Are Hyperbolic Representations in Graphs Created Equal?

Max Kochurov [1] **Sergey Ivanov** [2] **Eugeny Burnaev** [1]

Abstract

Recently there was an increasing interest in applications of graph neural networks in non-Euclidean geometry; however, are non-Euclidean representations always useful for graph learning tasks? For different problems such as node classification and link prediction we compute hyperbolic embeddings and conclude that for tasks that require global prediction consistency it might be useful to use non-Euclidean embeddings, while for other tasks Euclidean models are superior. To do so we first fix an issue of the existing models associated with the optimization process at zero curvature. Current hyperbolic models deal with gradients at the origin in ad-hoc manner, which is inefficient and can lead to numerical instabilities. We solve the instabilities of κ-Stereographic model at zero curvature cases and evaluate the approach of embedding graphs into the manifold in several graph representation learning tasks.

1. Introduction

Hierarchies in data are common in real world settings and can be observed in many scenarios. For example, languages have relations between words and contextual dependencies within a sentence. Words can be viewed as entities and one may define natural type dependencies on them. These dependencies may be entirely arbitrary and different relation graphs may exist for the same dictionary.

In general, there are several settings in graph problems: knowledge graph completion (Balažević et al., 2019), node classification and link prediction (Liu et al., 2019; Chami et al., 2019; Bachmann et al., 2020) or graph embedding (Gu et al., 2019). While Euclidean baselines are strong, there is a general trend of making Hyperbolic embeddings to be more efficient and interpretable.

[1]Skolkovo Institute of Science and Technology, Russia [2]Criteo AI Lab, France. Correspondence to: Max Kochurov <maksim.kochurov@skoltech.ru>.

Proceedings of the 37^{th} International Conference on Machine Learning, Vienna, Austria, PMLR 108, 2020.

In this work we highlight the shortcomings of the past research about Hyperbolic deep learning for graph data. We review the model from (Bachmann et al., 2020) and fix it to be more robust at zero curvature case. Furthermore, we evaluate the κ-Stereographic model for node classification, link prediction, graph classification, graph embedding problems.

2. Preliminaries

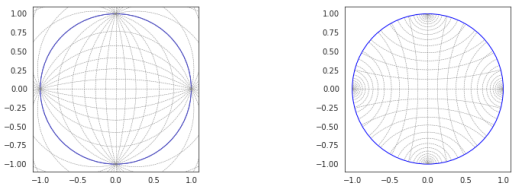
2.1. Base Model

κ-Stereographic model $\mathcal{M}_\kappa^n$ (Bachmann et al., 2020) is a unification of constant curvature manifolds: hyperboloid and sphere. The model is a Riemannian manifold that has constant sectional curvature κ and dimension n. Besides curvature, the parameter κ defines how parameters are constrained what is crucial for optimization algorithms:

$$\mathcal{M}_\kappa^n = \begin{cases} \left\{x \in \mathbb{R}^n : \|x\|_2 < 1/\sqrt{-\kappa}\right\},\ \lambda_x^\kappa & \kappa < 0 \\ \mathbb{R}^n,\ \lambda_x^\kappa & \kappa \geq 0 \end{cases} \tag{1}$$

where $\lambda_x^\kappa = \frac{2}{1+\kappa\|x\|_2^2}$ is conformal (preserving angles) metric tensor at point x.

For optimization we need exponential map (gradient update) and parallel transport (momentum update). For positive and negative cases they are well defined. Exponential map is a function that defines a unit time travel along geodesic (straight) line from a given point, i.e. $\exp_x^\kappa : T_x\mathcal{M}_\kappa^n \mapsto \mathcal{M}_\kappa^n$. For κ-Stereographic model exponential map is defined



(a) $\kappa = 1$ (b) $\kappa = -1$

Figure 1. Stereographic projection geodesics for sphere ($\kappa = 1$) and hyperboloid ($\kappa = -1$)

as follows:

$$\exp_x^\kappa(u) = x \oplus_\kappa \tan_\kappa(\|u\|_x/2)\frac{u}{\|u\|_2}, \tag{2}$$

where $\oplus_\kappa$ is defined as

$$x \oplus_\kappa y = \frac{(1 - 2\kappa\langle x, y\rangle - \kappa\|y\|_2^2)x + (1 + \kappa\|x\|_2^2)y}{1 - 2\kappa\langle x, y\rangle + \kappa^2\|x\|_2^2\|y\|_2^2} \tag{3}$$

and $\tan_\kappa$ as

$$\tan_\kappa(x) = \begin{cases} \kappa^{-1/2}\tan(x\kappa^{1/2}) & \kappa > 0 \\ x & \kappa = 0 \\ |\kappa|^{-1/2}\tanh(x|\kappa|^{1/2}) & \kappa < 0 \end{cases} \tag{4}$$

Parallel transport (e.g. for momentum update in SGD) along geodesic looks like

$$P_{x\to y}^\kappa(v) = \mathrm{gyr}[y, -x]v\lambda_x^\kappa/\lambda_y^\kappa, \tag{5}$$

where $\mathrm{gyr}[u, v]w$ is defined as

$$\mathrm{gyr}[u, v]w = (-(u \oplus_\kappa v)) \oplus (u \oplus_\kappa (v \oplus_\kappa w)) \tag{6}$$

Computing distances and similarity measured on the κ-Stereographic manifold is used for final classification layers in neural networks. Importantly, distances on the manifold are defined as follows:

$$d_\kappa(x, y) = 2\tan_\kappa^{-1}(\|(-x) \oplus_\kappa y\|_2), \tag{7}$$

Gromov product, an extension to inner product on manifolds

$$(x, y)_r = (d_\mathcal{M}(x, r)^2 + d_\mathcal{M}(y, r)^2 - d_\mathcal{M}(x, y)^2)/2, \tag{8}$$

and distance to the hyperplane

$$d_\kappa(x, \tilde{H}_{a,p}^\kappa) = \inf_{w\in\tilde{H}_{a,p}^\kappa} d_\kappa(x, w) \tag{9}$$

$$= \sin_\kappa^{-1}\left\{\frac{2|\langle(-p) \oplus_\kappa x, a\rangle|}{(1 + \kappa\|(-p) \oplus_\kappa \|x\|_2^2)\|a\|_2}\right\}, \tag{10}$$

where

$$\sin_\kappa(x) = \begin{cases} \kappa^{-1/2}\sin(x\kappa^{1/2}) & \kappa > 0 \\ x & \kappa = 0 \\ |\kappa|^{-1/2}\sinh(x|\kappa|^{1/2}) & \kappa < 0 \end{cases} \tag{11}$$

2.2. Fixing Missing Gradients

Equations 4 and 11 are incomplete. At the point of zero curvature, gradient for κ is not well defined. However, we can generalize it by taking left and right Taylor expansions. We can see gradients for κ appear:

$$\tan_\kappa(x) \approx x + \tfrac{1}{3}\kappa x^3 + \tfrac{2}{15}\kappa^2 x^5\Big|_{\kappa=+\varepsilon} \tag{12}$$

$$\tan_\kappa(x) \approx x + \tfrac{1}{3}(-\kappa)x^3 + \tfrac{2}{15}(-\kappa)^2 x^5\Big|_{\kappa=-\varepsilon} \tag{13}$$

Then the complete formula in the equation 4 for $\tan_\kappa(x)$ that is differentiable at $\kappa = 0$ should be the following:

$$\tan_\kappa(x) = \begin{cases} \kappa^{-1/2}\tan(x\kappa^{1/2}) & \kappa > 0 \\ x + \frac{1}{3}\kappa x^3 + \frac{2}{15}\kappa^2 x^5 & \kappa = 0 \\ |\kappa|^{-1/2}\tanh(x|\kappa|^{1/2}) & \kappa < 0 \end{cases}, \tag{14}$$

where $\frac{2}{15}\kappa^2 x^5$ term is optional and is required for higher order gradients only. Taylor expansions for $\tan_\kappa^{-1}$ and functions involved in computing distances to hyperplanes may be found in Appendix B.

2.3. Curvature optimization

Curvature optimization in Hyperbolic models is an important challenge. The optimization step changes the geometry of space and parameter constraints are also dependent on the curvature parameter. Therefore parameters and curvature are tied together and curvature update cannot be performed without parameter updates.

We are interested in optimizing parameters $\{p_i\}$ of the model that lie in κ-Stereographic model. Formally, the optimization problem looks as:

$$\min_{\kappa\in\mathbb{R},\ p_i\in\mathcal{M}_\kappa^n} \mathcal{L}(\kappa, \{p_i\}) \tag{15}$$

To work with parameters $\{p_i\}$ on the manifold, they are represented as tuples of numbers in some chart, i.e. a numerical parametrization of the manifold. One cannot make independent parameter updates. Fair to note, that all curvatures represent the same model, and one can be obtained with isomorphism. However, this is not the case for product manifolds and numerical precision is different for different curvatures (Gu et al., 2019; Sa et al., 2018).

Model decomposition changes loss landscape and this was shown to improve the performance (Salimans & Kingma, 2016). Parameter constraints vary with κ as seen in equation 1. Once negative κ changes, numerical representation of parameters may no longer satisfy boundary conditions, because the ball's radius changes (equation 1), while parameters remain fixed. In order to satisfy the constraints, we should project these parameters back to the domain. There are a few alternatives on how to perform optimization in such a complex parameter space.

Alternating optimization. One way to perform optimization is by doing alternation in parameter updates: with one step we optimize κ and project p_i, with another step we optimize p_i. This partially solves inconsistency but introduces another problem. After κ update, all mutual distances are updated and perturbed. Therefore, parameter distances to the origin are changed and momentum for these parameters correspond to a new tangent space, which does not

correspond to the old one. From one point of view, it is just a vector space around a specific point; from another, it is connected to a point on a concrete manifold with the metric tensor. After curvature updates, the metric tensor changes as well. It is not obvious how to take this into account. However, just ignoring both problems worked sufficiently in practice.

Joint optimization. The purpose of alternating optimization was to decouple curvature and parameters. However, it is not practical: forward model pass is done twice, one per each parameter group. More efficiently we can compute and apply gradients once for all parameter groups. There are two options on how to do that: one is to update κ first and then p_i, the second is vice-versa. In the former case, we have biased gradients for parameters, in the latter for curvature. Bias appears as we separate updates in two blocks. In parameter updates, curvature affects the exponential map and gradient norms. For curvature updates, we make parameter updates with one curvature and then change curvature without additional computing gradient at a new point. As there are much fewer parameters involved in the curvature updates, biasing curvature in update is more accurate.

Tangent space optimization. The most consistent way to formulate optimization problems is to reparametrize parameter space and remove constraints (Lezcano Casado, 2019). The intuitive way to do that is to put the problem into the tangent space of zero. The problem formulation changes as follows:

$$\min_{\kappa\in\mathbb{R},\ p_i\in\mathcal{M}_\kappa^n} \mathcal{L}(\kappa, \{p_i\}) \quad \Longleftrightarrow \tag{16}$$

$$\Longleftrightarrow \min_{\kappa\in\mathbb{R},\ \tilde{p}_i\in T_0\mathcal{M}_\kappa^n} \mathcal{L}(\kappa, \{\exp_0^\kappa(\tilde{p}_i)\}) \tag{17}$$

Next, we compare joint optimization and tangent space optimization with curvature in an empirical study. We start with embedding nodes of a graph into non-Euclidean space such that it preserves graph distances. This problem isolates optimization process from the architecture choice and serves as a good example to study various optimization techniques in non-Euclidean space.

3. Experiments

We consider four benchmark tasks: 1) node classification, 2) link prediction, 3) graph classification, 4) graph embedding. These tasks are conceptually different: in tasks 1) and 3) target label for an object does not depend on other objects such as other nodes or graphs. In contrast, to solve problems 2) and 4) it is important to keep global consistency. For link prediction it prevents spurious connections or their absence. For graph embedding task, the loss function captures all node interactions.

3.1. Graph Embedding

As shown in (Sa et al., 2018), Euclidean space cannot capture all tree-like data structures, while embeddings in Hyperbolic space can represent any tree without significant distortions. Other works (Gu et al., 2019) successfully applied mixed curvature spaces for graph embeddings and show that non-Euclidean spaces are good at capturing relations in the real-world graphs. The analysis of mixed curvature spaces is still not explored to a full extent. Curvature signs should be specified before training and cannot be changed while training. The limitation requires a costly grid search to tune hyperparameters. The problem is not self-contained to serve as a real-world example, but it clearly shows the advantages and disadvantages of Riemannian optimization compared to Euclidean optimization in terms of the number of iterations and running time. It allows us to compare purely the methods of optimization and the loss function rather than the complexity of neural networks.

In this work, we propose to use a κ-Stereographic model with smooth transitions in geometry. Tunable curvature allows only to specify the desired product space for embeddings without the sign. Gradient descent is supposed to fit optimal curvature for the graph dataset. We investigate curvature optimization approaches described in Section 2.3 for Facebook social network dataset. To embed a graph into metric space, we choose the following loss function (see also Algorithm 1)

$$D_{avg} = \mathbb{E}_{v_i,v_j\sim G}\left(\frac{d_{ij}}{d_{\mathcal{M}}(v_i,v_j)} - 1\right)^2 \to \min_{v_i,v_j} \tag{18}$$

The loss function optimizes the ratio between distances computed on the graph and in the metric space. When ideally optimized, the loss should be strictly zero. The results for four optimization approaches are presented in Table 1.

Table 1. Embedding nodes of a graph into κ-Stereographic space. Results for Facebook dataset D_{avg} (↓) (number of gradient descent iterations in brackets). Traditional Euclidean model (1st row) is outperformed once κ-Stereographic embeddings are trained on top (2nd row). Hyperbolic Riemannian optimization suffers from week convergence (3rd row), while tangent optimization (4th row) is able to improve the results given same number of iterations. Time per iteration for all the models is about the same.

Optimization way	$(\mathcal{M}_\kappa^2) \times 5$	$(\mathcal{M}_\kappa^5) \times 2$
Euclidean (20k)	0.0069	0.0069
Euclidean (20k) + $\mathcal{M}_\kappa^n$ (20k)	0.0056	0.0055
κ-Stereographic (20k)	0.0085	0.0226
Tangent κ-Stereographic (20k)	0.0075	**0.0025**

Table 2. Evaluation results (AUC) for link prediction on Cora, PubMed and CiteSeer datasets. Validation partition was used to select test scores to report and 11 independent experiments were run to report standard deviation.

Models	Cora	PubMed	CiteSeer
GCN(Kipf & Welling, 2016)-$\mathcal{M}_\kappa^n$-gromov	$\mathbf{0.977 \pm 0.004}$	$\mathbf{0.969 \pm 0.003}$	$\mathbf{0.990 \pm 0.002}$
GCN(Kipf & Welling, 2016)-$\mathbb{R}$-inner	0.864 ± 0.01	0.870 ± 0.005	0.882 ± 0.009

3.2. Node Classification

In this problem we project embeddings to κ-Stereographic manifold with a GCN model[1] described in (Kipf & Welling, 2016). The final layer to obtain logits was combined with Gromov product (equation 8), where the reference point is chosen to be zero. Gromov product is a natural generalization of inner products traditionally used for similarity measure purposes. The algorithm for node classification may be found in Algorithm 3. As can be seen from the Table 3 complex geometry did not lead to improvement and the Euclidean model performed better on hold out dataset partition. The explanation may be in the locality of information as no hierarchy is needed to express the solution, and additional degrees of freedom leads to overfitting.

Table 3. Node classification results for GCN model. Accuracy (↑) for the model on a test set for best performing model on validation part is used to measure the performance. The results suggest that embedding attributes in κ-Stereographic space does not lead to improvement.

Manifold	Cora	CiteSeer	PubMed
Baseline $\mathbb{R}^{16}$	**0.809**	**0.718**	**0.783**
$\mathcal{M}_\kappa^{16}$	0.8	0.68	0.76
$(\mathcal{M}_\kappa^{8}) \times 2$	0.781	0.673	0.772
$(\mathcal{M}_\kappa^{4}) \times 4$	0.777	0.666	0.765

3.3. Link Prediction

We were inspired by (Kipf & Welling, 2016) and extended GCN models to work in κ-Stereographic space. We embed graph neural network in κ-Stereographic space using the exponential map and measure similarity with Gromov Product. Experiment setup followed standard protocol as in reference implementation in `torch_geometric`[2]. Evaluation results are found in Table 2. Results suggest that non-Euclidean geometry of embeddings improves generalization across three benchmark datasets. No additional hyperparameters were introduced except tunable curvature. We believe this result is thanks to metric learning nature of the problem, GCN learned to accurately predict such embeddings for nodes that capture the unobserved topological structure.

3.4. Graph Classification

For benchmarking classification problems, we studied the Proteins dataset win a modification of GCN (see Algorithm 4). We embedded graph into κ-Stereographic model and used Gromov inner product or distance to hyperplanes to compute logit scores for the classification task. The results are presented in Table 4. The results suggest that for graph classification task using non Euclidean embedding may lead to overfitting and training is thus less stable.

Table 4. Graph classification results on PROTEINS dataset. Accuracy (↑) is reported for hold out dataset with std.

Model	$D = 64$	$D = 32$
Euclean baseline	$\mathbf{0.749 \pm 0.01}$	$\mathbf{0.795 \pm 0.02}$
$\mathcal{M}_\kappa^D$-planes	0.743 ± 0.03	0.737 ± 0.03
$\mathcal{M}_\kappa^D$-gromov	0.735 ± 0.03	0.736 ± 0.03

4. Discussion

Hyperbolic neural networks are a promising approach for tasks, where knowledge of the global structure is crucial for the prediction. That covers many graph representation learning problems such as link prediction or graph embedding task. From the experiments, we can conclude that more complicated models not necessarily perform better in the problems, where a local structure is enough for solving a task, Euclidean methods are as good as their extensions to non-Euclidean spaces.

Putting aside optimization of curvature for graph models, one should decide about the nature of the problem at hand and ask whether the answers to the problem depend on all the nodes, subset of them, or just a single object? Often the ground truth labels depend only on the local neighborhood of the node in which case complex Hyperbolic models may be inferior to their Euclidean counterparts.

[1] Reference implementation is in found in `torch_geometric` package (Fey & Lenssen, 2019): https://github.com/rusty1s/pytorch_geometric/blob/master/examples/gcn.py

[2] found in https://github.com/rusty1s/pytorch_geometric/blob/master/examples/link_pred.py

References

Bachmann, G., Becigneul, G., and Ganea, O.-E. Constant curvature graph convolutional networks. 2020.

Balažević, I., Allen, C., and Hospedales, T. Multi-relational poincaré graph embeddings. May 2019. URL http://arxiv.org/abs/1905.09791v3.

Chami, I., Ying, Z., Ré, C., and Leskovec, J. Hyperbolic graph convolutional neural networks. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 32*, pp. 4868–4879. Curran Associates, Inc., 2019. URL http://papers.nips.cc/paper/8733-hyperbolic-graph-convolutional-neural-networks.pdf.

Fey, M. and Lenssen, J. E. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

Gu, A., Sala, F., Gunel, B., and Ré, C. Learning mixed-curvature representations in product spaces. 2019. URL https://openreview.net/pdf?id=HJxeWnCcF7.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907, 2016. URL http://arxiv.org/abs/1609.02907.

Lezcano Casado, M. Trivializations for gradient-based optimization on manifolds. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 32*, pp. 9157–9168. Curran Associates, Inc., 2019. URL http://papers.nips.cc/paper/9115-trivializations-for-gradient-based-optimization-on-manifolds.pdf.

Liu, Q., Nickel, M., and Kiela, D. Hyperbolic graph neural networks. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 32*, pp. 8230–8241. Curran Associates, Inc., 2019. URL http://papers.nips.cc/paper/9033-hyperbolic-graph-neural-networks.pdf.

Sa, C. D., Gu, A., Ré, C., and Sala, F. Representation tradeoffs for hyperbolic embeddings. Apr 2018. URL http://arxiv.org/abs/1804.03329v2.

Salimans, T. and Kingma, D. P. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *CoRR*, abs/1602.07868, 2016. URL http://arxiv.org/abs/1602.07868.

A. Algorithms

Algorithm 1 Graph embedding

Require: G, V – graph
Run Floyd–Warshall to get graph distances $\{d_{ij}\}$ between nodes
Initialize nodes with random embeddigns $\{v_i\} \in \mathcal{M}\{\mathcal{M}$ can be any manifold$\}$
$D_{avg} = \mathbb{E}_{v_i, v_j \sim G} \left(\frac{d_{ij}}{d_{\mathcal{M}}(v_i, v_j)} - 1 \right)^2 \rightarrow \min_{v_i, v_j}$

Algorithm 2 GCN-$\mathbb{U}$ for link prediction

Require: $\mathcal{G}$ – input graph
$\tilde{\mathcal{G}} = \text{GCN}(\mathcal{G})\{$Calculate GCN for graph $\mathcal{G}\}$
if clip norm **then**
$\forall\, i : y_i = [\tilde{\mathcal{G}}]_i / \max(\|[\tilde{\mathcal{G}}]_i\|, 1)\{$Clip norm of embeddings to 1$\}$
else
$\forall\, i : y_i = [\tilde{\mathcal{G}}]_i\{$No norm clipping$\}$
end if
$\forall\, i : x_i = \exp_0^{\mathcal{M}}([\tilde{\mathcal{G}}]_i)\{$Map GCN output for i'th node to $\mathcal{M}$, e.g. $(\mathbb{U}_1^d, \ldots, \mathbb{U}_K^d)\}$
if Gromov **then**
$z_i^j = (x_i, x_j)_0\{$Calculate Gromov product score for link i-j as in equation 8$\}$
else
$z_i^j = -d_{\mathcal{M}}(x_i, x_j)^2 + r\{$Calculate distance score for link i-j proportional to distance$\}$
end if
$p(x_i, x_j) = \sigma(z_i^j)$ $\{$Calculate Sigmoid for decision$\}$
$\mathbb{I}(x_i, x_j) = p(x_i, x_j) > t$ $\{$Final decision is based on threshold $t\}$

Algorithm 3 κ-GCN for node classification

Require: $\mathcal{G}$ – input graph
$\tilde{\mathcal{G}} = \text{GCN}(\mathcal{G})\{$Calculate GCN for graph $\mathcal{G}\}$
$x_i = \exp_0^{\mathcal{M}}((\tilde{\mathcal{G}})_i)\{$Map GCN output for i'th node to $\mathcal{M}$, e.g. $(\mathbb{U}_1^d, \ldots, \mathbb{U}_K^d)\}$
$z_i^c = (x_i, w_c)_0 + b_c\{$Calculate Gromov product per class as in equation 8$\}$
$p(x_i) = \text{softmax}(z_i^1, \ldots, z_i^C)$ $\{$Calculate Softmax for predictions$\}$

Algorithm 4 κ-GCN for graph classification

Require: $\mathcal{G}$ – input graph
$\tilde{g} = \text{GCN}(\mathcal{G})\{$Calculate GCN aggregation for graph $\mathcal{G}\}$
$x = \exp_0^{\mathcal{M}}(\tilde{g})\{$Map GCN output for graph to $\mathcal{M}$, e.g. $(\mathbb{U}_1^d, \ldots, \mathbb{U}_K^d)\}$
if Gromov **then**
$z_c = (x, w_c)_0 + b_c\{$Calculate Gromov product per class as in equation 8$\}$
else
$z_c = d_{\mathcal{M}}(x, \tilde{H}^{\mathcal{M}}_{a_c, p_c})\{$Calculate signed distance to the class hyperplane$\}$
end if
$p(x) = \text{softmax}(z_1, \ldots, z_C)$ $\{$Calculate Softmax for predictions$\}$

B. Taylor Expansions

Proper gradients for zero curvature cases solve all these limitations at once, and no prior knowledge assumed to fit curvature. Moreover, the Tailor series allows us a convenient approach to pre-train Hyperbolic models with zero curvature and, at some point, turn on curvature optimization. Use-cases may involve future research in generalizing Euclidean convolutional neural networks to Hyperbolic spaces.

The approach to calculating gradients for κ correctly is to write tailor expansion at a problematic point.

$$\tan_\kappa(x) = \begin{cases} \kappa^{-1/2}\tan(x\kappa^{1/2}) & \kappa > 0 \\ x + \frac{1}{3}\kappa x^3 + \frac{2}{15}\kappa^2 x^5 + \frac{17}{315}\kappa^3 x^7 + \frac{62}{2835}\kappa^4 x^9 + \frac{1382}{155925}\kappa^5 x^{11} + \dots & \kappa = 0 \\ |\kappa|^{-1/2}\tanh(x|\kappa|^{1/2}) & \kappa < 0 \end{cases} \tag{19}$$

$$\tan^{-1}_\kappa(x) = \begin{cases} \kappa^{-1/2}\tan^{-1}(x\kappa^{1/2}) & \kappa > 0 \\ x - \frac{\kappa x^3}{3} + \frac{\kappa^2 x^5}{5} - \frac{\kappa^3 x^7}{7} + \frac{\kappa^4 x^9}{9} - \frac{\kappa^5 x^{11}}{11} + \dots & \kappa = 0 \\ |\kappa|^{-1/2}\tanh^{-1}(x|\kappa|^{1/2}) & \kappa < 0 \end{cases} \tag{20}$$

$$\sin_\kappa(x) = \begin{cases} \kappa^{-1/2}\sin(x\kappa^{1/2}) & \kappa > 0 \\ x - \frac{\kappa x^3}{6} + \frac{\kappa^2 x^5}{120} - \frac{\kappa^3 x^7}{5040} + \frac{\kappa^4 x^9}{362880} - \frac{\kappa^5 x^{11}}{39916800} + \dots & \kappa = 0 \\ |\kappa|^{-1/2}\sinh(x|\kappa|^{1/2}) & \kappa < 0 \end{cases} \tag{21}$$

$$\sin^{-1}_\kappa(x) = \begin{cases} \kappa^{-1/2}\sin^{-1}(x\kappa^{1/2}) & \kappa > 0 \\ x - \frac{1}{6}\kappa x^3 + \frac{3}{40}\kappa^2 x^5 - \frac{5}{112}\kappa^3 x^7 + \frac{35}{1152}\kappa^4 x^9 - \frac{63}{2816}\kappa^5 x^{11} + \dots & \kappa = 0 \\ |\kappa|^{-1/2}\sinh^{-1}(x|\kappa|^{1/2}) & \kappa < 0 \end{cases} \tag{22}$$

Tailor expansion[3] allows us to take gradient in regions where κ is zero. Only first-order expansion is required for the correct gradient and now will depend on the actual value of x. With this extension, there is no more gap between constant positive and negative curvature manifolds. They are interpolated smoothly, and gradients allow to determine the best curvature sign.

[3] Obtained with Wolfram Mathematica