Deep Riemannian Manifold Learning

Aaron Lou[1*] **Maximilian Nickel**[2] **Brandon Amos**[2]
[1]Cornell University [2] Facebook AI

Abstract

We present a new class of learnable Riemannian manifolds with a metric parameterized by a deep neural network. The core manifold operations–specifically the Riemannian exponential and logarithmic maps–are solved using approximate numerical techniques. Input and parameter gradients are computed with an adjoint sensitivity analysis. This enables us to fit geodesics and distances with gradient-based optimization of both on-manifold values **and** the manifold itself. We demonstrate our method's capability to model smooth, flexible metric structures in graph embedding tasks.

1 Introduction

Geometric domain knowledge is important for machine learning systems to interact, process, and produce data from inherently geometric spaces and phenomena [BBL+17, MBM+17]. Geometric knowledge often empowers the model with explicit operations and components that reflect the geometric properties of the domain.

One commonly used construct is a Riemannian manifold, the generalization of standard Euclidean space. These structures enable models to represent non-trivial geometric properties of data [NK17, DFDC+18, GDM+18, NK18, GBH18]. However, despite this capability, the current set of usable Riemannian manifolds is surprisingly small, as the core manifold operations must be provided in a closed form to enable gradient-based optimization. Worse still, these manifolds are often set a priori, as the techniques for interpolating between manifolds are extremely restricted [GSGR19, SGB20].

In this work, we study how to integrate a general class of Riemannian manifolds into learning systems. Specifically, we parameterize a class of Riemannian manifolds with a deep neural network and develop techniques to optimize through the manifold operations. We apply this manifold learning method in the context of graph embeddings to accurately reconstruct distances.

2 Background on Riemannian geometry

We assume that our readers are sufficiently familiar with the core definitions of Riemannian geometry (including manifolds, Riemannian metrics, and the Riemannian exponential and logarithmic maps). For a full mathematical introduction or for the explicit definitions and notations we used in this paper, we point interested readers to app. A.

We outline the formulation for geodesics, which are integral to our core computations. Suppose we are given a Riemannian manifold $(\mathcal{M}, g)$ and a set of local coordinates x^i. A curve $\gamma : I \to \mathcal{M}$ is a **geodesic** if it satisfies the second-order ordinary differential equation (ODE) in eq. (1) for all indices $i, j, k \in [n]$

$$\frac{d^2\gamma^i}{dt^2} + \Gamma^i_{jk} \frac{d\gamma^j}{dt} \frac{d\gamma^k}{dt} = 0, \tag{1}$$

*Work performed during an internship at Facebook AI

where γ^i are the local coordinate components of γ and the **Christoffel symbols** Γ^i_{jk} are defined by

$$\Gamma^i_{jk} := \sum_l \frac{1}{2} g^{il} \left(\frac{\partial g_{jl}}{\partial x^k} + \frac{\partial g_{kl}}{\partial x^j} - \frac{\partial g_{jk}}{\partial x^l} \right). \tag{2}$$

3 Related work

Our work primarily relates to problems in geometric machine learning. Several previous works exist for building systems for predefined manifolds [NK17, NK18, GSGR19, GDM+18, GBH18, LKJ+20, LLK+20, MN20]. Others attempt to learn a manifold structure through neural networks [BC20, AHH17, SKF18]. Our method augments both types of paper. For the first type we expand the scope of tractable Riemannian manifolds, and, for the second type, we allow for deep metric constructions independent of the manifold structure.

Our work can also be viewed as a deep metric learning method. There exist several approaches which construct deep metrics directly [HA15, VBL+16, SSZ17, MBL20, DSGRS18] and even ones which structure neural networks for metrics [PCJB20]. By contrast, our method constructs deep metrics endowed with Riemannian geometry, which allow for several desirable properties such as smoothness and interpretability. While other papers have explored usages of Riemannian geometry for metric learning [HFB12], ours is the first to unify this with the generality of deep neural networks.

4 Deep Riemannian Manifolds

We present our construction of a Riemannian manifold $(\mathcal{M}, g)$ parameterized by a neural network.

4.1 Manifold Construction

The manifold $\mathcal{M}$ will be constructed as an embedded submanifold parameterized by a **single chart** $\varphi : \mathbb{R}^n \to \mathcal{M} \subseteq \mathbb{R}^D$, where φ is taken to be any type of diffeomorphic neural network and $D \geq n$.

For all points $x \in \mathcal{M}$ we can directly model the local behavior through φ. Specifically, we can construct a chart φ_x centered at x by taking $\varphi_x(\cdot) := \varphi(x + \cdot)$, allowing for local coordinates $x_i := \varphi_x(e_i)$. Tangent vectors can either be computed *implicitly* as the dual to the local coordinates (in particular as vectors of $\mathbb{R}^n$), or can be modeled *explicitly* as tangent vectors of $T_x\mathcal{M} := D_x\varphi(\mathbb{R}^n)$.

Our approach is similar to previous works such as [BC20, SKF18], which also construct manifolds through a single chart. We will find that the local coordinates provide a nice computational frame for our metrics, so we will henceforth reference manifold values only by their $\mathbb{R}^n$ counterpart.

4.2 Metric Formulation

To construct the metric g, we first note that in local coordinates each g_x is an symmetric positive definite (SPD) matrix. Since our local coordinates remain fixed by our single chart construction, this means that our function $x \to g_x$ is in fact a smooth function from $\mathbb{R}^n \to \mathcal{S}_n^+$ where $\mathcal{S}_n^+$ is the space of $n \times n$ SPD matrices.

To parameterize this function with a neural network, we note that the matrix exponential map $\exp$ acts as a bijection between $\mathcal{S}_n$, the space of symmetric matrices, and $\mathcal{S}_n^+$. Let $\mathrm{sym} : \mathbb{R}^{\frac{n(n+1)}{2}} \to \mathcal{S}_n$ be the function which takes a $\mathbb{R}^{\frac{n(n+1)}{2}}$ vector and produces the corresponding symmetric matrix in $\mathbb{R}^{n \times n}$. Given a neural network $f_{\mathrm{nn}} : \mathbb{R}^n \to \mathbb{R}^{\frac{n(n+1)}{2}}$, then we define our deep metric as

$$g^d_x := (\exp \circ \mathrm{sym} \circ f_{\mathrm{nn}})(x).$$

We note that our construction is indeed a Riemannian metric, as it is a smooth inner product for tangent vectors. Furthermore, it allows for some notion of universal approximation.

Prop 4.1 (Universal Approximation of Riemannian Metrics)**.** *On a compact set of $\mathbb{R}^n$, our deep metric is capable to approximating any Riemannian metric on our single-chart manifolds.*

This follows since neural networks are universal approximators and the matrix exponential is bounded on a bounded set. We present this fully in app. B.1

5 Numerical Riemannian Manifold Operations

For our deep Riemannian manifold, the core operations of the exponential map, logarithmic map, and distance have no closed form. This means that these functions must be solved numerically. Furthermore, as there are no closed-form solutions, we must also develop differentiation techniques for our operations.

5.1 Exponential Map

The exponential map $\exp_x(v)$ is the time 1 solution to eq. (1) with initial value x and velocity v.

For our deep manifold, x, v and $\exp_x(v)$ are all elements of $\mathbb{R}^n$. Therefore, solving our geodesic equation with the initial value becomes a standard numerical ODE problem. By linearizing the ODE, our exponential map becomes a first order initial value problem (IVP), so computation and differentiation can be implemented with a Neural ODE [CRBD18].

5.2 Logarithmic Map

The log map $\log_x(y)$ is the initial value v for which $\exp_x(v) = y$. This is a boundary value problem (BVP), and, for our deep manifolds, reduces to a BVP on Euclidean space. As opposed to IVPs, these problems are conceptually harder, more difficult to solve, and lack a clean adjoint sensitivity analysis.

Computation There exist many computational methods for BVP problems. Common methods include shooting method and the the grid solver [SB02], and modern numerical solvers such as the Matlab bvp5c solver can even control for error [KS08]. However, for our purposes we found that using the BVP solver in [AHHS19], which we reimplement for batched computation in PyTorch [PGM+19], provided a faster and more stable computation.

Differentiation To calculate gradients for the input values and neural network parameters, we develop an adjoint sensitivity analysis for BVPs. We present the proof in app. B.2.

Theorem 5.1 (Adjoint Sensitivity For BVPs)**.** *Suppose we are given a BVP $B(x, y)$ with corresponding IVP $I(x, v)$ s.t. $I(x, B(x, y)) = y$. Suppose our solution to $B(x_0, y_0)$ is b. Then*

$$D_x B(x_0, y_0) = -D_v I(x_0, b)^{-1} \circ D_x I(x_0, b) \qquad D_y B(x_0, y_0) = -D_v I(x_0, b)^{-1} \tag{3}$$

Note that the derivatives of I can be computed through the adjoint-sensitivity analysis. Furthermore, if our ODE problems are controlled by some parameter θ, then

$$D_\theta B(x_0, y_0) = -D_v I(x_0, b)^{-1} \circ D_\theta I(x_0, b)$$

5.3 Other Manifold Operations

Note that distance has the property that $d_g(x, y) = \left\|\log_x(y)\right\|_g$ and that interpolation can be constructed directly with exp/log maps $\gamma_{x \to y}(t) = \exp_x(t \log_x(y))$. Since all of these components are differentiable with respect to x, y and neural network parameters θ, AutoDiff takes care of these derivatives [L+18].

6 Experiments

We experimentally validate our deep manifold formulation for graph embeddings.

6.1 Graph embeddings

In this experiment we embed graphs into our deep Riemannian manifolds. Since our manifolds are able to represent more complex geometric structures than previous methods, we expect that our

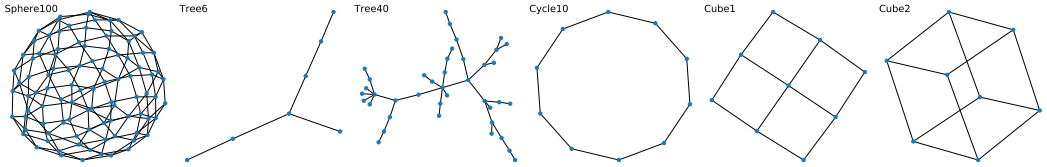

Figure 1: The synthetic graphs we consider from [CBG20].

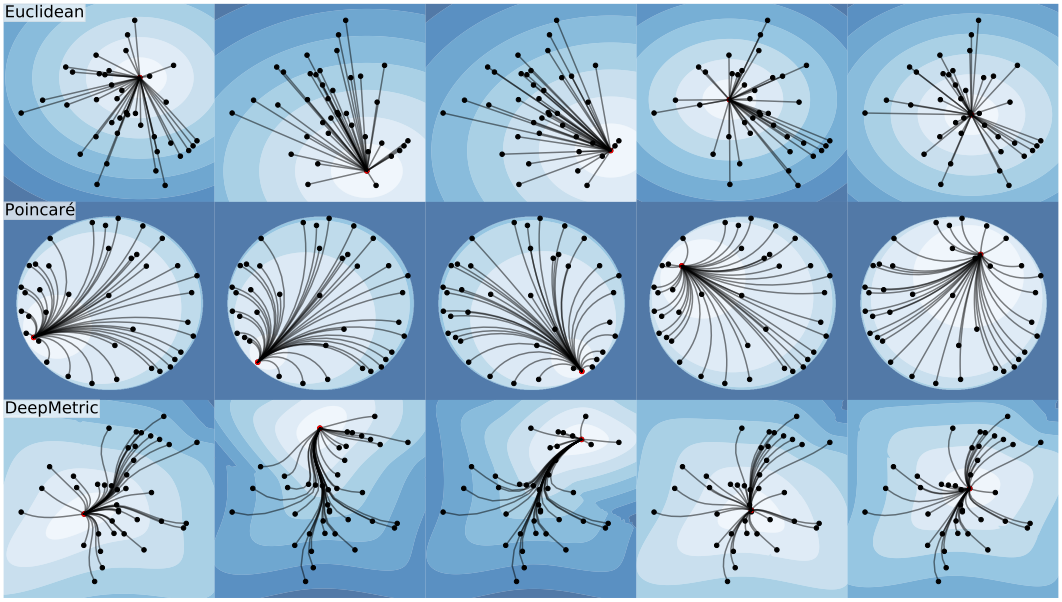

Figure 2: Geodesics and 2D embeddings on the Tree40 graph, where each column correspond to the geodesics connected to a point (highlighted in red). The contours show the geodesic distances. The deep metric accurately captures the node distances and induces non-trivial geodesics.

Table 1: Distortions for 2D, 5D, and 10D embeddings of the graph distances over 3 trials.

	Sphere100			**Tree6**			**Tree40**			**Cycle10**			**Cube1**			**Cube2**		
	2	5	10	2	5	10	2	5	10	2	5	10	2	5	10	2	5	10
Euclidean	42.64± 21.88	2.85± 0.14	2.82± 0.08	3.14± 2.80	1.43± 0.07	1.51± 0.10	44.64± 19.64	7.85± 1.26	5.62± 0.39	1.67± 0.06	1.91± 0.12	1.79± 0.04	6.21± 1.27	1.83± 0.09	1.74± 0.16	4.37± 0.54	2.05± 0.07	1.98± 0.09
PoincareBall	12.11± 0.60	2.42± 0.04	2.64± 0.05	1.65± 0.04	1.63± 0.00	1.63± 0.00	8.65± 3.14	3.45± 0.12	2.38± 0.12	1.93± 0.00	1.93± 0.00	1.94± 0.00	1.80± 0.01	1.81± 0.01	1.81± 0.01	2.66± 0.31	2.17± 0.01	2.16± 0.01
Sphere	3.86± 0.00	2.42± 0.00	2.56± 0.01	1.69± 0.02	1.71± 0.01	1.71± 0.00	11.61± 0.94	6.83± 0.90	5.51± 0.23	1.72± 0.00	1.73± 0.00	1.73± 0.00	1.78± 0.05	1.76± 0.01	1.76± 0.01	2.48± 0.04	2.15± 0.05	2.10± 0.01
DeepManifold	102.74± 62.74	2.41± 0.01	2.51± 0.06	2.01± 0.80	1.13± 0.04	1.13± 0.05	71.57± 15.44	4.47± 0.65	2.51± 0.37	8.23± 5.69	1.77± 0.55	1.34± 0.07	5.99± 1.49	1.55± 0.15	1.53± 0.09	3.69± 1.02	1.65± 0.03	1.58± 0.05

embeddings will be more faithful and have lower distortion. We consider the synthetic graphs from [CBG20], we which we visualize in sect. 6.

We present our experimental details in app. C. In table 1, we report the distortion of our embeddings, noting how our manifolds are generally capable of producing better embeddings. Geodesics are visualized in fig. 2; note how our method produces complex geodesics for the data geometry.

7 Conclusion

We have presented deep Riemannian manifolds, a type of Riemannian manifold with a metric parameterized by a deep neural network, and have shown how to optimize both manifold-valued data as well as the manifold itself. We hope our work enables future geometric machine learning models to better account for more general structures.

References

[AHH17] Georgios Arvanitidis, Lars Kai Hansen, and Søren Hauberg. Latent space oddity: on the curvature of deep generative models. *arXiv preprint arXiv:1710.11379*, 2017.

[AHHS19] Georgios Arvanitidis, Søren Hauberg, Philipp Hennig, and Michael Schober. Fast and robust shortest paths on manifolds learned from data. *arXiv preprint arXiv:1901.07229*, 2019.

[BBL+17] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

[BC20] Johann Brehmer and Kyle Cranmer. Flows for simultaneous manifold learning and density estimation. *arXiv preprint arXiv:2003.13913*, 2020.

[CBG20] Calin Cruceru, Gary Bécigneul, and Octavian-Eugen Ganea. Computationally tractable riemannian manifolds for graph embeddings. *arXiv preprint arXiv:2002.08665*, 2020.

[CRBD18] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 6571–6583. Curran Associates, Inc., 2018.

[Cyb89] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, 1989.

[DC16] Manfredo P Do Carmo. *Differential geometry of curves and surfaces: revised and updated second edition*. Courier Dover Publications, 2016.

[DFDC+18] Tim R Davidson, Luca Falorsi, Nicola De Cao, Thomas Kipf, and Jakub M Tomczak. Hyperspherical variational auto-encoders. *arXiv preprint arXiv:1804.00891*, 2018.

[DSGRS18] Christopher De Sa, Albert Gu, Christopher Ré, and Frederic Sala. Representation tradeoffs for hyperbolic embeddings. *Proceedings of machine learning research*, 80:4460, 2018.

[GBH18] Octavian Ganea, Gary Bécigneul, and Thomas Hofmann. Hyperbolic neural networks. In *Advances in neural information processing systems*, pages 5345–5355, 2018.

[GDM+18] Caglar Gulcehre, Misha Denil, Mateusz Malinowski, Ali Razavi, Razvan Pascanu, Karl Moritz Hermann, Peter Battaglia, Victor Bapst, David Raposo, Adam Santoro, et al. Hyperbolic attention networks. *arXiv preprint arXiv:1805.09786*, 2018.

[GSGR19] Albert Gu, Frederic Sala, Beliz Gunel, and Christopher Ré. Learning mixed-curvature representations in product spaces. In *International Conference on Learning Representations*, 2019.

[HA15] Elad Hoffer and Nir Ailon. Deep metric learning using triplet network. In *International Workshop on Similarity-Based Pattern Recognition*, pages 84–92. Springer, 2015.

[HFB12] Sø ren Hauberg, Oren Freifeld, and Michael J. Black. A geometric take on metric learning. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 2024–2032. Curran Associates, Inc., 2012.

[KS08] J Kierzenka and Lawrence Shampine. A bvp solver that controls residual and error. *European Society of Computational Methods in Sciences and Engineering (ESCMSE) Journal of Numerical Analysis, Industrial and Applied Mathematics*, 3:27–41, 01 2008.

[L+18] Allan M. M. Leal et al. autodiff, a modern, fast and expressive C++ library for automatic differentiation. `https://autodiff.github.io`, 2018.

[Lee97] J.M. Lee. *Riemannian Manifolds: An Introduction to Curvature*. Graduate Texts in Mathematics. Springer New York, 1997.

[Lee03] J.M. Lee. *Introduction to Smooth Manifolds*. Graduate Texts in Mathematics. Springer, 2003.

[LKJ+20] Aaron Lou, Isay Katsman, Qingxuan Jiang, Serge Belongie, Ser-Nam Lim, and Christopher De Sa. Differentiating through the fréchet mean, 2020.

[LLK+20] Aaron Lou, Derek Lim, Isay Katsman, Leo Huang, Qingxuan Jiang, Ser-Nam Lim, and Christopher De Sa. Neural manifold ordinary differential equations, 2020.

[MBL20] Kevin Musgrave, Serge Belongie, and Ser-Nam Lim. A metric learning reality check, 2020.

[MBM+17] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael M Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5115–5124, 2017.

[MN20] Emile Mathieu and Maximilian Nickel. Riemannian continuous normalizing flows, 2020.

[NK17] Maximillian Nickel and Douwe Kiela. Poincaré embeddings for learning hierarchical representations. In *Advances in neural information processing systems*, pages 6338–6347, 2017.

[NK18] Maximilian Nickel and Douwe Kiela. Learning continuous hierarchies in the lorentz model of hyperbolic geometry. *arXiv preprint arXiv:1806.03417*, 2018.

[PCJB20] Silviu Pitis, Harris Chan, Kiarash Jamali, and Jimmy Ba. An inductive bias for distances: Neural nets that respect the triangle inequality. *arXiv preprint arXiv:2002.05825*, 2020.

[PGM+19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[SB02] J. Stoer and R Bulirsch. *Introduction to numerical analysis*. Texts in applied mathematics. Springer, 2002.

[SDSGR18] Frederic Sala, Chris De Sa, Albert Gu, and Christopher Re. Representation tradeoffs for hyperbolic embeddings. volume 80 of *Proceedings of Machine Learning Research*, pages 4460–4469, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[SGB20] Ondrej Skopek, Octavian-Eugen Ganea, and Gary Bécigneul. Mixed-curvature variational autoencoders. In *International Conference on Learning Representations*, 2020.

[SKF18] H. Shao, A. Kumar, and P. T. Fletcher. The riemannian geometry of deep generative models. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 428–4288, 2018.

[SSZ17] Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. In *Advances in neural information processing systems*, pages 4077–4087, 2017.

[VBL+16] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. In *Advances in neural information processing systems*, pages 3630–3638, 2016.

[YDS19] Tao Yu and Christopher M De Sa. Numerically accurate hyperbolic embeddings using tiling-based models. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 2023–2033. Curran Associates, Inc., 2019.

A Background in Riemannian Geometry

We give a cursory overview of the core Riemannian geometry constructs integral to our construction. For a more thorough overview of this topic, we recommend interested readers consult a text such as [DC16, Lee97, Lee03].

The core object of study is the **differentiable manifold** $\mathcal{M}$ of dimension n, the higher dimensional analogue of a surface. Locally around each point, the space "resembles" Euclidean space $\mathbb{R}^n$. These are mathematically formulated by **charts**, diffeomorphic functions $\varphi : U \to V$ with U, V open subsets of $\mathbb{R}^n$ and $\mathcal{M}$ respectively. Our chart ranges cover our manifold, allowing the neighborhood of each point on our manifold to be controlled by **local coordinates** $x^1, \ldots, x^n$ which correspond to the standard directions in $\mathbb{R}^n$.

By taking linear approximations of functions at each point, one can construct **tangent vectors** v which form the $T_x\mathcal{M}$, the **tangent space of $\mathcal{M}$ at $\mathbf{x}$**. These tangent spaces are vector spaces of dimension n. With a local coordinate system, we can define the basis $dx^1, \ldots, dx^n$ of $T_x\mathcal{M}$, which we assume implicitly from the local coordinate system.

To endow our manifold with a notion of distance, we first construct **Riemannian metrics**. These are a collection of local inner products $g_x : T_x\mathcal{M} \times T_x\mathcal{M} \to \mathbb{R}$ that vary smoothly by x. We will denote g as the general metric for for $(g_x)_{x\in\mathcal{M}}$ and $\|\cdot\|_g$ to be the norm induced by g.

A Riemannian metric gives rise to a notion of distance. Specifically, for a curve $\gamma : I \to \mathcal{M}$, then the length of γ is $L(\gamma) = \sqrt{\int_0^1 \|\gamma'(t)\|_g^2 \, dt}$. For two points $x, y \in \mathcal{M}$, the distance $d_g(x, y)$ is defined as the minimum such distance for any such curve $\min_{\gamma:\gamma(0)=x,\gamma(y)} L(\gamma)$.

In practice, we can more efficiently find these minimizing curves by considering **geodesics**. Geodesics are curves which are governed by the geodesic equation eq. (1) and are the *locally* minimizing curves. This means that a minimizing curve will be a geodesic, but geodesics may not be minimizing (an example being the great arc on a sphere).

Given a point $x \in \mathcal{M}$ and an initial velocity $v \in T_x\mathcal{M}$, there is a unique constant speed geodesic with $\gamma(0) = x, \gamma'(0) = v$. $\gamma(1)$ is called the **exponential map** of x with v, or $\exp_x(v)$. The **logarithmic map** is the (local) inverse of this and is denoted $\log_x(y)$. Note that with this construction we can interpolate a geodesic using only the exp and log maps, in particular for a geodesic γ between x and y, $\gamma(t) = \exp_x(t \log_x(y))$. Furthermore, the distance can be represented as $d_g(x, y) = \|\log_x(y)\|_g$

B Proof of Claims

We formalize and prove the claims of the paper.

B.1 Universal Approximation

Prop B.1 (Universal Approximation of Riemannian Metrics Full)**.** *Suppose we have a compact set D of $\mathbb{R}^n$ and a neural network with sufficient approximation capacity as in [Cyb89]. For any metric g and $\epsilon > 0$, there exists a neural network s.t.* $\sup_{x\in K} \|g_x - g_x^d\| < \epsilon$.

Proof. Let g'_x be the function s.t. $g_x = (\exp \circ \mathrm{sym})(g'_x)$. We note that g'_x exists and is unique as $\exp$ is bijective as a function from $\mathcal{S}_n$ to $\mathcal{S}_n^+$.

There exists a neural network s.t. $\sup_{x\in K} \|f_{nn}(x) - g'_x\| \leq \frac{\epsilon}{K}$, where K is defined as $\sup_{a,b\in f_{nn}(D),g'(D)} \|(\exp \circ \mathrm{sym})(a) - (\exp \circ \mathrm{sym})(b)\| \leq K \|a - b\|$.

Then, we see that

$$\sup_{x\in D} \left\|g_x - g_x^d\right\| \leq K \sup_{x\in D} \|f_{nn}(x) - g'_x\| \leq \epsilon \tag{4}$$

as desired. □

B.2 Adjoint Sensitivity

Theorem B.1 (Adjoint Sensitivity For BVPs)**.** *Suppose we are given a BVP $B(x, y)$ with corresponding IVP $I(x, v)$ s.t. $I(x, B(x, y)) = y$. Suppose our solution to $B(x_0, y_0)$ is b. Then*

$$D_x B(x_0, y_0) = -D_v I(x_0, b)^{-1} \circ D_x I(x_0, b) \qquad D_y B(x_0, y_0) = -D_v I(x_0, b)^{-1} \tag{5}$$

Note that the derivatives of I can be computed through the adjoint-sensitivity analysis. Furthermore, if our ODE problems are controlled by some parameter θ, then

$$D_\theta B(x_0, y_0) = -D_v I(x_0, b)^{-1} \circ D_\theta I(x_0, b)$$

Proof. The core of this proof comes from the fact that $I(x_0, B(x_0, y_0)) = y_0$. Differentiating this identity w.r.t y using the chain rule gives us

$$\begin{aligned} I &= D_y y_0 \\ &= (D_y I)(x_0, B(x_0, y_0)) \\ &= \begin{bmatrix} D_x I(x_0, B(x_0, y_0)) \\ D_v I(x_0, B(x_0, y_0)) \end{bmatrix} \begin{bmatrix} 0 & D_y B(x_0, y_0) \end{bmatrix} \\ &= D_v I(x_0, B(x_0, y_0)) D_y B(x_0, y_0) \end{aligned}$$

rearranging and setting $b = B(x_0, y_0)$ gives us that $D_y B(x_0, y_0) = -D_v I(x_0, b)^{-1}$, as desired.

For x, differentiating through the same identity gives us that

$$\begin{aligned} 0 &= D_x y_0 \\ &= (D_x I)(x_0, B(x_0, y_0)) \\ &= \begin{bmatrix} D_x I(x_0, B(x_0, y_0)) \\ D_v I(x_0, B(x_0, y_0)) \end{bmatrix} \begin{bmatrix} I & D_x B(x_0, y_0) \end{bmatrix} \\ &= D_x I(x_0, B(x_0, y_0)) + D_v I(x_0, B(x_0, y_0)) D_x B(x_0, y_0) \end{aligned}$$

and rearranging gives us that $D_x B(x_0, y_0) = -D_v I(x_0, b)^{-1} \circ D_x I(x_0, b)$.

For θ, the proof is very similar to the proof for x. In particular, from the same differentiation we get that

$$\begin{aligned} 0 &= D_\theta y_0 \\ &= (D_\theta I)(x_0, B(x_0, y_0), \theta) \\ &= \begin{bmatrix} D_x I(x_0, B(x_0, y_0)) \\ D_v I(x_0, B(x_0, y_0)) \\ D_\theta I(x_0, B(x_0, y_0)) \end{bmatrix} \begin{bmatrix} 0 & D_\theta B(x_0, y_0) & I \end{bmatrix} \\ &= D_v I(x_0, B(x_0, y_0)) D_\theta B(x_0, y_0) + D_\theta I(x_0, B(x_0, y_0)) \end{aligned}$$

and rearranging gives us $D_\theta B(x_0, y_0) = -D_v I(x_0, b)^{-1} \circ D_\theta I(x_0, b)$. □

C Graph embeddings: Additional Details

We optimize global distances with respect to the stress metric. This metric is given below, where we denote the graph distance between points i and j by g_{ij} and the manifold distance as m_{ij}.

$$\mathcal{L}_s := \sum_{i<j} (g_{ij} - m_{ij})^2 \tag{6}$$

Table 2: Reconstruction losses (stress) for 2D, 5D, and 10D embeddings of the graph distances.

	Sphere100			**Tree6**			**Tree40**			**Cycle10**			**Cube1**			**Cube2**		
	2	5	10	2	5	10	2	5	10	2	5	10	2	5	10	2	5	10
Euclidean	$1.76_{\pm 0.26}$	$0.25_{\pm 0.00}$	$0.25_{\pm 0.00}$	$0.32_{\pm 0.46}$	$0.04_{\pm 0.01}$	$0.05_{\pm 0.01}$	$1.42_{\pm 0.28}$	$0.35_{\pm 0.03}$	$0.26_{\pm 0.05}$	$0.17_{\pm 0.00}$	$0.21_{\pm 0.01}$	$0.20_{\pm 0.01}$	$0.34_{\pm 0.16}$	$0.11_{\pm 0.01}$	$0.12_{\pm 0.01}$	$0.37_{\pm 0.06}$	$0.15_{\pm 0.00}$	$0.15_{\pm 0.00}$
PoincareBall	$1.83_{\pm 0.00}$	$0.13_{\pm 0.00}$	$0.13_{\pm 0.00}$	$0.07_{\pm 0.00}$	$0.07_{\pm 0.00}$	$0.07_{\pm 0.00}$	$1.87_{\pm 0.17}$	$0.25_{\pm 0.00}$	$0.04_{\pm 0.00}$	$0.22_{\pm 0.00}$	$0.22_{\pm 0.00}$	$0.22_{\pm 0.00}$	$0.18_{\pm 0.00}$	$0.18_{\pm 0.00}$	$0.18_{\pm 0.00}$	$0.29_{\pm 0.02}$	$0.24_{\pm 0.00}$	$0.24_{\pm 0.00}$
Sphere	$0.53_{\pm 0.00}$	$0.15_{\pm 0.00}$	$0.07_{\pm 0.00}$	$0.07_{\pm 0.00}$	$0.08_{\pm 0.00}$	$0.08_{\pm 0.00}$	$1.03_{\pm 0.00}$	$0.76_{\pm 0.00}$	$0.66_{\pm 0.00}$	$0.12_{\pm 0.00}$	$0.12_{\pm 0.00}$	$0.12_{\pm 0.00}$	$0.16_{\pm 0.00}$	$0.16_{\pm 0.00}$	$0.16_{\pm 0.00}$	$0.27_{\pm 0.00}$	$0.23_{\pm 0.00}$	$0.23_{\pm 0.00}$
DeepManifold	$1.92_{\pm 0.29}$	$0.11_{\pm 0.00}$	$0.18_{\pm 0.02}$	$0.14_{\pm 0.12}$	$0.00_{\pm 0.00}$	$0.00_{\pm 0.00}$	$1.48_{\pm 0.70}$	$0.18_{\pm 0.05}$	$0.08_{\pm 0.01}$	$0.62_{\pm 0.42}$	$0.11_{\pm 0.10}$	$0.03_{\pm 0.01}$	$0.67_{\pm 0.15}$	$0.05_{\pm 0.03}$	$0.05_{\pm 0.02}$	$0.28_{\pm 0.08}$	$0.06_{\pm 0.00}$	$0.06_{\pm 0.01}$

However, one will note that we report distortion as our loss function, which is given by

$$\mathcal{L}_d := \min\{\alpha : \alpha \geq 1, \exists \beta > 0 \text{ s.t. } \forall i < j, \beta m_{ij} \leq d_{ij} \leq \alpha\beta m_{ij}\} \tag{7}$$

In practice, distortion is calculated as $\max_{i<j} \frac{g_{ij}}{m_{ij}} \max_{i<j} \frac{m_{ij}}{d_{ij}}$. We choose to optimize stress to allow for signal to propagate along more than one node and to account for more global graph structures. In practice, this effects our Poincare Ball results for larger graphs as the Poincare Ball is incapable of numerically representing these larger distances without a more complex numerical system [SDSGR18, YDS19]. We report the stress of the embeddings in table 2 normalized by the number of distances.

Points are updated with Riemannian Stochastic Gradient Descent with a learning rate of 0.01 on the chart. We run with a batch size of 32 for most tasks except for small graphs, where we must optimize over smaller batches of 8 to avoid our optimization becoming full batch gradient descent. Finally, we train for 500 epochs.

For our deep manifold, we construct a 2 layer neural network with 32 hidden units for smaller graphs and 128 for larger ones. Neural network parameters are updated with Stochastic Gradient Descent with a learning rate of 0.01. To stabilize training, we "burnin" deep manifold embeddings by optimizing the Euclidean distances instead for 15 epochs before switching to Deep Manifolds. Since at initialization time our deep manifold metric is (approximately) Euclidean, this enables us to find a good for position our initial embeddings in which geodesics are less likely to overlap. Finally, we clamp the gradient of our neural network to 5.