}}(\mathbf{v})) = \mathbf{v}$ for any $\mathbf{v} \in \mathbb{H}^n$, and vice versa.

$$\text{For the Lorentz: } \log_{\mathbf{x}}(\mathbf{u}) = d_{\mathcal{L}}(\mathbf{x}, \mathbf{u}) \frac{\mathbf{u} + \langle \mathbf{x}, \mathbf{u} \rangle_{\mathcal{L}} \mathbf{x}}{\|\mathbf{u} + \langle \mathbf{x}, \mathbf{u} \rangle_{\mathcal{L}} \mathbf{x}\|_{\mathcal{L}}};$$

$$\text{For the Poincaré ball: } \log_{\mathbf{x}}(\mathbf{u}) = \frac{2}{\lambda_{\mathbf{x}}} \operatorname{artanh}\left(\| - \mathbf{x} \oplus \mathbf{u}\|\right) \frac{-\mathbf{x} \oplus \mathbf{u}}{\| - \mathbf{x} \oplus \mathbf{u}\|}.$$

Parallel Transport. For two points $\mathbf{x}, \mathbf{y} \in \mathbb{H}^n$, the parallel transport, also called the parallel translation, from $\mathbf{x}$ to $\mathbf{y}$, $\mathrm{PT}_{\mathbf{x} \to \mathbf{y}}$, is a map that "transports" or "translates" a tangent vector from $T_{\mathbf{x}}\mathbb{H}^n$ to $T_{\mathbf{y}}\mathbb{H}^n$ along the geodesic from $\mathbf{x}$ to $\mathbf{y}$.

For the Lorentz model:

$$\mathrm{PT}_{\mathbf{x} \to \mathbf{y}}(\mathbf{v}) = \frac{\langle \mathbf{y}, \mathbf{v} \rangle_{\mathcal{L}}}{-1/\kappa - \langle \mathbf{x}, \mathbf{y} \rangle_{\mathcal{L}}} (\mathbf{x} + \mathbf{y});$$

For the Poincaré ball model, in this paper we only need the parallel transport from $\mathbf{x}$ to $\mathbf{o}$, which enjoys a simple form:

$$\mathrm{PT}_{x \to \mathbf{o}}(\mathbf{v}) = \log_{\mathbf{o}}\left(\mathbf{o} \oplus \exp_{\mathbf{x}}(\mathbf{v})\right) = \frac{\lambda_{\mathbf{x}}}{\lambda_{\mathbf{o}}} \mathbf{v}.$$

A.2 Hyperbolic Neural Operations

Wrapped Normal Distribution. The wrapped normal distribution [2] is a hyperbolic distribution that resembles the normal distribution in Euclidean space. Although it was orinially designed for Lorentz model, it can be easily generalize to other hyperbolic models as well. Given $\boldsymbol{\mu} \in \mathbb{H}^n$ and $\boldsymbol{\Sigma} \in \mathbb{R}^{n \times n}$, to sample $\mathbf{z} \in \mathbb{H}^n$ from the wrapped normal distribution $\mathcal{G}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, one first samples a vector from the Euclidean normal distribution $\mathcal{N}(0, \boldsymbol{\Sigma})$ and applies a hyperbolic embedding to obtain $\mathbf{x} \in L^n$. Then, one applies parallel transport and produces $\mathbf{z} = \exp_{\boldsymbol{\mu}}(\mathrm{PT}_{\mathbf{o} \to \boldsymbol{\mu}}(\log_{\mathbf{o}}(\mathbf{x}))) \in \mathbb{H}^n$. The wrapped normal distribution is mainly used for generating kernel points for proposed model.

Hyperbolic Embedding. The core idea of embedding a Euclidean vector to hyperbolic space is to treat the Euclidean vector as the image of a hyperbolic vector's projection. For Lorentz, we can view the Euclidean vector as the image of hyperbolic vector's projection onto the spacial domain from the hyperboloid [2]. Suppose $\mathbf{x} \in \mathbb{R}^n$, then a simple embedding on the hyperboloid can be $\exp_{\mathbf{o}}(\begin{bmatrix} 0 \\ \mathbf{x} \end{bmatrix}) \in \mathbb{L}^n$, where 0 can be viewed as the padding element. For Poincaré, we can view the Euclidean vector as the image of hyperbolic vector's projection onto the tangent space of the origin of Poincaré ball. A simple embedding on the Poincaré ball can be $\exp_{\mathbf{o}}(\mathbf{x}) \in \mathbb{B}^n$.

Hyperbolic Centralization. The centralization in Euclidean space often refers to the average operation, which is essentially the Euclidean centroid. We can smoothly extends the idea of centralization to hyperbolic space by making use of hyperbolic centroid. In fact, the notion of a centroid is extended to $\mathbb{L}^n$ by Law et al. [33], defined to be the point μ^* that minimizes a weighted sum of squared Lorentzian distance:

$$\boldsymbol{\mu}^*_{\mathcal{L}} = \arg\min_{\boldsymbol{\mu} \in \mathcal{L}^n} \sum_{i=1}^{N} \nu_i d^2_{\mathcal{L}}(\mathbf{x}_i, \boldsymbol{\mu}_{\mathcal{L}}),$$

where $\{\mathbf{x}_i\}_{i=1}^N$ is the set of points to aggregate, $\boldsymbol{\nu}$ is the weight vector whose entries satisfy $\nu_i \geq 0, \sum_i \nu_i > 0, i = 1, \ldots, N$. A closed form of the hyperboloid centroid is given by:

$$\mathrm{HCent}(\{\mathbf{x}_i\}_{i=1}^N, \boldsymbol{\nu}) = \boldsymbol{\mu}_{\mathcal{L}}^* = \frac{\sum_{i=1}^N \nu_i \mathbf{x}_i}{\left| \left\| \sum_{i=1}^N \nu_i \mathbf{x}_i \right\|_{\mathcal{L}} \right|}.$$

And a closed form of klein midpoint is given by:

$$\mathrm{KMid}\left(\{\mathbf{x}_i\}_{i=1}^N, \{\alpha_i\}_{i=1}^N\right) = \boldsymbol{\mu}_{\mathcal{K}}^* = \frac{\sum_{i=1}^N \alpha_i \gamma_{\mathbf{x}_i} \mathbf{x}_i}{\sum_{j=1}^N \alpha_j \gamma_{\mathbf{x}_j}}, \text{ where } \gamma_{\mathbf{x}} = \frac{1}{\sqrt{1 - \|\mathbf{x}\|^2}}.$$

Hyperbolic Output Layer. Two simple approaches can be used to output Euclidean features from Hyperbolic embeddings. An intuitive way is to directly project hyperbolic vectors onto the tangent space and use a simple Euclidean MLP to adjust output dimension,

$$\mathrm{MLP}_{\mathbb{R}^n \to \mathbb{R}^m}\{\log_o^{\mathcal{H}}(\mathbf{y})\}, \mathbf{y} \in \mathbb{H}^n. \tag{15}$$

However, a more sophisticated approach is to apply the hyperbolic distance layer [33], which is a numerically stable method of outputting Euclidean features. It maps points from $\mathbb{L}^n$ to $\mathbb{R}^m$. Given an input $\mathbf{x} \in H^n$, it first initializes m trainable centroids $\{\mathbf{c}_i\}_{i=1}^m \subset H^n$, then produces a vector of distances

$$\mathbf{y} = \mathrm{HCDist}_{n,m}(\mathbf{x}) = [d_{\mathcal{H}}(\mathbf{x}, \mathbf{c}_1) \cdots d_{\mathcal{H}}(\mathbf{x}, \mathbf{c}_m)]^\top. \tag{16}$$

B Proofs

B.1 Proof of Theorem 1

Proof. For any $\mathbf{x}_i \in \mathcal{N}(\mathbf{x})$, according to the definition in (8),

$$\begin{aligned}
\mathrm{T}_{\mathbf{x}\to\mathbf{y}}(\mathbf{x}_i) \ominus \mathrm{T}_{\mathbf{x}\to\mathbf{y}}(\mathbf{x}) &= \exp_{\mathbf{y}}\left(\mathrm{PT}_{\mathbf{x}\to\mathbf{y}}(\log_{\mathbf{x}}(\mathbf{x}_i))\right) \ominus \mathbf{y} \\
&= \mathrm{T}_{\mathbf{y}\to\mathbf{o}}\left(\exp_{\mathbf{y}}\left(\mathrm{PT}_{\mathbf{x}\to\mathbf{y}}(\log_{\mathbf{x}}(\mathbf{x}_i))\right)\right) \\
&= \exp_{\mathbf{o}}\left(\mathrm{PT}_{\mathbf{y}\to\mathbf{o}}\left(\log_{\mathbf{y}}\left(\exp_{\mathbf{y}}\left(\mathrm{PT}_{\mathbf{x}\to\mathbf{y}}(\log_{\mathbf{x}}(\mathbf{x}_i))\right)\right)\right)\right) \\
&= \exp_{\mathbf{o}}\left(\mathrm{PT}_{\mathbf{y}\to\mathbf{o}}\left(\mathrm{PT}_{\mathbf{x}\to\mathbf{y}}(\log_{\mathbf{x}}(\mathbf{x}_i))\right)\right) \\
&= \exp_{\mathbf{o}}\left(\mathrm{PT}_{\mathbf{x}\to\mathbf{o}}(\log_{\mathbf{x}}(\mathbf{x}_i))\right) \\
&= \mathbf{x}_i \ominus \mathbf{x}.
\end{aligned}$$

Consequently, the input of $\mathrm{HLinear}_k$, $k = 1, \cdots, K$, in (8) is invariant under the operator $\mathrm{T}_{\mathbf{x}\to\mathbf{y}}$. This implies that $\mathbf{x}'$ in (11) is also invariant under $\mathrm{T}_{\mathbf{x}\to\mathbf{y}}$, since it only depends on $\{\mathbf{x}_i \ominus \mathbf{x} : \mathbf{x}_i \in \mathcal{N}(\mathbf{x})\}$ and the fixed kernel $\{\tilde{\mathbf{x}}_k\}_{k=1}^K$. We have thus proven (13). □

B.2 Proof of Theorem 2

Proof. We first identify the isometry χ. Since $\mathbf{o}$ is a fixed point of χ, it is necessary that $d_{\mathcal{L}}(\mathbf{o}, \mathbf{u}) = d_{\mathcal{L}}(\mathbf{o}, \chi(\mathbf{u}))$ for any $\mathbf{u} \in \mathbb{L}^n$. This is equivalent to

$$\langle \mathbf{o}, \mathbf{u} \rangle_{\mathcal{L}} = \langle \mathbf{o}, \chi(\mathbf{u}) \rangle_{\mathcal{L}}. \tag{17}$$

Recall that $\mathbf{o} = [1, \mathbf{0}_n^\top]^\top$. Therefore, (17) simply reduces to $u_t = \chi(\mathbf{u})_t$, where $\chi(\mathbf{u})_t$ denotes the time component of $\chi(\mathbf{u})$. Therefore, χ only acts as an isometry on the spatial component of the hyperbolic space. However, since the spatial component of $\mathbb{L}^n$ can be identified as $\mathbb{R}^n$, the spatial component of χ is an Euclidean isometry which preserves $\mathbf{0}_n$. Therefore, it has to be either a rotation or reflection in $\mathbb{R}^n$. Hence, χ is linear and has a matrix representation

$$\chi(\mathbf{u}) = \mathbf{U}_{n+1}\mathbf{u} := \begin{bmatrix} 1 & \\ & \mathbf{U}_n \end{bmatrix} \mathbf{u},$$

where $\mathbf{U}_n \in \mathbb{R}^{n \times n}$ is unitary.

Next, for each $k = 1, \cdots, K$, define $\mathbf{V}_k = \mathbf{U}_{n+1}\mathbf{W}_k$ and $\mathbf{a}_k = \mathbf{U}_{n+1}\mathbf{b}_k$. Clearly,

$$h(\ \cdot\ ;\ \mathbf{V}_k, \mathbf{a}_k) = \mathbf{U}_n h(\ \cdot\ ;\ \mathbf{W}_k, \mathbf{b}_k).$$

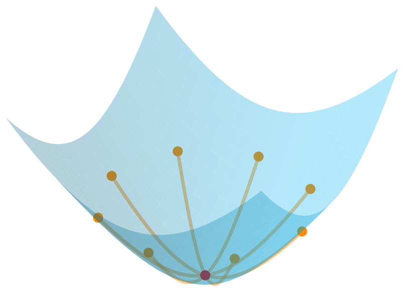
Consequently, denoting the HLinear layers of $\mathrm{HKConv}^{(\nu)}$ as $\mathrm{HLinear}^{(\nu)}$, $\nu = 0, 1$, it holds that $\mathrm{HLinear}_k^{(1)}(\cdot) = \mathbf{U}_{n+1}\mathrm{HLinear}_k^{(0)}(\cdot)$. Hence, the distance between $\mathbf{x}_{ik}$ and $\tilde{\mathbf{x}}_k$ in (9) is invariant. Specifically, if $\mathbf{x}_{ik} = \mathrm{HLinear}_k^{(0)}(\mathbf{x}_i \ominus \mathbf{x})$ and $\mathbf{z}_{ik} = \mathrm{HLinear}_k^{(1)}(\mathbf{x}_i \ominus \mathbf{x})$, then $\mathbf{z}_{ik} = \mathbf{U}_{n+1}\mathbf{x}_{ik}$ and thus

$$d_{\mathcal{L}}(\mathbf{z}_{ik}, \tilde{\mathbf{z}}_k) = d_{\mathcal{L}}(\mathbf{U}_{n+1}\mathbf{x}_{ik}, \mathbf{U}_{n+1}\tilde{\mathbf{x}}_k) = d_{\mathcal{L}}(\mathbf{x}_{ik}, \tilde{\mathbf{x}}_k).$$

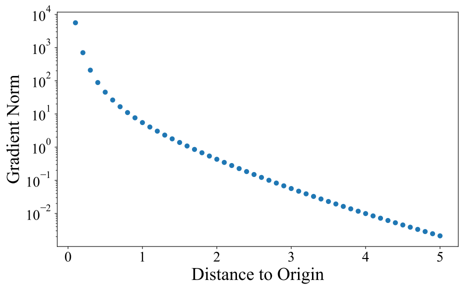
This yields (14) since HKConv depends on $\tilde{\mathbf{x}}_k$ only via $d_{\mathcal{L}}(\cdot, \tilde{\mathbf{x}}_k)$ in (9). $\square$

C Kernel Points Generation

We adopt the following experiment on hyperboloid to show this phenomenon. We uniformly sample $K = 8$ points along the unit circle in $\mathbb{L}^2$, as illustrated in Figure 3a. Then, we translate them along the geodesics from the origin, in the manner that all the points have the same distance to the origin. For each distance value, we record the gradient norm of the first term of 5, i.e. the reciprocals of the pairwise distances. The results are shown in Figure 3b. It is evident that the gradient norm decreases exponentially with distance, illustrating a typical scenario where gradients vanish when the distance between the kernel points and the origin is large.



(a)



(b)

Figure 3: Illustration of gradient vanishment when the kernel points are too distance. (a) The sampled points in the experiment; (b) Gradient norm decreases as points goes aways from the origin.

D More on Node Classification

The size of the model is mainly determined by three hyper-parameters: number of kernel points, hidden dimensions, and layers. In Table 2, we provide a simple experiment and show how hidden dimensions and layers of the model influence overall performance. Specifically, we run SKN model on two benchmarks, and the result is given by Table 2. Nevertheless, We emphasize more on the number of kernel points.

Analysis on Number of Kernels. For node classification, we record the performance of HKNs using K = 2, 3, $\cdots$, 9 kernel points while fixing the other hyper-parameters. The results are displayed in Table 3 and plotted in Figures 4 and 5. In those figures, we only include the mean but not the standard deviation for clarity. We notice that the best performance is achieved with a moderate K, which accommodates the tradeoff between expressivity and overfitting. We conclude that the kernel points in HKConv are effective in extracting hyperbolic features. We also notice that when learning on hyperboloid, model performance is stable w.r.t different kernel size, while it is much more unstable when learning on Poincaré. Additionally, CUDA memory can easily reach its limitation when the kernel size is large. This can be a potential challenge when performing node classification on large graphs.

However, it is worth to notice that the number of kernel points is not same as the meaning of kernel size in traditional convolutions. For a traditional convolution, the receptive field is determined by

Table 2: Model size effect.

Dataset	Embedding Dim	Num Layers	Num KPs	Params	F1 (%)
Actor	16	4	4	18501	59.41 ± 0.85
Actor	32	4	4	49659	65.92 ± 1.11
Actor	48	4	4	73893	66.52 ± 1.15
Actor	32	2	4	34413	65.03 ± 2.30
Actor	32	3	4	38769	65.87 ± 1.24
Actor	32	4	4	49659	65.92 ± 1.11
Squirrel	8	3	4	17425	71.39 ± 1.05
Squirrel	16	3	4	35857	72.25 ± 0.20
Squirrel	24	3	4	55313	71.71 ± 1.63
Squirrel	16	2	4	34701	68.53 ± 1.17
Squirrel	16	3	4	35857	72.25 ± 0.20
Squirrel	16	4	4	37013	74.55 ± 0.33

the size of the kernel, but for graph related tasks, the receptive field is determined by the number of times (layers) the message propagates. The kernel in hkconv is only used for feature transformation (discussed in Appendix 4.4), thus, its receptive field is not related to the number of kernel points, but determined by the number of hkconv layers.

Table 3: Node classification results with different numbers of kernel points. The mean F1 score (%) and standard deviation are reported. (OOM indicates out of memory)

SKN	Cornell	Texas	Wisconsin	Chameleon	Squirrel	Actor
2	84.85±4.73	92.42±4.73	90.12±4.66	80.77±0.63	70.94±2.21	65.83±1.69
3	86.36±3.94	90.91±6.01	89.51±4.66	82.54±1.30	72.33±0.95	65.53±1.60
4	86.36±3.94	89.39±8.60	**92.59**±1.85	82.17±1.39	72.25±0.20	65.92±1.11
5	86.36±3.94	**93.94**±6.56	90.12±7.01	82.42±1.28	72.73±1.36	66.28±0.63
6	86.36±3.94	87.88±10.25	92.59±5.56	81.69±0.63	**73.05**±1.80	**66.42**±1.42
7	**87.12**±2.62	88.64±8.19	91.98±6.50	82.30±0.94	72.86±1.00	65.76±1.92
8	87.12±2.62	92.42±3.47	88.27±5.66	**82.60**±1.14	73.00±0.79	OOM
9	86.36±3.94	90.91±6.01	92.59±5.56	82.42±0.37	OOM	OOM

BKN	Cornell	Texas	Wisconsin	Chameleon	Squirrel	Actor
2	78.03±3.47	75.00±2.27	81.48±3.21	65.93±1.60	**63.25±1.50**	**65.46±2.54**
3	76.52±4.73	76.52±4.73	74.07±9.62	66.97±1.22	62.82±0.87	58.71±8.66
4	78.03±4.73	75.00±2.27	**82.10±2.83**	**68.25±3.32**	OOM	OOM
5	77.27±3.94	74.24±5.72	74.07±11.11	62.76±10.28	OOM	OOM
6	**78.03±3.47**	76.52±4.73	77.78±12.83	67.09±3.91	OOM	OOM
7	76.52±4.73	**77.27**±3.94	69.75±5.95	60.38±10.11	OOM	OOM
8	77.27±3.94	77.27±3.94	72.22±1.85	59.34±11.28	OOM	OOM
9	77.27±3.94	75.76±5.25	80.25±2.83	59.89±10.34	OOM	OOM

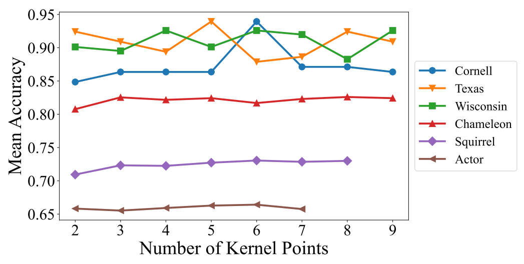


Figure 4: Node classification results for SKN.

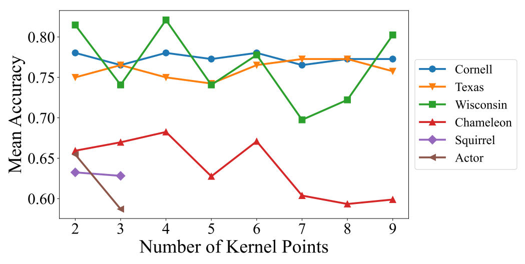


Figure 5: Node classification results for BKN.

Embedding Visualizations. Figure 6 and Figure 7 show the embedding of GCN, HGCN (one representative graph convolution, and one representative hyperbolic convolution) and SKN (ours) on three representative node classification dataset: Chameleon and Squirrel [37]; Actor [38].

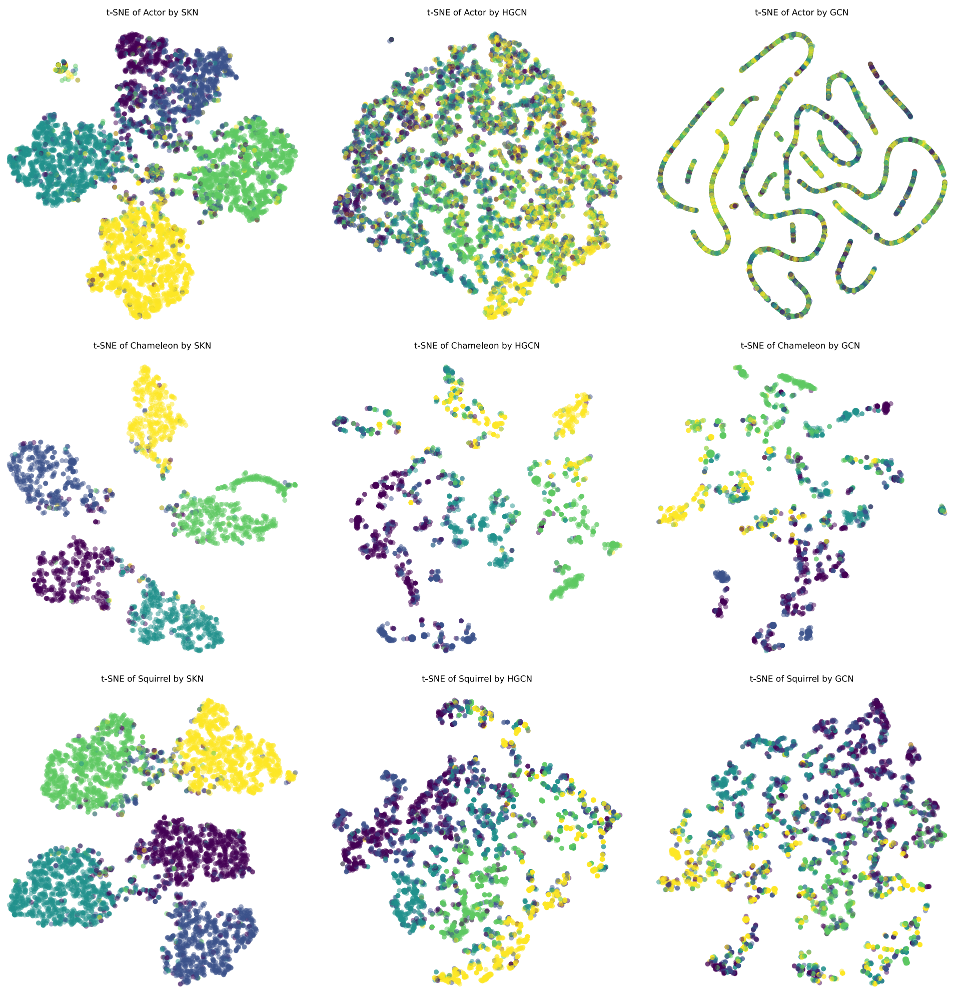


Figure 6: Embeddings (t-SNE).

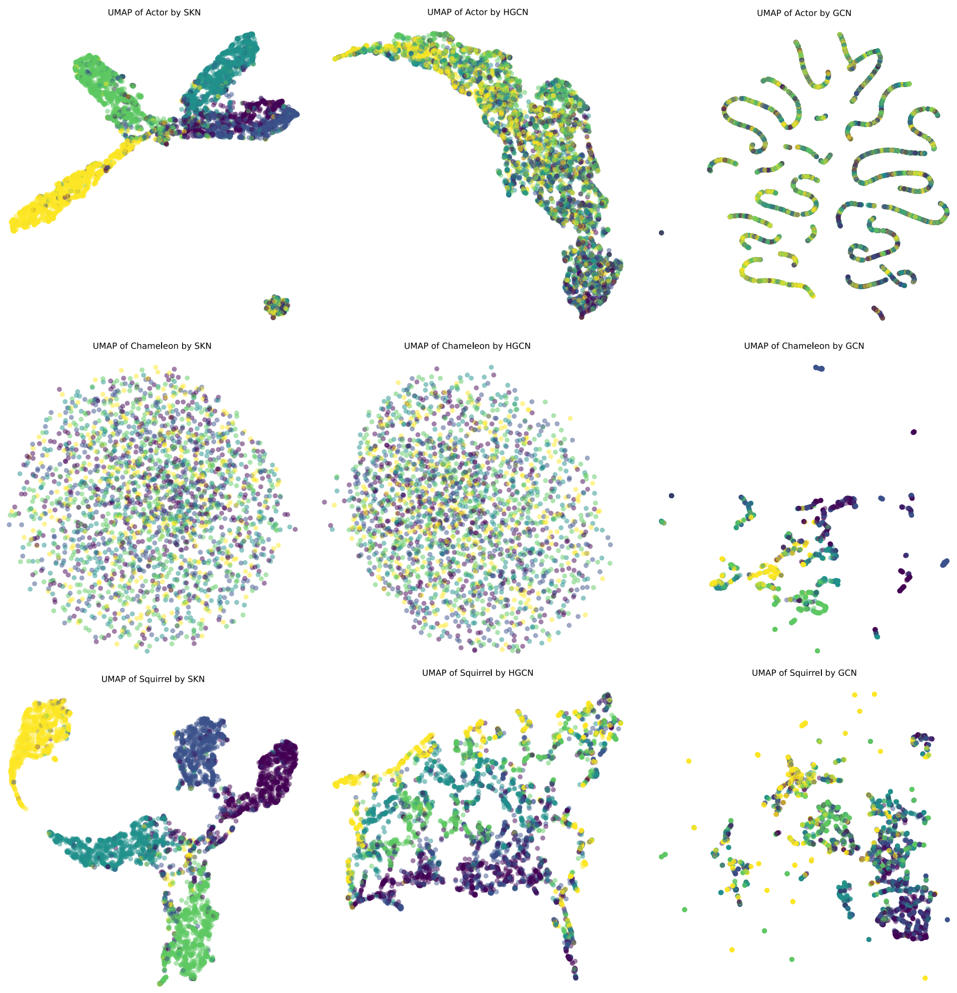


Figure 7: Embeddings (UMAP).

Table 4: Graph classification results with different numbers of kernel points. Mean accuracy (%) and standard deviation are reported.

		PTC	ENZYMES	PROTEIN	IMDB-B	IMDB-M
Data Statistics	# of graphs	344	600	1113	1000	1500
	# of classes	2	6	2	2	3
	Avg # of nodes	25.56	32.63	39.06	19.77	13
	Avg # of edges	25.96	62.14	72.82	96.53	65.94
	Hyperbolicity (δ)	0.725	1.15	1.095	0.2385	0.1157
Euclidean GNN	GCN [27]	63.87±2.65	66.39±6.91	74.54±0.45	73.32±0.39	50.27±0.38
	GIN [34]	66.58±6.78	59.79±4.31	70.67±1.08	72.78±0.86	47.91±1.03
	GMT [53]	65.89±2.16	67.52±4.28	75.09±0.59	73.48±0.76	50.66±0.82
Hyperbolic	HGCN [6]	55.17±3.21	53.63±5.12	68.41±2.15	61.71±0.97	50.12±0.71
	H2H-GCN [15]	66.14±3.91	60.84±4.72	72.12±3.63	68.19±0.82	48.31±0.63
	HyboNet [8]	65.56±4.19	56.82±5.39	65.36±2.81	71.26±1.28	54.35±0.98
	WLHN [46]	-	62.5±5.0	75.9±1.9	73.4±3.7	49.7±3.6
Ours(HKNs)	SKN	**73.69**±4.81	**82.46**±5.62	**83.22**±5.19	**79.92**±1.31	**56.53**±1.02
	BKN	63.33±8.82	53.67±6.66	58.52±0.64	61.33±1.53	49.83±3.69
Ablations	SKN-direct	66.42±3.13	58.31±4.81	64.78±3.51	67.65±0.89	46.31±0.61
	SKN-random	61.24±8.32	67.41±8.62	68.85±7.81	68.97±4.38	52.14±3.83
	SKN-train	NaN	NaN	NaN	NaN	NaN

For Figure 6, we project hyperbolic embedding to $T_{\mathbf{o}}\mathbb{H}^n$ and use t-SEN [47] with Euclidean metric for dimensionality reduction. For Figure 7, we directly use hyperbolic distance metric and apply UMAP [48] for dimensionality reduction.

E Graph Classification

In graph classification, one assigns class labels to full graphs. Unlike node classification, a global pooling is taken after node embeddings.

Settings. We use the following graph datasets to evaluate our model: chemical graphs including PTC [49], ENZYMES [50], PROTEIN [51]; social graphs including IMDB-B and IMDB-M [52]. In particular, these datasets all have small hyperbolicity (the average δ over all graphs in each dataset is reported in Table 4). The task is supervised and each dataset is partitioned into a training set, a validation set and a test set.

To build the HKNs, we use the validation sets to determine the number of kernel points K from $\{2, 3, 4, 5, 6, 7, 8, 9\}$. To avoid overfitting, dropout is applied to the input of each layer. In all experiments, we use the Riemannian Adam [32] as the optimizer. We consider the standard hyperboloid with curvature $\kappa = -1$ as our underlying hyperbolic space.

For this task, the HKN architecture contains an embedding layer where graph node features are represented in the hyperbolic space; a cascading of HKConv layers where the number of layers is validated from $\{2, 3, 4, 5, 6, 7\}$; a global pooling by taking the mean for all graph node features; an output layer via (16) for SKN and (15) for BKN, whose output represents the likelihood of the classes.

Baselines. We compare HKNs with graph neural networks including GCN [27], GIN [34], GMT [53], and recent hyperbolic neural networks including HGCN [6], H2H-GCN [15], HyboNet [8], WLHN [46]. For large-scale graph datasets, we compare SKN with the above methods for which relevant results have been reported. We report the mean and the standard deviation from three independent implementations. The results are reported in Table 4 and 5 respectively.

Results. SKN consistently achieves the best performance in all the datasets. Moreover, its advantage over the second best is highly significant (approximately 8% in PTC, 15% in ENZYMES, 8% in PROTEIN, 6% in IBDB-B and 2% in IBDB-M). Although BKN does not provide such impressive performance, it is still competent with other hyperbolic models on certain tasks. We note that the hyperbolic neural network baselines do not show a clear advantage over graph neural networks. This

Table 5: Graph classification results for ogbg-molhiv and ogbg-molpcba datasets. Mean accuracy (%) and standard deviation are reported under the standard metrics used in [54]

	ogbg-molhiv ROC-AUC	**ogbg-molpcba** Avg. Precision
GCN [27]	76.06±0.97	20.20±0.24
GIN [34]	75.58±1.40	22.66±0.28
HGCN [6]	75.91±1.48	17.52±0.20
HyboNet [8]	76.83±0.52	20.17±0.31
WLHN [46]	78.41±0.31	22.90±0.25
SKN	**79.75**±0.47	**27.87**±0.32

indicates that these baselines do not extract effective geometric information other than the graph structure. We believe the considerable improvement reflects that SKN has the capacity in extracting useful patterns from local geometric embedding of features in full graphs.

Ablations. We further analyze the effect of the important components in HKConv, namely, the way we perform translation in (8) and the fixed hyperbolic kernel points that minimize (5). We consider the following three ablations (specifically, we use SKN for ablation since it has previously yielded the best results):

- **SKN-direct**: we apply hyperbolic linear layers directly to each $\mathbf{x}_i$ in (8) instead of $\mathbf{x}_i \ominus \mathbf{x}$, and use $d_{\mathcal{L}}(\mathbf{x}_i, \mathrm{PT}_{\mathbf{o}\to\mathbf{x}}(\tilde{\mathbf{x}}_k))$ to translate the kernels to the neighborhood of $\mathbf{x}$ instead of $d_{\mathcal{L}}(\mathbf{x}_i \ominus \mathbf{x}, \tilde{\mathbf{x}}_k)$ in (9).
- **SKN-random**: we randomly sample the kernel points $\{\tilde{\mathbf{x}}_k\}_{k=1}^{K}$, independently from a wrapped normal distribution [2] with unit standard deviation, instead of solving (5).
- **SKN-train**: we regard the positions of the kernel points as trainable parameters, which are optimized together with other network parameters.

We apply these alternatives to the same graph classification tasks and present the results in the same Table 4. Indeed, the performance deteriorates significantly if we perform translation by moving the kernels to the neighborhood of input features. We believe that SKN-direct fails to capture critical geometric information due to the varying distortion of the kernel across different locations. Similarly, generating the kernel points randomly results in low accuracy. This result validates the formulation of the loss function in (5). Moreover, if we consider the locations of the kernel points as learnable parameters, the training process will be numerically unstable and produce NaN for all datasets. The ablation studies have validated the necessity and competitiveness of the key components of HKConv.

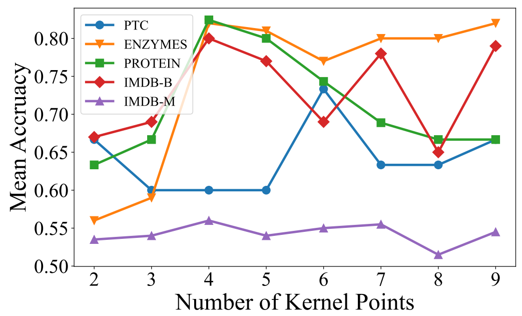


Figure 8: Graph classification results with different numbers of kernel points for SKN.

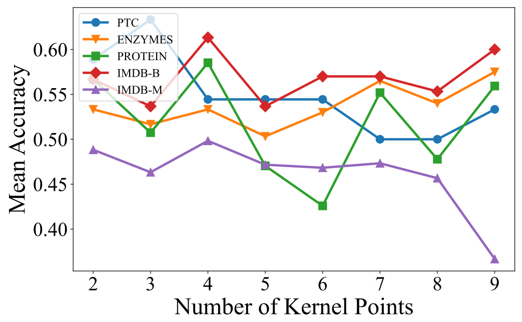


Figure 9: Graph classification results with different numbers of kernel points for BKN.

Analysis on Number of Kernels. For graph classification, we also study how the performance of HKNs changes with different numbers of kernels. We record the performance using SKN and BKN with $K = \{2, 3, 4, 5, 6, 7, 8, 9\}$ kernel points. To focus on the number of kernels, the other hyper-parameters are fixed according to the main task. The results are reported in Table 6 and plotted

Table 6: Graph classification results with different numbers of kernel points. The mean accuracy (%) and standard deviation are reported.

SKN	PTC	ENZYMES	PROTEIN	IMDB-B	IMDB-M
2	66.73±4.21	56.05±5.22	63.60±4.00	66.83±1.29	53.36±0.95
3	60.12±4.24	59.78±5.60	66.83±3.82	70.02±1.53	54.48±0.97
4	60.50±4.63	**82.46**±5.62	**83.22**±5.19	**79.92**±1.31	55.51±0.99
5	60.58±4.22	80.92±5.21	80.12±3.82	76.67±1.40	53.77±0.84
6	**73.69**±4.81	76.25±5.27	74.15±4.10	69.37±1.40	55.13±1.53
7	62.74±4.34	80.12±5.65	68.61±3.89	77.46±1.55	**56.53**±1.02
8	62.86±4.30	80.42±5.81	66.50±4.55	65.15±1.13	51.87±1.17
9	67.28±4.35	82.27±5.31	66.44±3.89	79.13±1.22	54.33±0.93

BKN	PTC	ENZYMES	PROTEIN	IMDB-B	IMDB-M
2	58.89±5.09	53.33±4.51	56.67±2.94	56.67±6.66	48.83±3.51
3	**63.33±8.82**	51.67±5.03	50.74±10.68	53.67±11.85	46.33±2.93
4	54.44±1.92	53.33±2.08	**58.52±0.64**	**61.33±1.53**	**49.83±3.68**
5	54.44±6.93	50.33±7.09	47.04±7.40	53.67±6.35	47.17±4.01
6	54.44±11.71	53.00±6.92	42.59±1.70	57.00±6.08	46.83±2.02
7	50.00±3.33	56.50±2.12	55.19±10.26	57.00±6.08	47.33±3.82
8	50.00±12.01	54.00±11.31	47.78±8.01	55.33±8.96	45.67±3.33
9	53.33±12.01	**57.50±0.71**	55.93±5.59	60.00±1.00	36.67±6.25

in Figure 8 and Figure 9. The results reveal that indeed, a moderate number of kernel points leads to the best performance. In particular, for SKN, we observe that when the number of kernels is K = 2, the performance is on par with the baseline methods. This suggests that utilizing only a limited number of kernels is insufficient to fully capture the hyperbolic pattern. Conversely, further increasing the number of kernel points does not lead to continuous performance improvement. With a large number of kernel points, the neural network becomes complex and may overfit the data. The results for BKN are also consistent with what we found in SKN, although the overall performance on all datasets is worse than SKN.

F Link Prediction

We further conducted experiments for link prediction (LP), where one predicts missing edges within a graph. We also provide some more node classification (NC) results since these tasks are mutually related. Due to its superior performance compared to other models within the HKN framework, experiments are conducted exclusively using SKN.

Settings. Following many previous works in hyperbolic neural networks, we take three datasets preprocessed by Chami et al. [6]: Disease, Airport, and PubMed [55]. The first two datasets are created by Chami et al. [6], whereas the last dataset is well known citation networks.

Baselines. We compare SKN with the benchmark results compiled in Chen et al. [8]: GCN [27], GAT [39], SAGE [40], SGC [56], HGCN [6], LGCN [16], HyboNet [8], H2H-GCN [15] and NHGCN [21].

Results. We test SKN and HyboNet on our machine and show the other results using numbers in respective papers. The results are reported in Table 7.

Our model performs the best in Airport, on both LP and NC tasks. It better supports that SKN can make good use of local geometry, especially noting that the hyperbolicity of Airport (1) is closest to the hyperboloid (0.88). In the remaining datasets, SKN is still comparable with other hyperbolic methods.

G Non-Graph Task

In addition to the previously introduced graph-based tasks, we also consider a non-graph task involving language data that exhibits hierarchical structures. Machine translation translates one language to another and is a sequence-to-sequence modeling task. Dependency tree probing then

Table 7: Link prediction (LP) and node classification (NC) results. Mean and standard deviation are reported: area under the ROC curve (%) for LP and F1 score (%) for NC.

	Disease		Airport		PubMed	
# of nodes	1,044		3,188		19,717	
# of edges	1,043		18,631		44,327	
Hyperbolicity δ	0		1		3.5	
	LP	NC	LP	NC	LP	NC
GCN	64.7±0.5	69.7±0.4	89.3±0.4	81.4±0.6	91.1±0.5	78.1±0.2
GAT	69.8±0.3	70.4±0.4	90.5±0.3	81.5±0.3	91.2±0.1	79.0±0.3
SAGE	65.9±0.3	69.1±0.6	90.4±0.5	82.1±0.5	86.2±1.0	77.4±2.2
SGC	65.1±0.2	69.5±0.2	89.8±0.3	80.6±0.1	94.1±0.0	78.9±0.0
HGCN	90.8±0.3	74.5±0.9	96.4±0.1	90.6±0.2	96.3±0.0	80.3±0.3
LGCN	96.6±0.6	84.4±0.8	-	-	96.6±0.1	78.6±0.7
H2H-GCN	**97.0**±0.3	88.6±1.7	96.4±0.1	89.3±0.5	**96.9**±0.0	79.9±0.5
NHGCN	92.8±0.2	**91.7**±0.7	97.2±0.3	92.4±0.7	**96.9**±0.1	**80.5**±0.0
HyboNet	96.5±0.4	90.4±0.9	95.2±0.4	90.4±0.5	94.3±0.2	78.6±0.2
SKN (Ours)	86.9±0.3	85.4±0.2	**97.3**±0.3	**95.8**±0.3	94.4±0.1	78.6±0.2

analyzes the obtained model. Similar as the link prediction experiments F, experiments are conducted exclusively using SKN.

Settings. For a fair comparison, we adopt the network architecture of the hyperbolic transformer model used by Chen et al. [8], with the modification that HKConv is applied for aggregation as follows. Given the query set $\mathcal{Q} = \{\mathbf{q}_1, \dots, \mathbf{q}_{|\mathcal{Q}|}\}$, the key set $\mathcal{K} = \{\mathbf{k}_1, \dots, \mathbf{k}_{|\mathcal{K}|}\}$, and the value set $\mathcal{V} = \{\mathbf{v}_1, \dots, \mathbf{v}_{|\mathcal{V}|}\}$, where $|\mathcal{K}| = |\mathcal{V}|$, We implement HKConv for each $i = 1, \cdots, |\mathcal{V}|$ as follows:

$$\begin{aligned} \mathbf{v}_{ijk} &= \mathrm{HLinear}_k(\mathbf{v}_j \ominus \mathbf{v}_i), \quad k = 1, \cdots, K; \\ \mathbf{v}'_{ij} &= \mathrm{HCent}(\{\mathbf{v}_{ijk}\}_{k=1}^K, \{d_{\mathcal{L}}(\mathbf{v}_j \ominus \mathbf{v}_i, \tilde{\mathbf{x}}_k)\}_{k=1}^K); \\ \boldsymbol{\mu}_i &= \mathrm{HCent}(\{\mathbf{v}'_{ij}\}_{j=1}^{|\mathcal{V}|}, \{w_{ij}\}_{j=1}^{|\mathcal{V}|})), \end{aligned}$$

where for $j = 1, \cdots, |\mathcal{V}|$,

$$w_{ij} = \frac{\exp\left(\frac{-d_{\mathcal{L}}^2(\boldsymbol{q}_i, \boldsymbol{k}_j)}{\sqrt{n}}\right)}{\sum_{k=1}^{|\mathcal{K}|} \exp\left(\frac{-d_{\mathcal{L}}^2(\boldsymbol{q}_i, \boldsymbol{k}_k)}{\sqrt{n}}\right)}.$$

Here, the first two equations are exactly (8) and (9), presented for clear notation. The last equation is (11) where the weights are determined by the query set and the key set. Overall, HKConv performs a further transformation of $\mathbf{v}_i$ that utilizes the local geometric embedding.

Following Chen et al. [8], we take two machine translation datasets: IWSLT'14 and WMT'14[3] English-German. Dependency tree probing is tested on the former.

Baselines. We compare with ConvSeq2Seq [57], Transformer [58], DynamicConv [59], BiBERT [60]; as well as hyperbolic methods including HNN++ [7], HATT [22] and HyboNet [8]. Notably, HNN++ employs its hyperbolic convolution, which, although unsuitable for graph classification tasks due to the heterogeneous graph structures, is well-suited for this task. We report the results in Table 8, where we compare our method with DynamicConv and BiBERT. The benchmark results for ConvSeq2Seq, Transformer, HNN++, HATT and HyboNet are taken from Shimizu et al. [7] and Chen et al. [8].

Results. SKN consistently outperforms all baseline methods across both tasks and metrics. Notably, since we intentionally maintain the same attention mechanism as HyboNet but replace the aggregation with HKConv, the improved performance is not attributed to attention. We believe the encouraging results are due to the better expressivity of HKConv as well as the capability of learning local information to countervail the global attention in the transformer.

[3]We have corrected the typo of the year number in the reference.

Table 8: Machine translation and dependency tree probing results. The BLEU (BiLingual Evaluation Understudy) scores on the test set are reported for machine translation. UUAS, Dspr., Root%, Nspr. are reported for dependency tree probing.

	Machine Translation				Dependency Tree Probing			
	IWSLT' 14	WMT' 14			Distance		Depth	
	d=64	d=64	d=128	d=256	UUAS	Dspr.	Root%	Nspr.
ConvSeq2Seq [57]	23.6	14.9	20.0	21.8	-	-	-	-
Transformer [58]	23.0	17.0	21.7	25.1	0.36	0.3	12	0.88
DynamicConv [59]	24.1	15.2	20.4	22.1	-	-	-	-
BiBERT [60]	23.8	18.7	22.5	27.8	-	-	-	-
HNN++ [7]	22.0	17.0	19.4	21.8	-	-	-	-
HATT [22]	23.7	18.8	22.5	25.5	0.5	0.64	49	0.88
HyboNet [8]	25.9	19.7	23.3	26.2	0.59	0.7	64	0.92
SKN (Ours)	**27.3**	**20.1**	**25.6**	**29.1**	**0.62**	**0.74**	**72**	**0.94**