$\mathcal{T}_p\mathcal{G}$ Geoopt: Riemannian Optimization in PyTorch

Max Kochurov [1] **Rasul Karimov** [1] **Serge Kozlukov** [2 1]

Abstract

Geoopt is a research-oriented modular open-source package for Riemannian Optimization in PyTorch. The core of Geoopt is a standard `Manifold` interface that allows for the generic implementation of optimization algorithms (Bécigneul & Ganea, 2018). Geoopt supports basic Riemannian SGD as well as adaptive optimization algorithms. Geoopt also provides several algorithms and arithmetic methods for supported manifolds, which allow composing geometry-aware neural network (Ganea et al., 2018; Liu et al., 2019; Brooks et al., 2019) layers that can be integrated with existing models.

```python
import geoopt
from geoopt.optim import (
    RiemannianAdam
)
manifold = geoopt.Stiefel()
orth_mat = geoopt.Parameter(
    manifold.random(10, 10)
)
opt = RiemannianAdam([orth_mat])
```

Figure 1. Creation of a manifold valued parameter.

1. Introduction

Geooptis built on top of PyTorch (Paszke et al., 2019), a dynamic computation graph backend. This allows us to use all the capabilities of PyTorch for geometric deep learning, including auto-differentiation, GPU acceleration, and exporting models (e.g., ONNX (Bai et al., 2019)). Geoopt optimizers implement the interface of native PyTorch optimizers and can serve as a drop-in replacement during training. The only difference is how parameters are declared[1], see Figure 1. The created manifold parameters can be used transparently with PyTorch functions and its serialization utils. All native PyTorch tensors by Geoopt optimizers are treated as regular Euclidean parameters.

The work on the package is mostly motivated by experiments with hyperbolic embeddings and hyperbolic neural networks. We provide several models of hyperbolic space, including the Poincarè ball model, the Hyperboloid model, and general κ-Stereographic model which generalizes Hyperbolic, Euclidean, and Spherical geometries (Bachmann et al., 2019).

2. Riemannian optimization

For a thorough introduction to geometry and differential geometry we refer to (Schuller, 2015b;a; Lee, 2006; 2013; Thurston, 1997), for synthetic description in general metric spaces to (Yokota, 2012), and concerned specifically with optimization and automatic differentiation (Betancourt; Absil et al., 2007; Elliott, 2018; Elliott).

Figure 2 visualizes a gradient descent step on the Poincaré disk. The concept of "directions" on a manifold corresponds to length-minimizing paths emanating from a point. Restricted to a single source point, these paths, in a delicate way, form a vector space, denoted $\mathcal{T}_p M$ and called the "tangent space" at point p. Given such a path segment X, we can obtain its destination point using the operation called "exponential map", $p_{t+1} = \exp X$. In a small neighbourhood, one can find a unique shortest path connecting one point to another – this is called the logarithmic map, $X = \log_{p_t} p_{t+1}$. The linear approximation (the derivative) of a function between manifolds is thus a linear map that takes directions in the input manifold into directions on the output manifold. For an objective function $\mathcal{J} : M \to \mathbb{R}$ this means that derivative at a point p_t is an operator $\mathcal{J}'(p_t) : \mathcal{T}_{p_t} M \to \mathbb{R}$, i.e a linear functional. Given an inner product (a Riemannian local metric) g_{p_t}, there is unique direction $\mathcal{J}'(p_t)^\flat \in \mathcal{T}_{p_t} M$ that corresponds to this linear functional, in such a way that $\mathcal{J}'(p_t) = g_{p_t}\left(\mathcal{J}'(p_t)^\flat, \cdot\right)$, assuming convenient placeholder notation. It is the sought for ascent direction. Thus

[1]Skolkovo Institute of Science and Technology, Moscow, Russia [2]HSE University, Russian Federation Skolkovo Institute of Science and Technology, Russian Federation. Correspondence to: Max Kochurov <maxim.v.kochurov@gmail.com>.

Proceedings of the 37^{th} International Conference on Machine Learning, Vienna, Austria, PMLR 108, 2020. Copyright 2020 by the author(s).

[1]More examples can be found here: `https://github.com/geoopt/geoopt/tree/master/examples`

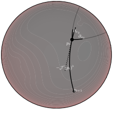

Figure 2. A gradient descent step on the Poincaré disk. Contour lines visualize the objective function; p_t is the current estimate; $-\flat(\mathrm{d}\,\mathcal{J})$ is the descent direction, visualized as a geodesic curve; p_{t+1} is the final point of that curve and the new estimate; ∂_{x_1}, ∂_{x_2} are basis vectors in the space of directions at p_t; stroked line visualizes the (downscaled) "Euclidean" gradient.

the update rule is

$$p_{t+1} = \exp(-\eta \mathcal{J}'(p_t)^\flat),$$

where $\eta \in \mathbb{R}$ is the learning rate.

In Geoopt, points and directions are numerically represented using embeddings of manifolds into ambient vector spaces (often embedding is the identity map). Objective functions, too, are defined in this ambient space. Using PyTorch's `backward` we can obtain the derivative of this "extended" function, acting on "Euclidean" directions. As an embedding map allows to "push" a direction on the manifold into a direction in the ambient vector space, this "Euclidean" derivative naturally corresponds to a linear functional acting on directions on the manifold (which pushes directions to ambient space and applies the "Euclidean" derivative). This functional is exactly the derivative of our original objective function defined on the manifold, and we can use the inner product to convert it into the ascent direction, as discussed earlier. This whole procedure – the transition from ambient space to the manifold, and application of inner product – is performed in Geoopt by a single operation, `egrad2rgrad`.

3. Design goals

Optimization on manifolds is a fairly general problem and designing a general-purpose package accounting for possible use-cases may not be a tractable problem. Geoopt is specifically concerned with geometric deep learning *research* and its development is guided by a couple of rather pragmatic principles:

1. **Smooth integration with the PyTorch ecosystem.** This assumes "familiar" PyTorch-esque interfaces. For instance, `geoopt.optim` optimizers can serve as drop-in replacements of `torch.optim`. This also implies compatibility with third-party packages based on PyTorch, for example, experiment management systems (Falcon, 2019; Kolesnikov, 2018).

2. **Broadcasting.** Support broadcasting for all operations and broadcasting semantics for product manifolds.

3. **Robustness and numerical stability.** Hyperbolic models such as Poincaré disk and the Lorentz model have an unbounded numerical error as points get far from the origin. Therefore it is important that Geoopt users don't have to deal with more `NaN`s that they would have to otherwise. Whenever possible, algorithms in Geoopt are implemented to work even with `float32` precision. The instabilities of specific functions are described in documentation appropriately.

4. **Efficiency and extendibility.** The previous bullets are concerned with "not getting in the way". When those are satisfied, we strive to provide reasonable efficiency and leave place for extendibility.

4. Implementation details

The basic primitive of Geoopt is `geoopt.ManifoldTensor` which is a "tensor" (a multi-dimensional array) that stores a reference to its containing `geoopt.Manifold`. We inherit from `torch.Tensor` and `torch.nn.Parameter`. This ensures compatibility with the rest of PyTorch ecosystem and suggests just one "right way" to use Geoopt within PyTorch code, which we consider Pythonic (van Rossum et al., 2001).

Array manipulations in Geoopt should support broadcasting. Simple product manifolds are implemented with broadcast along first dimensions, by convention. More complex cases are handled by `geoopt.ProductManifold` class.

The original goal of Geoopt is Riemannian optimization, and it is supposed to be efficient: this requires optimizations in the update step, merging retractions followed by parallel transport, etc. In product manifolds, the adaptive term is computed per manifold parameter, and product structure is exploited (Bécigneul & Ganea, 2018). This is a part of Geoopt in the first place, and any possibility to make effective use of the adaptive term is implemented.

The `geoopt.Manifold` base class describes a methodset expected by `geoopt.optim` optimizers. The `geoopt.Manifold` inherits from `torch.nn.Module`: this way it is captured by `state_dict()` and its parameters can be optimized for.

The minimal methodset for the `geoopt.Manifold` subclass includes:

- `Retr`action: retr takes an array of points, an array of tangent vectors at these points, and outputs an array of points. Retraction is a first-order approximation of the exponential map used in optimization, and often we have a separate `expmap` method. However, for some manifolds, we provide variants that perform the actual exponential map instead of retraction during optimization.
- Vector `transp`ort: transp takes an array of source points, an array of target points, an array of tangent vectors attached to source points, and produces an array of tangent vectors at target points. It is the first-order approximation of parallel transport.
- `Inner` product: inner takes an array of points and two arrays of tangent vectors at these points and returns an array of inner products of those vectors.
- `egrad2rgrad` is used to convert the covector in the ambient vector space (produced by PyTorch's `backward`) into a corresponding tangent vector on the actual manifold.

Points and tangent vectors in Geoopt are always represented by coordinates in the (assumed) ambient vector space. In case of `PoincareBall`, the embedding coincides with the natural global chart, and corresponds to the chart-induced basis vector fields. Such consistency is only possible because of negative curvature of Hyperbolic space and conformality of Poincaré Ball. On a sphere, one could neither allocate a non-vanishing smooth vector field, nor expect unique geodesics to exist between all points, nor measures to have unique barycentres. For this reason, on a Sphere one has to either use local charts or take on the extrinsic approach (assume an ambient vector space, which is what we do). The array of numbers representing a tangent vector (e.g., one gets after taking a logarithmic map) in Geoopt stores the coordinates of the push-forward of that vector under the assumed embedding into ambient vector space. This representation is somewhat restrictive (e.g., it complicates implementing the tiling-based parameterizations of Hyperbolic space (Yu & De Sa, 2019)) but rather convenient and follows the spirit of (Bécigneul & Ganea, 2018).

To extend Geoopt, one should implement basic methods such as retraction or exponential map on the manifold, parallel or vector transport for tangent vectors, and make them properly broadcastable. The latter might be the hardest in implementation, and as maintainers, we are more than ready to help with it.

5. Features

To help researches Geoopt has implementation of standard manifolds (Absil et al., 2007):

- `geoopt.Sphere` manifold – for unit norm constrained problems (embeddings, eigenvalue problems)
$$\mathbb{S} = \{x \in \mathbb{R}^n : \|x\| = 1\} \tag{1}$$
- `geoopt.Stiefel` manifold – for basis reconstruction
$$\mathrm{St} = \left\{X \in \mathbb{R}^{n\times m} : X^\top X = I\right\} \tag{2}$$
- `geoopt.BirkhoffPolytope` (Douik & Hassibi, 2018) – for inferring permutations in data
$$\mathbb{B} = \left\{X \in \mathbb{R}^{n\times n} : \mathbf{1}^\top X = \mathbf{1} = X\mathbf{1}\right\} \tag{3}$$
- `geoopt.Stereographic` model (Bachmann et al., 2019) and `geoopt.Lorentz` manifold – for Hyperbolic deep learning
- `geoopt.Product` and `geoopt.Scaled` manifolds – to combine and extend any of above

Geoopt supports most important and widely used optimizers:

- `geoopt.optim.RiemannianAdam` – a Riemannian version for popular Adam optimizer (Kingma & Ba, 2014)
- `geoopt.optim.SparseRiemannianAdam` – Adam implementation to support sparse gradients
- `geoopt.optim.RiemannianSGD` – SGD with (Nesterov) momentum implementation
- `geoopt.optim.SparseRiemannianSGD` – SGD implementation that supports sparse gradients

6. Advanced Usage

The advanced usage of Geoopt covers Hyperbolic deep learning pioneered in recent years (Sun et al., 2015; Nickel & Kiela, 2017; Sa et al., 2018; Gromov, 1987; Dhingra et al., 2017). In Geoopt, we provide a robust implementation for the Poincare Ball model along with methods for performing supplementary math. In addition to constant negative curvature support, positive curvature stereographic model of a sphere is also a part of the unified implementation of Möbius arithmetics in projected spacetime domain. Users can find supplementary functions as methods of `geoopt.Stereographic` class. Derivatives for curvature are supported by the whole domain, especially for zero curvature case, so curvature optimization is possible.

6.1. Other Applications

Geoopt is a general-purpose optimization library for PyTorch. Manifold optimization appears in many applications.

Language models. For example, in NLP, when training recurrent neural networks, it is useful to constraint the transition matrix to be unitary (Arjovsky et al., 2015). The unitary matrix keeps the gradient norm unchanged, and the network is able to learn long-range dependencies. Unitary matrices form a smooth Riemannian manifold, and Riemannian optimization can be easily applied to them. Another kind of constrained parameterization used in RNNs is Stiefel manifold (Helfrich et al., 2017). It also helps to avoid problems of vanishing or exploding gradients.

Computer vision. In the field of computer vision, doubly stochastic matrices can be used to match keypoints between views (Birdal & Simsekli, 2019). In (Birdal & Simsekli, 2019) the probabilistic approach was proposed to compare images from a completely different time and viewpoints. To calculate uncertainty bounds, MCMC is run over the solution space. Combined with cycle consistency energy function method is available not only to match keypoints but also to provide estimates guiding to pick the most promising connections.

Time series. For multidimensional time series analysis and classification, it was shown promising to look at the covariance matrix of stationary representation. The covariance matrix is passed to SPD neural networks that perform final classification (Nguyen et al., 2019; Brooks et al., 2019), e.g., processes or gestures. The approach proposed in (Brooks et al., 2019) allows Riemannian batch normalization for SPD matrices, further improving time series classification benchmarks and training stability.

Hyperbolic deep learning. An active area of research is using hyperbolic representations to account for "implicit hierarchical relationships" in data. Geoopt allows for optimization with parameters in several models of real Hyperbolic spaces, and provides basic operations of hyperbolic geometry. Hyperbolic embeddings appear in NLP (Balažević et al., 2019; Nickel & Kiela, 2017), image understanding (Khrulkov et al., 2019), and general representation learning (Gu et al., 2019). Some works also focus on graph learning tasks (Chami et al., 2019; Liu et al., 2019; Bachmann et al., 2019) and extend the message passing framework proposed by (Fey & Lenssen, 2019). With Geoopt, implementation of such extensions become simpler, as demonstrated by (Chami et al., 2019). An extensible implementation of Hyperbolic message passing framework may rely on `torch_geometric` library modifying `aggregate` method in `MessagePassing` class.

Summary. Riemannian optimization is important for current research in geometric deep learning. Geoopt tries to fill the niche of Riemannian optimization in PyTorch. The library has helped to conduct research in computer vision (Khrulkov et al., 2019; Birdal & Simsekli, 2019; Chen et al., 2019), navigation (Comer et al., 2020), optimal transport (Titouan et al., 2019), time-series analysis (Vayer et al., 2020), and Hyperbolic deep learning (Shen et al., 2020; Skopek et al., 2020; Alvarez-Melis et al., 2019; Chami et al., 2019).

7. Related projects

There were other Riemannian optimization projects prior to Geoopt. Notable examples include PyManOpt (Townsend et al., 2016) and GeomStats (Miolane et al., 2020). The main distinction between Geoopt and other solutions is interface-wise. PyManOpt is a Python reimplementation of the original Manopt (Boumal et al., 2014) and follows the original interface closely with its `solver.solve(Problem(manifold, cost))` semantics. PyManOpt currently provides an admittedly broader collection of algorithms (trusted region methods, Nelder-Mead, etc) and manifolds than Geoopt. Manopt is the MATLAB package accompanying the Absil's book (Absil et al., 2007). Geomstats is designed around sklearn's `fit-transform` semantics. Both solutions are great general-purpose tools for Riemannian optimization. Geoopt is concerned explicitly with neural networks and geometric deep learning: its interfaces are designed to integrate well with PyTorch-based projects. Geoopt users define neural networks and cost functions in the usual "PyTorch" way and don't have to construct a PyManOpt `Problem`. In this aspect, similar to Geoopt is McTorch. It takes on the approach of forking PyTorch and extending it on the C++ back-end side. This is heavy on infrastructure. Maintaining a fork up to date demands a considerable and continuous effort. Using a fork complicates integration with other third-party libraries, which could pin to specific versions of PyTorch. It could complicate it to the point that one runs into the task of re-compilation of entire PyTorch and further distribution of binary packages. Geoopt avoids such infra-structural costs and aims to keep the bar low – both for new contributors and users.

References

Absil, P.-A., Mahony, R., and Sepulchre, R. *Optimization Algorithms on Matrix Manifolds*. Princeton University Press, USA, 2007. ISBN 0691132984.

Alvarez-Melis, D., Mroueh, Y., and Jaakkola, T. S. Unsupervised hierarchy matching with optimal transport over hyperbolic spaces. In *OTML Workshop in Advances in Neural Information Processing Systems*, 2019. URL https://arxiv.org/abs/1911.02536.

Arjovsky, M., Shah, A., and Bengio, Y. Unitary evolution recurrent neural networks, 2015.

Bachmann, G., Bécigneul, G., and Ganea, O.-E. Constant curvature graph convolutional networks. Nov 2019. URL http://arxiv.org/abs/1911.05076v1.

Bai, J., Lu, F., Zhang, K., et al. Onnx: Open neural network exchange. https://github.com/onnx/onnx, 2019.

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

Betancourt, M. Tweet: "i am continuously amazed at how delightfully illuminating differential geometry is. if everyone listened to it then we would have so many fewer janky algorithms out there". URL https://twitter.com/betanalpha/status/1246150679976083456.

Birdal, T. and Simsekli, U. Probabilistic permutation synchronization using the riemannian structure of the birkhoff polytope. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 11105–11116, 2019.

Boumal, N., Mishra, B., Absil, P.-A., and Sepulchre, R. Manopt, a Matlab toolbox for optimization on manifolds. *Journal of Machine Learning Research*, 15(42):1455–1459, 2014. URL https://www.manopt.org.

Brooks, D., Schwander, O., Barbaresco, F., Schneider, J.-Y., and Cord, M. Riemannian batch normalization for spd neural networks. Sep 2019. URL http://arxiv.org/abs/1909.02414v2.

Bécigneul, G. and Ganea, O.-E. Riemannian adaptive optimization methods. Oct 2018. URL http://arxiv.org/abs/1810.00760v2.

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.

Chen, B., Huang, X., Xiao, L., Cai, Z., and Jing, L. Hyperbolic interaction model for hierarchical multi-label classification. *arXiv preprint arXiv:1905.10802*, 2019.

Comer, J. F., Andrews, R. W., Naderializadeh, N., Kolouri, S., and Hoffman, H. SAR automatic target recognition with less labels. In Hammoud, R. I., Overman, T. L., and Mahalanobis, A. (eds.), *Automatic Target Recognition XXX*, volume 11394, pp. 73 – 83. International Society for Optics and Photonics, SPIE, 2020. doi: 10.1117/12.2564875. URL https://doi.org/10.1117/12.2564875.

Dhingra, B., Shallue, C. J., Norouzi, M., and Andrew M. Dai, G. E. D. Poincaré embeddings for learning hierarchical representations. 2017. URL https://arxiv.org/pdf/1705.08039.

Douik, A. and Hassibi, B. Manifold optimization over the set of doubly stochastic matrices: A second-order geometry. Technical report, 2018.

Elliott, C. Beautiful differentiation (extended version). Technical report, Technical Report 2009-02, LambdaPix, March 2009a. URL http://conal. net . . . .

Elliott, C. The simple essence of automatic differentiation, 2018.

Falcon, W. Pytorch lightning. *GitHub. Note: https://github.com/williamFalcon/pytorch-lightning Cited by*, 3, 2019.

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

Ganea, O., Becigneul, G., and Hofmann, T. Hyperbolic neural networks. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 31*, pp. 5345–5355. Curran Associates, Inc., 2018. URL http://papers.nips.cc/paper/7780-hyperbolic-neural-networks.pdf.

Gromov, M. Hyperbolic groups. In *Essays in group theory*, pp. 75–263. Springer, 1987.

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

Helfrich, K., Willmott, D., and Ye, Q. Orthogonal recurrent neural networks with scaled cayley transform, 2017.

Khrulkov, V., Mirvakhabova, L., Ustinova, E., Oseledets, I., and Lempitsky, V. Hyperbolic image embeddings. *arXiv preprint arXiv:1904.02239*, 2019.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. Dec 2014. URL http://arxiv.org/abs/1412.6980v9.

Kolesnikov, S. Accelerated dl r&d. https://github.com/catalyst-team/catalyst, 2018.

Lee, J. M. *Riemannian manifolds: an introduction to curvature*, volume 176. Springer Science & Business Media, 2006.

Lee, J. M. Smooth manifolds. In *Introduction to Smooth Manifolds*, pp. 1–31. Springer, 2013.

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.

Miolane, N., Brigant, A. L., Mathe, J., Hou, B., Guigui, N., Thanwerdas, Y., Heyder, S., Peltre, O., Koep, N., Zaatiti, H., Hajri, H., Cabanes, Y., Gerald, T., Chauchat, P., Shewmake, C., Kainz, B., Donnat, C., Holmes, S., and Pennec, X. Geomstats: A python package for riemannian geometry in machine learning, 2020.

Nguyen, X., Brun, L., Lezoray, O., and Bougleux, S. A neural network based on spd manifold learning for skeleton-based hand gesture recognition, 04 2019.

Nickel, M. and Kiela, D. Poincaré embeddings for learning hierarchical representations. In Guyon, I., Luxburg, U., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 30*. 2017.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library. Dec 2019. URL http://arxiv.org/abs/1912.01703v1.

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

Schuller, F. Lectures on the geometric anatomy of theoretical physics, 2015a. URL https://www.youtube.com/playlist?list=PLPH7f_7ZlzxTi6kS4vCmv4ZKm9u8g5yic. Unofficial typeset notes: http://mathswithphysics.blogspot.com/.

Schuller, F. The we-heraeus international winter school on gravity and light, 2015b. URL https://www.youtube.com/playlist?list=PLFeEvEPtX_0S6vxxiiNPrJbLu9aK1UVC_.

Shen, H. T., Long, T., Mettes, P., and Snoek, C. Searching for actions on the hyperbole. In *IEEE International Conference on Computer Vision and Pattern Recognition*, 2020.

Skopek, O., Ganea, O.-E., and Bécigneul, G. Mixed-curvature variational autoencoders. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=S1g6xeSKDS.

Sun, K., Wang, J., Kalousis, A., and Marchand-Maillet, S. Space-time local embeddings. In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 28*, pp. 100–108. Curran Associates, Inc., 2015. URL http://papers.nips.cc/paper/5971-space-time-local-embeddings.pdf.

Thurston, W. P. *Three-dimensional geometry and topology*, volume 35. Princeton university press, 1997.

Titouan, V., Flamary, R., Courty, N., Tavenard, R., and Chapel, L. Sliced gromov-wasserstein. 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. 14753–14763. Curran Associates, Inc., 2019. URL http://papers.nips.cc/paper/9615-sliced-gromov-wasserstein.pdf.

Townsend, J., Koep, N., and Weichwald, S. Pymanopt: A python toolbox for optimization on manifolds using automatic differentiation. *Journal of Machine Learning Research*, 17(137):1–5, 2016. URL http://jmlr.org/papers/v17/16-177.html.

van Rossum, G., Warsaw, B., and Coghlan, N. Pep-8, 2001. URL https://www.python.org/dev/peps/pep-0008/.

Vayer, T., Chapel, L., Courty, N., Flamary, R., Soullard, Y., and Tavenard, R. Time series alignment with global invariances, 2020.

Yokota, T. A rigidity theorem in alexandrov spaces with lower curvature bound. *Mathematische Annalen*, 353(2): 305–331, 2012.

Yu, T. and De Sa, C. M. Numerically accurate hyperbolic embeddings using tiling-based models. In *Advances in Neural Information Processing Systems*, pp. 2021–2031, 2019.