Hyperbolic Neural Networks

Attila-Balázs Kis

University of Stuttgart, Stuttgart, Germany

Abstract. Neural networks have been successful in solving a multitude of classification and regression tasks in different fields. A relevant condition to their success is the amount of data these networks use to learn spatial and/or temporal connections, and through the generalization power manage to achieve impressive results on real-world problems. The Euclidean geometry is used as the state-of-the-art representation and computation in the neural networks. In the subset of problems in which data is represented as graphs and trees, Euclidean geometry is not capable to comprehensively encode the data such that neural networks can take advantage of it. Searching for a geometry which has at its core functions and operations which can be parameterized to represent tree and graph-like data with small distortion, we present a *notable alternative*: hyperbolic geometry. In this article we will break down the building blocks of neural networks and by using the power of hyperbolic geometry, construct appropriate correspondents to Euclidean variants, such as *basic operations, activation functions and network layers*. We will present how utilizing the representational advantages of hyperbolic networks in capturing hierarchy and low distortion in embedding the data, hyperbolic networks outperformed their Euclidean baselines in multiple thorough experiments presented by the literature.
Furthermore, we will analyze how hyperbolic neural networks can learn *graph embeddings* in a more suitable way than their Euclidean counterparts, following the experiments presented in the literature.

Keywords: neural networks · hyperbolic geometry · Riemannian geometry · Poincaré model · graph embeddings

1 Introduction

Neural networks are a subset of machine learning, and are a powerful tool for both classification and regressions tasks, vastly outperforming other solutions. Applied successfully in many fields such as computer vision, signal processing, it proved to be a solution which, with the appropriate adaptations for the task on hand, can yield solution to many tasks.
Neural networks, and the majority of machine learning concepts and implementations are performed in Euclidean geometry, for the simple and logical reason of being intuitive to humans. Following convenient factors are the vectorial representation which yield simple distance and inner product formulas together with basic operations, all being necessary in building solutions solving regression and classification problems.

The subset real world problems of our interest in this article boils down to a very interesting structural difference compared to the usual input data in neural networks, this being *hierarchy*. In real life, hierarchical data became, with the increase of the data flow, more and more present in the data science, machine learning and big data fields. Euclidean geometry offers a linear growth, while following the above-mentioned structure, the number of data samples - named, conveniently aligning with tree and graph representation, *nodes* - grows exponentially. Using Euclidean representation, cluttering while representing data is unavoidable, and this issue grows to the point where Euclidean geometry doesn't offer a suitable representation without a notable distortion of the underlying reality in the data.
The introduction of hyperbolic geometry in the field of neural networks can be argumented using Fig. 1. in why the new representation can handle better tree-like data structures. Although the explanation seems straight-forward, to be able to use the hyperbolic geometry, suitable tools have to be created using their Euclidean counterparts and applying the appropriate changes. We need to change:

- operators to hyperbolic ones, i.e. use gyrovector space operations;
- neural network layers to manifold versions;
- point-wise non-linear activation functions to their hyperbolic counterparts.

in order to be able to apply the newly approached non-Euclidean domain *of constant negative curvature* - i.e. a space with hyperbolic properties - to our tasks of interest.

In the following sections, we will discuss the theoretical grounds needed to switch domains. Next, will apply the switch on important blocks of neural networks and gyrovector operations. As an in-depth study, we will analyze how using Euclidean inputs and multiple hyperbolic layers the data can be transformed in such a way that the *negative curvature* of the data can be learned, thus the embedding of the data in hyperbolic geometry can be optimized to yield the best representation for e.g. classification problems.

We will point out the advantages over Euclidean geometry, and possible places where there is room for improvement in the taken approach, following [1]. Overall, we will see the way that the hyperbolic geometry can capture, thus benefit of the inherent hierarchical structure of data.

2 Related work

A comprehensive and very well structured approach to the theoretical grounds of the subject at hand is presented in [1] and [2], where together with a methodological presentation of the underlying mathematics needed to create hyperbolic neural networks, an interesting set of experiments is conducted to numerically evaluate the superiority of the new networks over the well-known Euclidean versions.

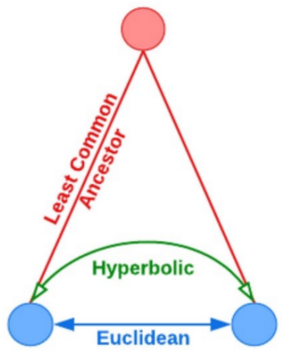

Fig. 1. Visual representation of the difference measure between using Euclidean and hyperbolic geometry in tree-like structures

In [2], the authors discuss the approach developed to handle well data resembling tree structure in learning graph embeddings so that classification tasks can be performed by using a representation with less distortion. The measure of distortion is measured by the performance of the model which "learned" the data, by well-known performance metrics in neural networks. Deep graph convolutional networks with multiple layers transform the data passing through the hidden layers. The network can learn in such a way the best curvature of the hyperbolic space in which to embed the data, so that *distortion* is kept to the minimum.

Sources [3] and [4] give very good additional theoretical grounds in order to understand better the subjects of *neural networks operating in hyperbolic space* and *embedding graph data* in the most efficient way.

Throughout further chapters other sources such as blog posts/web links will be cited, used for linking additional explanations to particular paragraphs, such as [5] and [6].

3 Theoretical background

In the following chapter we will tackle the theory behind concepts needed to construct and use hyperbolic neural networks. The amount of details will be correlated to the target of the paper, focusing on giving a comprehensive overview of the subject. Proofs and further reading will be linked to the sources of citations, so that the reader can follow up thoroughly on the sections presented.

3.1 Differential geometry

We will define important terms used throughout the subject to generalize the Euclidean geometry of neural networks so that they can be used in hyperbolic geometry. For an elaborate and more detailed description of the theoretical terms we refer you to [1] and Appendix A of [2] .

Manifold – The manifold is the generalization of the notion of 2D surfaces in high dimensions. Formally defined, a multi-dimensional manifold M is a topological space that *locally resembles the topological space* $\mathbb{R}^d$. This means that for any point on M there exists a bijection containing also *an inverse* between the neighborhood of that point and $\mathbb{R}^d$.

Tangent space – Thinking of a manifold M, at a point x there exists the so-called *tangent space* $\mathrm{T}_x M$ which locally approximates M around x.
This approximation will be highly useful when converting neural network components to their hyperbolic components, since there are operations which cannot be performed in hyperbolic space, and in Euclidean space they can be performed with ease. By moving a point from the manifold to its tangent space, performing an operation and then moving it back to the manifold this issue can be averted quite easily while not violating points in the hyperbolic space.

Riemannian manifold – Is a pair of manifold equipped with a Riemannian metric $(M,\ g)$. $g = (g_x)_{x \in M}$ on M is a collection of inner-products varying smoothly with x. The metric g is used to induce *global distances* by integrating the length of a shortest path between two points.
Importantly, a *smooth path* between two points having a *minimal length* on a manifold is called a *geodesic*, and it is the generalization of the distance, i.e. the straight line in Euclidean geometry.

We further define two relevant operations between the manifold M and it's tangent space $\mathrm{T}_x M$, namely moving a point between the two spaces. The *logarithmic* map, log_x gives a way to project a point x located on the manifold to it's tangent space, and the *exponential* map exp_x gives a way to project back from the tangent space to the manifold.
Additionally we specify the *parallel transport*, which is defined as being the way to move tangent vectors along geodesics and be able to connect the resulting point of the operation to another tangent space. This operation is going to be used as the *addition of bias in hyperbolic neural networks.*

3.2 Hyperbolic space

A comprehensive definition of the hyperbolic space is: an n-dimensional hyperbolic space is an *n-dimensional complete Riemannian manifold* with a *constant negative curvature.* Fig. 2. [1] (a) - (c) is a comparison of the three geometries defined by curvature. Relevant to note the sum of angles in an arbitrary triangle in the hyperbolic geometry (a) do not add up to the usual 180, since the space is "deformed" compared to the intuitive Euclidean space.

[1] Image taken from [7]

Furthermore looking at (d)-(f) on the same figure we can note how on graphs with negative curvature (d), one can very easily create multiple *partially ordered subsets*. These subsets are hierarchically connected together, and looking e.g. at one subset containing nodes {0, 2, 10, 11, 12} a tree-like structure is clearly visible. [3] defines the hyperbolic space in the following way: it can be intuitively understood as a continuous tree: the volume of the ball grows *exponentially with its radius*.

Defining formally the hyperbolic metric space, one can see it as a metric space satisfying certain metric relations between points in the space. These relations can be quantified using a *non-negative real number* δ, also called the Gromov's hyperbolicity. δ is a metric that tells us how tree-like the data is.
The lower δ is, the more hyperbolic is the graph dataset, and $\delta = 0$ for trees [2] chapter 5.1.

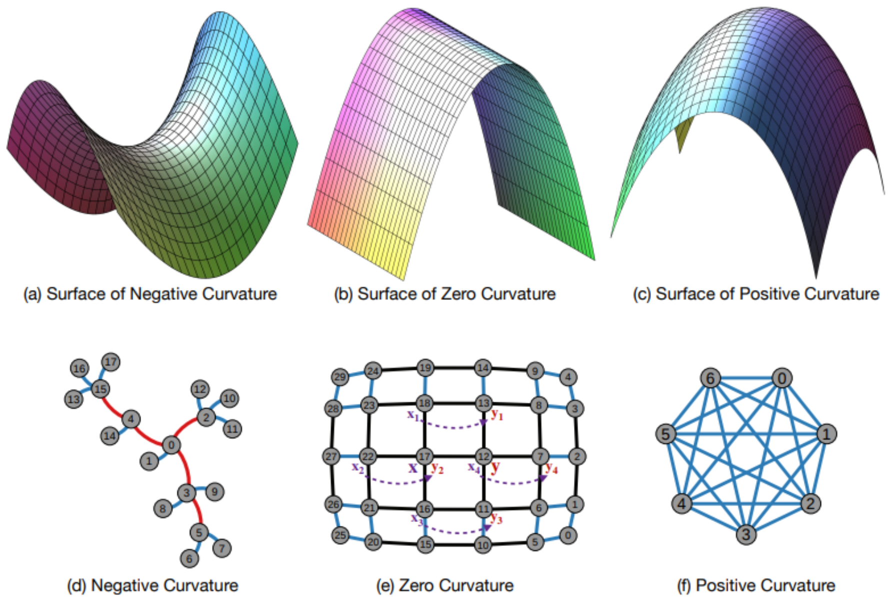
(a) Surface of Negative Curvature
(b) Surface of Zero Curvature
(c) Surface of Positive Curvature
(d) Negative Curvature
(e) Zero Curvature
(f) Positive Curvature

Fig. 2. Comparison of surfaces and graphs, based on surface curvature

Poincaré ball – The hyperbolic space has multiple models with which can work with, and one of the most expressive ones is the Poincaré ball. One of the most outstanding advantages of the aforementioned ball is the ease of visualization of the embeddings in the ball. This is a relevant point to look after when interested in powerful graph embeddings, which we will discuss in a later section in detail. Thus the Poincaré ball is the choice for the hyperbolic space model. In the context of gyrovector spaces (presented in section 3.3), the Poincaré ball has constant curvature (c) *equal to 1.*

The Poincaré ball model is utilizing the manifold $\mathbb{D}^n = \{x \in \mathbb{R}^n : ||x|| < 1\}$ i.e. unit ball, and it is defined as $(D^n, g^{\mathbb{R}})$. The Riemannian metric is defined in Eq. 1.

$$g_x^{\mathbb{D}} = \lambda_x^2 \mathbb{I}_n,\ where\ \lambda_x = \frac{2}{1 - ||x||^2} \tag{1}$$

Furthermore [1] in chapter 2.2 defines the *induced distance* between two points x, y $\in \mathbb{D}^n$ and the *angle between* two vectors u, v $\in \mathrm{T}_x\mathbb{D}^n \setminus\{0\}$. Both functions are going to be used in connecting gyrovector spaces and Riemannian geometry of the Poincaré ball.
Overall, the Poincaré model is good to *capture hierarchies and visualization.*

3.3 Gyrovector spaces

The Euclidean space, since it is a vectorial space, the operations performed on vectors are inherited from the structure of the space. Gyrovector spaces give a simplistic and well correlated construction of much needed operations in neural networks to be done in hyperbolic space.

Möbius addition The operation is defined in Euclidean space as the translation of vector x by vector y. Two values x, y $\in \mathbb{D}_c^n$ is defined in Eq. 2. Variable c denotes the curvature. If not said differently, c is assumed to be the constant negative curvature of 1.

$$x \oplus_c y = \frac{(1 + 2c\langle x, y\rangle + c||x||^2)x + (1 - c||x||^2)y}{1 + 2c\langle x, y\rangle + c^2||x||^2||y||^2} \tag{2}$$

The operation $\oplus_c$ is *not commutative nor associative.*
Additionally the *Möbius subtraction* is naturally defines in Eq. 3.

$$x \ominus_c y = x \oplus_c (-y) \tag{3}$$

An interesting note is that if c = 0, then the Möbius addition and subtraction give back the Euclidean space addition and subtraction as we can see in Eq. 4.

$$\begin{aligned} x \oplus_0 y &= x + y \\ x \ominus_0 y &= x - y \end{aligned} \tag{4}$$

Möbius scalar-vector multiplication Multiplying x $\in \mathbb{D}_c^n \setminus\{0\}$ with a constant $a \in \mathbb{R}$ is computed using Eq. 5.

$$a \otimes_c x = \frac{1}{\sqrt{c}} tanh(atanh^{-1}(\sqrt{c}||x||)\frac{x}{||x||} \tag{5}$$

Similarly as in the case of the addition, if c = 0, we recover the Euclidean scalar multiplication. Furthermore the Möbius scalar multiplication suffices basic properties of the Euclidean counterpart, which is going to be useful in the followings.

Distance A generalized version of the distance on $(\mathbb{D}_c^n, g^c)$ is given by Eq. 6.

$$d_c(x, y) = \frac{2}{\sqrt{c}} tanh^{-1}(\sqrt{c}||-x \oplus_c y||) \tag{6}$$

Analogously to the previous two subchapters, if fixing c = 0, we get back to $d_c(x, y) = 2||x - y||$, which is exactly the Euclidean distance. Even more interestingly, fixing c = 1, we get back the distance equation defined in [1] chapter 2.2, the distance in the Poincaré ball model.

3.4 Connecting Gyrovector spaces to the Poincaré ball

The subsection follows the logical structure of [1] chapter 2.4, and for proofs and more thorough detailing we refer the user to the aforementioned paper.

We are particularly interested in describing the *exponential and logarithmic maps*. Defining the two maps accordingly will give us an easy way to define operations such as scalar and matrix multiplication and parallel transport, since the tangent space is Euclidean, and moving points to the tangent space, performing operation and moving back to the manifold will result in exactly the hyperbolic operations we are interested in.

Exponential and logarithmic map – For any point x $\in \mathbb{D}_c^n$ and a given v $\neq 0$ the exponential map can be expressed using Eq. 7. Analogously the logarithmic map is expressed using Eq. 8.

$$\exp_x^c : \mathrm{T}_c\mathbb{D}_c^n \to \mathbb{D}_c^n, \quad \exp_x^c(v) = x \oplus_c (tanh(\sqrt{c}\frac{\lambda^c||v||}{2})\frac{v}{\sqrt{c}||v||}) \tag{7}$$

$$\log_x^c : \mathbb{D}_c^n \to \mathrm{T}_c\mathbb{D}_c^n, \quad \log_x^c(v) = \frac{2}{\sqrt{c}\lambda_x^c} tanh^{-1}(\sqrt{c}||-x \oplus_c y||)\frac{-x \oplus_c y}{||-x \oplus_c y||} \tag{8}$$

Interesting mention is that fixing c = 0, $\exp_x^0(v) = x + v$ and analogously $\log_x^x(v) = x - v$, quickly observing the Euclidean versions of the exponential and logarithmic maps on a surface without curvature.

Redefining Möbius multiplication – Using the logarithmic and exponential maps at x = 0 $\in \mathbb{D}_c^n$, the Möbius multiplication $r \otimes_c x$ can be defined quite simply in the following way, furthermore presented in Fig. 3[2].

$$r \otimes_c x = \exp_0^c(r\log_0^c) \tag{9}$$

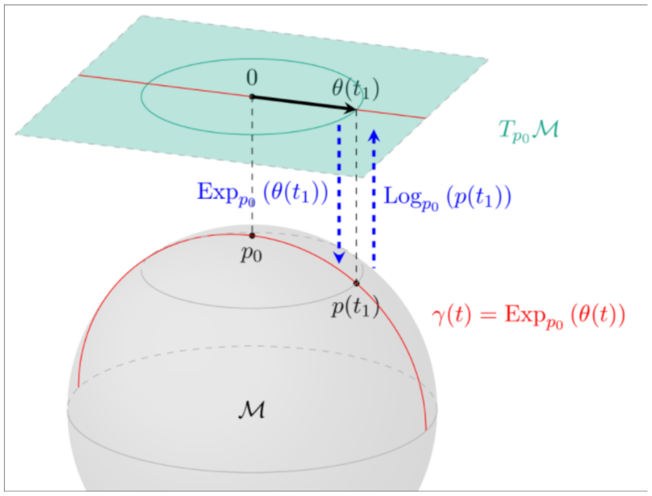
0
$\theta(t_1)$
$T_{p_0}\mathcal{M}$
$\mathrm{Exp}_{p_0}(\theta(t_1))$
$\mathrm{Log}_{p_0}(p(t_1))$
p_0
$p(t_1)$
$\gamma(t) = \mathrm{Exp}_{p_0}(\theta(t))$
$\mathcal{M}$

Fig. 3. Möbius scalar multiplication, $p_0 = x$ and $\theta(t_1) = rm$ in Euclidean space. $p(t_1) = r \otimes_c x$.

This reinterpretation together with multiplication rules help us in defining an important operation, namely the *matrix-vector product*, which thus follows:

$$\begin{bmatrix} M_1 & M_2 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = M_1 \otimes_c v_1 \oplus M_2 \otimes_c v_2 \tag{10}$$

Keeping in mind the natural order of operations, the matrix-vector product is yet another very important component of the neural networks defined in the hyperbolic space.

Parallel Transport – The final building block, which is highly relevant in optimizing the parameters between hyperbolic layers, is the parallel transport. Utilized in *bias addition*, to be able to learn the embedding space as good as possible.

In a manifold $(\mathbb{D}_c^n, g^c)$, the parallel transport with respect to the Levi-Civita connection of a vector (the unique ∇) $\mathrm{v} \in \mathrm{T}_0\mathbb{D}_c^n$ to another tangent space $\in \mathrm{T}_y\mathbb{D}_c^n$ is given by Eq. 11

$$\mathrm{P}_{0\to y}^c(v) = \log_y^c(x \oplus_c \exp_0^c(v)) = \frac{\lambda_0^c}{\lambda_y^c} v. \tag{11}$$

Having built the mathematical components needed to work with hyperbolic spaces in neural networks, in the next chapter we will see a comprehensive connection between the theoretical background and the elements of neural networks

[2] Image taken from Lecture - Theoretical and Methodological foundations of Autonomous Systems, University of Stuttgart, Machine Learning and Robotics Lab

so that at the end we can construct neural networks which are exploiting the power of hyperbolic geometry in case of tree-like data.

4 Hyperbolic building blocks

Throughout converting Euclidean neural network concepts to their hyperbolic counterparts, a *generic approach* will be noticeable in multiple cases, which resembles the following structure: When there is a need to perform an Euclidean operation, move the point from the manifold to the tangent space (here Euclidean operations are valid), perform the operation and then move back to the Poincaré ball.

4.1 Activation function

Thinking of Euclidean point-wise non-linear activation functions such as *ReLU or tanh* (denoted by f), using the theory defined above we can define the *hyperbolic activation function* in the following way:

$$\mathrm{f}^{\otimes_c}(\mathrm{x}) = \exp_0^c(\mathrm{f}(\log_0^c)) \tag{12}$$

4.2 Multinomial logistic regression (MLR)

MLR in Euclidean space is defined using hyperplanes, and the probabilities per class depends on distance from hyperplanes. Thus defining Euclidean MLR using Euclidean separator hyperplane $\mathbb{E}_k : \langle a_k, x\rangle - b_k = 0$ for k $\in$ K as: $\mathrm{P}(y = k|x) \propto \exp(\langle a_k, x\rangle - b_k)$.

Rewriting the hyperplane equation, we get $\mathrm{P}(y = k|x) \propto \exp(sign(\langle a_k, x\rangle - b_k)||a_k||d_{\mathbb{R}}(x, \mathbb{E}_k))$, where d is the Euclidean distance. If we replace b_k with $a_k p_k$, then equation transforms to:

$\mathrm{P}(y = k|x) \propto \exp(sign(\langle a_k, -p_k + x\rangle)||a_k||d_{\mathbb{R}}(x, \mathbb{E}_k))$.
From here, the hyperbolic equivalent of the separator is: $\mathbb{H}_k^c : \langle -p_k \oplus_c x, a_k\rangle = 0$.

Thus, the hyperbolic multinomial logistic regression can be written as Eq. 13, utilizing the hyperbolic distance $d_{\mathbb{H}}(x, y) = \frac{2}{\sqrt{c}} tanh^{-1}(\sqrt{c}|| - x \oplus_c y||)$ from [1] chapter 3.1.

$$\mathrm{P}(y = k|x) \propto \exp(\frac{\lambda_{pk}^c ||a_k||}{\sqrt{c}} sinh^{-1}(\frac{2\sqrt{c}\langle -p_k \oplus_c x, a_k\rangle}{(1 - c|| - p_k \oplus_c x||^2)||a_k||})) \tag{13}$$

A more elaborated mathematical explanation is provided by [1] chapter 3.1, together with an alternative formulation of the multiclass logistic regression.

However performing optimization over a_k is not a trivial task since living in the tangent space from point p_k, it depends on it. Using the parallel transport from $a_k \to p_k$ the equation $a_k = \mathrm{P}_{0\to p_k}^c(a_k') = \frac{\lambda_0^c}{\lambda_{pk}^c} a_k'$, where $a_k' \in \mathrm{T}_0\mathbb{D}_c^n = \mathbb{R}^n$, which means it is in Euclidean space, thus a_k' can be optimized as an Euclidean parameter.

4.3 Riemannian optimization

In this subsection we are going to define the Riemannian Stochastic Gradient Descent, the hyperbolic alternative of the well-known SGD. [8] presents a very complex and detailed explanation of the mathematical background.

Taking the *generic approach*, the stochastic gradient descent $SGD : x_{t+1} \leftarrow x_t + \alpha g_t$ can be converted to it's Riemannian counterpart in a quite straightforward way following the steps from Algorithm 1.

Algorithm 1 Riemannian SGD (RSGD)

```
1: Evaluate the gradient $g_t$ at $x_t$
2: Project $g_t$ to the tangent space
3: Perform gradient step towards the negative gradient:
4:     $RSGD : x_{t+1} \leftarrow \exp_{x_t}(-\alpha g_t)$
```

Alternatively we could define any other optimization algorithm, such as Adam, Adagrad, etc., but keeping to the target of the paper we refer the user to the specified paper above.

4.4 Feed-forward layer

The feed forward layer is the first component actively utilizing previously converted neural network components, such as activation functions and matrix-vector product. The only component we have to define more specifically is the *bias addition*, which was mentioned as being a byproduct of the Riemannian *parallel transport* previously.

Translating a point $x \in \mathbb{D}_c^n$ by a bias $b \in \mathbb{D}_c^n$, is given by Eq. 14, utilizing the formulas derived previously, together with the realisation that the Möbius addition is a parallel transform (Eq. 11).

$$x \leftarrow x \oplus_c b = \exp_x^c(\mathrm{P}_{0\to x}^c(\log_0^c(b)) = \exp_x^c(\frac{\lambda_0^c}{\lambda_x^c}\log_0^c(b)) \tag{14}$$

Finalizing with the *bias addition* we have every necessary building block to create a feed-forward linear layer being described by $x_{t+1} = \sigma(\mathrm{x_t} w + b)$ but in hyperbolic space.

4.5 Recurrent layer

In the following section we are going to define the basic recurrent neural network layer in the hyperbolic space, and for deriving one of the most important RNN architecture, the *Gated Recurrent Unit*'s hyperbolic counterpart, we refer the reader to [1] chapter 3.3.

RNN – the naive architecture of the recurrent neural network is given by the following equation: $h_{x+1} = \sigma(Wh_t + Ux_t + b)$, where σ is a point-wise non-linearity, i.e. an activation function. Fig4 is a visual representation of RNN [3]. Utilizing Eq. 12, the formula can naturally be rewritten to Eq. 15, additionally specifying parameters $W \in \mathrm{M}_{m,n}(\mathbb{R}), \mathrm{U} \in \mathrm{M}_{m,d}(\mathbb{R}), b \in \mathbb{D}_c^m$.

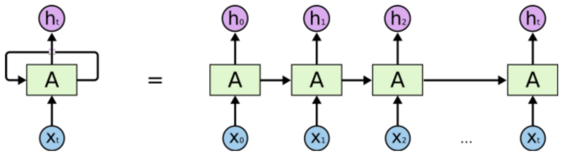

Fig. 4. Compact (left) and unrolled (right) representation of recurrent neural networks

$$h_{x+1} = \sigma^{\oplus_c}(W \otimes_c h_t \oplus_c U \otimes_c x_t \oplus_c b), \quad h_t \in \mathbb{D}_c^n, x_t \in \mathbb{D}_c^d \tag{15}$$

One last thing to be added is that usual *Euclidean input* x_t has to be rewritten to the hyperbolic geometry, which is possible in a straightforward manner: $x_t = \exp_0^c(x_t)$ and thus converting the input to the layer so that it can perform computations successfully.

Note that by following [1] chapter 3.3., one can rewrite the main alternative of GRUs, the Long Short-Term Memory (LSTM) architecture also, and experiments can be concluded with both architectures. Their dissimilarities yield different results in different scenarios.

4.6 Connecting the building blocks

With this chapter we conclude the theoretical derivation of the neural network components into their hyperbolic counterparts. Using the previous formulas and definitions, fully hyperbolic, or even mixed architectures between Euclidean and hyperbolic layers can be created.

In [1] chapter 4 a notorious set of experiments is concluded, using different architectures and different datasets. Overall proving that on tree-like dataset, the neural networks *consisting of hyperbolic elements* are on par, and most of the time perform better than their Euclidean counterparts.
Notable details related to the experiments and overall conclusion of [1]:

– hyperbolic optimization (RSGD) is needed to optimize hyperbolic layers
– even though the fully Euclidean network might have the advantage of more notorious optimization, the hyperbolic networks perform on average better, capturing better the hierarchical structure of the input data

[3] Image taken from [9]

– between three variants of *1. fully Euclidean MLR*, *2. fully hyperbolic MLR* and *3. Euclidean MLR performed embeddings mapped to the tangent space* the fully hyperbolic MLR performs the best, not surprisingly

Through a notorious set of examples and experiments [1] it is proven that if Euclidean building blocks of neural networks are correctly converted to their *hyperbolic counterparts*, HNNs can surpass their Euclidean base-architecture if the input data has a strongly hierarchical structure.

5 Learning Graph Embeddings

Graph embeddings, regardless of the geometry used, is a transformation operation. They are the transformations of property graphs to a vector or a set of vectors to enclose the information about the graph, in a more compact manner. Embedding nodes of a graph into Euclidean space has successfully done by Graph Convolutional Neural Networks (GCN), which have a main drawback: the **distortion** generated in the process of embedding the *often large-scale hierarchical data* into Euclidean geometry.
Hyperbolic geometry is an alternative to minimize the distortion when embedding the graph data, but transforming the operations of *feature transformation and aggregation* of GCNs into hyperbolic space is a challenge on it's own. The solution presented by [2] tackles the transformation of the aforementioned operations. Furthermore enables Euclidean input features mapping to hyperbolic spaces of different curvature so that the *distortion is minimized.* This way the expressiveness of the network is maximized. The term of *trainable curvature* between layers of the NNs is presented, as a way to further increase the representational power of the networks operating in the hyperbolic geometry.

5.1 Graph Convolutional Neural Networks (GCN)

In this chapter, we are going to utilize the theoretical background presented in Chapter 3 and the elements of hyperbolic neural networks in Chapter 4, not presenting them again. We will focus on describing the approach of the GCNs together with the advantages of transforming different components to hyperbolic geometry.

Embeddings are compressed representations. Even when GCNs are using a mapping technique which should in theory aggregate neighborhoods in such a way that the networks becomes better representable in low dimensions, distortion is going to be growing with the exponential growth of nodes in tree-like data together with the so-called width of the data, i.e. the properties of the nodes.

In the following we will describe the architecture of a GCN for a layer l at a certain vertex i in Eq. 16, given a set of $(\mathrm{W}^l, \mathrm{b}^l)$ weights and biases, together with an activation function $\sigma()$. Furthermore we define the concept of *neighborhood* of a vertex i, which is defined as follows: $\mathrm{N}(i) = j : (i,j) \in \mathrm{V}$, where V and E are the set of vertices and edges of a graph, respectively. Intuitively one can understand the concept of neighborhood as the set of vertices being directly linked to the vertex of interest.

$$\begin{aligned} h_i^l &= \mathrm{W}^l x_i^{l-1} + b^l \quad \text{(feature transform)} \\ x_i^l &= \sigma(h_i^l + \sum_{j \in \mathrm{N}(i)} \mathrm{w}_{ij} h_j^l) \quad \text{(neighborhood aggregation)} \end{aligned} \tag{16}$$

And while the architecture of the GCN is utilizing the hierarchy of the graph, namely the neighborhoods to create meaningful aggregations to reduce cluttering, it's main issue is still the representation space. Even if the architecture manages to aggregate successfully neighborhoods and reduce the space needed to represent the graph, the Euclidean space is still imposing quite strong limits even from the beginning, thus the distortion is still going to strongly affect the results.

5.2 Adaptation to Hyperbolic GCNs (HGCN)

In the following section we will define the hyperbolic counterpart of the GCNs, utilizing the theoretical background set in previous chapters. Furthermore we will utilize the experiments from [2] to prove the hypothesis that the more hyperbolic (i.e. Gromov's δ-hyperbolicity is small) the dataset is, HGCNs are performing better compared to GCNs.

First we define c > 0 being the value defining the curvature $-1/\mathrm{c}$ of a manifold. This c is going to be a parameter of each HGCN stacked layer in a deep architecture, and we are going to define the notion of *trainable curvature*, i.e. changing the c of each layer so that the representational power of the network is maximized.

Input feature mapping – The necessity to map Euclidean input features to hyperbolic space can be quite straightforwardly written as: $\mathrm{x}^{0,H} = \exp_0^c(0, x^{0,E})$, where H and E are denoting hyperbolic and Euclidean spaces, and **0** is the reference, the *origin* of the mapping between the manifold $\mathrm{H}^{d,c}$ and it's tangent space. Note that $x^{0,E} \in \mathbb{R}^d$.

Feature transform – Using the previously defined exp and log maps, the feature transform is no different from any feed-forward layer's internal structure, as presented in subsection here.

Neighborhood aggregation – This operation in hyperbolic space is redefined as the *attention-based neighborhood aggregation*, since the hyperbolic space maintains successfully the structure of the input graph. The redefined attention-based neighborhood aggregation is built, thus, as presented in Eq. 17, using hyperbolic Multilayer Perceptron (MLP).

$$\begin{aligned} w_{ij} &= \text{softmax}_{j\in \mathrm{N}(i)}(MLP(\log_{\mathrm{o}}^{\mathrm{K}}(\mathrm{x}_{\mathrm{i}}^{\mathrm{H}})||\log_{\mathrm{o}}^{\mathrm{K}}(\mathrm{x}_{\mathrm{j}}^{\mathrm{K}}))) \\ \text{agg}^{K}(x^{H})_i &= \exp_{x_i^H}^{K}(\sum_{j\in \mathrm{N}(i)} w_{ij}\log_{x_i^H}^{K}(x_j^H)) \end{aligned} \tag{17}$$

Two mentions related to Eq. 17 would be that w_{ij} is the new *attention of node i to node j*, which can be seen as the new weight of aggregation, and concatenation of the two embeddings is possible only in tangent space.

HGCN architecture – Before adding every component up to define the final architecture of HGCNs, we would like to specify a particular version of the hyperbolic activation function, which connects two successive layers, having hyperbolic curvatures $-1/\mathrm{K}_{l-1}$ and $-1/\mathrm{K}_l$, respectively:

$$\sigma^{\oplus^{K_{l-1},K_l}}(x^H) = \exp^{K_l}(\sigma(\log^{K_{l-1}}(x^H))) \tag{18}$$

Having all the components, the HGCN architecture is presented using Fig. 5 [4], connecting every step to a graphical representation.

Furthermore [2] presents in detail the idea of *trainable curvature* as a powerful and successful method to learn the embeddings of the right scale for each layer of a neural network. Even though theoretically a fixed curvature should already result in an overall boost of performance, the forward pass of the HGCN architecture allows the adjustment of curvature of each hyperbolic layer. By an optimization procedure the final result is proved by multiple experiments to be even higher than the fixed curvature alternative. For further reading we refer the reader to [2] Appendix B, where the mathematical background is thoroughly covered.

Using the architecture presented above, [2] proves the hypothesis that the more hyperbolic the dataset, i.e. a graph is, HGCN are outperforming GCNs, and going a step further, trainable curvature is a definite performance booster element. Yet not trivial, the aforementioned paper provides a good understanding ground for dealing with the tricky subject of trainable curvatures.

6 Conclusions

We presented the theroetical background behind the hyperbolic geometry needed to convert neural networks utilizing Euclidean spaces into a structure that provides a better representational power in case of input data which resembles a hierarchical structure, i.e. tree-like data.
Utilizing the impressive literature and experiments provided by [1] and [2] together with further linked articles, papers we showed the power of hyperbolic neural networks in the context of hierarchical data. Finally we gave a comprehensive explanation on how these networks can learn graph embeddings in a better

[4] Image taken from [2]

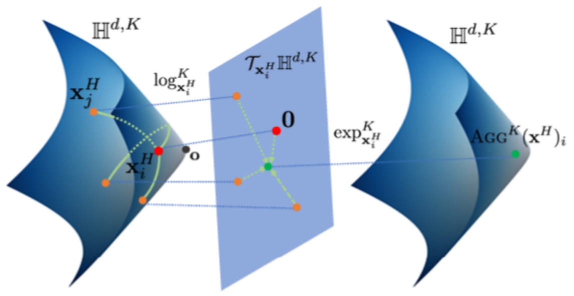
$\mathbb{H}^{d,K}$
$\mathbf{x}_j^H$
$\log^K_{\mathbf{x}_i^H}$
$\mathcal{T}_{\mathbf{x}_i^H}\mathbb{H}^{d,K}$
$\mathbf{0}$
$\mathbf{x}_i^H$
$\mathbf{o}$
$\exp^K_{\mathbf{x}_i^H}$
$\mathbb{H}^{d,K}$
$\text{AGG}^K(\mathbf{x}^H)_i$

Fig. 5. HGCN attention-based neighborhood aggregation: from left to right, feature transform, aggregation, mapping back to new hyperbolic space by activation

way than their Euclidean counterparts, making them a better choice when the data resembles a hyperbolic structure.

References

1. O.-E. Ganea, G. Bécigneul, and T. Hofmann, "Hyperbolic neural networks," 2018.
2. I. Chami, R. Ying, C. Ré, and J. Leskovec, "Hyperbolic graph convolutional neural networks," 2019.
3. G. Bachmann, G. Bécigneul, and O. Ganea, "Constant curvature graph convolutional networks," in *International Conference on Machine Learning*, pp. 486–496, PMLR, 2020.
4. B. P. Chamberlain, J. R. Clough, and M. P. Deisenroth, "Hybed: Hyperbolic neural graph embedding," 2018.
5. B. Keng, "Hyperbolic geometry and poincaré embeddings." `https://bjlkeng.github.io/posts/hyperbolic-geometry-and-poincare-embeddings/`, Jun 2018.
6. P. Godec, "Graph embeddings - the summary." `https://towardsdatascience.com/graph-embeddings-the-summary-cc6075aba007`, Jun 2019.
7. C.-C. Ni, Y.-Y. Lin, F. Luo, and J. Gao, "Community detection on networks with ricci flow," 2019.
8. S. Bonnabel, "Stochastic gradient descent on riemannian manifolds," *IEEE Transactions on Automatic Control*, vol. 58, pp. 2217–2229, sep 2013.
9. C. Lioma, B. Larsen, C. Petersen, and J. G. Simonsen, "Deep learning relevance: Creating relevant information (as opposed to retrieving it)."