Hyperbolic neural networks, ML 2019 report

Max Kochurov [*1] **Rasul Kerimov** [*1] **Sergei Kozlukov** [*1]

Abstract

Recent research (Nickel & Kiela, 2017; hyperbolicdeeplearning.com, 2019) has proven that it might be beneficial, in case of graph-like data, to endow embeddings with a non-Euclidean metric of negative curvature. Furthering this line, (Ganea et al., 2018) proposed means to construct neural network layers accounting for alleged curved geometry of hierarchical data latent spaces.

In this work we reproduce the results and provide an analyzis of (Ganea et al., 2018). One of the by-products of this work is an open-source Riemannian optimization package for `pytorch` (Geoopt, 2019).

1. Introduction

One of the long-standing problems in ML is that of finding fixed-length vector representations of data, which would capture domain-specific "important" features of observations and, hopefully, provide means to measure similarity of objects, combine them, etc. We're considering this problem in the context of hierarchical data such as texts, trees, graphs, et cetera. This setting is relevant for problems of machine translation, question answering, community detection, link prediction, and alike. Classical approaches include: `Word2Vec` (Mikolov et al., 2013), `GLoVe` (Pennington et al., 2014), `FastText` (Bojanowski et al., 2016) for texts and `Node2Vec` (Grover & Leskovec, 2016) and `DeepWalk` (Perozzi et al., 2014) for graphs. One of the caveats of all these approaches is dependence of their performance on dimensionality of embeddings. A few studies (Nickel et al., 2014) have shown that the representation of big graphs can require prohibitively large dimension to show certain relations between nodes. Hyperbolic embeddings do not suffer from that problem.

Embeddings trees in hyperbolic spaces is not a new idea (Gromov, 1987; Hamann, 2017), however its adoption in machine learning is a relatively new trend yet. One of the prerequisites for building models that account for negative curvature of the domain space might be generalization of basic operations (e.g. matrix-vector product, vector addition, vector inner product), and closed-form expressions for basic objects (e.g. distances, geodesics, parallel transport). (Ganea et al., 2018) attempt to provide such tools and apply them to generalize multinomial logistic regression, FFNN and RNN. We reproduce and analyze their results, provide our own interpretation, and consider ways to further develop this subject.

[*]Equal contribution [1]Skoltech, Moscow, Russia. Correspondence to: Max Kochurov <maksim.kochurov@skoltech.ru>.

Preliminary work.

2. Background

2.1. Riemannian geometry

A d-dimensional smooth manifold $\mathcal{M}$ is a topological space, each point $x \in M$ of which has a neighbourhood U_x homeomorphic to some open $V_0 \in \mathbb{R}^d$ in such a way that when to covering neighbourhoods overlap, the change of coordinates is a diffeomorphic map. At each point x of manifold we construct a *tangent space* $\mathcal{T}_x M$ – a vector space of *derivations*, i.e. operators taking initial velocities of functions $\mathcal{M} \to \mathbb{R}$ along paths emanating from x. Riemannian manifold is a smooth manifold with an inner product map (called *metric tensor*) $g(x) = \langle \cdot, \cdot \rangle_x : \mathcal{T}_x M \times \mathcal{T}_x M \to \mathbb{R}$ defined for each point $x \in M$. Riemannian metric allows to calculate the length of any curve $\gamma : [0, 1] \to \mathcal{M}$:

$$L(\gamma) = \int_0^1 \sqrt{g_{y_t}(\dot{y}_t, \dot{y}_t)}\, \mathrm{d}\, t.$$

In regular circumstances, this induces global distance on $\mathcal{M}$ defined as the length of a shortest path between given two points:

$$d(x, y) = \inf_{\gamma : \gamma_0 = x, \gamma_1 = y} L(\gamma). \tag{1}$$

Thus $\gamma : [0, 1] \to \mathcal{M}$ is called a shortest path between γ_0 and γ_1 if $d(\gamma_0, \gamma_1) = L(\gamma)$.

$\exp_x$ is the crucial operation that allows us to project back vector v on tangent space $\mathcal{T}_x \mathcal{M}$ at x to the point $\exp_x(v) \in \mathcal{M}$ on manifold.

Another important notion here is the relation of conformality between metric tensors. Metric tensor $\tilde{g}$ is said to be

conformal to g if it defines same angles:

$$\frac{\tilde{g}_x(u,v)}{\sqrt{\tilde{g}_x(u,u)}\sqrt{\tilde{g}_x(v,v)}} = \frac{g_x(u,v)}{\sqrt{g_x(u,u)}\sqrt{g_x(v,v)}} \tag{2}$$

for all $x \in \mathcal{M}$, $u, v \in \mathcal{T}_x\mathcal{M} \setminus \{0\}$. This is equivalent to the existence of a smooth function $\lambda : \mathcal{M} \to \mathbb{R}$, called comformal factor, such that $\tilde{g}_x = \lambda_x^2 g_x$ for all $x \in \mathcal{M}$.

2.2. Curvature

2.3. Poincare ball model

One of the model spaces of negative curvature (Cannon et al., 1997) is the Poincare ball model: a n-dimensional Poincare ball is the usual unit ball $\mathcal{D} = \{x \in \mathbb{R}^n : \|x\| < 1\}$, taken with the following metric tensor:

$$g_x^{\mathcal{D}} = \lambda_x^2 g^E, \lambda_x = \frac{2}{1 - \|x\|^2}, \tag{3}$$

where $g^E = I_n$ is the Euclidian metric tensor [1]. Distance between two points $x, y \in \mathcal{D}$ is:

$$d_{\mathcal{D}}(x,y) = \cosh^{-1}\left(1 + 2\frac{\|x-y\|^2}{(1-\|x\|^2)(1-\|y\|^2)}\right) \tag{4}$$

The Poincare ball is conformal to Euclidian space, therefore angles between two vectors $u, v \in \mathcal{T}_x\mathcal{M}\setminus\{0\}$ are preserved:

$$\cos_x(\angle(u,v)) = \frac{g_x^{\mathcal{D}}(u,v)}{\sqrt{g_x^{\mathcal{D}}(u,v)}\sqrt{g_x^{\mathcal{D}}(u,v)}} = \frac{\langle u, v\rangle_x}{\|u\|_x\|v\|_x} \tag{5}$$

2.4. Mobius arithmetics

To build a hyperbolic counterpart of GRU we're going to use the following primitives:

Mobius addition

$$x \oplus_c y = \frac{(1 + 2c\langle x, y\rangle + c\|y\|_2^2)x + (1 - c\|x\|_2^2)y}{1 + 2c\langle x, y\rangle + c^2\|x\|_2^2\|y\|_2^2}, \tag{6}$$

Mobius subtraction

$$x \ominus_c y = x \oplus_c (-y), \tag{7}$$

Mobius scalar multiplication

$$r \otimes_c x = \frac{1}{\sqrt{c}} \tanh(r \tanh^{-1}(\sqrt{c}\|x\|_2))\frac{x}{\|x\|_2}, \tag{8}$$

[1] We're implying a convention, that bilinear maps on $\mathbb{R}^n$ are identified with $n \times n$ matrices, and, thus, metric tensors are identified with Gramian matrices at each point.

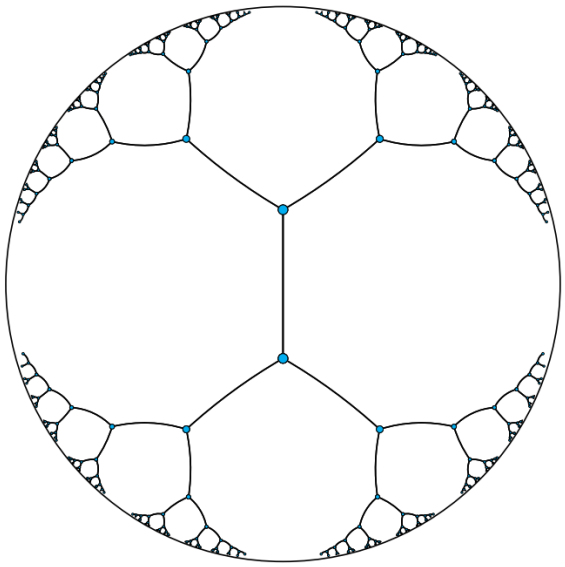

Figure 1. Poincare ball model. All the edges have the same hyperbolic length.

Mobius pointwise multiplication

$$\begin{aligned} &\operatorname{diag}(w) \otimes_c x = \\ &= \frac{1}{\sqrt{c}} \tanh\left(\frac{\|\operatorname{diag}(w)x\|_2}{x} \tanh^{-1}(\sqrt{c}\|x\|_2)\right) \frac{\|\operatorname{diag}(w)x\|_2}{\|x\|_2} \end{aligned} \tag{9}$$

Mobius "mat-vec"

$$M \otimes_c x = (1/\sqrt{c}) \tanh\left(\frac{\|Mx\|_2}{\|x\|_2} \tanh^{-1}(\sqrt{c}\|x\|_2)\right) \frac{Mx}{\|Mx\|_2}, \tag{10}$$

Mobius "function application"

$$f^{\otimes_c}(x) = \operatorname{Exp}_0^c(f(\operatorname{Log}_0^c(y))). \tag{11}$$

This works as follows: find a tangent vector represeting a geodesic curve from 0 to y via Log_0^c, apply f to the coordinate vector of $\operatorname{Log}_0^c y$, getting a new tangent vector, and follow this direction, obtaining the resulting manifold point via Exp_0^c.

We're not focusing on derivation of these operations, and refer the reader to the original paper (Ganea et al., 2018) and the seminal work (Ungar, 2009).

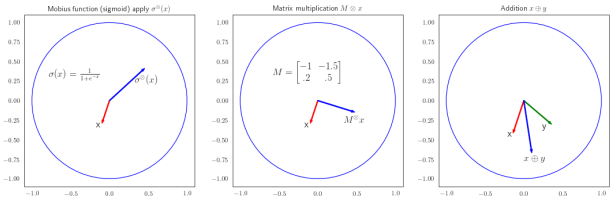

Figure 2. Example of different operations done on Poincare ball. The results were obtained using geoopt[3].

3. Hyperbolic RNN

Classical (Euclidean) GRU cell works as follows:

$$
\begin{aligned}
r_t &= \sigma(W^r h_{t-1} + U^r x_t + b^r) \\
z_t &= \sigma(W^z h_{t-1} + U^z x_t + b^z) \\
\tilde{h}_t &= \varphi(W(r_t \odot h_{t-1}) + U x_t + b) \\
h_t &= (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t
\end{aligned}
$$

Using (11) we we could try to "convert" a map $f : \mathbb{R}^n \times \mathbb{R}^p \to \mathbb{R}^m$ into a map between Poincare balls as [4]:

$$
f^{\otimes_c} : \mathcal{D}_c^n \times \mathcal{D}_c^p \ni (h, h') \mapsto \exp_0^c \left(f \left(\log_0^c(h), \log_0^c(h') \right) \right),
$$

Basing on that, authors end up with the following model:

$$
\begin{aligned}
r_t &= \sigma \log_0^c(W^r \otimes_c h_{t-1} \oplus U^r \otimes_c x_t \oplus_c b^r) \\
z_t &= \sigma \log_0^c(W^z \otimes_c h_{t-1} \oplus U^z \otimes_c x_t \oplus_c b^z) \\
\tilde{h}_t &= \varphi^{\otimes_c}((W \operatorname{diag}(r_t)) \otimes_c h_{t-1} \oplus_c U \otimes_c x_t \oplus b) \\
h_t &= h_{t-1} \oplus_c \operatorname{diag}(z_t) \otimes_c (-h_{t-1} \oplus \tilde{h}_t)
\end{aligned}
$$

where $\operatorname{diag}(x)$ is the squared diagonal matrix with x_i in the i'th diagonal entry and zeroes elsewhere, $\oplus$ is the Mobius addition. Mark non-associativity and non-commutativity of this operation.

It's easy to see that all the properties of usual GRU are preserved. Moreover, in the flat ($c = 0$) case we get exactly the classical GRU. Follow (Tallec & Ollivier, 2018) for more information.

4. Experiments

4.1. Dataset

As the authors of the original paper, we also tried evaluating our model on a toy dataset, with the task of detecting noisy prefixes. Specifically, given a pair of sentences, we need to detect if the second could be a prefix of the first, or if it's just a random sentence. The dataset has been prepared by the authors and we have re-used it for our experiments so that results be comparable. The structure of dataset PREFIX-Z% (Z being 10,30 or 50) is following: for each random sentence of random length at most 20 words and random prefix of it, a second sentence is generated by replacing Z% of the words of the prefix. Word vocabulary size is 100, we have 500K training, 10K validation and 10K test pairs.

4.2. Model

The model is pretty simple, we are embedding two sentences then feed their to the recurrent network (either with memory cells or not). Then the distance between the two hidden representations of sentences and the hidden representations are fed together to the FFNN (Euclidian of hyperbolic) and then fed to MLR that gives us probabilites of each classes. And on top of it all we have a cross entropy to get the loss.

4.3. Results

We have trained network both on hyperbolic GRU and Euclidean GRU and compared the results. Each the model was trained using `RAdam` (Bécigneul & Ganea, 2018; Kingma & Ba, 2014), we will later see experiments on different optimizer. The learning curve can be seen on Figure 3. We see that the curves differ significantly. Euclidean GRU starts from pretty high accuracy but Hyperbolic GRU, on the other hand, is not able to increase accuracy of validation set up into 8th epoch.

Table 1. Results for Hyperbolic and Euclidean GRUs with different batch sizes. We have increased batch size for Hyperbolic version got get the results in a feasible time frame.

Model	Value 2
Fully Euclidean GRU / B=64	93.25
Fully Hyperbolic GRU / B=1024	96.8

While the reason why the accuracy value for the Euclidean version of GRU starts from pretty high values is not easily intractable. We can say that the problem is easy enough and therefore standard realization of the algorithm can easily capture the relation even after the first epoch. But for the Hyperbolic realization we can relate the initial inactiveness of the curve to the batch size of the model. Due to our realization of hyperbolic cells we were not able to get the required results in meaningful time and, therefore, we increased the batch size up to 1024. Using which we were able to debug the model and get the validation results to show in the report. As it was analyzed by (Smith et al., 2018), batch size has a huge affect on the the training and it is, usually, a good practice to just increase it instead of finding the right learning rate.

[4] Assuming natural bases in tangent spaces and identifying them with coordinate spaces for simlicity

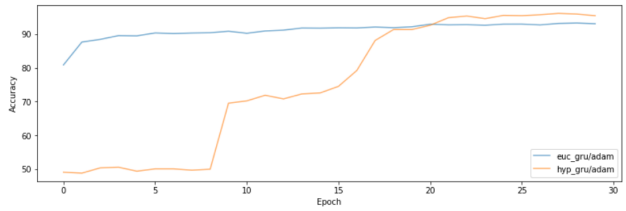

Figure 3. Learning curve of hyperbolic GRU and Euclidean GRU. We see that the Euclidean curve starts with much higher accuracy on start but still is not able to compete with hyperbolical version after only 30 epochs.

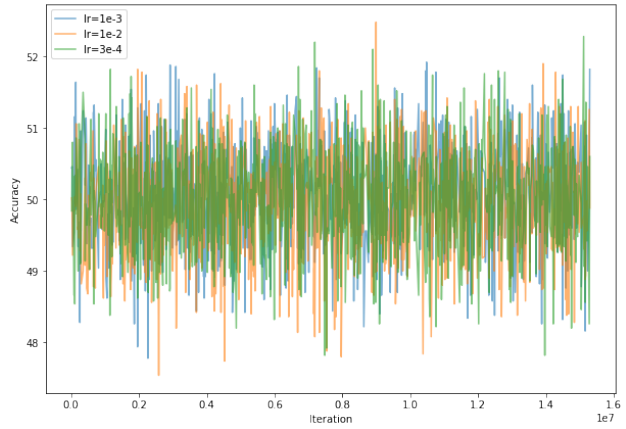

Figure 4. Training curve for `RSGDs` with different learning rate hyperparameter. We see a high instability of training. After 30 epochs we still do not see any improvements in accuracy.

4.4. Optimization choice

During our experiments we found that the `Riemannian Adam` is much better suitable to this problem. We compared `RAdam` with the `RSGD` and the results of former one were much more higher. Even more, it was hard to tune hyperparameters of `RSGD` to give us some meaningful results, usually we have seen high volatilities (see Figure 4) in the loss and the average accuracy during training didn't change by much. More experiments should be conducted to find the reason of this phenomena. But for now in the scope of this work, we can't say more.

4.5. Code

For performing experiments and testing our ideas, we have implemented various Riemannion optimization techniques based on (Bécigneul & Ganea, 2018). All the code is open source (Geoopt, 2019). Current status is as follows:

- ☒ Abstract interface for Riemannian manifolds, embedded in ambient $\mathbb{R}^n$.
- ☒ Compatible generic `RSGD`.
- ☒ Compatible generic `RAdam`.
- ☒ Compatible implementation of Poincare ball and Mobius arithmetics.
- ☒ Test coverage for optimization routines.
- ☒ GRU based on Mobius arithmetics, API-compatible with `torch.nn.GRU`.
- ☐ Layers parameterized by pivots of Log and Exp, as opposed to fixed pivot of 0 in Mobius arithmetics-based layers.
- ☐ Test coverage for "Mobius" layers and RNN loops.
- ☒ Numerical stability with `float64`.
- ☐ Numerical stability with `float32`
- ☐ Investigation of possibility of using `cudnn` loop.
- ☐ `C++` implementation of core operations and loops. [5]

5. Conclusion

We have implemented the models, conducted experiments and reached the results that were described in the original paper by the authors. Now, there are few directions to continue our research in this topic. First, and the most important thing from now on, is to optimize our programming code, making it parallel and more efficient, consequently. This is

[5] Authors hard-coded `float64`

the most important step, because we made a lot of effort to get these results, usually we waited for 10-20 hours for the model to finish its work. It is not possible for debugging and model evaluation in future research.

Another issues was that the `RSGD` didn't give the results we were waiting for. It was hard to tune parameters and usually the loss didn't decrease well even after 30 epochs. More experiments should be conducted in that direction to find the cause of this instability.

Overall, we have seen that the Hyperbolic GRUs are much more suitable for the task that we working on. We are planning to work on the programming code and making it more available for the scientists in the field to use it in their own experiments.

References

Bécigneul, G. and Ganea, O.-E. Riemannian adaptive optimization methods. *CoRR*, abs/1810.00760, 2018.

Bojanowski, P., Grave, E., Joulin, A., , and Mikolov, T. Enriching word vectors with subword information. 2016.

Cannon, J. W., Floyd, W. J., Kenyon, R., Parry, W. R., and et al. Hyperbolic geometry. 1997.

Ganea, O.-E., Bécigneul, G., and Hofmann, T. Hyperbolic neural networks. In *NeurIPS*, 2018.

Geoopt. Geoopt: a Riemannian optimization package for pytorch, 2019. URL https://github.com/geoopt/geoopt.

Gromov, M. Hyperbolic groups. 1987.

Grover, A. and Leskovec, J. node2vec: Scalable feature learning for networks. 2016.

Hamann, M. On the tree-likeness of hyperbolic spaces. 2017.

hyperbolicdeeplearning.com. Hyperbolic deep learning bibliography in chronological order, 2019. URL http://hyperbolicdeeplearning.com/?page_id=42.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. 2014.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G., , and Dean, J. Distributed representations of words and phrases and their compositionality. 2013.

Nickel, M. and Kiela, D. Poincaré embeddings for learning hierarchical representations. In *NIPS*, 2017.

Nickel, M., Jiang, X., , and Tresp, V. Reducing the rank in relational factoriza-tion models by including observable patterns. 2014.

Pennington, J., Socher, R., , and Mannin, C. D. Glove: Global vectors forword representation. 2014.

Perozzi, B., Al-Rfou, R., , and Skiena, S. Deepwalk: Online learning of social representations. 2014.

Smith, S. L., Kindermans, P.-J., Ying, C., and Le, Q. V. Don't decay the learning rate, increase the batch size. 2018.

Tallec, C. and Ollivier, Y. Can recurrent neural networks warp time? 2018.

Ungar, A. A. A gyrovector space approach to hyperbolic geometry. 2009.