geoopt Documentation

Release 0.0.1

Max Kochurov

Oct 30, 2019

Contents

CHAPTER 1

geoopt

Manifold aware `pytorch.optim`.

Unofficial implementation for "Riemannian Adaptive Optimization Methods" ICLR2019 and more.

1.1 Installation

Make sure you have pytorch>=1.1.0 installed

There are two ways to install geoopt:

1. GitHub (preferred so far) due to active development

```
pip install git+https://github.com/geoopt/geoopt.git
```

2. pypi (this might be significantly behind master branch)

```
pip install geoopt
```

The preferred way to install geoopt will change once stable project stage is achieved. Now, pypi is behind master as we actively develop and implement new features.

1.2 What is done so far

Work is in progress but you can already use this. Note that API might change in future releases.

1.2.1 Tensors

- `geoopt.ManifoldTensor` – just as torch.Tensor with additional `manifold` keyword argument.

- `geoopt.ManifoldParameter` – same as above, recognized in `torch.nn.Module.parameters` as correctly subclassed.

All above containers have special methods to work with them as with points on a certain manifold

- `.proj_()` – inplace projection on the manifold.
- `.proju(u)` – project vector `u` on the tangent space. You need to project all vectors for all methods below.
- `.egrad2rgrad(u)` – project gradient `u` on Riemannian manifold
- `.inner(u, v=None)` – inner product at this point for two **tangent** vectors at this point. The passed vectors are not projected, they are assumed to be already projected.
- `.retr(u)` – retraction map following vector `u`
- `.expmap(u)` – exponential map following vector `u` (if expmap is not available in closed form, best approximation is used)
- `.transp(v, u)` – transport vector `v` with direction `u`
- `.retr_transp(v, u)` – transport `self`, vector `v` (and possibly more vectors) with direction `u` (returns are plain tensors)

1.2.2 Manifolds

- `geoopt.Euclidean` – unconstrained manifold in `R` with Euclidean metric
- `geoopt.Stiefel` – Stiefel manifold on matrices `A in R^{n x p} : A^t A=I, n >= p`
- `geoopt.Sphere` - Sphere manifold `||x||=1`
- `geoopt.PoincareBall` - Poincare ball model (wiki)
- `geoopt.ProductManifold` - Product manifold constructor
- `geoopt.Scaled` - Scaled version of the manifold. Similar to Learning Mixed-Curvature Representations in Product Spaces if combined with `ProductManifold`

All manifolds implement methods necessary to manipulate tensors on manifolds and tangent vectors to be used in general purpose. See more in documentation.

1.2.3 Optimizers

- `geoopt.optim.RiemannianSGD` – a subclass of `torch.optim.SGD` with the same API
- `geoopt.optim.RiemannianAdam` – a subclass of `torch.optim.Adam`

1.2.4 Samplers

- `geoopt.samplers.RSGLD` – Riemannian Stochastic Gradient Langevin Dynamics
- `geoopt.samplers.RHMC` – Riemannian Hamiltonian Monte-Carlo
- `geoopt.samplers.SGRHMC` – Stochastic Gradient Riemannian Hamiltonian Monte-Carlo

1.2.5 Citing Geoopt

If you find this project useful in your research, please kindly add this bibtex entry in references.

```
@misc{geoopt,
    author = {Max Kochurov and Sergey Kozlukov and Rasul Karimov and Viktor Yanush},
    title = {Geoopt: Adaptive Riemannian optimization in PyTorch},
    year = {2019},
    publisher = {GitHub},
    journal = {GitHub repository},
    howpublished = {\url{https://github.com/geoopt/geoopt}},
}
```

CHAPTER 2

API

2.1 Manifolds

All manifolds share same API. Some manifols may have several implementations of retraction operation, every implementation has a corresponding class.

class geoopt.manifolds.**Euclidean**(*ndim=0*)

Simple Euclidean manifold, every coordinate is treated as an independent element.

Parameters **ndim** (*int*) – number of trailing dimensions treated as manifold dimensions. All the operations acting on cuch as inner products, etc will respect the `ndim`.

component_inner(*x: torch.Tensor*, *u: torch.Tensor*, *v=None*)

Inner product for tangent vectors at point x according to components of the manifold.

The result of the function is same as `inner` with `keepdim=True` for all the manifolds except ProductManifold. For this manifold it acts different way computing inner product for each component and then building an output correctly tiling and reshaping the result.

Parameters

- **x** (*tensor*) – point on the manifold
- **u** (*tensor*) – tangent vector at point x
- **v** (*tensor (optional)*) – tangent vector at point x

Returns inner product component wise (broadcasted)

Return type tensor

Notes

The purpose of this method is better adaptive properties in optimization since ProductManifold will "hide" the structure in public API.

dist (*x: torch.Tensor*, *y: torch.Tensor*, *, *keepdim=False*)
Compute distance between 2 points on the manifold that is the shortest path along geodesics.

Parameters

- **x** (*tensor*) – point on the manifold
- **y** (*tensor*) – point on the manifold
- **keepdim** (*bool*) – keep the last dim?

Returns distance between two points

Return type scalar

dist2 (*x: torch.Tensor*, *y: torch.Tensor*, *, *keepdim=False*)
Compute squared distance between 2 points on the manifold that is the shortest path along geodesics.

Parameters

- **x** (*tensor*) – point on the manifold
- **y** (*tensor*) – point on the manifold
- **keepdim** (*bool*) – keep the last dim?

Returns squared distance between two points

Return type scalar

egrad2rgrad (*x: torch.Tensor*, *u: torch.Tensor*)
Transform gradient computed using autodiff to the correct Riemannian gradient for the point x.

Parameters

- **x** (*tensor*) – point on the manifold
- **u** (*tensor*) – gradient to be projected

Returns grad vector in the Riemannian manifold

Return type tensor

expmap (*x: torch.Tensor*, *u: torch.Tensor*)
Perform an exponential map $\operatorname{Exp}_x(u)$.

Parameters

- **x** (*tensor*) – point on the manifold
- **u** (*tensor*) – tangent vector at point x

Returns transported point

Return type tensor

extra_repr ()
Set the extra representation of the module

To print customized extra information, you should reimplement this method in your own modules. Both single-line and multi-line strings are acceptable.

inner (*x: torch.Tensor*, *u: torch.Tensor*, *v: torch.Tensor = None*, *, *keepdim=False*)
Inner product for tangent vectors at point x.

Parameters

- **x** (*tensor*) – point on the manifold

- **u** (*tensor*) – tangent vector at point x
- **v** (*tensor (optional)*) – tangent vector at point x
- **keepdim** (*bool*) – keep the last dim?

Returns inner product (broadcasted)

Return type scalar

logmap (*x: torch.Tensor*, *y: torch.Tensor*)
Perform an logarithmic map $\operatorname{Log}_x(y)$.

Parameters

- **x** (*tensor*) – point on the manifold
- **y** (*tensor*) – point on the manifold

Returns tangent vector

Return type tensor

norm (*x: torch.Tensor*, *u: torch.Tensor*, *, *keepdim=False*)
Norm of a tangent vector at point x.

Parameters

- **x** (*tensor*) – point on the manifold
- **u** (*tensor*) – tangent vector at point x
- **keepdim** (*bool*) – keep the last dim?

Returns inner product (broadcasted)

Return type scalar

origin (**size*, *dtype=None*, *device=None*, *seed=42*)
Zero point origin.

Parameters

- **size** (*shape*) – the desired shape
- **device** (*torch.device*) – the desired device
- **dtype** (*torch.dtype*) – the desired dtype
- **seed** (*int*) – ignored

Returns

Return type *ManifoldTensor*

proju (*x: torch.Tensor*, *u: torch.Tensor*)
Project vector u on a tangent space for x, usually is the same as `egrad2rgrad()`.

Parameters

- **x** (*tensor*) – point on the manifold
- **u** (*tensor*) – vector to be projected

Returns projected vector

Return type tensor

projx (*x: torch.Tensor*)
Project point x on the manifold.

Parameters **x** (*tensor*) – point to be projected

Returns projected point

Return type tensor

random (**size*, *mean=0.0*, *std=1.0*, *device=None*, *dtype=None*)
Create a point on the manifold, measure is induced by Normal distribution.

Parameters

- **size** (*shape*) – the desired shape
- **mean** (*float | tensor*) – mean value for the Normal distribution
- **std** (*float | tensor*) – std value for the Normal distribution
- **device** (*torch.device*) – the desired device
- **dtype** (*torch.dtype*) – the desired dtype

Returns random point on the manifold

Return type *ManifoldTensor*

random_normal (**size*, *mean=0.0*, *std=1.0*, *device=None*, *dtype=None*)
Create a point on the manifold, measure is induced by Normal distribution.

Parameters

- **size** (*shape*) – the desired shape
- **mean** (*float | tensor*) – mean value for the Normal distribution
- **std** (*float | tensor*) – std value for the Normal distribution
- **device** (*torch.device*) – the desired device
- **dtype** (*torch.dtype*) – the desired dtype

Returns random point on the manifold

Return type *ManifoldTensor*

retr (*x: torch.Tensor*, *u: torch.Tensor*)
Perform a retraction from point x with given direction u.

Parameters

- **x** (*tensor*) – point on the manifold
- **u** (*tensor*) – tangent vector at point x

Returns transported point

Return type tensor

transp (*x: torch.Tensor*, *y: torch.Tensor*, *v: torch.Tensor*)
Perform vector transport $\mathfrak{T}_{x\to y}(v)$.

Parameters

- **x** (*tensor*) – start point on the manifold
- **y** (*tensor*) – target point on the manifold
- **v** (*tensor*) – tangent vector at point x

Returns transported tensor(s)

Return type tensor or tuple of tensors

class geoopt.manifolds.**Stiefel**(***kwargs*)

Manifold induced by the following matrix constraint:

$$X^\top X = I$$
$$X \in \mathrm{R}^{n\times m}$$
$$n \geq m$$

Parameters **canonical** (*bool*) – Use canonical inner product instead of euclidean one (defaults to canonical)

See also:

CanonicalStiefel, *EuclideanStiefel*, *EuclideanStiefelExact*

origin(**size, dtype=None, device=None, seed=42*)

Identity matrix point origin.

Parameters

- **size** (*shape*) – the desired shape
- **device** (*torch.device*) – the desired device
- **dtype** (*torch.dtype*) – the desired dtype
- **seed** (*int*) – ignored

Returns

Return type *ManifoldTensor*

projx(*x: torch.Tensor*)

Project point x on the manifold.

Parameters **x** (*tensor*) – point to be projected

Returns projected point

Return type tensor

random(**size, dtype=None, device=None*)

Naive approach to get random matrix on Stiefel manifold.

A helper function to sample a random point on the Stiefel manifold. The measure is non-uniform for this method, but fast to compute.

Parameters

- **size** (*shape*) – the desired output shape
- **dtype** (*torch.dtype*) – desired dtype
- **device** (*torch.device*) – desired device

Returns random point on Stiefel manifold

Return type *ManifoldTensor*

random_naive(**size, dtype=None, device=None*)

Naive approach to get random matrix on Stiefel manifold.

A helper function to sample a random point on the Stiefel manifold. The measure is non-uniform for this method, but fast to compute.

Parameters

- **size** (*shape*) – the desired output shape
- **dtype** (*torch.dtype*) – desired dtype
- **device** (*torch.device*) – desired device

Returns random point on Stiefel manifold

Return type *ManifoldTensor*

class geoopt.manifolds.**CanonicalStiefel**(***kwargs*)

Stiefel Manifold with Canonical inner product

Manifold induced by the following matrix constraint:

$$X^\top X = I$$
$$X \in \mathrm{R}^{n\times m}$$
$$n \geq m$$

egrad2rgrad(*x: torch.Tensor*, *u: torch.Tensor*)

Project vector u on a tangent space for x, usually is the same as `egrad2rgrad()`.

Parameters

- **x** (*tensor*) – point on the manifold
- **u** (*tensor*) – vector to be projected

Returns projected vector

Return type tensor

expmap(*x: torch.Tensor*, *u: torch.Tensor*)

Perform a retraction from point x with given direction u.

Parameters

- **x** (*tensor*) – point on the manifold
- **u** (*tensor*) – tangent vector at point x

Returns transported point

Return type tensor

expmap_transp(*x: torch.Tensor*, *u: torch.Tensor*, *v: torch.Tensor*)

Calculate an optimized retr_transp for Stiefel Manifold.

inner(*x: torch.Tensor*, *u: torch.Tensor*, *v: torch.Tensor = None*, ***, *keepdim=False*)

Inner product for tangent vectors at point x.

Parameters

- **x** (*tensor*) – point on the manifold
- **u** (*tensor*) – tangent vector at point x
- **v** (*tensor (optional)*) – tangent vector at point x
- **keepdim** (*bool*) – keep the last dim?

Returns inner product (broadcasted)

Return type scalar

proju(*x: torch.Tensor*, *u: torch.Tensor*)

Project vector u on a tangent space for x, usually is the same as `egrad2rgrad()`.

Parameters

- **x** (*tensor*) – point on the manifold
- **u** (*tensor*) – vector to be projected

Returns projected vector

Return type tensor

retr (*x: torch.Tensor*, *u: torch.Tensor*)
Perform a retraction from point x with given direction u.

Parameters

- **x** (*tensor*) – point on the manifold
- **u** (*tensor*) – tangent vector at point x

Returns transported point

Return type tensor

retr_transp (*x: torch.Tensor*, *u: torch.Tensor*, *v: torch.Tensor*)
Calculate an optimized retr_transp for Stiefel Manifold.

transp_follow_expmap (*x: torch.Tensor*, *u: torch.Tensor*, *v: torch.Tensor*)
Perform vector transport following u: $\mathfrak{T}_{x\to\operatorname{retr}(x,u)}(v)$.

This operation is sometimes is much more simpler and can be optimized.

Parameters

- **x** (*tensor*) – point on the manifold
- **u** (*tensor*) – tangent vector at point x
- **v** (*tensor*) – tangent vector at point x to be transported

Returns transported tensor

Return type tensor

transp_follow_retr (*x: torch.Tensor*, *u: torch.Tensor*, *v: torch.Tensor*)
Perform vector transport following u: $\mathfrak{T}_{x\to\operatorname{retr}(x,u)}(v)$.

This operation is sometimes is much more simpler and can be optimized.

Parameters

- **x** (*tensor*) – point on the manifold
- **u** (*tensor*) – tangent vector at point x
- **v** (*tensor*) – tangent vector at point x to be transported

Returns transported tensor

Return type tensor

class geoopt.manifolds.**EuclideanStiefel** (***kwargs*)
Stiefel Manifold with Euclidean inner product

Manifold induced by the following matrix constraint:

$$X^\top X = I$$
$$X \in \mathrm{R}^{n\times m}$$
$$n \geq m$$

egrad2rgrad (*x: torch.Tensor*, *u: torch.Tensor*)

Project vector u on a tangent space for x, usually is the same as `egrad2rgrad()`.

Parameters

- **x** (*tensor*) – point on the manifold
- **u** (*tensor*) – vector to be projected

Returns projected vector

Return type tensor

expmap (*x: torch.Tensor*, *u: torch.Tensor*)

Perform an exponential map $\operatorname{Exp}_x(u)$.

Parameters

- **x** (*tensor*) – point on the manifold
- **u** (*tensor*) – tangent vector at point x

Returns transported point

Return type tensor

inner (*x: torch.Tensor*, *u: torch.Tensor*, *v: torch.Tensor = None*, ***, *keepdim=False*)

Inner product for tangent vectors at point x.

Parameters

- **x** (*tensor*) – point on the manifold
- **u** (*tensor*) – tangent vector at point x
- **v** (*tensor (optional)*) – tangent vector at point x
- **keepdim** (*bool*) – keep the last dim?

Returns inner product (broadcasted)

Return type scalar

proju (*x: torch.Tensor*, *u: torch.Tensor*)

Project vector u on a tangent space for x, usually is the same as `egrad2rgrad()`.

Parameters

- **x** (*tensor*) – point on the manifold
- **u** (*tensor*) – vector to be projected

Returns projected vector

Return type tensor

retr (*x: torch.Tensor*, *u: torch.Tensor*)

Perform a retraction from point x with given direction u.

Parameters

- **x** (*tensor*) – point on the manifold
- **u** (*tensor*) – tangent vector at point x

Returns transported point

Return type tensor

transp(*x: torch.Tensor*, *y: torch.Tensor*, *v: torch.Tensor*)
Perform vector transport $\mathfrak{T}_{x\to y}(v)$.

Parameters

- **x** (*tensor*) – start point on the manifold
- **y** (*tensor*) – target point on the manifold
- **v** (*tensor*) – tangent vector at point x

Returns transported tensor(s)

Return type tensor or tuple of tensors

class geoopt.manifolds.**EuclideanStiefelExact**(***kwargs*)
Stiefel Manifold with Euclidean inner product

Manifold induced by the following matrix constraint:

$$X^\top X = I$$
$$X \in \mathrm{R}^{n\times m}$$
$$n \geq m$$

Notes

The implementation of retraction is an exact exponential map, this retraction will be used in optimization

extra_repr()
Set the extra representation of the module

To print customized extra information, you should reimplement this method in your own modules. Both single-line and multi-line strings are acceptable.

retr(*x: torch.Tensor*, *u: torch.Tensor*)
Perform an exponential map $\mathrm{Exp}_x(u)$.

Parameters

- **x** (*tensor*) – point on the manifold
- **u** (*tensor*) – tangent vector at point x

Returns transported point

Return type tensor

retr_transp(*x: torch.Tensor*, *u: torch.Tensor*, *v: torch.Tensor*)
Perform an exponential map and vector transport from point x with given direction u.

Parameters

- **x** (*tensor*) – point on the manifold
- **u** (*tensor*) – tangent vector at point x
- **v** (*tensor*) – tangent vector at point x to be transported

Returns transported point

Return type tensor

transp_follow_retr (*x: torch.Tensor*, *u: torch.Tensor*, *v: torch.Tensor*)
Perform vector transport following u: $\mathfrak{T}_{x\to\mathrm{Exp}(x,u)}(v)$.

Here, Exp is the best possible approximation of the true exponential map. There are cases when the exact variant is hard or impossible implement, therefore a fallback, non-exact, implementation is used.

Parameters

- **x** (*tensor*) – point on the manifold
- **u** (*tensor*) – tangent vector at point x
- **v** (*tensor*) – tangent vector at point x to be transported

Returns transported tensor

Return type tensor

class geoopt.manifolds.**Sphere** (*intersection: torch.Tensor = None*, *complement: torch.Tensor = None*)
Sphere manifold induced by the following constraint

$$\|x\| = 1$$
$$x \in \operatorname{span}(U)$$

where U can be parametrized with compliment space or intersection.

Parameters

- **intersection** (*tensor*) – shape `(..., dim, K)`, subspace to intersect with
- **complement** (*tensor*) – shape `(..., dim, K)`, subspace to compliment

See also:

SphereExact

dist (*x: torch.Tensor*, *y: torch.Tensor*, ***, *keepdim=False*)
Compute distance between 2 points on the manifold that is the shortest path along geodesics.

Parameters

- **x** (*tensor*) – point on the manifold
- **y** (*tensor*) – point on the manifold
- **keepdim** (*bool*) – keep the last dim?

Returns distance between two points

Return type scalar

egrad2rgrad (*x: torch.Tensor*, *u: torch.Tensor*)
Project vector u on a tangent space for x, usually is the same as `egrad2rgrad()`.

Parameters

- **x** (*tensor*) – point on the manifold
- **u** (*tensor*) – vector to be projected

Returns projected vector

Return type tensor

expmap (*x: torch.Tensor*, *u: torch.Tensor*)
Perform an exponential map $\mathrm{Exp}_x(u)$.

Parameters

- **x** (*tensor*) – point on the manifold
- **u** (*tensor*) – tangent vector at point x

Returns transported point

Return type tensor

inner(*x: torch.Tensor*, *u: torch.Tensor*, *v: torch.Tensor = None*, ***, *keepdim=False*)
Inner product for tangent vectors at point x.

Parameters

- **x** (*tensor*) – point on the manifold
- **u** (*tensor*) – tangent vector at point x
- **v** (*tensor (optional)*) – tangent vector at point x
- **keepdim** (*bool*) – keep the last dim?

Returns inner product (broadcasted)

Return type scalar

logmap(*x: torch.Tensor*, *y: torch.Tensor*)
Perform an logarithmic map $\operatorname{Log}_x(y)$.

Parameters

- **x** (*tensor*) – point on the manifold
- **y** (*tensor*) – point on the manifold

Returns tangent vector

Return type tensor

proju(*x: torch.Tensor*, *u: torch.Tensor*)
Project vector u on a tangent space for x, usually is the same as `egrad2rgrad()`.

Parameters

- **x** (*tensor*) – point on the manifold
- **u** (*tensor*) – vector to be projected

Returns projected vector

Return type tensor

projx(*x: torch.Tensor*)
Project point x on the manifold.

Parameters **x** (*tensor*) – point to be projected

Returns projected point

Return type tensor

random(**size*, *dtype=None*, *device=None*)
Uniform random measure on Sphere manifold.

Parameters

- **size** (*shape*) – the desired output shape
- **dtype** (*torch.dtype*) – desired dtype
- **device** (*torch.device*) – desired device

Returns random point on Sphere manifold

Return type *ManifoldTensor*

Notes

In case of projector on the manifold, dtype and device are set automatically and shouldn't be provided. If you provide them, they are checked to match the projector device and dtype

random_uniform(**size*, *dtype=None*, *device=None*)
Uniform random measure on Sphere manifold.

Parameters

- **size** (*shape*) – the desired output shape
- **dtype** (*torch.dtype*) – desired dtype
- **device** (*torch.device*) – desired device

Returns random point on Sphere manifold

Return type *ManifoldTensor*

Notes

In case of projector on the manifold, dtype and device are set automatically and shouldn't be provided. If you provide them, they are checked to match the projector device and dtype

retr(*x: torch.Tensor*, *u: torch.Tensor*)
Perform a retraction from point x with given direction u.

Parameters

- **x** (*tensor*) – point on the manifold
- **u** (*tensor*) – tangent vector at point x

Returns transported point

Return type tensor

transp(*x: torch.Tensor*, *y: torch.Tensor*, *v: torch.Tensor*)
Perform vector transport $\mathfrak{T}_{x\to y}(v)$.

Parameters

- **x** (*tensor*) – start point on the manifold
- **y** (*tensor*) – target point on the manifold
- **v** (*tensor*) – tangent vector at point x

Returns transported tensor(s)

Return type tensor or tuple of tensors

class geoopt.manifolds.**SphereExact**(*intersection: torch.Tensor = None*, *complement: torch.Tensor = None*)
Sphere manifold induced by the following constraint

$$\|x\| = 1$$
$$x \in \sim\imath\partial\ltimes(U)$$

where U can be parametrized with compliment space or intersection.

Parameters

- **intersection** (*tensor*) – shape `(..., dim, K)`, subspace to intersect with
- **complement** (*tensor*) – shape `(..., dim, K)`, subspace to compliment

See also:

Sphere

Notes

The implementation of retraction is an exact exponential map, this retraction will be used in optimization

extra_repr()

Set the extra representation of the module

To print customized extra information, you should reimplement this method in your own modules. Both single-line and multi-line strings are acceptable.

retr(*x: torch.Tensor*, *u: torch.Tensor*)

Perform an exponential map $\mathrm{Exp}_x(u)$.

Parameters

- **x** (*tensor*) – point on the manifold
- **u** (*tensor*) – tangent vector at point x

Returns transported point

Return type tensor

retr_transp(*x: torch.Tensor*, *u: torch.Tensor*, *v: torch.Tensor*)

Perform an exponential map and vector transport from point x with given direction u.

Parameters

- **x** (*tensor*) – point on the manifold
- **u** (*tensor*) – tangent vector at point x
- **v** (*tensor*) – tangent vector at point x to be transported

Returns transported point

Return type tensor

transp_follow_retr(*x: torch.Tensor*, *u: torch.Tensor*, *v: torch.Tensor*)

Perform vector transport following u: $\mathfrak{T}_{x\to\mathrm{Exp}(x,u)}(v)$.

Here, Exp is the best possible approximation of the true exponential map. There are cases when the exact variant is hard or impossible implement, therefore a fallback, non-exact, implementation is used.

Parameters

- **x** (*tensor*) – point on the manifold
- **u** (*tensor*) – tangent vector at point x
- **v** (*tensor*) – tangent vector at point x to be transported

Returns transported tensor

Return type tensor

class geoopt.manifolds.**PoincareBall**(*c=1.0*)

Poincare ball model, see more in *Poincare Ball model*.

Parameters **c** (*float | tensor*) – ball negative curvature

Notes

It is extremely recommended to work with this manifold in double precision

See also:

PoincareBallExact

dist(*x: torch.Tensor*, *y: torch.Tensor*, ***, *keepdim=False*, *dim=-1*)

Compute distance between 2 points on the manifold that is the shortest path along geodesics.

Parameters

- **x** (*tensor*) – point on the manifold
- **y** (*tensor*) – point on the manifold
- **keepdim** (*bool*) – keep the last dim?

Returns distance between two points

Return type scalar

egrad2rgrad(*x: torch.Tensor*, *u: torch.Tensor*, ***, *dim=-1*)

Transform gradient computed using autodiff to the correct Riemannian gradient for the point x.

Parameters

- **x** (*tensor*) – point on the manifold
- **u** (*tensor*) – gradient to be projected

Returns grad vector in the Riemannian manifold

Return type tensor

expmap(*x: torch.Tensor*, *u: torch.Tensor*, ***, *project=True*, *dim=-1*)

Perform an exponential map $\mathrm{Exp}_x(u)$.

Parameters

- **x** (*tensor*) – point on the manifold
- **u** (*tensor*) – tangent vector at point x

Returns transported point

Return type tensor

expmap_transp(*x: torch.Tensor*, *u: torch.Tensor*, *v: torch.Tensor*, *dim=-1*, *project=True*)

Perform an exponential map and vector transport from point x with given direction u.

Parameters

- **x** (*tensor*) – point on the manifold
- **u** (*tensor*) – tangent vector at point x
- **v** (*tensor*) – tangent vector at point x to be transported

Returns transported point

Return type tensor

inner (*x: torch.Tensor*, *u: torch.Tensor*, *v: torch.Tensor = None*, ***, *keepdim=False*, *dim=-1*)
Inner product for tangent vectors at point x.

Parameters

- **x** (*tensor*) – point on the manifold
- **u** (*tensor*) – tangent vector at point x
- **v** (*tensor (optional)*) – tangent vector at point x
- **keepdim** (*bool*) – keep the last dim?

Returns inner product (broadcasted)

Return type scalar

logmap (*x: torch.Tensor*, *y: torch.Tensor*, ***, *dim=-1*)
Perform an logarithmic map $\operatorname{Log}_x(y)$.

Parameters

- **x** (*tensor*) – point on the manifold
- **y** (*tensor*) – point on the manifold

Returns tangent vector

Return type tensor

norm (*x: torch.Tensor*, *u: torch.Tensor*, ***, *keepdim=False*, *dim=-1*)
Norm of a tangent vector at point x.

Parameters

- **x** (*tensor*) – point on the manifold
- **u** (*tensor*) – tangent vector at point x
- **keepdim** (*bool*) – keep the last dim?

Returns inner product (broadcasted)

Return type scalar

origin (**size*, *dtype=None*, *device=None*, *seed=42*)
Zero point origin.

Parameters

- **size** (*shape*) – the desired shape
- **device** (*torch.device*) – the desired device
- **dtype** (*torch.dtype*) – the desired dtype
- **seed** (*int*) – ignored

Returns random point on the manifold

Return type *ManifoldTensor*

proju (*x: torch.Tensor*, *u: torch.Tensor*)
Project vector u on a tangent space for x, usually is the same as `egrad2rgrad()`.

Parameters

- **x** (*tensor*) – point on the manifold
- **u** (*tensor*) – vector to be projected

Returns projected vector

Return type tensor

projx(*x: torch.Tensor*, *dim=-1*)
Project point x on the manifold.

Parameters **x** (*tensor*) – point to be projected

Returns projected point

Return type tensor

random(**size*, *mean=0*, *std=1*, *dtype=None*, *device=None*)
Create a point on the manifold, measure is induced by Normal distribution on the tangent space of zero.

Parameters

- **size** (*shape*) – the desired shape
- **mean** (*float | tensor*) – mean value for the Normal distribution
- **std** (*float | tensor*) – std value for the Normal distribution
- **dtype** (*torch.dtype*) – target dtype for sample, if not None, should match Manifold dtype
- **device** (*torch.device*) – target device for sample, if not None, should match Manifold device

Returns random point on the PoincareBall manifold

Return type *ManifoldTensor*

Notes

The device and dtype will match the device and dtype of the Manifold

random_normal(**size*, *mean=0*, *std=1*, *dtype=None*, *device=None*)
Create a point on the manifold, measure is induced by Normal distribution on the tangent space of zero.

Parameters

- **size** (*shape*) – the desired shape
- **mean** (*float | tensor*) – mean value for the Normal distribution
- **std** (*float | tensor*) – std value for the Normal distribution
- **dtype** (*torch.dtype*) – target dtype for sample, if not None, should match Manifold dtype
- **device** (*torch.device*) – target device for sample, if not None, should match Manifold device

Returns random point on the PoincareBall manifold

Return type *ManifoldTensor*

Notes

The device and dtype will match the device and dtype of the Manifold

retr(*x: torch.Tensor*, *u: torch.Tensor*, ***, *dim=-1*)
Perform a retraction from point x with given direction u.

Parameters

- **x** (*tensor*) – point on the manifold
- **u** (*tensor*) – tangent vector at point x

Returns transported point

Return type tensor

retr_transp (*x: torch.Tensor*, *u: torch.Tensor*, *v: torch.Tensor*, *dim=-1*)
Perform a retraction + vector transport at once.

Parameters

- **x** (*tensor*) – point on the manifold
- **u** (*tensor*) – tangent vector at point x
- **v** (*tensor*) – tangent vector at point x to be transported

Returns transported point and vectors

Return type tuple of tensors

Notes

Sometimes this is a far more optimal way to preform retraction + vector transport

transp (*x: torch.Tensor*, *y: torch.Tensor*, *v: torch.Tensor*, *dim=-1*)
Perform vector transport $\mathfrak{T}_{x\to y}(v)$.

Parameters

- **x** (*tensor*) – start point on the manifold
- **y** (*tensor*) – target point on the manifold
- **v** (*tensor*) – tangent vector at point x

Returns transported tensor(s)

Return type tensor or tuple of tensors

transp_follow_expmap (*x: torch.Tensor*, *u: torch.Tensor*, *v: torch.Tensor*, *dim=-1*, *project=True*)
Perform vector transport following u: $\mathfrak{T}_{x\to\mathrm{Exp}(x,u)}(v)$.

Here, Exp is the best possible approximation of the true exponential map. There are cases when the exact variant is hard or impossible implement, therefore a fallback, non-exact, implementation is used.

Parameters

- **x** (*tensor*) – point on the manifold
- **u** (*tensor*) – tangent vector at point x
- **v** (*tensor*) – tangent vector at point x to be transported

Returns transported tensor

Return type tensor

transp_follow_retr (*x: torch.Tensor*, *u: torch.Tensor*, *v: torch.Tensor*, *dim=-1*)
Perform vector transport following u: $\mathfrak{T}_{x\to\mathrm{retr}(x,u)}(v)$.

This operation is sometimes is much more simpler and can be optimized.

Parameters

- **x** (*tensor*) – point on the manifold
- **u** (*tensor*) – tangent vector at point x
- **v** (*tensor*) – tangent vector at point x to be transported

Returns transported tensor

Return type tensor

class geoopt.manifolds.**PoincareBallExact**(*c=1.0*)

Poincare ball model, see more in *Poincare Ball model*.

Parameters c (*float | tensor*) – ball negative curvature

Notes

It is extremely recommended to work with this manifold in double precision

The implementation of retraction is an exact exponential map, this retraction will be used in optimization.

See also:

PoincareBall

extra_repr()

Set the extra representation of the module

To print customized extra information, you should reimplement this method in your own modules. Both single-line and multi-line strings are acceptable.

retr(*x: torch.Tensor*, *u: torch.Tensor*, ***, *project=True*, *dim=-1*)

Perform an exponential map $\operatorname{Exp}_x(u)$.

Parameters

- **x** (*tensor*) – point on the manifold
- **u** (*tensor*) – tangent vector at point x

Returns transported point

Return type tensor

retr_transp(*x: torch.Tensor*, *u: torch.Tensor*, *v: torch.Tensor*, *dim=-1*, *project=True*)

Perform an exponential map and vector transport from point x with given direction u.

Parameters

- **x** (*tensor*) – point on the manifold
- **u** (*tensor*) – tangent vector at point x
- **v** (*tensor*) – tangent vector at point x to be transported

Returns transported point

Return type tensor

transp_follow_retr(*x: torch.Tensor*, *u: torch.Tensor*, *v: torch.Tensor*, *dim=-1*, *project=True*)

Perform vector transport following u: $\mathfrak{T}_{x\to\operatorname{Exp}(x,u)}(v)$.

Here, Exp is the best possible approximation of the true exponential map. There are cases when the exact variant is hard or impossible implement, therefore a fallback, non-exact, implementation is used.

Parameters

- **x** (*tensor*) – point on the manifold
- **u** (*tensor*) – tangent vector at point x
- **v** (*tensor*) – tangent vector at point x to be transported

Returns transported tensor

Return type tensor

class geoopt.manifolds.**Scaled**(*manifold: geoopt.manifolds.base.Manifold*, *scale=1.0*, *learnable=False*)

Scaled manifold.

Scales all the distances on tha manifold by a constant factor. Scaling may be learnable since the underlying representation is canonical.

Examples

Here is a simple example of radius 2 Sphere

```
>>> import geoopt, torch, numpy as np
>>> sphere = geoopt.Sphere()
>>> radius_2_sphere = Scaled(sphere, 2)
>>> p1 = torch.tensor([-1., 0.])
>>> p2 = torch.tensor([0., 1.])
>>> np.testing.assert_allclose(sphere.dist(p1, p2), np.pi / 2)
>>> np.testing.assert_allclose(radius_2_sphere.dist(p1, p2), np.pi)
```

egrad2rgrad(*x: torch.Tensor*, *u: torch.Tensor*, ***kwargs*)

Transform gradient computed using autodiff to the correct Riemannian gradient for the point x.

Parameters

- **x** (*tensor*) – point on the manifold
- **u** (*tensor*) – gradient to be projected

Returns grad vector in the Riemannian manifold

Return type tensor

inner(*x: torch.Tensor*, *u: torch.Tensor*, *v: torch.Tensor = None*, ***, *keepdim=False*, ***kwargs*)

Inner product for tangent vectors at point x.

Parameters

- **x** (*tensor*) – point on the manifold
- **u** (*tensor*) – tangent vector at point x
- **v** (*tensor (optional)*) – tangent vector at point x
- **keepdim** (*bool*) – keep the last dim?

Returns inner product (broadcasted)

Return type scalar

norm(*x: torch.Tensor*, *u: torch.Tensor*, ***, *keepdim=False*, ***kwargs*)

Norm of a tangent vector at point x.

Parameters

- **x** (*tensor*) – point on the manifold

- **u** (*tensor*) – tangent vector at point x
- **keepdim** (*bool*) – keep the last dim?

Returns inner product (broadcasted)

Return type scalar

proju (*x: torch.Tensor, u: torch.Tensor, **kwargs*)
Project vector u on a tangent space for x, usually is the same as `egrad2rgrad()`.

Parameters

- **x** (*tensor*) – point on the manifold
- **u** (*tensor*) – vector to be projected

Returns projected vector

Return type tensor

projx (*x: torch.Tensor, **kwargs*)
Project point x on the manifold.

Parameters **x** (*tensor*) – point to be projected

Returns projected point

Return type tensor

random (**size, dtype=None, device=None, **kwargs*)
Random sampling on the manifold.

The exact implementation depends on manifold and usually does not follow all assumptions about uniform measure, etc.

transp (*x: torch.Tensor, y: torch.Tensor, v: torch.Tensor, **kwargs*)
Perform vector transport $\mathfrak{T}_{x\to y}(v)$.

Parameters

- **x** (*tensor*) – start point on the manifold
- **y** (*tensor*) – target point on the manifold
- **v** (*tensor*) – tangent vector at point x

Returns transported tensor(s)

Return type tensor or tuple of tensors

class geoopt.manifolds.**ProductManifold** (**manifolds_with_shape*)
Product Manifold.

Examples

A Torus

```
>>> import geoopt
>>> sphere = geoopt.Sphere()
>>> torus = ProductManifold((sphere, 2), (sphere, 2))
```

component_inner (*x: torch.Tensor, u: torch.Tensor, v=None*)
Inner product for tangent vectors at point x according to components of the manifold.

The result of the function is same as `inner` with `keepdim=True` for all the manifolds except Product-Manifold. For this manifold it acts different way computing inner product for each component and then building an output correctly tiling and reshaping the result.

Parameters

- **x** (*tensor*) – point on the manifold
- **u** (*tensor*) – tangent vector at point x
- **v** (*tensor (optional)*) – tangent vector at point x

Returns inner product component wise (broadcasted)

Return type tensor

Notes

The purpose of this method is better adaptive properties in optimization since ProductManifold will "hide" the structure in public API.

dist (*x*, *y*, *, *keepdim=False*)
Compute distance between 2 points on the manifold that is the shortest path along geodesics.

Parameters

- **x** (*tensor*) – point on the manifold
- **y** (*tensor*) – point on the manifold
- **keepdim** (*bool*) – keep the last dim?

Returns distance between two points

Return type scalar

dist2 (*x: torch.Tensor*, *y: torch.Tensor*, *, *keepdim=False*)
Compute squared distance between 2 points on the manifold that is the shortest path along geodesics.

Parameters

- **x** (*tensor*) – point on the manifold
- **y** (*tensor*) – point on the manifold
- **keepdim** (*bool*) – keep the last dim?

Returns squared distance between two points

Return type scalar

egrad2rgrad (*x: torch.Tensor*, *u: torch.Tensor*)
Transform gradient computed using autodiff to the correct Riemannian gradient for the point x.

Parameters

- **x** (*tensor*) – point on the manifold
- **u** (*tensor*) – gradient to be projected

Returns grad vector in the Riemannian manifold

Return type tensor

expmap (*x: torch.Tensor*, *u: torch.Tensor*)
Perform an exponential map $\mathrm{Exp}_x(u)$.

Parameters

- **x** (*tensor*) – point on the manifold
- **u** (*tensor*) – tangent vector at point x

Returns transported point

Return type tensor

expmap_transp (*x: torch.Tensor*, *u: torch.Tensor*, *v: torch.Tensor*)
Perform an exponential map and vector transport from point x with given direction u.

Parameters

- **x** (*tensor*) – point on the manifold
- **u** (*tensor*) – tangent vector at point x
- **v** (*tensor*) – tangent vector at point x to be transported

Returns transported point

Return type tensor

classmethod from_point (**parts*, *batch_dims=0*)
Construct Product manifold from given points.

Parameters

- **parts** (*tuple[geoopt.ManifoldTensor]*) – Manifold tensors to construct Product manifold from
- **batch_dims** (*int*) – number of first dims to treat as batch dims and not include in the Product manifold

Returns

Return type *ProductManifold*

inner (*x: torch.Tensor*, *u: torch.Tensor*, *v=None*, ***, *keepdim=False*)
Inner product for tangent vectors at point x.

Parameters

- **x** (*tensor*) – point on the manifold
- **u** (*tensor*) – tangent vector at point x
- **v** (*tensor (optional)*) – tangent vector at point x
- **keepdim** (*bool*) – keep the last dim?

Returns inner product (broadcasted)

Return type scalar

logmap (*x: torch.Tensor*, *y: torch.Tensor*)
Perform an logarithmic map $\operatorname{Log}_x(y)$.

Parameters

- **x** (*tensor*) – point on the manifold
- **y** (*tensor*) – point on the manifold

Returns tangent vector

Return type tensor

origin(**size*, *dtype=None*, *device=None*, *seed=42*) → geoopt.tensor.ManifoldTensor
Create some reasonable point on the manifold in a deterministic way.

For some manifolds there may exist e.g. zero vector or some analogy. In case it is possible to define this special point, this point is returned with the desired size. In other case, the returned point is sampled on the manifold in a deterministic way.

Parameters

- **size** (*shape*) – the desired shape
- **device** (*torch.device*) – the desired device
- **dtype** (*torch.dtype*) – the desired dtype
- **seed** (*int*) – A parameter controlling deterministic randomness for manifolds that do not provide `.origin`, but provide `.random`. (default: 42)

Returns

Return type tensor

pack_point(**tensors*) → torch.Tensor
Construct a tensor representation of a manifold point.

In case of regular manifolds this will return the same tensor. However, for e.g. Product manifold this function will pack all non-batch dimensions.

Parameters **tensors** (*List[torch.Tensor]*) –

Returns

Return type torch.Tensor

proju(*x: torch.Tensor*, *u: torch.Tensor*)
Project vector u on a tangent space for x, usually is the same as `egrad2rgrad()`.

Parameters

- **x** (*tensor*) – point on the manifold
- **u** (*tensor*) – vector to be projected

Returns projected vector

Return type tensor

projx(*x: torch.Tensor*)
Project point x on the manifold.

Parameters **x** (*tensor*) – point to be projected

Returns projected point

Return type tensor

retr(*x: torch.Tensor*, *u: torch.Tensor*)
Perform a retraction from point x with given direction u.

Parameters

- **x** (*tensor*) – point on the manifold
- **u** (*tensor*) – tangent vector at point x

Returns transported point

Return type tensor

retr_transp (*x: torch.Tensor*, *u: torch.Tensor*, *v: torch.Tensor*)
Perform a retraction + vector transport at once.

Parameters

- **x** (*tensor*) – point on the manifold
- **u** (*tensor*) – tangent vector at point x
- **v** (*tensor*) – tangent vector at point x to be transported

Returns transported point and vectors

Return type tuple of tensors

Notes

Sometimes this is a far more optimal way to preform retraction + vector transport

take_submanifold_value (*x: torch.Tensor*, *i: int*, *reshape=True*) → torch.Tensor
Take i'th slice of the ambient tensor and possibly reshape.

Parameters

- **x** (*tensor*) – Ambient tensor
- **i** (*int*) – submanifold index
- **reshape** (*bool*) – reshape the slice?

Returns

Return type torch.Tensor

transp (*x: torch.Tensor*, *y: torch.Tensor*, *v: torch.Tensor*)
Perform vector transport $\mathfrak{T}_{x\to y}(v)$.

Parameters

- **x** (*tensor*) – start point on the manifold
- **y** (*tensor*) – target point on the manifold
- **v** (*tensor*) – tangent vector at point x

Returns transported tensor(s)

Return type tensor or tuple of tensors

transp_follow_expmap (*x: torch.Tensor*, *u: torch.Tensor*, *v: torch.Tensor*)
Perform vector transport following u: $\mathfrak{T}_{x\to\mathrm{Exp}(x,u)}(v)$.

Here, Exp is the best possible approximation of the true exponential map. There are cases when the exact variant is hard or impossible implement, therefore a fallback, non-exact, implementation is used.

Parameters

- **x** (*tensor*) – point on the manifold
- **u** (*tensor*) – tangent vector at point x
- **v** (*tensor*) – tangent vector at point x to be transported

Returns transported tensor

Return type tensor

transp_follow_retr (*x*, *u*, *v*)
Perform vector transport following u: $\mathfrak{T}_{x\to\operatorname{retr}(x,u)}(v)$.

This operation is sometimes is much more simpler and can be optimized.

Parameters

- **x** (*tensor*) – point on the manifold
- **u** (*tensor*) – tangent vector at point x
- **v** (*tensor*) – tangent vector at point x to be transported

Returns transported tensor

Return type tensor

unpack_tensor (*tensor: torch.Tensor*) → Tuple[torch.Tensor]
Construct a point on the manifold.

This method should help to work with product and compound manifolds. Internally all points on the manifold are stored in an intuitive format. However, there might be cases, when this representation is simpler or more efficient to store in a different way that is hard to use in practice.

Parameters **tensor** (*torch.Tensor*) –

Returns

Return type torch.Tensor

2.2 Optimizers

class geoopt.optim.**RiemannianAdam** (**args*, *stabilize=None*, ***kwargs*)
Riemannian Adam with the same API as `torch.optim.Adam`.

Parameters

- **params** (*iterable*) – iterable of parameters to optimize or dicts defining parameter groups
- **lr** (*float (optional)*) – learning rate (default: 1e-3)
- **betas** (*Tuple[float, float] (optional)*) – coefficients used for computing running averages of gradient and its square (default: (0.9, 0.999))
- **eps** (*float (optional)*) – term added to the denominator to improve numerical stability (default: 1e-8)
- **weight_decay** (*float (optional)*) – weight decay (L2 penalty) (default: 0)
- **amsgrad** (*bool (optional)*) – whether to use the AMSGrad variant of this algorithm from the paper On the Convergence of Adam and Beyond (default: False)

Other Parameters **stabilize** (*int*) – Stabilize parameters if they are off-manifold due to numerical reasons every `stabilize` steps (default: `None` – no stabilize)

step (*closure=None*)
Performs a single optimization step.

Parameters **closure** (*callable, optional*) – A closure that reevaluates the model and returns the loss.

class geoopt.optim.**RiemannianSGD**(*params*, *lr*, *momentum=0*, *dampening=0*, *weight_decay=0*, *nesterov=False*, *stabilize=None*)

Riemannian Stochastic Gradient Descent with the same API as `torch.optim.SGD`.

Parameters

- **params** (*iterable*) – iterable of parameters to optimize or dicts defining parameter groups
- **lr** (*float*) – learning rate
- **momentum** (*float (optional)*) – momentum factor (default: 0)
- **weight_decay** (*float (optional)*) – weight decay (L2 penalty) (default: 0)
- **dampening** (*float (optional)*) – dampening for momentum (default: 0)
- **nesterov** (*bool (optional)*) – enables Nesterov momentum (default: False)

Other Parameters **stabilize** (*int*) – Stabilize parameters if they are off-manifold due to numerical reasons every `stabilize` steps (default: `None` – no stabilize)

step(*closure=None*)

Performs a single optimization step (parameter update).

Parameters **closure** (*callable*) – A closure that reevaluates the model and returns the loss. Optional for most optimizers.

2.3 Tensors

class geoopt.tensor.**ManifoldTensor**

Same as `torch.Tensor` that has information about its manifold.

Other Parameters **manifold** (`geoopt.Manifold`) – A manifold for the tensor, (default: `geoopt.Euclidean`)

as_point()

Construct a point on the manifold.

This method should help to work with product and compound manifolds. Internally all points on the manifold are stored in an intuitive format. However, there might be cases, when this representation is simpler or more efficient to store in a different way that is hard to use in practice.

Parameters **tensor** (*torch.Tensor*) –

Returns

Return type torch.Tensor

dist(*other*, *p=2*, ***kwargs*)

Return euclidean or geodesic distance between points on the manifold. Allows broadcasting.

Parameters

- **other** (*tensor*) –
- **p** (*str|int*) – The norm to use. The default behaviour is not changed and is just euclidean distance. To compute geodesic distance, `p` should be set to `"g"`

Returns

Return type scalar

expmap(*u*, ***kwargs*)
Perform an exponential map $\text{Exp}_x(u)$.

Parameters **u** (*tensor*) – tangent vector at point x

Returns transported point

Return type tensor

expmap_transp(*u*, *v*, ***kwargs*)
Perform an exponential map and vector transport from point x with given direction u.

Parameters

- **u** (*tensor*) – tangent vector at point x
- **v** (*tensor*) – tangent vector at point x to be transported

Returns transported point

Return type tensor

inner(*u*, *v=None*, ***kwargs*)
Inner product for tangent vectors at point x.

Parameters

- **u** (*tensor*) – tangent vector at point x
- **v** (*tensor (optional)*) – tangent vector at point x
- **keepdim** (*bool*) – keep the last dim?

Returns inner product (broadcasted)

Return type scalar

logmap(*y*, ***kwargs*)
Perform an logarithmic map $\text{Log}_x(y)$.

Parameters **y** (*tensor*) – point on the manifold

Returns tangent vector

Return type tensor

proj_()
Inplace projection to the manifold.

Returns same instance

Return type tensor

proju(*u*, ***kwargs*)
Project vector u on a tangent space for x, usually is the same as `egrad2rgrad()`.

Parameters **u** (*tensor*) – vector to be projected

Returns projected vector

Return type tensor

retr(*u*, ***kwargs*)
Perform a retraction from point x with given direction u.

Parameters **u** (*tensor*) – tangent vector at point x

Returns transported point

Return type tensor

retr_transp (*u*, *v*, ***kwargs*)

Perform a retraction + vector transport at once.

Parameters

- **u** (*tensor*) – tangent vector at point x
- **v** (*tensor*) – tangent vector at point x to be transported

Returns transported point and vectors

Return type tuple of tensors

Notes

Sometimes this is a far more optimal way to preform retraction + vector transport

transp (*y*, *v*, ***kwargs*)

Perform vector transport $\mathfrak{T}_{x\to y}(v)$.

Parameters

- **y** (*tensor*) – target point on the manifold
- **v** (*tensor*) – tangent vector at point x

Returns transported tensor(s)

Return type tensor or tuple of tensors

transp_follow_expmap (*u*, *v*, ***kwargs*)

Perform vector transport following u: $\mathfrak{T}_{x\to \mathrm{Exp}(x,u)}(v)$.

Here, Exp is the best possible approximation of the true exponential map. There are cases when the exact variant is hard or impossible implement, therefore a fallback, non-exact, implementation is used.

Parameters

- **u** (*tensor*) – tangent vector at point x
- **v** (*tensor*) – tangent vector at point x to be transported

Returns transported tensor

Return type tensor

transp_follow_retr (*u*, *v*, ***kwargs*)

Perform vector transport following u: $\mathfrak{T}_{x\to \mathrm{retr}(x,u)}(v)$.

This operation is sometimes is much more simpler and can be optimized.

Parameters

- **u** (*tensor*) – tangent vector at point x
- **v** (*tensor*) – tangent vector at point x to be transported

Returns transported tensor

Return type tensor

class geoopt.tensor.**ManifoldParameter**

Same as `torch.nn.Parameter` that has information about its manifold.

It should be used within `torch.nn.Module` to be recognized in parameter collection.

Other Parameters **manifold** (`geoopt.Manifold` (optional)) – A manifold for the tensor if `data` is not a `geoopt.ManifoldTensor`

2.4 Samplers

class `geoopt.samplers.`**`RHMC`**(*params*, *epsilon=0.001*, *n_steps=1*)

Riemannian Hamiltonian Monte-Carlo.

Parameters

- **params** (*iterable*) – iterables of tensors for which to perform sampling
- **epsilon** (*float*) – step size
- **n_steps** (*int*) – number of leapfrog steps

step(*closure*)

Perform a single sampling step.

Parameters **closure** (*callable*) – A closure that reevaluates the model and returns the log probability.

class `geoopt.samplers.`**`RSGLD`**(*params*, *epsilon=0.001*)

Riemannian Stochastic Gradient Langevin Dynamics.

Parameters

- **params** (*iterable*) – iterables of tensors for which to perform sampling
- **epsilon** (*float*) – step size

step(*closure*)

Perform a single sampling step.

Parameters **closure** (*callable*) – A closure that reevaluates the model and returns the log probability.

class `geoopt.samplers.`**`SGRHMC`**(*params*, *epsilon=0.001*, *n_steps=1*, *alpha=0.1*)

Stochastic Gradient Riemannian Hamiltonian Monte-Carlo.

Parameters

- **params** (*iterable*) – iterables of tensors for which to perform sampling
- **epsilon** (*float*) – step size
- **n_steps** (*int*) – number of leapfrog steps
- **alpha** (*float*) – $(1 - alpha)$ – momentum term

step(*closure*)

Perform a single sampling step.

Parameters **closure** (*callable*) – A closure that reevaluates the model and returns the log probability.

2.5 Extended Guide

2.5.1 Poincare Ball model

Poincare ball model is a compact representation of hyperbolic space. To have a nice introduction into this model we should start from simple concepts, putting them all together to build a more complete picture.

Hyperbolic spaces

Hyperbolic space is a constant negative curvature Riemannian manifold. A very simple example of Riemannian manifold with constant, but positive curvature is sphere.

An (N+1)-dimensional hyperboloid spans the manifold that can be embedded into N-dimensional space via projections.

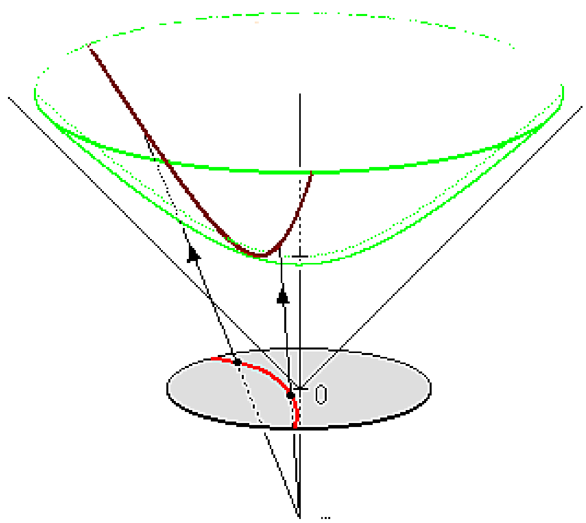

Fig. 1: img source Wikipedia, Hyperboloid Model

Originally, the distance between points on the hyperboloid is defined as

$$d(x, y) = \operatorname{arccosh}(x, y)$$

It is difficult to work in (N+1)-dimensional space and there is a range of useful embeddings exist in literature

Klein Model

Poincare Model

Here we go.

First of all we note, that Poincare ball is embedded in a Sphere of radius $r = 1/\sqrt{c}$, where c is negative curvature. We also note, as c goes to 0, we recover infinite radius ball. We should expect this limiting behaviour recovers Euclidean geometry.

To connect Euclidean space with its embedded manifold we need to get g_x. It is done via *conformal factor* λ^c_x.

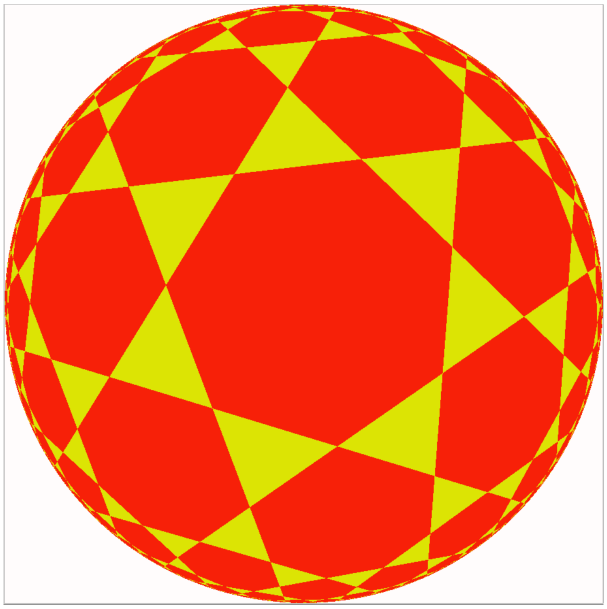

Fig. 2: img source Wikipedia, Klein Model

Fig. 3: img source Bulatov, Poincare Model

`geoopt.manifolds.poincare.math.`**`lambda_x`**(*x*, ***, *c=1.0*, *keepdim=False*, *dim=-1*)

Compute the conformal factor λ^c_x for a point on the ball.

$$\lambda^c_x = \frac{1}{1 - c\|x\|_2^2}$$

Parameters

- **x** (*tensor*) – point on the Poincare ball
- **c** (*float | tensor*) – ball negative curvature
- **keepdim** (*bool*) – retain the last dim? (default: false)
- **dim** (*int*) – reduction dimension

Returns conformal factor

Return type tensor

λ^c_x connects Euclidean inner product with Riemannian one

`geoopt.manifolds.poincare.math.`**`inner`**(*x*, *u*, *v*, ***, *c=1.0*, *keepdim=False*, *dim=-1*)

Compute inner product for two vectors on the tangent space w.r.t Riemannian metric on the Poincare ball.

$$\langle u, v\rangle_x = (\lambda^c_x)^2 \langle u, v\rangle$$

Parameters

- **x** (*tensor*) – point on the Poincare ball
- **u** (*tensor*) – tangent vector to x on Poincare ball
- **v** (*tensor*) – tangent vector to x on Poincare ball
- **c** (*float | tensor*) – ball negative curvature

- **keepdim** (*bool*) – retain the last dim? (default: false)
- **dim** (*int*) – reduction dimension

Returns inner product

Return type tensor

`geoopt.manifolds.poincare.math.`**`norm`**(*x*, *u*, ***, *c=1.0*, *keepdim=False*, *dim=-1*)

Compute vector norm on the tangent space w.r.t Riemannian metric on the Poincare ball.

$$\|u\|_x = \lambda_x^c \|u\|_2$$

Parameters

- **x** (*tensor*) – point on the Poincare ball
- **u** (*tensor*) – tangent vector to x on Poincare ball
- **c** (*float | tensor*) – ball negative curvature
- **keepdim** (*bool*) – retain the last dim? (default: false)
- **dim** (*int*) – reduction dimension

Returns norm of vector

Return type tensor

`geoopt.manifolds.poincare.math.`**`egrad2rgrad`**(*x*, *grad*, ***, *c=1.0*, *dim=-1*)

Translate Euclidean gradient to Riemannian gradient on tangent space of x.

$$\nabla_x = \nabla_x^E / (\lambda_x^c)^2$$

Parameters

- **x** (*tensor*) – point on the Poincare ball
- **grad** (*tensor*) – Euclidean gradient for x
- **c** (*float | tensor*) – ball negative curvature
- **dim** (*int*) – reduction dimension for operations

Returns Riemannian gradient $u \in T_x\mathbb{D}_c^n$

Return type tensor

Math

The good thing about Poincare ball is that it forms a Gyrogroup. Minimal definition of a Gyrogroup assumes a binary operation $*$ defined that satisfies a set of properties.

Left identity For every element $a \in G$ there exist $e \in G$ such that $e * a = a$.

Left Inverse For every element $a \in G$ there exist $b \in G$ such that $b * a = e$

Gyroassociativity For any $a, b, c \in G$ there exist $gyr[a, b]c \in G$ such that $a * (b * c) = (a * b) * gyr[a, b]c$

Gyroautomorphism $gyr[a, b]$ is a magma automorphism in G

Left loop $gyr[a, b] = gyr[a * b, b]$

As mentioned above, hyperbolic space forms a Gyrogroup equipped with geoopt.manifolds.poincare.math.**mobius_add**(*x*, *y*, *, *c=1.0*, *dim=-1*)

Compute Mobius addition is a special operation in a hyperbolic space.

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

In general this operation is not commutative:

$$x \oplus_c y \ne y \oplus_c x$$

But in some cases this property holds:

- zero vector case

$$\mathbf{0} \oplus_c x = x \oplus_c \mathbf{0}$$

- zero negative curvature case that is same as Euclidean addition

$$x \oplus_0 y = y \oplus_0 x$$

Another useful property is so called left-cancellation law:

$$(-x) \oplus_c (x \oplus_c y) = y$$

Parameters

- **x** (*tensor*) – point on the Poincare ball
- **y** (*tensor*) – point on the Poincare ball
- **c** (*float|tensor*) – ball negative curvature
- **dim** (*int*) – reduction dimension for operations

Returns the result of mobius addition

Return type tensor

geoopt.manifolds.poincare.math.**gyration**(*a*, *b*, *u*, *, *c=1.0*, *dim=-1*)

Apply gyration $\operatorname{gyr}[u, v]w$.

Guration is a special operation in hyperbolic geometry. Addition operation $\oplus_c$ is not associative (as mentioned in *mobius_add()*), but gyroassociative which means

$$u \oplus_c (v \oplus_c w) = (u\oplus_c v) \oplus_c \operatorname{gyr}[u, v]w,$$

where

$$\operatorname{gyr}[u, v]w = \ominus (u \oplus_c v) \oplus (u \oplus_c (v \oplus_c w))$$

We can simplify this equation using explicit formula for Mobius addition [1]. Recall

$$\begin{aligned} A = -c^2 \langle u, w\rangle \langle v, v\rangle + c\langle v, w\rangle + 2c^2 \langle u, v\rangle \langle v, w\rangle\\ B = -c^2 \langle v, w\rangle \langle u, u\rangle - c\langle u, w\rangle\\ D = 1 + 2c\langle u, v\rangle + c^2 \langle u, u\rangle \langle v, v\rangle \end{aligned}$$

$$\operatorname{gyr}[u, v]w = w + 2\frac{Au + Bv}{D}$$

Parameters

- **a** (*tensor*) – first point on Poincare ball
- **b** (*tensor*) – second point on Poincare ball
- **u** (*tensor*) – vector field for operation
- **c** (*float | tensor*) – ball negative curvature
- **dim** (*int*) – reduction dimension for operations

Returns the result of automorphism

Return type tensor

References

[1] A. A. Ungar (2009), A Gyrovector Space Approach to Hyperbolic Geometry

Using this math, it is possible to define another useful operations

geoopt.manifolds.poincare.math.**mobius_sub**(*x, y, *, c=1.0, dim=-1*)
Compute Mobius substraction.

Mobius substraction can be represented via Mobius addition as follows:

$$x \ominus_c y = x \oplus_c (-y)$$

Parameters

- **x** (*tensor*) – point on Poincare ball
- **y** (*tensor*) – point on Poincare ball
- **c** (*float | tensor*) – ball negative curvature
- **dim** (*int*) – reduction dimension for operations

Returns the result of mobius substraction

Return type tensor

geoopt.manifolds.poincare.math.**mobius_scalar_mul**(*r, x, *, c=1.0, dim=-1*)
Compute left scalar multiplication on the Poincare ball.

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

This operation has properties similar to Euclidean

- *n-addition* property

$$r \otimes_c x = x \oplus_c \cdots \oplus_c x$$

- Distributive property

$$(r_1 + r_2) \otimes_c x = r_1 \otimes_c x \oplus r_2 \otimes_c x$$

- Scalar associativity

$$(r_1 r_2) \otimes_c x = r_1 \otimes_c (r_2 \otimes_c x)$$

- Scaling property

$$|r| \otimes_c x / \|r \otimes_c x\|_2 = x / \|x\|_2$$

Parameters

- **r** (*float | tensor*) – scalar for multiplication
- **x** (*tensor*) – point on Poincare ball
- **c** (*float | tensor*) – ball negative curvature
- **dim** (*int*) – reduction dimension for operations

Returns the result of mobius scalar multiplication

Return type tensor

`geoopt.manifolds.poincare.math.`**`mobius_pointwise_mul`**(*w*, *x*, *, *c=1.0*, *dim=-1*)

Compute a generalization for point-wise multiplication to hyperbolic space.

Mobius pointwise multiplication is defined as follows

$$\operatorname{diag}(w) \otimes_c x = (1/\sqrt{c}) \tanh\left(\frac{\|\operatorname{diag}(w)x\|_2}{x} \tanh^{-1}(\sqrt{c}\|x\|_2)\right) \frac{\|\operatorname{diag}(w)x\|_2}{\|x\|_2}$$

Parameters

- **w** (*tensor*) – weights for multiplication (should be broadcastable to x)
- **x** (*tensor*) – point on Poincare ball
- **c** (*float | tensor*) – negative ball curvature
- **dim** (*int*) – reduction dimension for operations

Returns Mobius point-wise mul result

Return type tensor

`geoopt.manifolds.poincare.math.`**`mobius_matvec`**(*m*, *x*, *, *c=1.0*, *dim=-1*)

Compute a generalization for matrix-vector multiplication to hyperbolic space.

Mobius matrix vector operation is defined as follows:

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

Parameters

- **m** (*tensor*) – matrix for multiplication. Batched matmul is performed if `m.dim() > 2`, but only last dim reduction is supported
- **x** (*tensor*) – point on Poincare ball
- **c** (*float | tensor*) – negative ball curvature
- **dim** (*int*) – reduction dimension for operations

Returns Mobius matvec result

Return type tensor

`geoopt.manifolds.poincare.math.`**`mobius_fn_apply`**(*fn*, *x*, **args*, *c=1.0*, *dim=-1*, ***kwargs*)

Compute a generalization for function application in hyperbolic space.

First, hyperbolic vector is mapped to a Euclidean space via Log^c_0 and nonlinear function is applied in this tangent space. The resulting vector is then mapped back with Exp^c_0

$$f^{\otimes_c}(x) = \operatorname{Exp}^c_0(f(\operatorname{Log}^c_0(y)))$$

Parameters

- **x** (*tensor*) – point on Poincare ball
- **fn** (*callable*) – function to apply
- **c** (*float | tensor*) – ball negative curvature
- **dim** (*int*) – reduction dimension for operations

Returns Result of function in hyperbolic space

Return type tensor

`geoopt.manifolds.poincare.math.`**`mobius_fn_apply_chain`**(*x*, **fns*, *c=1.0*, *dim=-1*)

Compute a generalization for sequential function application in hyperbolic space.

First, hyperbolic vector is mapped to a Euclidean space via Log^c_0 and nonlinear function is applied in this tangent space. The resulting vector is then mapped back with Exp^c_0

$$f^{\otimes_c}(x) = \operatorname{Exp}^c_0(f(\operatorname{Log}^c_0(y)))$$

The definition of mobius function application allows chaining as

$$y = \operatorname{Exp}^c_0(\operatorname{Log}^c_0(y))$$

Resulting in

$$(f \circ g)^{\otimes_c}(x) = \operatorname{Exp}^c_0((f \circ g)(\operatorname{Log}^c_0(y)))$$

Parameters

- **x** (*tensor*) – point on Poincare ball
- **fns** (*callable[]*) – functions to apply
- **c** (*float | tensor*) – ball negative curvature
- **dim** (*int*) – reduction dimension for operations

Returns Apply chain result

Return type tensor

Manifold

Now we are ready to proceed with studying distances, geodesics, exponential maps and more

`geoopt.manifolds.poincare.math.`**`dist`**(*x*, *y*, ***, *c=1.0*, *keepdim=False*, *dim=-1*)

Compute geodesic distance on the Poincare ball.

$$d_c(x, y) = \frac{2}{\sqrt{c}} \tanh^{-1}(\sqrt{c}\|(-x) \oplus_c y\|_2)$$

Parameters

- **x** (*tensor*) – point on Poincare ball
- **y** (*tensor*) – point on Poincare ball
- **c** (*float | tensor*) – ball negative curvature
- **keepdim** (*bool*) – retain the last dim? (default: false)
- **dim** (*int*) – reduction dimension

Returns geodesic distance between x and y

Return type tensor

`geoopt.manifolds.poincare.math.`**`dist2plane`**(*x*, *p*, *a*, ***, *c=1.0*, *keepdim=False*, *signed=False*, *dim=-1*)

Compute geodesic distance from x to a hyperbolic hyperplane in Poincare ball.

The distance is computed to a plane that is orthogonal to a and contains p.

To form an intuition what is a hyperbolic hyperplane, let's first consider Euclidean hyperplane

$$H_{a,b} = \{x \in \mathbb{R}^n \ : \ \langle x, a\rangle - b = 0\},$$

where $a \in \mathbb{R}^n \backslash \{\mathbf{0}\}$ and $b \in \mathbb{R}^n$.

This formulation of a hyperplane is hard to generalize, therefore we can rewrite $\langle x, a\rangle - b$ utilizing orthogonal completion. Setting any p s.t. $b = \langle a, p\rangle$ we have

$$\begin{aligned} H_{a,b} &= \{x \in \mathbb{R}^n \ : \ \langle x, a\rangle - b = 0\} \\ &= H_{a,\langle a,p\rangle} = \tilde{H}_{a,p} \\ &= \{x \in \mathbb{R}^n \ : \ \langle x, a\rangle - \langle a, p\rangle = 0\} \\ &= \{x \in \mathbb{R}^n \ : \ \langle -p + x, a\rangle = 0\} \\ &= p + \{a\}^\perp \end{aligned}$$

Naturally we have a set $\{a\}^\perp$ with applied $+$ operator to each element. Generalizing a notion of summation to the hyperbolic space we replace $+$ with $\oplus_c$.

Next, we should figure out what is $\{a\}^\perp$ in the Poincare ball.

First thing that we should acknowledge is that notion of orthogonality is defined for vectors in tangent spaces. Let's consider now $p \in \mathbb{D}_c^n$ and $a \in T_p\mathbb{D}_c^n \backslash \{\mathbf{0}\}$.

Slightly deviating from traditional notation let's write $\{a\}_p^\perp$ highlighting the tight relationship of $a \in T_p\mathbb{D}_c^n \backslash \{\mathbf{0}\}$ with $p \in \mathbb{D}_c^n$. We then define

$$\{a\}_p^\perp := \{z \in T_p\mathbb{D}_c^n \ : \ \langle z, a\rangle_p = 0\}$$

Recalling that a tangent vector z for point p yields $x = \mathrm{Exp}_p^c(z)$ we rewrite the above equation as

$$\{a\}_p^\perp := \left\{x \in \mathbb{D}_c^n \ : \ \langle \mathrm{Log}_p^c(x), a\rangle_p = 0\right\}$$

This formulation is something more pleasant to work with. Putting all together

$$\begin{aligned} \tilde{H}_{a,p}^c &= p + \{a\}_p^\perp \\ &= \left\{x \in \mathbb{D}_c^n \ : \ \langle \mathrm{Log}_p^c(x), a\rangle_p = 0\right\} \\ &= \{x \in \mathbb{D}_c^n \ : \ \langle -p \oplus_c x, a\rangle = 0\} \end{aligned}$$

To compute the distance $d_c(x, \tilde{H}_{a,p}^c)$ we find

$$\begin{aligned} d_c(x, \tilde{H}_{a,p}^c) &= \inf_{w \in \tilde{H}_{a,p}^c} d_c(x, w) \\ &= \frac{1}{\sqrt{c}} \sinh^{-1}\left\{\frac{2\sqrt{c}|\langle(-p) \oplus_c x, a\rangle|}{(1 - c\|(-p) \oplus_c x\|_2^2)\|a\|_2}\right\} \end{aligned}$$

Parameters

- **x** (*tensor*) – point on Poincare ball
- **a** (*tensor*) – vector on tangent space of p

- **p** (*tensor*) – point on Poincare ball lying on the hyperplane
- **c** (*float | tensor*) – ball negative curvature
- **keepdim** (*bool*) – retain the last dim? (default: false)
- **signed** (*bool*) – return signed distance
- **dim** (*int*) – reduction dimension for operations

Returns distance to the hyperplane

Return type tensor

`geoopt.manifolds.poincare.math.`**`parallel_transport`**(*x, y, v, *, c=1.0, dim=-1*)

Perform parallel transport on the Poincare ball.

Parallel transport is essential for adaptive algorithms in Riemannian manifolds. For Hyperbolic spaces parallel transport is expressed via gyration.

To recover parallel transport we first need to study isomorphism between gyrovectors and vectors. The reason is that originally, parallel transport is well defined for gyrovectors as

$$P_{x\to y}(z) = \operatorname{gyr}[y, -x]z,$$

where $x,\ y,\ z \in \mathbb{D}_c^n$ and $\operatorname{gyr}[a, b]c = \ominus(a \oplus_c b) \oplus_c (a \oplus_c (b \oplus_c c))$

But we want to obtain parallel transport for vectors, not for gyrovectors. The blessing is isomorphism mentioned above. This mapping is given by

$$U^c_p\ :\ T_p\mathbb{D}^n_c \to \mathbb{G} = v \mapsto \lambda^c_p v$$

Finally, having points $x,\ y \in \mathbb{D}^n_c$ and a tangent vector $u \in T_x\mathbb{D}^n_c$ we obtain

$$\begin{aligned} P^c_{x\to y}(v) &= (U^c_y)^{-1}\left(\operatorname{gyr}[y, -x]U^c_x(v)\right) \\ &= \operatorname{gyr}[y, -x]v\lambda^c_x/\lambda^c_y \end{aligned}$$

Parameters

- **x** (*tensor*) – starting point
- **y** (*tensor*) – end point
- **v** (*tensor*) – tangent vector to be transported
- **c** (*float | tensor*) – ball negative curvature
- **dim** (*int*) – reduction dimension for operations

Returns transported vector

Return type tensor

`geoopt.manifolds.poincare.math.`**`geodesic`**(*t, x, y, *, c=1.0, dim=-1*)

Compute geodesic at the time point t.

Geodesic (the shortest) path connecting x and y. The path can be treated as and extension of a line segment between points but in a Riemannian manifold. In Poincare ball model, the path is expressed using Mobius addition and scalar multiplication:

$$\gamma_{x\to y}(t) = x \oplus_c r \otimes_c ((-x) \oplus_c y)$$

The required properties of this path are the following:

$$\gamma_{x\to y}(0) = x$$
$$\gamma_{x\to y}(1) = y$$
$$\dot\gamma_{x\to y}(t) = v$$

Moreover, as geodesic path is not only the shortest path connecting points and Poincare ball. This definition also requires local distance minimization and thus another property appears:

$$d_c(\gamma_{x\to y}(t_1), \gamma_{x\to y}(t_2)) = v|t_1 - t_2|$$

"Natural parametrization" of the curve ensures unit speed geodesics which yields the above formula with $v = 1$. However, for Poincare ball we can always compute the constant speed v from the points that particular path connects:

$$v = d_c(\gamma_{x\to y}(0), \gamma_{x\to y}(1)) = d_c(x, y)$$

Parameters

- **t** (*float | tensor*) – travelling time
- **x** (*tensor*) – starting point on Poincare ball
- **y** (*tensor*) – target point on Poincare ball
- **c** (*float | tensor*) – ball negative curvature
- **dim** (*int*) – reduction dimension for operations

Returns point on the Poincare ball

Return type tensor

`geoopt.manifolds.poincare.math.`**`geodesic_unit`**(*t, x, u, *, c=1.0, dim=-1*)

Compute unit speed geodesic at time t starting from x with direction $u/\|u\|_x$.

$$\gamma_{x,u}(t) = x \oplus_c \tanh(t\sqrt{c}/2)\frac{u}{\sqrt{c}\|u\|_2}$$

Parameters

- **t** (*tensor*) – travelling time
- **x** (*tensor*) – initial point
- **u** (*tensor*) – direction
- **c** (*float | tensor*) – ball negative curvature
- **dim** (*int*) – reduction dimension for operations

Returns the point on geodesic line

Return type tensor

`geoopt.manifolds.poincare.math.`**`expmap`**(*x, u, *, c=1.0, dim=-1*)

Compute exponential map on the Poincare ball.

Exponential map for Poincare ball model. This is tightly related with `geodesic()`. Intuitively Exponential map is a smooth constant travelling from starting point x with speed u.

A bit more formally this is travelling along curve $\gamma_{x,u}(t)$ such that

$$\gamma_{x,u}(0) = x$$
$$\dot\gamma_{x,u}(0) = u$$
$$\|\dot\gamma_{x,u}(t)\|_{\gamma_{x,u}(t)} = \|u\|_x$$

The existence of this curve relies on uniqueness of differential equation solution, that is local. For the Poincare ball model the solution is well defined globally and we have.

$$\operatorname{Exp}_x^c(u) = \gamma_{x,u}(1) = \\ x \oplus_c \tanh(\sqrt{c}/2\|u\|_x)\frac{u}{\sqrt{c}\|u\|_2}$$

Parameters

- **x** (*tensor*) – starting point on Poincare ball
- **u** (*tensor*) – speed vector on Poincare ball
- **c** (*float | tensor*) – ball negative curvature
- **dim** (*int*) – reduction dimension for operations

Returns $\gamma_{x,u}(1)$ end point

Return type tensor

`geoopt.manifolds.poincare.math.`**`expmap0`**(*u*, ***, *c=1.0*, *dim=-1*)

Compute exponential map for Poincare ball model from 0.

$$\operatorname{Exp}_0^c(u) = \tanh(\sqrt{c}/2\|u\|_2)\frac{u}{\sqrt{c}\|u\|_2}$$

Parameters

- **u** (*tensor*) – speed vector on Poincare ball
- **c** (*float | tensor*) – ball negative curvature
- **dim** (*int*) – reduction dimension for operations

Returns $\gamma_{0,u}(1)$ end point

Return type tensor

`geoopt.manifolds.poincare.math.`**`logmap`**(*x*, *y*, ***, *c=1.0*, *dim=-1*)

Compute logarithmic map for two points x and y on the manifold.

$$\operatorname{Log}_x^c(y) = \frac{2}{\sqrt{c}\lambda_x^c}\tanh^{-1}(\sqrt{c}\|(-x)\oplus_c y\|_2) * \frac{(-x)\oplus_c y}{\|(-x)\oplus_c y\|_2}$$

The result of Logarithmic map is a vector such that

$$y = \operatorname{Exp}_x^c(\operatorname{Log}_x^c(y))$$

Parameters

- **x** (*tensor*) – starting point on Poincare ball
- **y** (*tensor*) – target point on Poincare ball
- **c** (*float | tensor*) – ball negative curvature
- **dim** (*int*) – reduction dimension for operations

Returns tangent vector that transports x to y

Return type tensor

`geoopt.manifolds.poincare.math.`**`logmap0`**(*y*, ***, *c=1.0*, *dim=-1*)

Compute logarithmic map for y from 0 on the manifold.

$$\operatorname{Log}_0^c(y) = \tanh^{-1}(\sqrt{c}\|y\|_2)\frac{y}{\|y\|_2}$$

The result is such that

$$y = \operatorname{Exp}_0^c(\operatorname{Log}_0^c(y))$$

Parameters

- **y** (*tensor*) – target point on Poincare ball
- **c** (*float|tensor*) – ball negative curvature
- **dim** (*int*) – reduction dimension for operations

Returns tangent vector that transports 0 to y

Return type tensor

Stability

Numerical stability is a pain in this model. It is strongly recommended to work in `float64`, so expect adventures in `float32` (but this is not certain).

`geoopt.manifolds.poincare.math.`**`project`**(*x*, ***, *c=1.0*, *dim=-1*, *eps=None*)

Safe projection on the manifold for numerical stability.

Parameters

- **x** (*tensor*) – point on the Poincare ball
- **c** (*float|tensor*) – ball negative curvature
- **dim** (*int*) – reduction dimension to compute norm
- **eps** (*float*) – stability parameter, uses default for dtype if not provided

Returns projected vector on the manifold

Return type tensor

2.6 Developer Guide

2.6.1 Base Manifold

The common base class for all manifolds is `geoopt.manifolds.base.Manifold`.

class `geoopt.manifolds.base.`**`Manifold`**(***kwargs*)

`_assert_check_shape`(*shape*, *name*)

Util to check shape and raise an error if needed.

Exhaustive implementation for checking if a given point has valid dimension size, shape, etc. It will raise a ValueError if check is not passed

Parameters

- **shape** (*tuple*) – shape of point on the manifold
- **name** (*str*) – name to be present in errors

Raises ValueError

_check_point_on_manifold (*x: torch.Tensor*, *, *atol=1e-05*, *rtol=1e-05*)
Util to check point lies on the manifold.

Exhaustive implementation for checking if a given point lies on the manifold. It should return boolean and a reason of failure if check is not passed. You can assume assert_check_point is already passed beforehand

Parameters

- **x** (*tensor*) – point on the manifold
- **atol** (*float*) – absolute tolerance as in numpy.allclose()
- **rtol** (*float*) – relative tolerance as in numpy.allclose()

Returns check result and the reason of fail if any

Return type bool, str or None

_check_shape (*shape*, *name*)
Util to check shape.

Exhaustive implementation for checking if a given point has valid dimension size, shape, etc. It should return boolean and a reason of failure if check is not passed

Parameters

- **shape** (*tuple*) – shape of point on the manifold
- **name** (*str*) – name to be present in errors

Returns check result and the reason of fail if any

Return type bool, str or None

_check_vector_on_tangent (*x: torch.Tensor*, *u: torch.Tensor*, *, *atol=1e-05*, *rtol=1e-05*)
Util to check a vector belongs to the tangent space of a point.

Exhaustive implementation for checking if a given point lies in the tangent space at x of the manifold. It should return a boolean indicating whether the test was passed and a reason of failure if check is not passed. You can assume assert_check_point is already passed beforehand

Parameters

- **x** (*tensor*) –
- **u** (*tensor*) –
- **atol** (*float*) – absolute tolerance
- **rtol** – relative tolerance

Returns check result and the reason of fail if any

Return type bool, str or None

assert_check_point (*x: torch.Tensor*)
Check if point is valid to be used with the manifold and raise an error with informative message on failure.

Parameters **x** (*tensor*) – point on the manifold

Notes

This check is compatible to what optimizer expects, last dimensions are treated as manifold dimensions

assert_check_point_on_manifold(*x: torch.Tensor*, ***, *atol=1e-05*, *rtol=1e-05*)
Check if point :math'x' is lying on the manifold and raise an error with informative message on failure.

Parameters

- **x** (*tensor*) – point on the manifold
- **atol** (*float*) – absolute tolerance as in `numpy.allclose()`
- **rtol** (*float*) – relative tolerance as in `numpy.allclose()`

assert_check_vector(*u: torch.Tensor*)
Check if vector is valid to be used with the manifold and raise an error with informative message on failure.

Parameters u (*tensor*) – vector on the tangent plane

Notes

This check is compatible to what optimizer expects, last dimensions are treated as manifold dimensions

assert_check_vector_on_tangent(*x: torch.Tensor*, *u: torch.Tensor*, ***, *ok_point=False*, *atol=1e-05*, *rtol=1e-05*)
Check if u u is lying on the tangent space to x and raise an error on fail.

Parameters

- **x** (*tensor*) – point on the manifold
- **u** (*tensor*) – vector on the tangent space to x
- **atol** (*float*) – absolute tolerance as in `numpy.allclose()`
- **rtol** (*float*) – relative tolerance as in `numpy.allclose()`
- **ok_point** (*bool*) – is a check for point required?

check_point(*x: torch.Tensor*, ***, *explain=False*)
Check if point is valid to be used with the manifold.

Parameters

- **x** (*tensor*) – point on the manifold
- **explain** (*bool*) – return an additional information on check

Returns boolean indicating if tensor is valid and reason of failure if False

Return type bool

Notes

This check is compatible to what optimizer expects, last dimensions are treated as manifold dimensions

check_point_on_manifold(*x: torch.Tensor*, ***, *explain=False*, *atol=1e-05*, *rtol=1e-05*)
Check if point x is lying on the manifold.

Parameters

- **x** (*tensor*) – point on the manifold
- **atol** (*float*) – absolute tolerance as in `numpy.allclose()`

- **rtol** (*float*) – relative tolerance as in `numpy.allclose()`
- **explain** (*bool*) – return an additional information on check

Returns boolean indicating if tensor is valid and reason of failure if False

Return type bool

Notes

This check is compatible to what optimizer expects, last dimensions are treated as manifold dimensions

check_vector(*u: torch.Tensor*, ***, *explain=False*)
Check if vector is valid to be used with the manifold.

Parameters

- **u** (*tensor*) – vector on the tangent plane
- **explain** (*bool*) – return an additional information on check

Returns boolean indicating if tensor is valid and reason of failure if False

Return type bool

Notes

This check is compatible to what optimizer expects, last dimensions are treated as manifold dimensions

check_vector_on_tangent(*x: torch.Tensor*, *u: torch.Tensor*, ***, *ok_point=False*, *explain=False*, *atol=1e-05*, *rtol=1e-05*)
Check if u is lying on the tangent space to x.

Parameters

- **x** (*tensor*) – point on the manifold
- **u** (*tensor*) – vector on the tangent space to x
- **atol** (*float*) – absolute tolerance as in `numpy.allclose()`
- **rtol** (*float*) – relative tolerance as in `numpy.allclose()`
- **explain** (*bool*) – return an additional information on check
- **ok_point** (*bool*) – is a check for point required?

Returns boolean indicating if tensor is valid and reason of failure if False

Return type bool

component_inner(*x: torch.Tensor*, *u: torch.Tensor*, *v=None*)
Inner product for tangent vectors at point x according to components of the manifold.

The result of the function is same as `inner` with `keepdim=True` for all the manifolds except ProductManifold. For this manifold it acts different way computing inner product for each component and then building an output correctly tiling and reshaping the result.

Parameters

- **x** (*tensor*) – point on the manifold
- **u** (*tensor*) – tangent vector at point x
- **v** (*tensor (optional)*) – tangent vector at point x

Returns inner product component wise (broadcasted)

Return type tensor

Notes

The purpose of this method is better adaptive properties in optimization since ProductManifold will "hide" the structure in public API.

device
Manifold device.

Returns

Return type Optional[torch.device]

dist (*x: torch.Tensor*, *y: torch.Tensor*, *, *keepdim=False*)
Compute distance between 2 points on the manifold that is the shortest path along geodesics.

Parameters

- **x** (*tensor*) – point on the manifold
- **y** (*tensor*) – point on the manifold
- **keepdim** (*bool*) – keep the last dim?

Returns distance between two points

Return type scalar

dist2 (*x: torch.Tensor*, *y: torch.Tensor*, *, *keepdim=False*)
Compute squared distance between 2 points on the manifold that is the shortest path along geodesics.

Parameters

- **x** (*tensor*) – point on the manifold
- **y** (*tensor*) – point on the manifold
- **keepdim** (*bool*) – keep the last dim?

Returns squared distance between two points

Return type scalar

dtype
Manifold dtype.

Returns

Return type Optional[torch.dtype]

egrad2rgrad (*x: torch.Tensor*, *u: torch.Tensor*)
Transform gradient computed using autodiff to the correct Riemannian gradient for the point x.

Parameters

- **x** (*tensor*) – point on the manifold
- **u** (*tensor*) – gradient to be projected

Returns grad vector in the Riemannian manifold

Return type tensor

expmap(*x: torch.Tensor*, *u: torch.Tensor*)
Perform an exponential map $\mathrm{Exp}_x(u)$.

Parameters

- **x** (`tensor`) – point on the manifold
- **u** (`tensor`) – tangent vector at point x

Returns transported point

Return type tensor

expmap_transp(*x: torch.Tensor*, *u: torch.Tensor*, *v: torch.Tensor*)
Perform an exponential map and vector transport from point x with given direction u.

Parameters

- **x** (`tensor`) – point on the manifold
- **u** (`tensor`) – tangent vector at point x
- **v** (`tensor`) – tangent vector at point x to be transported

Returns transported point

Return type tensor

extra_repr()
Set the extra representation of the module

To print customized extra information, you should reimplement this method in your own modules. Both single-line and multi-line strings are acceptable.

inner(*x: torch.Tensor*, *u: torch.Tensor*, *v=None*, ***, *keepdim=False*)
Inner product for tangent vectors at point x.

Parameters

- **x** (`tensor`) – point on the manifold
- **u** (`tensor`) – tangent vector at point x
- **v** (`tensor (optional)`) – tangent vector at point x
- **keepdim** (`bool`) – keep the last dim?

Returns inner product (broadcasted)

Return type scalar

logmap(*x: torch.Tensor*, *y: torch.Tensor*)
Perform an logarithmic map $\mathrm{Log}_x(y)$.

Parameters

- **x** (`tensor`) – point on the manifold
- **y** (`tensor`) – point on the manifold

Returns tangent vector

Return type tensor

norm(*x: torch.Tensor*, *u: torch.Tensor*, ***, *keepdim=False*)
Norm of a tangent vector at point x.

Parameters

- **x** (*tensor*) – point on the manifold
- **u** (*tensor*) – tangent vector at point x
- **keepdim** (*bool*) – keep the last dim?

Returns inner product (broadcasted)

Return type scalar

origin(**size*, *dtype=None*, *device=None*, *seed=42*)

Create some reasonable point on the manifold in a deterministic way.

For some manifolds there may exist e.g. zero vector or some analogy. In case it is possible to define this special point, this point is returned with the desired size. In other case, the returned point is sampled on the manifold in a deterministic way.

Parameters

- **size** (*shape*) – the desired shape
- **device** (*torch.device*) – the desired device
- **dtype** (*torch.dtype*) – the desired dtype
- **seed** (*int*) – A parameter controlling deterministic randomness for manifolds that do not provide `.origin`, but provide `.random`. (default: 42)

Returns

Return type tensor

pack_point(**tensors*) → torch.Tensor

Construct a tensor representation of a manifold point.

In case of regular manifolds this will return the same tensor. However, for e.g. Product manifold this function will pack all non-batch dimensions.

Parameters **tensors** (*List[torch.Tensor]*) –

Returns

Return type torch.Tensor

proju(*x: torch.Tensor*, *u: torch.Tensor*)

Project vector u on a tangent space for x, usually is the same as `egrad2rgrad()`.

Parameters

- **x** (*tensor*) – point on the manifold
- **u** (*tensor*) – vector to be projected

Returns projected vector

Return type tensor

projx(*x: torch.Tensor*)

Project point x on the manifold.

Parameters **x** (*tensor*) – point to be projected

Returns projected point

Return type tensor

random (**size*, *dtype=None*, *device=None*, ***kwargs*)
Random sampling on the manifold.

The exact implementation depends on manifold and usually does not follow all assumptions about uniform measure, etc.

retr (*x: torch.Tensor*, *u: torch.Tensor*)
Perform a retraction from point x with given direction u.

Parameters

- **x** (*tensor*) – point on the manifold
- **u** (*tensor*) – tangent vector at point x

Returns transported point

Return type tensor

retr_transp (*x: torch.Tensor*, *u: torch.Tensor*, *v: torch.Tensor*)
Perform a retraction + vector transport at once.

Parameters

- **x** (*tensor*) – point on the manifold
- **u** (*tensor*) – tangent vector at point x
- **v** (*tensor*) – tangent vector at point x to be transported

Returns transported point and vectors

Return type tuple of tensors

Notes

Sometimes this is a far more optimal way to preform retraction + vector transport

transp (*x: torch.Tensor*, *y: torch.Tensor*, *v: torch.Tensor*)
Perform vector transport $\mathfrak{T}_{x\to y}(v)$.

Parameters

- **x** (*tensor*) – start point on the manifold
- **y** (*tensor*) – target point on the manifold
- **v** (*tensor*) – tangent vector at point x

Returns transported tensor(s)

Return type tensor or tuple of tensors

transp_follow_expmap (*x: torch.Tensor*, *u: torch.Tensor*, *v: torch.Tensor*)
Perform vector transport following u: $\mathfrak{T}_{x\to\mathrm{Exp}(x,u)}(v)$.

Here, Exp is the best possible approximation of the true exponential map. There are cases when the exact variant is hard or impossible implement, therefore a fallback, non-exact, implementation is used.

Parameters

- **x** (*tensor*) – point on the manifold
- **u** (*tensor*) – tangent vector at point x
- **v** (*tensor*) – tangent vector at point x to be transported

Returns transported tensor

Return type tensor

transp_follow_retr (*x: torch.Tensor*, *u: torch.Tensor*, *v: torch.Tensor*)
Perform vector transport following u: $\mathfrak{T}_{x\to\text{retr}(x,u)}(v)$.

This operation is sometimes is much more simpler and can be optimized.

Parameters

- **x** (*tensor*) – point on the manifold
- **u** (*tensor*) – tangent vector at point x
- **v** (*tensor*) – tangent vector at point x to be transported

Returns transported tensor

Return type tensor

unpack_tensor (*tensor: torch.Tensor*) → torch.Tensor
Construct a point on the manifold.

This method should help to work with product and compound manifolds. Internally all points on the manifold are stored in an intuitive format. However, there might be cases, when this representation is simpler or more efficient to store in a different way that is hard to use in practice.

Parameters **tensor** (*torch.Tensor*) –

Returns

Return type torch.Tensor

class geoopt.manifolds.base.**ScalingStorage**
Helper class to make implementation transparent.

This is just a dictionary with additional overriden `__call__` for more explicit and elegant API to declare members. A usage example may be found in `Manifold`.

Methods that require rescaling when wrapped into `Scaled` should be defined as follows

1. Regular methods like `dist`, `dist2`, `expmap`, `retr` etc. that are already present in the base class do not require registration, it has already happened in the base `Manifold` class.

2. New methods (like in `PoincareBall`) should be treated with care.

```
class PoincareBall(Manifold):
    # make a class copy of __scaling__ info. Default methods are already present␣
↪there
    __scaling__ = Manifold.__scaling__.copy()
    ... # here come regular implementation of the required methods

    @__scaling__(ScalingInfo(1))  # rescale output according to rule `out *␣
↪scaling ** 1`
    def dist0(self, x: torch.Tensor, *, dim=-1, keepdim=False):
        return math.dist0(x, c=self.c, dim=dim, keepdim=keepdim)

    @__scaling__(ScalingInfo(u=-1))  # rescale argument `u` according to the rule␣
↪`out * scaling ** -1`
    def expmap0(self, u: torch.Tensor, *, dim=-1, project=True):
        res = math.expmap0(u, c=self.c, dim=dim)
        if project:
            return math.project(res, c=self.c, dim=dim)
```

(continues on next page)

(continued from previous page)

```
        else:
            return res
    ... # other special methods implementation
```

3. Some methods are not compliant with the above rescaling rules. We should mark them as *NotCompatible*

```
# continuation of the PoincareBall definition
@__scaling__(ScalingInfo.NotCompatible)
def mobius_fn_apply(
    self, fn: callable, x: torch.Tensor, *args, dim=-1, project=True, **kwargs
):
    res = math.mobius_fn_apply(fn, x, *args, c=self.c, dim=dim, **kwargs)
    if project:
        return math.project(res, c=self.c, dim=dim)
    else:
        return res
```

copy() → a shallow copy of D

class geoopt.manifolds.base.**ScalingInfo**(**results*, ***kwargs*)

Scaling info for each argument that requires rescaling.

```
scaled_value = value * scaling ** power if power != 0 else value
```

For results it is not always required to set powers of scaling, then it is no-op.

The convention for this info is the following. The output of a function is either a tuple or a single object. In any case, outputs are treated as positionals. Function inputs, in contrast, are treated by keywords. It is a common practice to maintain function signature when overriding, so this way may be considered as a sufficient in this particular scenario. The only required info for formula above is `power`.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

g

Index

A

C

D

E

F

G

I

L

M

N

O

P

R

S

T

U