Hyperbolic Convolution via Kernel Point Aggregation

Eric Qu
Division of Natural and Applied Sciences
Duke Kunshan University
Kunshan, Jiangsu 215316, China
zhonghang.qu@duke.edu

Dongmian Zou
Division of Natural and Applied Sciences
Duke Kunshan University
Kunshan, Jiangsu 215316, China
dongmian.zou@duke.edu

Abstract

For non-Euclidean data, it is of vital importance to learn representations according to their underlying geometry. Studies have revealed that the hyperbolic space can effectively embed hierarchical or tree-like data. In particular, the few past years have witnessed a rapid development 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 do not embrace the convolution operation and ignore 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 not only expressively learns local features according to the hyperbolic geometry, but also enjoys equivariance to permutation of hyperbolic points and invariance to parallel transport of a local neighborhood. We show that using HKConv layers, the Hyperbolic Kernel Network (HKN) advances state-of-the-art in various tasks.

1 Introduction

Recent advances of geometric deep learning have achieved remarkable successes for data in non-Euclidean domains. In applications, many datasets show a hierarchical or tree-like structure. For these data, the hyperbolic space has proved effective as it has large capacity and allows embedding with small distortion (Sonthalia and Gilbert, 2020). Many hyperbolic neural network methods have arisen and found wide applications such as word embedding (Nagano et al., 2019), knowledge graph completion (Chami et al., 2020), and image segmentation (Atigh et al., 2022). In those neural networks, Euclidean neural operations are generalized to one of the hyperbolic coordinate models, such as the Poincaré ball model or the Lorentz model.

One core component in hyperbolic neural networks is the linear layers, known as fully-connected layers, in the hyperbolic space (Ganea et al., 2018; Chami et al., 2019; Shimizu et al., 2021; Chen et al., 2022). However, hyperbolic linear layers are unaware of the structure within the input data, especially local patterns. In the Euclidean domain, convolution is often used to extract local information from the input. The local connectivity has made convolutional layers an important building block that fostered the early success of deep learning (LeCun et al., 2015). In convolutional neural networks (CNN), convolutional filters extract local patterns from the input by measuring and aggregating correlations between the filter and different regions of the input. Unfortunately, hyperbolic neural networks seldom contain convolutional layers. Some hyperbolic networks (Khrulkov et al., 2020; Ahmad and Lecue, 2022; Atigh et al., 2022) carry out convolutions in regular CNN layers, which are not regarded as hyperbolic convolutions. The only existing work that explicitly defines hyperbolic convolution is HNN++ (Shimizu et al., 2021), 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. This makes it impossible to extend to graph input.

Preprint.

On the other hand, many works (Liu et al., 2019; Chami et al., 2019; Bachmann et al., 2020; Dai et al., 2021; Zhang et al., 2021) have 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, such graph convolution depends only on the topology of the underlying graph and neglects the rich geometric patterns in the hyperbolic space itself.

One difficulty with designing hyperbolic convolution lies in the fact that principles in designing CNN convolutions do not extend to the hyperbolic space. On one hand, the hyperbolic space is usually used as the embedding space of features, not a domain where the features are defined as functions. On the other hand, the hyperbolic space is not a regular grid and cannot facilitate alignment of convolutional kernels and input features. To address the former issue, the convolutional filter has to be defined also in the same hyperbolic space; to address the latter 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 (Atzmon et al., 2018; Thomas et al., 2019; Lin et al., 2020) and use fixed point kernels in the hyperbolic space to design a hyperbolic convolutional layer, called the hyperbolic kernel convolution (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. We outline our contributions as follows.

- We formulate HKConv, a novel hyperbolic convolution method based on point kernels. To the best of our knowledge, this is the first hyperbolic neural layer that effectively extracts local hyperbolic patterns.
- HKConv is equivariant to permutation of ordering of the input hyperbolic features. Moreover, it is invariant to parallel translation of a local neighborhood.
- We demonstrate the effectiveness of HKConv by using it in hyperbolic neural networks, which achieve state-of-the-art performance in various tasks.

2 Hyperbolic Geometry

Hyperbolic geometry is a non-Euclidean geometry with a constant negative curvature (Cannon et al., 1997; Anderson, 2006). In order to define operations in the hyperbolic space, it is convenient to work with a Cartesian-like model (coordinate system) such as the Lorentz model and the Poincaré ball model (both are isometric). We choose to work with the Lorentz model since it is known to be numerically stable and provide sufficient space for optimization (Nickel and Kiela, 2018; Chami et al., 2019).

The Lorentz Model Given a constant negative curvature $\kappa < 0$, the Lorentz model $\mathbb{L}^n = \mathbb{L}^n_\kappa = (\mathcal{L}, \mathfrak{g})$, known as the hyperboloid, is an n-dimensional manifold $\mathcal{L}$ embedded in $\mathbb{R}^{n+1}$. Every point in $\mathbb{L}^n$ is represented by $\boldsymbol{x} = \begin{bmatrix} x_t \\ \boldsymbol{x}_s \end{bmatrix}$, where $x_t > 0$ is the "time component" and $\boldsymbol{x}_s \in \mathbb{R}^n$ is the "spatial component". They satisfy $\langle \boldsymbol{x}, \boldsymbol{x} \rangle_\mathcal{L} = 1/\kappa$, where $\langle \cdot, \cdot \rangle_\mathcal{L}$ is the Lorentz inner product induced by the metric tensor $\mathfrak{g} = \operatorname{diag}([-1, \mathbf{1}_n^\top])$ ($\mathbf{1}_n \in \mathbb{R}^n$ is the n-dimensional vector whose entries are all 1's):

$$\langle \boldsymbol{x}, \boldsymbol{y} \rangle_\mathcal{L} := \boldsymbol{x}^\top \mathfrak{g} \boldsymbol{y} = -x_t y_t + \boldsymbol{x}_s^\top \boldsymbol{y}_s,\ \boldsymbol{x}, \boldsymbol{y} \in \mathbb{L}^n. \quad (1)$$

Geodesics Shortest paths in the hyperbolic space are called geodesics. The distance between two points in hyperbolic space is the length of the geodesic between them. With the Lorentz model, the distance between two points $\boldsymbol{x}$ and $\boldsymbol{y}$ is given by, e.g., Chami et al. (2019):

$$d_\mathcal{L}(\boldsymbol{x}, \boldsymbol{y}) = (-\kappa)^{-1/2} \cosh^{-1}(\kappa \langle \boldsymbol{x}, \boldsymbol{y} \rangle_\mathcal{L}). \quad (2)$$

Tangent Space The tangent space at $\boldsymbol{x} \in \mathbb{L}^n$, $\mathcal{T}_{\boldsymbol{x}}\mathbb{L}^n$, is the first order approximation of the manifold around $\boldsymbol{x}$, and can be represented as an n-dimensional subspace of $\mathbb{R}^{n+1}$. Specifically, $\mathcal{T}_{\boldsymbol{x}}\mathbb{L}^n := \{\boldsymbol{y} \in \mathbb{R}^{n+1} : \langle \boldsymbol{y}, \boldsymbol{x} \rangle_\mathcal{L} = 0\}$.

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

$$\exp_{\boldsymbol{x}}(\boldsymbol{v}) = \cosh(\phi)\boldsymbol{x} + \phi^{-1}\sinh(\phi)\boldsymbol{v}, \quad \phi = \sqrt{-\kappa}\|\boldsymbol{v}\|_\mathcal{L},$$

where $\|\cdot\|_\mathcal{L} := \sqrt{\langle \cdot, \cdot \rangle_\mathcal{L}}$. The logarithmic map $\log_{\boldsymbol{x}} : \mathbb{L}^n \to \mathcal{T}_{\boldsymbol{x}}\mathbb{L}^n$ projects hyperbolic vectors into the tangent space. It is the inverse map of $\exp_{\boldsymbol{x}}$ in the sense that $\log_{\boldsymbol{x}}(\exp_{\boldsymbol{x}}(\boldsymbol{v})) = \boldsymbol{v}$ for any $\boldsymbol{v} \in \mathbb{L}^n$ and $\exp_{\boldsymbol{x}}(\log_{\boldsymbol{x}}(\boldsymbol{u})) = \boldsymbol{u}$ for any $\boldsymbol{u} \in \mathcal{T}_{\boldsymbol{x}}\mathbb{L}^n$. Using Lorentz coordinates,

$$\log_{\boldsymbol{x}}(\boldsymbol{u}) = \frac{\cosh^{-1}(\psi)}{\sqrt{-\kappa}} \frac{\boldsymbol{u} - \psi\boldsymbol{x}}{\|\boldsymbol{u} - \psi\boldsymbol{x}\|_\mathcal{L}}, \psi = \kappa\langle \boldsymbol{x}, \boldsymbol{u} \rangle_\mathcal{L}.$$

Parallel Translation For two points $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{L}^n$, the parallel transport, also called the parallel translation, from $\boldsymbol{x}$ to $\boldsymbol{y}$, $\mathrm{PT}_{\boldsymbol{x}\to\boldsymbol{y}}$, is a map that "transports" or "translates" a tangent vector from $\mathcal{T}_{\boldsymbol{x}}\mathbb{L}^n$ to $\mathcal{T}_{\boldsymbol{y}}\mathbb{L}^n$ along the geodesic from $\boldsymbol{x}$ to $\boldsymbol{y}$. The parallel transport in $\mathbb{L}^n$ is

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

3 Proposed Model

3.1 Hyperbolic Point Kernels

To construct an HKConv layer, the first step is to fix K kernel points $\{\tilde{\boldsymbol{x}}_k\}_{k=1}^K$ in the hyperbolic space $\mathbb{L}^m$, where m

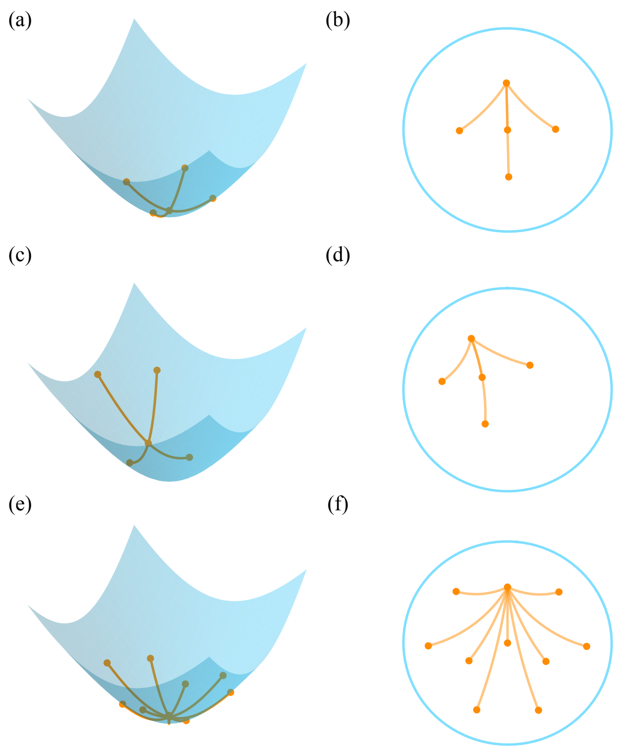

Figure 1: Illustration of kernel points in $\mathbb{L}^2$. (a) and (b) illustrate $K = 5$ points in the Lorentz and Poincaré models, respectively; (c) and (d) apply a parallel transport to all the points in (a) and (b); (e) and (f) show $K = 10$ points in the Lorentz and Poincaré models, respectively.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

is the dimension of the input features. The kernel points can be regarded as located in the neighborhood of the hyperbolic origin $\boldsymbol{o} = [1, \mathbf{0}_m^{\mathrm{T}}]^{\mathrm{T}} \in \mathbb{L}^m$, where $\mathbf{0}_m$ is the m-dimensional zero vector.

The kernel points are pre-determined since it will be numerically unstable to train them in the hyperbolic space together with other network parameters (see Section 4.2). This does not restrict expressivity of HKConv, since learnable transformations are still applied to the input features.

The locations of $\{\tilde{\boldsymbol{x}}_k\}_{k=1}^K$ are determined according to the principle of Thomas et al. (2019): the kernel points should be as far as possible from each other, but also not too distant from the origin. Specifically, the kernels are determined by solving the optimization problem where the following loss function is minimized:

$$\mathfrak{L}(\{\tilde{\boldsymbol{x}}_k\}_{k=1}^K) = \sum_{k=1}^K \sum_{l \neq k} \frac{1}{d_{\mathcal{L}}(\tilde{\boldsymbol{x}}_l, \tilde{\boldsymbol{x}}_k)} + \sum_{k=1}^K d_{\mathcal{L}}(\boldsymbol{o}, \tilde{\boldsymbol{x}}_k). \quad (3)$$

In our context, the first term in (3) guarantees expressivity as it associates distinct features with each kernel point. If the kernel points are too close to each other, the output from correlation of the input and the kernel points will be too similar. The second term penalizes the case where the kernel points are too far from the origin, which would cause vanishing gradients when calculating correlation because of the $\cosh^{-1}$ function in (2).

In general, it is difficult to determine an analytical solution to the minimization of (3). We use the Riemannian gradient descent method (Becigneul and Ganea, 2019) with a learning rate of 0.0001 until it converges. Figure 1 shows examples of 5 and 10 kernel points in $\mathbb{L}^2$. 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 (3).

3.2 Hyperbolic Kernel Convolution

Analogous to convolution in the Euclidean space, HKConv applies the kernel points to local neighborhoods of input hyperbolic features. Such neighborhoods are not necessarily formed according to hyperbolic distances. For instance, when the hyperbolic features are embedding of graph features, we can consider the 1-hop neighbors according to graph topology. Let $\mathbb{X} \subset \mathbb{L}^m$ denote the collection of input hyperbolic features. For each $\boldsymbol{x} \in \mathbb{X}$, we use $\mathcal{N}(\boldsymbol{x}) \subset \mathbb{X} - \{\boldsymbol{x}\}$ to denote the neighbors of $\boldsymbol{x}$. 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 with optional attention. Figure 2 presents an illustration and we next describe the details.

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

$$\mathrm{T}_{\boldsymbol{x}\to\boldsymbol{y}}(\boldsymbol{u}) := \exp_{\boldsymbol{y}}\left(\mathrm{PT}_{\boldsymbol{x}\to\boldsymbol{y}}(\log_{\boldsymbol{x}}(\boldsymbol{u}))\right). \quad (4)$$

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

$$\boldsymbol{u} \ominus \boldsymbol{x} := \mathrm{T}_{\boldsymbol{x}\to\boldsymbol{o}}(\boldsymbol{u}). \quad (5)$$

Clearly, $\boldsymbol{x} \ominus \boldsymbol{x} = \boldsymbol{o}$. Moreover, because logarithmic and exponential maps are distance preserving, (5) preserves the relative distance between $\boldsymbol{x}$ and its neighbors. That is,

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

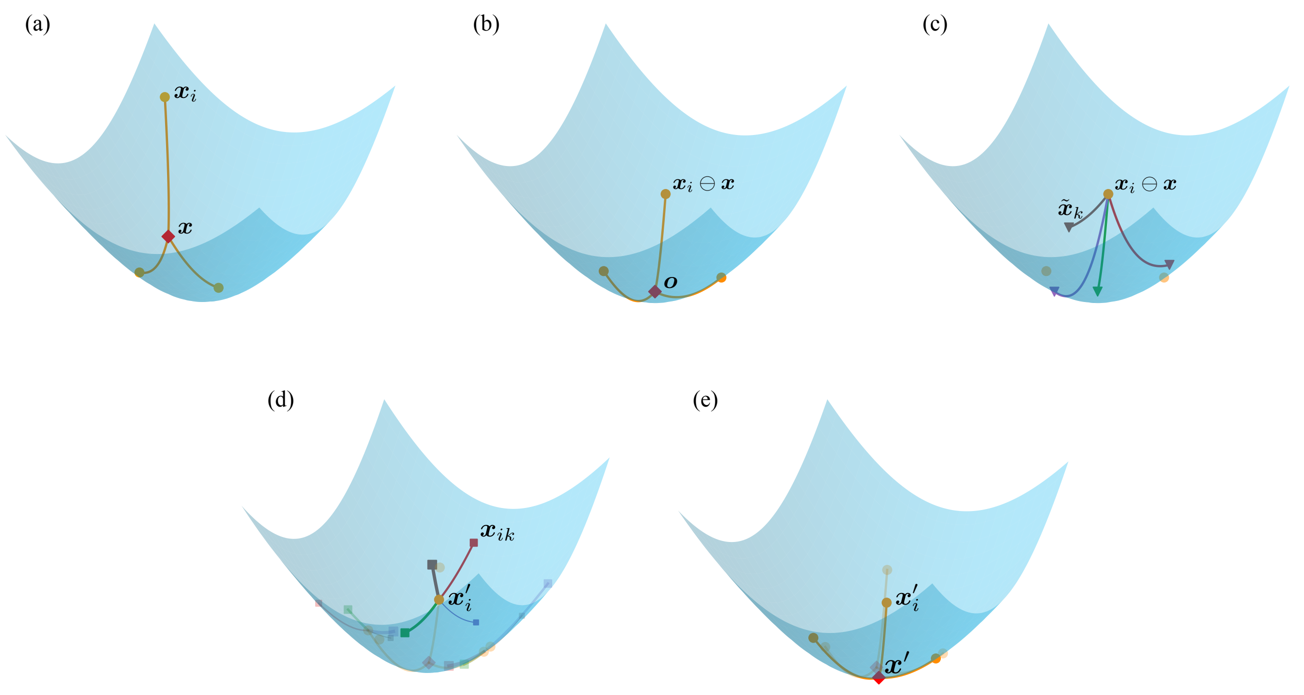

Figure 2: Illustration of HKConv. In this example, there are four kernel points and an input feature $\boldsymbol{x}$ with three neighbors in $\mathcal{N}(\boldsymbol{x})$. (a) The original positions of $\boldsymbol{x}$ (the diamond) and $\mathcal{N}(\boldsymbol{x})$ (the circle points). (b) $\{\boldsymbol{x}\}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

We remark that (5) is similar to "adding bias" in some designs of hyperbolic linear layers (Ganea et al., 2018; Chami et al., 2019).

For each $\boldsymbol{x}_i \in \mathcal{N}(\boldsymbol{x})$, the translation $\boldsymbol{x}_i \ominus \boldsymbol{x}$ gives the relative input feature of $\boldsymbol{x}_i$ with respect to $\boldsymbol{x}$. The first step of HKConv is to extract information using a learnable hyperbolic linear layer (Chen et al., 2022) for each kernel point $\tilde{\boldsymbol{x}}_k$, $k = 1, \cdots, K$. We denote it by $\mathrm{HLinear}_k : \mathbb{L}^m \to \mathbb{L}^n$ and extract

$$\boldsymbol{x}_{ik} = \mathrm{HLinear}_k(\boldsymbol{x}_i \ominus \boldsymbol{x}), \quad k = 1, \cdots, K. \tag{6}$$

Step 2. Similar to Euclidean convolution, we take the correlation between the input features and the kernel points $\tilde{\boldsymbol{x}}_k$. However, unlike the Euclidean domain where the kernels are moved to the neighborhood of the input, when we perform convolution, we move $\mathcal{N}(\boldsymbol{x})$ to the neighborhood of $\boldsymbol{o}$. This, on one hand, avoids different extent of distortion of the kernel points at different locations, and on the other hand ensures simple forms of the expressions of both the exponential and logarithmic maps since the spatial component of the origin is zero. Specifically, each neighbor $\boldsymbol{x}_i \in \mathcal{N}(\boldsymbol{x})$ is moved via the translation $\boldsymbol{x}_i \ominus \boldsymbol{x}$, and the correlation with the kernel point $\tilde{\boldsymbol{x}}_k$ is given by the hyperbolic distance $d_{\mathcal{L}}(\boldsymbol{x}_i \ominus \boldsymbol{x}, \tilde{\boldsymbol{x}}_k)$. Unlike Euclidean convolution, in our design, this correlation is used as the weights for aggregating the $\boldsymbol{x}_{ik}$ obtained in (6). In the hyperbolic space, one way of aggregating features is to find their hyperbolic centroid (Law et al., 2019), whose output feature is in the same hyperbolic space. More specifically, given a set of input features $\{\boldsymbol{u}_i\}_{i=1}^N$ and their corresponding weights $\{\nu_i\}_{i=1}^N$, the hyperbolic centroid (denoted by HCent) of $\{\boldsymbol{u}_i\}_{i=1}^N$ is given by

$$\mathrm{HCent}(\{\boldsymbol{u}_i\}_{i=1}^N, \boldsymbol{\nu}) = \frac{\sum_{i=1}^N \nu_i \boldsymbol{u}_i}{\sqrt{-\kappa}\left|\|\sum_{i=1}^N \nu_i \boldsymbol{u}_i\|_{\mathcal{L}}\right|}.$$

Therefore, aggregating the features yields an output feature for each neighbor of $\boldsymbol{x}$:

$$\boldsymbol{x}_i' = \mathrm{HCent}(\{\boldsymbol{x}_{ik}\}_{k=1}^K, \{d_{\mathcal{L}}(\boldsymbol{x}_i \ominus \boldsymbol{x}, \tilde{\boldsymbol{x}}_k)\}_{k=1}^K). \tag{7}$$

In (7), the correlation $d_{\mathcal{L}}(\boldsymbol{x}_i \ominus \boldsymbol{x}, \tilde{\boldsymbol{x}}_k)$ endows $\boldsymbol{x}_{ik}$ with information of local geometry in the feature space. On the contrary, other hyperbolic message passing (Liu et al., 2019; Chami et al., 2019) or attention (Gulcehre et al., 2019) methods only use graph structural information, but not take into account the geometry of the feature embedding space.

Step 3. Different input features may have different numbers of neighbors. The last step in HKconv is to aggregate all the $\boldsymbol{x}_i'$'s from the neighborhood. When there is no other information, we simply apply equal weights to all the neighbors. Nevertheless, it is also possible to apply the attention regime so that the weights are learnable. That is,

$$\boldsymbol{x}' = \mathrm{HCent}(\{\boldsymbol{x}_i'\}_{i=1}^N, \{w_i\}_{i=1}^N), \tag{8}$$

where $\{w_i\}_{i=1}^N$ is either taken to be $\mathbf{1}_N$ or learned.

Altogether, for each input feature $\boldsymbol{x} \in \mathbb{L}^m$, our hyperbolic convolution produces a new hyperbolic feature $\boldsymbol{x}' \in \mathbb{L}^n$ by cascading the operations (6), (7) and (8). The learnable parameters are contained in $\mathrm{HLinear}_k$'s in (6). For convenience, we call the chain of operations (6)–(8) an HKConv layer, and denote $\boldsymbol{x}' = \mathrm{HKConv}(\boldsymbol{x}; \mathcal{N}(\boldsymbol{x}))$. When $\mathcal{N}(\boldsymbol{x})$ is clear from the context, we also directly denote $\boldsymbol{x}' = \mathrm{HKConv}(\boldsymbol{x})$.

3.3 Properties of HKConv

In the above manner, (6) and (7) analogize to the Euclidean convolution, whereas (8) constitutes a local pooling operation. Since the aggregation only depends on the neighborhood, which is unaffected by any matrix representation, HKConv is equivariant to permutation of 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. We summarize this property in the following proposition.

Proposition 1 (Equivariance to permutation of the order of data points)**.** *Let $\boldsymbol{X} \in \mathbb{R}^{N\times(m+1)}$ be the data matrix whose j-th row $\boldsymbol{x}_j \in \mathbb{R}^{m+1}$ is identified as a point in $\mathbb{L}^m$. For any matrix $\boldsymbol{P} \in \mathbb{R}^{N\times N}$ that permutes the rows of $\boldsymbol{X}$,*

$$\mathrm{HKConv}(\boldsymbol{PX}) = \boldsymbol{P}\mathrm{HKConv}(\boldsymbol{X}),$$

where HKConv is applied to each row of $\boldsymbol{X}$.

The above result indicates that HKConv does not depend on a specific Euclidean representation. As to the hyperbolic features themselves, HKConv also enjoys invariance under local translations in the hyperbolic space. Specifically, given an HKConv layer, if we perform a parallel translation from an input point $\boldsymbol{x}$ to another point along the geodesic in the hyperbolic space, the output $\mathrm{HKConv}(\boldsymbol{x})$ will not change. This result holds locally, i.e., for a given $\boldsymbol{x}$ and its neighbors. We formulate it as the following theorem. The proof can be found in the supplementary materials.

Theorem 2 (Local translation invariance)**.** *Fix $\boldsymbol{x} \in \mathbb{X}$ and its neighborhood $\mathcal{N}(\boldsymbol{x}) \subset \mathbb{X}$. For any $\boldsymbol{y} \in \mathbb{L}^n$ in the geodesic from $\boldsymbol{o}$ to $\boldsymbol{x}$,*

$$\begin{aligned} &\mathrm{HKConv}(\mathrm{T}_{\boldsymbol{x}\to\boldsymbol{y}}(\boldsymbol{x}); \mathrm{T}_{\boldsymbol{x}\to\boldsymbol{y}}(\mathcal{N}(\boldsymbol{x}))) \\ =\ &\mathrm{HKConv}(\boldsymbol{x}; \mathcal{N}(\boldsymbol{x})). \end{aligned} \tag{9}$$

Theorem 2 indicates that HKConv is indeed an operation that depends on local geometry. In particular, along the geodesics, as long as the relative positions between $\boldsymbol{x}$ and its neighbors stay constant, the output of HKConv will not change, regardless of the distance between $\boldsymbol{x}$ and the hyperbolic origin.

3.4 Hyperbolic Kernel Network

From the above discussion, HKConv can be defined for any collection of input hyperbolic features with a specified neighborhood for each feature. One possible approach is to choose the nearest neighbors according to the hyperbolic distance. If the data are represented in a graph, then the graph edges can be inherited to the hyperbolic space and used to naturally define neighbors. In particular, unlike HNN++ (Shimizu et al., 2021), our hyperbolic convolution allows heterogeneous structures, where different features can have different numbers of neighbors.

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 an Euclidean vector. To this end, we apply the hyperbolic distance layer (Liu et al., 2019) that maps from $\mathbb{L}^n$ to $\mathbb{R}^\ell$. Specifically, given an input $\boldsymbol{x} \in \mathbb{L}^n$, the output is the vector of distances between $\boldsymbol{x}$ and ℓ learnable points $\{\boldsymbol{c}_i\}_{i=1}^\ell \subset \mathbb{L}^n$, that is,

$$[d_{\mathcal{L}}(\boldsymbol{x}, \boldsymbol{c}_1), \cdots, d_{\mathcal{L}}(\boldsymbol{x}, \boldsymbol{c}_\ell)]^{\mathrm{T}} \in \mathbb{R}^\ell.$$

4 Experiments

We evaluate the performance of HKN on the following two tasks: (1) graph classification; (2) machine translation and dependency tree probing. In graph classification, each node in a graph has a hyperbolic representation; in machine translation, there is no graph structure, but the feature vectors are treated as points lying in the hyperbolic space. In the first task, we focus on comparison with graph neural networks and hyperbolic GCNs. In the second task, we focus on comparison with hyperbolic convolution and attention. All the experiments were executed on a GPU server with NVIDIA GeForce RTX 3090 GPUs (24G memory). Each experiment uses a single GPU.

Table 1: Graph classification results. 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
	Avg Hyperbolicity (δ)	0.725	1.15	1.095	0.2385	0.1157
Euclidean GNN	GCN (Kipf and Welling, 2017)	63.87±2.65	66.39±6.91	74.54±0.45	73.32±0.39	50.27±0.38
	GIN (Xu et al., 2019)	66.58±6.78	59.79±4.31	70.67±1.08	72.78±0.86	47.91±1.03
	GMT (Baek et al., 2021)	65.89±2.16	67.52±4.28	75.09±0.59	73.48±0.76	50.66±0.82
Hyperbolic	HGCN (Chami et al., 2019)	55.17±3.21	53.63±5.12	68.41±2.15	61.71±0.97	50.12±0.71
	H2H-GCN (Dai et al., 2021)	66.14±3.91	60.84±4.72	72.12±3.63	68.19±0.82	48.31±0.63
	HyboNet (Chen et al., 2022)	65.56±4.19	56.82±5.39	65.36±2.81	71.26±1.28	54.35±0.98
Ours	HKN	**73.69**±4.81	**82.46**±5.62	**83.22**±5.19	**79.92**±1.31	**56.53**±1.02
Ablations	HKN-direct	66.42±3.13	58.31±4.81	64.78±3.51	67.65±0.89	46.31±0.61
	HKN-random	61.24±8.32	67.41±8.62	68.85±7.81	68.97±4.38	52.14±3.83
	HKN-train	NaN	NaN	NaN	NaN	NaN

4.1 Graph Classification

In graph classification, one assigns class labels to full graphs. 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 $\mathbb{L}^n$ (Sonthalia and Gilbert, 2020). We expect hyperbolic models to work well for datasets where the graphs have overall similar δ values.

Settings We use the following graph datasets to evaluate our model: chemical graphs including PTC (Helma et al., 2001), ENZYMES (Schomburg et al., 2004), PROTEIN (Borgwardt et al., 2005); social graphs including IMDB-B and IMDB-M (Yanardag and Vishwanathan, 2015). In particular, these datasets all have small hyperbolicity (the average δ over all graphs in each dataset is reported in Table 1). The task is supervised and each dataset is partitioned into a training set, a validation set and a test set.

To build the HKN, 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 (Becigneul and Ganea, 2019) as the optimizer. We consider the standard hyperboloid with curvature $\kappa = -1$ as our underlying hyperbolic space. We validate the learning rate from $\{0.001, 0.005, 0.01\}$; the weight decay rate from $\{0, 0.01, 0.001\}$; and the dropout rate from $\{0.25, 0.5, 0.75, 0.9\}$.

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 centroid of all graph node features; a hyperbolic distance layer whose output represents the likelihood of the classes.

Baselines We compare HKN with graph neural networks including GCN (Kipf and Welling, 2017), GIN (Xu et al., 2019), GMT (Baek et al., 2021), and recent hyperbolic neural networks including HGCN (Chami et al., 2019), H2H-GCN (Dai et al., 2021), HyboNet (Chen et al., 2022). We test all the methods on our machine. We report the mean and the standard deviation from three independent implementations in Table 1.

Results HKN 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). We also note that the hyperbolic neural network baselines do not show clear advantage over graph neural networks. This indicates that these baselines do not extract effective geometric information other than the graph structure. We believe the considerable improvement reflects that HKN has the capacity in extracting useful patterns from local geometric embedding of features in full graphs.

4.2 Additional Results of Graph Classification

Ablations We further analyze the effect of the important components in HKConv, namely, the way we perform translation in (6) and the fixed hyperbolic kernel points that minimize (3). To this end, we compare HKN with three alternative models as ablations:

- **HKN-direct**: we apply hyperbolic linear layers directly to each $\boldsymbol{x}_i$ in (6) instead of $\boldsymbol{x}_i \ominus \boldsymbol{x}$, and use $d_{\mathcal{L}}(\boldsymbol{x}_i, \mathrm{PT}_{\boldsymbol{o}\to\boldsymbol{x}}(\tilde{\boldsymbol{x}}_k))$ to translate the kernels to the neighborhood of $\boldsymbol{x}$ instead of $d_{\mathcal{L}}(\boldsymbol{x}_i \ominus \boldsymbol{x}, \tilde{\boldsymbol{x}}_k)$ in (7).

- **HKN-random**: we randomly sample the kernel points $\{\tilde{\boldsymbol{x}}_k\}_{k=1}^K$, independently from a wrapped normal distribution (Nagano et al., 2019) with unit standard deviation, instead of solving (3).
- **HKN-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 as above and present the results in the same Table 1. Indeed, the performance deteriorates significantly if we perform translation by moving the kernels to the neighborhood of input features. We believe that HKN-direct fails to capture some important geometric information due to different distortion of the kernel at different locations. Similarly, we get low accuracy if we randomly generate the kernel points. This result validates the formulation of the loss function in (3). 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.

Analysis on number of kernels In the above experiment, we determined the number of kernel points by validation. Intuitively, there is tradeoff between expressivity and overfitting. In the following, we study how the performance of HKN varies with the different numbers of kernels to further validate the contribution of the kernel points. We record the performance using HKN with $K = \{2, 3, 4, 5, 6, 7, 8, 9\}$ kernel points. To focus on the number of kernels, the other hyperparameters are fixed according to the main task. The results are reported in Table 2 and plotted in Figure 3. In Figure 3, we only include the mean but not the standard deviation for clarity.

The results reveal that indeed, a moderate number of kernel points leads to the best performance. We observe that when the number of kernels is $K = 2$, the performance is on par with the baseline methods. This implies that using only few kernels cannot fully extract the hyperbolic pattern. On the other hand, if we keep increasing the number of kernel points, the performance will not keep improving. With a large number of kernel points, the neural network becomes complex and may overfit the data.

4.3 Node Classification

We show experimental results of HKN for an additional graph task: node classification. In node classification, one classifies nodes within a graph according to some nodes with known labels. Unlike graph classification, no global pooling is taken in node classification. The final output is a vector of distances for each node, representing likelihood of all the classes.

Table 2: Graph classification results with different numbers of kernel points. Mean accuracy (%) and standard deviation are reported.

	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

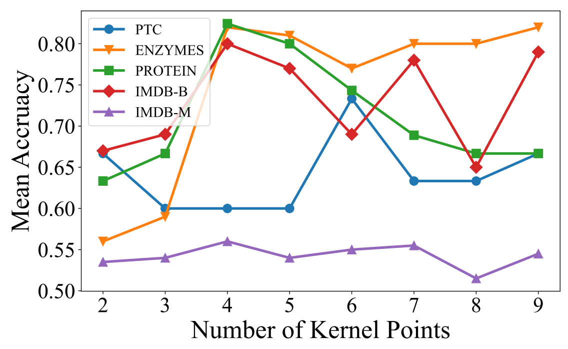

Figure 3: Graph classification results with different numbers of kernel points.

Settings As discussed in §4.1 of the main test, we expect HKN to work well for graph datasets whose Gromov hyperbolicity δ (Sonthalia and Gilbert, 2020) is similar to that of a hyperboloid. We test our HKN on the following datasets: the WebKB datasets (Craven et al., 2000) including Cornell, Texas and Wisconsin, where nodes are web pages and edges are hyperlinks; the Wikipedia Network datasets (Rozemberczki et al., 2021) including Chameleon and Squirrel, where nodes are Wikipedia pages and edges are links between two pages; the Actor Co-occurrence Network dataset (Tang et al., 2009) (Actor), where nodes represent actors and two nodes are connected if they co-appear in the same Wikipedia page. In these datasets, the hyperbolicity δ is either 1 or 1.5, very close to that of a hyperboloid. Moreover, we also test HKN on well-known citation networks including Cora (Namata et al., 2012) and PubMed (Sen et al., 2008), which show larger (and thus weaker) hyperbolicities.

Baselines We compare HKN with the following baseline methods: (1) multi-layer perceptron (MLP); (2) general graph neural networks including GCN (Kipf and Welling, 2017), GAT (Veličković et al., 2018), GraphSAGE (Hamilton et al., 2017), GCNII (Chen et al., 2020); (3) methods for heterophilous graphs including Geom-GCN (Pei et al., 2019), WRGAT (Suresh et al., 2021), LINKX (Lim et al., 2021), GloGNN and GloGNN++ (Li et al., 2022); (4) hyperbolic GCN methods: HGCN (Chami et al., 2019), Hy-

Table 3: Node classification results. Mean F1 score (%) and standard deviation are reported. NaN indicates a numerically unstable result.

	Cornell	Texas	Wisconsin	Chameleon	Squirrel	Actor	Cora	Pubmed
# of Nodes	183	183	251	2,277	5,201	7,600	2,708	19,717
# of Edges	280	295	466	31,421	198,493	26,752	5,278	44,327
# of Features	1,703	1,703	1,703	2,325	2,089	931	1,433	500
# of Classes	5	5	5	5	5	5	6	3
Hyperbolicity	$\delta = 1$	$\delta = 1$	$\delta = 1$	$\delta = 1.5$	$\delta = 1.5$	$\delta = 1.5$	$\delta = 11$	$\delta = 3.5$
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	75.69±2.00	87.16±0.37
GCN (Kipf and Welling, 2017)	60.54±5.30	55.14±5.16	51.76±3.06	64.82±2.24	53.43±2.01	27.32±1.10	86.98±1.27	88.42±0.50
GAT (Veličković et al., 2018)	61.89±5.05	52.16±6.63	49.41±4.09	60.26±2.50	40.72±1.55	27.44±0.89	87.30±1.10	86.33±0.48
GraphSAGE (Hamilton et al., 2017)	75.95±5.01	82.43±6.14	81.18±5.56	58.73±1.68	41.61±0.74	34.23±0.99	86.90±1.04	88.45±0.50
GCNII (Chen et al., 2020)	77.86±3.79	77.57±3.83	80.39±3.40	63.86±3.04	38.47±1.58	37.44±1.30	**88.37**±1.25	**90.15**±0.43
Geom-GCN (Pei et al., 2019)	60.54±3.67	66.76±2.72	64.51±3.66	60.00±2.81	38.15±0.92	31.59±1.15	85.35±1.57	89.95±0.47
WRGAT (Suresh et al., 2021)	81.62±3.90	83.62±5.50	86.98±3.78	65.24±0.87	48.85±0.78	36.53±0.77	88.20±2.26	88.52±0.92
LINKX (Lim et al., 2021)	77.84±5.81	74.60±8.37	75.49±5.72	68.42±1.38	61.81±1.80	36.10±1.55	84.64±1.13	87.86±0.77
GloGNN (Li et al., 2022)	83.51±4.26	84.32±4.15	87.06±3.53	69.78±2.42	57.54±1.39	37.35±1.30	88.31±1.13	89.62±0.35
GloGNN++ (Li et al., 2022)	**85.95**±5.10	84.05±4.90	88.04±3.22	71.21±1.84	57.88±1.76	37.70±1.40	88.33±1.09	89.24±0.39
HGCN (Chami et al., 2019)	79.43±0.47	70.13±0.32	83.26±0.51	NaN	62.31±0.57	36.58±0.79	79.84±0.27	80.41±0.42
HyboNet (Chen et al., 2022)	77.27±0.71	72.23±0.94	86.52±0.51	74.91±0.58	69.07±0.64	45.74±0.82	81.42±0.93	78.45±1.17
HKN (Ours)	84.14±0.53	**90.94**±0.66	**91.31**±0.36	**85.27**±0.57	**75.26**±0.38	**67.20**±0.68	80.34±0.59	77.62±0.54

Table 4: Node classification results with different numbers of kernel points. The mean F1 score (%) and standard deviation are reported.

	Cornell	Texas	Wisconsin	Chameleon	Squirrel	Actor	Cora	Pubmed
2	82.08±0.44	79.08±0.36	87.20±0.75	83.08±0.58	75.11±0.42	65.87±0.55	**80.34**±0.59	**77.62**±0.54
3	76.71±0.47	**90.94**±0.66	88.90±0.33	79.79±0.50	75.03±0.50	65.14±0.45	78.97±0.48	77.27±0.27
4	83.92±0.31	84.19±0.90	**91.31**±0.36	80.84±0.42	**75.26**±0.38	66.96±0.34	80.16±0.44	77.06±0.56
5	76.87±0.34	88.50±0.24	88.82±0.44	83.20±0.72	74.14±0.25	66.52±0.64	78.55±0.74	76.33±0.29
6	72.03±0.62	81.77±0.27	83.19±0.48	**85.27**±0.57	74.62±0.38	65.90±0.83	78.27±0.74	75.27±0.49
7	74.75±0.46	86.64±0.24	81.87±1.29	82.51±0.40	74.39±0.54	63.96±0.43	77.90±0.65	75.94±0.36
8	**84.14**±0.53	84.06±0.34	88.73±0.63	82.55±0.43	74.82±0.37	**67.20**±0.68	78.10±1.16	75.03±0.29
9	75.11±0.39	86.43±0.46	89.65±0.57	78.95±0.64	74.62±0.61	66.10±0.56	76.27±0.97	70.05±0.62

boNet (Chen et al., 2022). We report in Table 3 the mean F1 score (%) and standard deviation from three independent runs.

Results Our HKN achieves statistically significant improvement over other methods on five out of six benchmarks (Texas, Wisconsin, Chameleon, Squirrel, Actor) which show strong hyperbolicity. On Cornell, its performance is also very close to the best result. In particular, in all these benchmarks, HKN excels the other hyperbolic methods by a large margin. On the other hand, general graph neural networks are better than hyperbolic neural networks in Cora and PubMed, which only show very weak hyperbolicity. In these non-hyperbolic datasets, our method is still comparable to other hyperbolic neural networks.

Analysis on number of kernels We also study how the performance of HKN changes with different numbers of kernels. Fixing the other hyperparameters, we record the performance of HKN using $K = \{2, 3, 4, 5, 6, 7, 8, 9\}$ kernel points. The results are displayed in Table 4 and plotted in Figure 4.

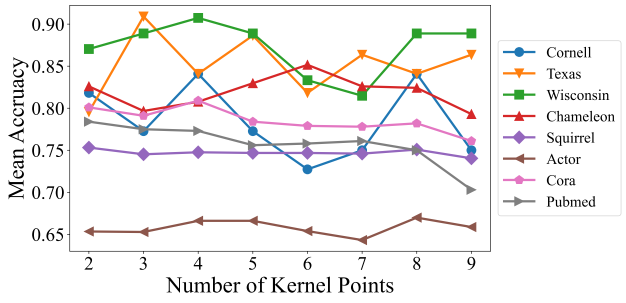

Figure 4: Node classification results with different numbers of kernel points.

We notice that for datasets with smaller hyperbolicity (the first six columns), the best performance is achieved with a moderate K, which accommodates the tradeoff between expressivity and overfitting. However, for Cora and PubMed, which are less hyperbolic, increasing the number of kernels will not enhance the performance and the best performance is already achieved with $K = 2$ kernels. We conclude that the kernel points in HKConv are effective in extracting hyperbolic features, while non-hyperbolic data

Table 5: 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' 17			Distance		Depth	
	d=64	d=64	d=128	d=256	UUAS	Dspr.	Root%	Nspr.
ConvSeq2Seq (Gehring et al., 2017)	23.6	14.9	20.0	21.8	-	-	-	-
Transformer (Ott et al., 2018)	23.0	17.0	21.7	25.1	0.36	0.3	12	0.88
HNN++ (Shimizu et al., 2021)	22.0	17.0	19.4	21.8	-	-	-	-
HATT (Gulcehre et al., 2019)	23.7	18.8	22.5	25.5	0.5	0.64	49	0.88
HyboNet (Chen et al., 2022)	25.9	19.7	23.3	26.2	0.59	0.7	64	0.92
HKN (Ours)	**27.3**	**20.1**	**25.6**	**29.1**	**0.62**	**0.74**	**72**	**0.94**

may not benefit from more kernel points.

4.4 Machine Translation and Dependency Tree Probing

In this task, we consider language data that show hierarchical structures. Machine translation translates one language to another and is a sequence-to-sequence modeling task. Dependency tree probing then analyzes the obtained model.

Settings For fair comparison, we use the network architecture of the hyperbolic transformer model in Chen et al. (2022), except that we apply HKConv for the aggregation in the following manner. Given the query set $\mathcal{Q} = \{\boldsymbol{q}_1, \ldots, \boldsymbol{q}_{|\mathcal{Q}|}\}$, the key set $\mathcal{K} = \{\boldsymbol{k}_1, \ldots, \boldsymbol{k}_{|\mathcal{K}|}\}$, and the value set $\mathcal{V} = \{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_{|\mathcal{V}|}\}$, where $|\mathcal{K}| = |\mathcal{V}|$, We implement HKConv for each $i = 1, \cdots, |\mathcal{V}|$ as follows:

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

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 (6) and (7), presented for clear notation. The last equation is (8) where the weights are determined by the query set and the key set. Overall, HKConv performs a further transformation of $\boldsymbol{v}_i$ that utilizes the local geometric embedding.

Following Chen et al. (2022), we take two machine translation datasets: IWSLT'14 and WMT'17 English-German. Dependency tree probing is tested on the former.

Baselines We compare with ConvSeq2Seq (Gehring et al., 2017) and Transformer (Ott et al., 2018), as well as hyperbolic methods including HNN++ (Shimizu et al., 2021), HATT (Gulcehre et al., 2019) and HyboNet (Chen et al., 2022). Note particularly that HNN++ uses their hyperbolic convolution which, while not applicable in the graph classification tasks due to the heterogeneous graph structures, naturally works in this task. We report the results in Table 5, where the benchmark results are taken from Shimizu et al. (2021) and Chen et al. (2022).

Results HKN consistently excels all the baseline methods in both tasks and in all the metrics. Since we have intentionally kept the same attention regime as HyboNet but replacing aggregation with HKConv, our better performance does not result from 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.

5 Related Work

Hyperbolic Neural Networks The earliest hyperbolic network was HNN (Ganea et al., 2018), where linear layers and recurrent layers were defined according to the Poincaré ball model. HNN was recently generalized to HNN++ (Shimizu et al., 2021), which introduced concatenation, convolution and attention mechanisms in the Poincaré ball. Nickel and Kiela (2018) pointed out the numerical instability of operations in the Poincaré ball and promoted the use of the Lorentz model. More recently, Chen et al. (2022) 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. (2022) 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. (2019) 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. (2021).

Non-Euclidean Convolution Recently, CNNs have been generalized to different manifolds. Geometric convolutional models include the spherical CNN (Cohen et al., 2018), the mesh CNN (Hanocka et al., 2019) and the icosahedral CNN (Cohen et al., 2019). 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 (Kipf and Welling, 2017). 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 (Yang et al., 2022). Many hyperbolic networks (Liu et al., 2019; Chami et al., 2019; Bachmann et al., 2020; Dai et al., 2021; Zhang et al., 2021) 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 them, our HKConv transforms relative features in each neighborhood and therefore encodes more local geometric information when applied to graph data.

6 Conclusion, Societal Impact 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 graph and non-graph tasks. We have particularly showed the advantage of the important components in our convolution.

One positive societal impact of our proposed model is that, as introduced in Section 1, hyperbolic neural networks are effective tools for many practical applications. In particular, our approach is sufficiently generic and can be applied to many different types of data that have hierarchical structures. On the other hand, it might have negative societal impact when a hyperbolic neural network is applied to social networks: its success might encourage collection of privacy behavior data and enable discriminatory pricing.

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. To address this issue, a future working direction is to explore sampling regimes for reducing model complexity.

Acknowledgments

References

Ahmad, O. and Lecue, F. (2022). 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.

Anderson, J. W. (2006). *Hyperbolic geometry*. Springer Science & Business Media.

Atigh, M. G., Schoep, J., Acar, E., van Noord, N., and Mettes, P. (2022). Hyperbolic image segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4453–4462.

Atzmon, M., Maron, H., and Lipman, Y. (2018). Point convolutional neural networks by extension operators. *ACM Trans. Graph.*, 37(4).

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

Baek, J., Kang, M., and Hwang, S. J. (2021). Accurate learning of graph representations with graph multiset pooling. In *International Conference on Learning Representations*.

Becigneul, G. and Ganea, O.-E. (2019). Riemannian adaptive optimization methods. In *International Conference on Learning Representations*.

Borgwardt, K. M., Ong, C. S., Schönauer, S., Vishwanathan, S., Smola, A. J., and Kriegel, H.-P. (2005). Protein function prediction via graph kernels. *Bioinformatics*, 21(suppl_1):i47–i56.

Cannon, J. W., Floyd, W. J., Kenyon, R., Parry, W. R., et al. (1997). Hyperbolic geometry. *Flavors of geometry*, 31(59-115):2.

Chami, I., Wolf, A., Juan, D.-C., Sala, F., Ravi, S., and Ré, C. (2020). Low-dimensional hyperbolic knowledge graph embeddings. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6901–6914, Online. Association for Computational Linguistics.

Chami, I., Ying, Z., Ré, C., and Leskovec, J. (2019). Hyperbolic graph convolutional neural networks. *Advances in neural information processing systems*, 32:4868–4879.

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

Chen, W., Han, X., Lin, Y., Zhao, H., Liu, Z., Li, P., Sun, M., and Zhou, J. (2022). Fully hyperbolic neural networks. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5672–5686.

Cohen, T., Weiler, M., Kicanaoglu, B., and Welling, M. (2019). Gauge equivariant convolutional networks and the icosahedral cnn. In *International conference on Machine learning*, pages 1321–1330. PMLR.

Cohen, T. S., Geiger, M., Köhler, J., and Welling, M. (2018). Spherical CNNs. In *International Conference on Learning Representations*.

Craven, M., DiPasquo, D., Freitag, D., McCallum, A., Mitchell, T., Nigam, K., and Slattery, S. (2000). Learning to construct knowledge bases from the world wide web. *Artificial intelligence*, 118(1-2):69–113.

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

Fan, X., Yang, C.-H., and Vemuri, B. C. (2022). 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.

Ganea, O., Bécigneul, G., and Hofmann, T. (2018). Hyperbolic neural networks. *Advances in neural information processing systems*, 31:5345–5355.

Gehring, J., Auli, M., Grangier, D., Yarats, D., and Dauphin, Y. N. (2017). Convolutional sequence to sequence learning. In *International conference on machine learning*, pages 1243–1252. PMLR.

Gulcehre, C., Denil, M., Malinowski, M., Razavi, A., Pascanu, R., Hermann, K. M., Battaglia, P., Bapst, V., Raposo, D., Santoro, A., and de Freitas, N. (2019). Hyperbolic attention networks. In *International Conference on Learning Representations*.

Hamilton, W., Ying, Z., and Leskovec, J. (2017). Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30.

Hanocka, R., Hertz, A., Fish, N., Giryes, R., Fleishman, S., and Cohen-Or, D. (2019). Meshcnn: a network with an edge. *ACM Transactions on Graphics (TOG)*, 38(4):1–12.

Helma, C., King, R. D., Kramer, S., and Srinivasan, A. (2001). The predictive toxicology challenge 2000–2001. *Bioinformatics*, 17(1):107–108.

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

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

Law, M., Liao, R., Snell, J., and Zemel, R. (2019). Lorentzian distance learning for hyperbolic representations. In *International Conference on Machine Learning*, pages 3672–3681. PMLR.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *nature*, 521(7553):436–444.

Li, X., Zhu, R., Cheng, Y., Shan, C., Luo, S., Li, D., and Qian, W. (2022). Finding global homophily in graph neural networks when meeting heterophily. In *International Conference on Machine Learning*, pages 13242–13256. PMLR.

Lim, D., Hohne, F., Li, X., Huang, S. L., Gupta, V., Bhalerao, O., and Lim, S. N. (2021). Large scale learning on non-homophilous graphs: New benchmarks and strong simple methods. *Advances in Neural Information Processing Systems*, 34:20887–20902.

Lin, Z.-H., Huang, S.-Y., and Wang, Y.-C. F. (2020). 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.

Liu, Q., Nickel, M., and Kiela, D. (2019). Hyperbolic graph neural networks. *Advances in Neural Information Processing Systems*, 32:8230–8241.

Nagano, Y., Yamaguchi, S., Fujita, Y., and Koyama, M. (2019). A wrapped normal distribution on hyperbolic space for gradient-based learning. In *International Conference on Machine Learning*, pages 4693–4702. PMLR.

Namata, G., London, B., Getoor, L., Huang, B., and Edu, U. (2012). Query-driven active surveying for collective classification. In *10th International Workshop on Mining and Learning with Graphs*, volume 8, page 1.

Nickel, M. and Kiela, D. (2018). Learning continuous hierarchies in the lorentz model of hyperbolic geometry. In *International Conference on Machine Learning*, pages 3779–3788. PMLR.

Ott, M., Edunov, S., Grangier, D., and Auli, M. (2018). Scaling neural machine translation. In *Proceedings of the Third Conference on Machine Translation, Volume 1: Research Papers*, pages 1–9, Belgium, Brussels. Association for Computational Linguistics.

Pei, H., Wei, B., Chang, K. C.-C., Lei, Y., and Yang, B. (2019). Geom-gcn: Geometric graph convolutional networks. In *International Conference on Learning Representations*.

Peng, W., Varanka, T., Mostafa, A., Shi, H., and Zhao, G. (2021). Hyperbolic deep neural networks: A survey. *IEEE Transactions on Pattern Analysis & Machine Intelligence*.

Rozemberczki, B., Allen, C., and Sarkar, R. (2021). Multi-scale attributed node embedding. *Journal of Complex Networks*, 9(2):cnab014.

Schomburg, I., Chang, A., Ebeling, C., Gremse, M., Heldt, C., Huhn, G., and Schomburg, D. (2004). Brenda, the enzyme database: updates and major new developments. *Nucleic acids research*, 32(suppl_1):D431–D433.

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.

Shimizu, R., Mukuta, Y., and Harada, T. (2021). Hyperbolic neural networks++. In *International Conference on Learning Representations*.

Sonthalia, R. and Gilbert, A. (2020). Tree! i am no tree! i am a low dimensional hyperbolic embedding. *Advances in Neural Information Processing Systems*, 33:845–856.

Suresh, S., Budde, V., Neville, J., Li, P., and Ma, J. (2021). Breaking the limit of graph neural networks by improving the assortativity of graphs with local mixing patterns. In *KDD*.

Tang, J., Sun, J., Wang, C., and Yang, Z. (2009). 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.

Thomas, H., Qi, C. R., Deschaud, J.-E., Marcotegui, B., Goulette, F., and Guibas, L. J. (2019). Kpconv: Flexible and deformable convolution for point clouds. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 6411–6420.

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

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. (2019). How powerful are graph neural networks? In *International Conference on Learning Representations*.

Yanardag, P. and Vishwanathan, S. (2015). Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1365–1374.

Yang, M., Zhou, M., Li, Z., Liu, J., Pan, L., Xiong, H., and King, I. (2022). Hyperbolic graph neural networks: A review of methods and applications.

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

Appendix

The Appendix contain the following contents. §A reviews common hyperbolic layers used in the paper, §B proves Theorem 2 of the main text, §C shows the tables of best hyperparameters obtained from validation.

A Preliminaries on Hyperbolic Neural Operations

Hyperbolic Linear Layers Cascading many exponential and logarithmic maps may lead to numerical instability (Chami et al., 2019; Chen et al., 2022). Therefore, when designing linear layers in complex hyperbolic neural networks, it is beneficial to take a "fully hyperbolic" approach, originally proposed by Chen et al. (2022). A fully hyperbolic linear layer applies a linear transformation to the spatial component of the input, and fills the time component so that the whole vector lies in the hyperboloid. Specifically, it maps $\boldsymbol{x} \in \mathbb{L}^m$ to $\begin{bmatrix} \sqrt{\|\boldsymbol{W}\boldsymbol{x}\|^2 - 1/\kappa} \\ \boldsymbol{W}\boldsymbol{x} \end{bmatrix}$, where $\boldsymbol{W} \in \mathbb{R}^{n\times(m+1)}$. With additional activation, bias and normalization, a general expressive linear layer transforms an input $\boldsymbol{x} \in \mathbb{L}^m$ to

$$\boldsymbol{y} = \mathrm{HLinear}(\boldsymbol{x}) = \begin{bmatrix} \sqrt{\|h(\boldsymbol{x})\|^2 - 1/\kappa} \\ h(\boldsymbol{x}) \end{bmatrix} \in \mathbb{L}^n.$$

Here,

$$h(\boldsymbol{x}) = \frac{\lambda\sigma\left(\boldsymbol{v}^\top \boldsymbol{x} + b'\right)}{\|\boldsymbol{W}\tau(\boldsymbol{x}) + \boldsymbol{b}\|}(\boldsymbol{W}\tau(\boldsymbol{x}) + \boldsymbol{b}),$$

where $\boldsymbol{v} \in \mathbb{R}^{n+1}$ and $\boldsymbol{W} \in \mathbb{R}^{n\times(m+1)}$ are trainable weights, $\boldsymbol{b} \in \mathbb{R}^n$ 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 $h(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{b})$ to emphasize its dependence on $\boldsymbol{W}$ and $\boldsymbol{b}$.

Hyperbolic Centroid The notion of a centroid is extended to $\mathbb{L}^n$ by Law et al. (2019), defined to be the point $\boldsymbol{\mu}^*$ that minimizes a weighted sum of squared hyperbolic distances:

$$\boldsymbol{\mu}^* = \underset{\boldsymbol{\mu}\in\mathbb{L}^n}{\arg\min} \sum_{i=1}^{N} \nu_i d_{\mathcal{L}}^2(\boldsymbol{x}_i, \boldsymbol{\mu}),$$

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

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

Hyperbolic Distance Layer The hyperbolic distance layer (Liu et al., 2019) is a numerically stable way of output Euclidean features. It maps points from $\mathbb{L}^n$ to $\mathbb{R}^m$. Given an input $\boldsymbol{x} \in \mathbb{L}^n$, it first initializes m trainable centroids $\{\boldsymbol{c}_i\}_{i=1}^m \subset \mathbb{L}^n$, then produces a vector of distances

$$\boldsymbol{y} = \mathrm{HCDist}_{n,m}(\boldsymbol{x}) = [d_{\mathcal{L}}(\boldsymbol{x}, \boldsymbol{c}_1) \cdots d_{\mathcal{L}}(\boldsymbol{x}, \boldsymbol{c}_m)]^\top,$$

Hyperbolic Embedding A simple approach to embedding a vector $\boldsymbol{x} \in \mathbb{R}^n$ in $\mathbb{L}^n$, first proposed by Nagano et al. (2019), is as follows. First, a zero padding is added to the first coordinate of $\boldsymbol{x}$, as if $\boldsymbol{x}$ is the spatial component of $\hat{\boldsymbol{x}} = \begin{bmatrix} 0 \\ \boldsymbol{x} \end{bmatrix} \in \mathcal{T}_{\boldsymbol{o}}\mathbb{L}^n$, where $\mathcal{T}_{\boldsymbol{o}}\mathbb{L}^n$ is the tangent space of $\mathbb{L}^n$ at the hyperbolic origin $\boldsymbol{o}$. Then, the embedding is produced by taking the exponential map of $\hat{\boldsymbol{x}}$ at $\boldsymbol{o}$.

Wrapped Normal Distribution The wrapped normal distribution (Nagano et al., 2019) is a hyperbolic distribution that resembles the normal distribution in Euclidean space. Given $\boldsymbol{\mu} \in \mathbb{L}^n$ and $\boldsymbol{\Sigma} \in \mathbb{R}^{n\times n}$, to sample $\boldsymbol{z} \in \mathbb{L}^n$ from the wrapped normal distribution $\mathcal{G}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, one first samples a vector from the Euclidean normal distribution $\mathcal{N}(\boldsymbol{0}, \boldsymbol{\Sigma})$ and applies an hyperbolic embedding to obtain $\boldsymbol{x} \in \mathbb{L}^n$. Then, one applies parallel transport and produces $\boldsymbol{z} = \exp_{\boldsymbol{\mu}}\left(\mathrm{PT}_{\boldsymbol{o}\to\boldsymbol{\mu}}\left(\log_{\boldsymbol{o}}(\boldsymbol{x})\right)\right) \in \mathbb{L}^n$. We used the wrapped normal distribution as an alternative way of generating the kernel points. However, our ablation study showed its deteriorate performance.

B Proof of Theorem 2

Proof. For any $\boldsymbol{x}_i \in \mathcal{N}(\boldsymbol{x})$, according to the definition in (4),

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

Consequently, the input of $\mathrm{HLinear}_k$, $k = 1, \cdots, K$, in (6) is invariant under the operator $\mathrm{T}_{\boldsymbol{x}\to\boldsymbol{y}}$. This implies that $\boldsymbol{x}'$ in (8) is also invariant under $\mathrm{T}_{\boldsymbol{x}\to\boldsymbol{y}}$, since it only depends on $\{\boldsymbol{x}_i \ominus \boldsymbol{x} : \boldsymbol{x}_i \in \mathcal{N}(\boldsymbol{x})\}$ and the fixed kernel $\{\tilde{\boldsymbol{x}}_k\}_{k=1}^{K}$. We have thus proved (9). $\square$

C Hyperparameters

We list the hyperparameters corresponding to the best validated results. The validation settings have been described in the main experimental parts.

C.1 Graph Classification

We list the hyperparameters used in the graph classification task in Table 6. These correspond to §4.1-4.2 of the main text.

Table 6: Hyperparameters for graph classification

	PTC	ENZYMES	PROTEIN	IMDB-B	IMDB-M
# of Kernels	4	4	4	4	7
Learning Rate	0.005	0.005	0.005	0.005	0.005
Weight Decay	0	0	0	0	0
Dropout	0	0.05	0	0	0
# of Layers	6	4	6	6	6
Batch Size	32	32	32	32	32
Patience	1000	1000	1000	1000	1000

C.2 Machine Translation and Dependency Tree Probing

We list the hyperparameters used in the machine translation and dependency tree probing tasks in Tables 7 and 8, respectively. These correspond to §4.3 of the main text.

Table 7: Hyper-parameters for machine translation.

Hyper-parameter	IWSLT'14	WMT'17
# of Kernels	5	5
# of GPUs	4	4
Embedding Dimension	64	64
Feed-forward Dimension	256	256
Batch Type	Token	Token
Batch Size	3300	4096
Gradient Accumulation Steps	3	3
Training Steps	30000	200000
Dropout	0.0	0.0
Attention Dropout	0.1	0.1
Max Gradient Norm	0.5	0.5
Warmup Steps	2000	6000
Decay Method	noam	noam
Label Smoothing	0.1	0.1
# of Layers	6	6
# of Heads	6	6
Learning Rate	2	2
Adam Beta2	0.998	0.998

Table 8: Hyper-parameters for dependency tree probing.

Hyper-parameter	Euclidean	HyboNet	HKNet
# of Kernels	5	5	5
Learning Rate	5e-4	5e-4	1e-4
Best Layer	0	4	3
Warmup Steps	1000	1000	1000
Dropout	0.0	0.0	0.0
Batch Size	32	32	32
Epochs	5	5	5

C.3 Node Classification

We list the hyperparameters used in the node classification tasks in Table 9. These correspond to §4.3 of the supplementary material.

Table 9: Hyperparameters for node classification and link prediction

	Cornell	Texas	Wisconsin	Chameleon	Squirrel	Actor	Cora	PubMed
# of Kernels	8	3	4	6	4	8	2	2
Learning Rate	0.0005	0.0005	0.0005	0.005	0.005	0.005	0.0005	0.0005
Weight Decay	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001
Dropout	0	0.1	0	0.5	0.2	0.05	0.9	0.9
Layers	2	2	2	2	3	4	2	2
Weight Clip	None	None	None	None	None	None	0.5	0.5