Fast geometric libraries for vision and data sciences

Jean Feydy
Imperial College London, INRIA Paris

18th of November, 2021
Université de Picardie Jules Verne
Équipe Perception Robotique
Who am I?

Background in **mathematics** and **data sciences**:


2014–2015  M2 mathematics, vision, learning at ENS Cachan.

2016–2019  PhD thesis in *medical imaging* with Alain Trouvé at ENS Cachan.

2019–2021  **Geometric deep learning** with Michael Bronstein at Imperial College.

2021+  **Medical data analysis** in the HeKA INRIA team (Paris).

Close ties with **healthcare**:

2015  Image denoising with **Siemens Healthcare** in Princeton.

2019+  MasterClass AI–Imaging, for **radiology interns** in the University of Paris.

2020+  Colloquium on **Medical imaging in the AI era** at the Paris Brain Institute.
My motivation: medical data analysis

Three main characteristics:

- **Heterogeneous data**: patient history, images, etc.
- Small stratified samples: 10 – 1 000 patients per group.
- Dealing with **outliers** and the **heavy tails** of our distributions is a priority.
Computational anatomy [CSG19, LSG\textsuperscript{+}18, CMN14]

Detect a pattern.  
Analyze a variation.  
Register a model.

Some characteristics, in the wider context of computer vision research:

- **Standard acquisitions**, without occlusions.
- **Precision** work (at millimeter scale).
- Need for **guarantees** of robustness and regularity.
A field that is moving fast

Target. Design models that combine medical expertise with modern datasets.

Challenge. The advent of Graphics Processing Units (GPU):

- Incredible value for money:
  \[1 \text{,000€} \approx 1 \text{,000 cores} \approx 10^{12} \text{ operations/s}.\]
- Bottleneck: constraints on register usage.

“User-friendly” Python ecosystem, consolidated around a small number of key operations.

7,000 cores in a single GPU.
My project: a long-term investment in the foundations of our field

Solution. Expand the standard toolbox in data sciences to deal with the challenges of the healthcare industry.

Ease the development of advanced models, without compromising on numerical performance.

In-depth work, numerical foundations $\rightarrow$ high-level applications:

1. Efficient manipulation of "symbolic" matrices (distances, kernel, etc.).
3. Geometric deep learning and biomedical applications.

Future of these tools and clinical perspectives.
1. Symbolic matrices
Computing libraries represent most objects as tensors

**Context.** Constrained **memory accesses** on the GPU:

- **Long access times** to the registers penalize the use of large **dense** arrays.
- Hard-wired **contiguous** memory accesses penalize the use of **sparse** matrices.

**Challenge.** In order to reach optimal run times:

- **Restrict** ourselves to operations that are supported by the constructor: convolutions, FFT, etc.
- Develop new routines from scratch in C++/CUDA (FAISS, KPConv...): **several months of work**.
The KeOps library: efficient support for symbolic matrices

**Solution.** KeOps – [www.kernel-operations.io](http://www.kernel-operations.io):

- For PyTorch, NumPy, Matlab and R, on **CPU and GPU**.
- **Automatic differentiation**.
- Just-in-time **compilation** of optimized C++ schemes, triggered for every new **reduction**: sum, min, etc.

If the formula “F” is simple (\(\leq 100\) arithmetic operations):

- “100k \times 100k” computation \(\rightarrow\) 10ms – 100ms,
- “1M \times 1M” computation \(\rightarrow\) 1s – 10s.

**Hardware ceiling of** \(10^{12}\) operations/s.

- **\(\times 10\)** to **\(\times 100\) speed-up** vs standard GPU implementations for a wide range of problems.

Symbolic matrix

Formula + data

- Distances \(d(x_i, y_j)\).
- Kernel \(k(x_i, y_j)\).
- Numerous transforms.
A first example: efficient nearest neighbor search in dimension 50

Create large point clouds using standard PyTorch syntax:

```python
import torch
N, M, D = 10**6, 10**6, 50
x = torch.rand(N, 1, D).cuda() # (1M, 1, 50) array
y = torch.rand(1, M, D).cuda() # (1, 1M, 50) array
```

Turn dense arrays into symbolic matrices:

```python
from pykeops.torch import LazyTensor
x_i, y_j = LazyTensor(x), LazyTensor(y)
```

Create a large symbolic matrix of squared distances:

```python
D_ij = ((x_i - y_j) ** 2).sum(dim=2) # (1M, 1M) symbolic
```

Use an `.argmin()` reduction to perform a nearest neighbor query:

```python
indices_i = D_ij.argmin(dim=1) # -> standard torch tensor
```
The KeOps library combines performance with flexibility

Script of the previous slide = efficient nearest neighbor query, on par with the bruteforce CUDA scheme of the FAISS library...

And can be used with any metric!

\[
\begin{align*}
D_{ij} &= ((x_i - x_j) ** 2).sum(dim=2) & \# \text{ Euclidean} \\
M_{ij} &= (x_i - x_j).abs().sum(dim=2) & \# \text{ Manhattan} \\
C_{ij} &= 1 - (x_i \mid x_j) & \# \text{ Cosine} \\
H_{ij} &= D_{ij} / (x_i[...,0] * x_j[...,0]) & \# \text{ Hyperbolic}
\end{align*}
\]

KeOps supports arbitrary formulas and variables with:

- **Reductions**: sum, log-sum-exp, K-min, matrix-vector product, etc.
- **Operations**: +, ×, sqrt, exp, neural networks, etc.
- **Advanced schemes**: batch processing, block sparsity, etc.
- **Automatic differentiation**: seamless integration with PyTorch.
KeOps lets users work with millions of points at a time.

Benchmark of a Gaussian convolution between clouds of N 3D points on a RTX 2080 Ti GPU.

<table>
<thead>
<tr>
<th>Number of points N</th>
<th>NumPy (CPU)</th>
<th>PyTorch (GPU)</th>
<th>KeOps (GPU)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1k</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10k</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100k</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1M</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

out of memory!
Applications
KeOps is a good fit for machine learning research

K-Means.

Gaussian Mixture Model.

Use any kernel, metric or formula you like!
KeOps is a good fit for machine learning research

Spectral analysis.

UMAP in hyperbolic space.

Use any kernel, metric or formula you like!
Applications to Kriging, spline, Gaussian process, kernel regression

A standard tool for regression [Lec18]:

Under the hood, solve a kernel linear system:

\[(\lambda \text{Id} + K_{xx})a = b\]  
\[\text{i.e.} \quad a \leftarrow (\lambda \text{Id} + K_{xx})^{-1}b\]

where \(\lambda \geq 0\) et \((K_{xx})_{i,j} = k(x_i, x_j)\) is a positive definite matrix.
KeOps symbolic tensors \( (K_{xx})_{i,j} = k(x_i, x_j) \):

- Can be fed to **standard solvers**: SciPy, GPyTorch, etc.
- GPytorch on the 3DRoad dataset \((N = 278k, D = 3)\):
  
  \[ 7h \text{ avec } 8 \text{ GPUs} \quad \rightarrow \quad 15mn \text{ avec } 1 \text{ GPU}. \]

- Provide a **fast backend for research codes**: see e.g. *Kernel methods through the roof: handling billions of points efficiently*, by G. Meanti, L. Carratino, L. Rosasco, A. Rudi (2020).
2. Optimal transport
Optimal transport (OT) generalizes sorting to spaces of dimension $D > 1$

**Context.** If $A = (x_1, \ldots, x_N)$ and $B = (y_1, \ldots, y_N)$ are two clouds of $N$ points in $\mathbb{R}^D$, we define:

$$ OT(A, B) = \min_{\sigma \in S_N} \frac{1}{2N} \sum_{i=1}^{N} \| x_i - y_{\sigma(i)} \|^2 $$

Generalizes sorting to metric spaces.

**Linear problem** on the permutation matrix $P$:

$$ OT(A, B) = \min_{P \in \mathbb{R}^{N \times N}} \frac{1}{2N} \sum_{i,j=1}^{N} P_{i,j} \cdot \| x_i - y_j \|^2, $$

s.t. $P_{i,j} \geq 0$  $\sum_j P_{i,j} = 1$  $\sum_i P_{i,j} = 1$.

Each source point is transported onto the target.

assignment $\sigma : [1, 5] \rightarrow [1, 5]$
Key properties of this distance “up to permutations”

The Wasserstein distance $\sqrt{\text{OT}(A, B)}$ is:

- **Symmetric**: $\text{OT}(A, B) = \text{OT}(B, A)$.
- **Positive**: $\text{OT}(A, B) \geq 0$.
- **Definite**: $\text{OT}(A, B) = 0 \iff A = B$.
- **Translation-aware**: $\text{OT}(A, \text{Translate}_\vec{v}(A)) = \frac{1}{2} \| \vec{v} \|^2$.
- More generally, $\text{OT}$ retrieves the unique gradient of a convex function $T = \nabla \phi$ that maps $A$ onto $B$:
  - In dimension 1, $(x_i - x_j) \cdot (y_{\sigma(i)} - y_{\sigma(j)}) \geq 0$.
  - In dimension $D$, $\langle x_i - x_j, T(x_i) - T(x_j) \rangle_{\mathbb{R}^D} \geq 0$.

$\implies$ Appealing generalization of an increasing mapping.
A simple example in 2D

\[ t = .00 \]
A simple example in 2D

t = .25
A simple example in 2D

\[ t = 0.50 \]
A simple example in 2D

t = 1.00
A simple example in 2D

$t = 5.00$
A simple example in 2D

\[ t = 10.00 \]
An efficient model... but beware of tears!

Before

After
An efficient model... but beware of tears!

Before

After
An efficient model... but beware of tears!

Before

After
An efficient model... but beware of tears!
Geometric solutions to least square problems [AC11]

Barycenter \( A^* = \arg \min_A \sum_{i=1}^{4} \lambda_i \text{Loss}(A, B_i) \).

Euclidean barycenters.
\[
\text{Loss}(A, B) = \|A - B\|_{L^2}^2
\]

Wasserstein barycenters.
\[
\text{Loss}(A, B) = \text{OT}(A, B)
\]
Regularization and entropic bias

**Challenge.** Linear assignment: hard to solve in the general case.

**Structure** of the distance matrix $\| x_i - y_j \|$

$\implies$ **Speed-up** computations.

Fundamental tool: **regularized transport**

$\text{OT}_\varepsilon(A, B) \simeq \text{OT}(A, B) + \text{entropic penalty}$ with strength $\varepsilon > 0$.

**Smooth** and strictly **convex** approximation: easier to study, most popular **Sinkhorn** (or “SoftAssign”) algorithm.

On the other hand, does not define a **distance**:

$\text{OT}_\varepsilon(B, B) > 0$. 

$\sigma = \sqrt{\varepsilon}$

$\arg \min_A \text{OT}_\varepsilon(A, B)$. 

Solution. Sinkhorn divergences are defined with:

\[ S_\varepsilon(A, B) = OT_\varepsilon(A, B) - \frac{1}{2} OT_\varepsilon(A, A) - \frac{1}{2} OT_\varepsilon(B, B) \]

in order to get a null value when \( A = B \).

Theorem (\( S_\varepsilon \) is well suited for optimization)

For all samples \( A \) and \( B \):

\[ S_\varepsilon(A, B) \geq 0 \quad \text{with equality iff.} \quad A = B, \]

\( A \mapsto S_\varepsilon(A, B) \) is convex, differentiable

and metrizes the convergence in law.

We generalize this result to positive Radon measures, arbitrary metrics \( \| x_i - y_j \| \) and to the “unbalanced” setting.

\[ \arg \min_A S_\varepsilon(A, B). \]
How should we solve the OT problem?

Key dates for discrete optimal transport with N points:

- [Kan42]: Dual problem of Kantorovitch.
- [Kuh55]: Hungarian methods in $O(N^3)$.
- [Ber79]: Auction algorithm in $O(N^2)$.
- [KY94]: SoftAssign = Sinkhorn + simulated annealing, in $O(N^2)$.
- [GRL+98, CR00]: Robust Point Matching = Sinkhorn as a loss.
- [Cut13]: Start of the GPU era.
- [Mér11, Lév15, Sch19]: multi-scale solvers in $O(N \log N)$.

**Solution, today:** Multiscale Sinkhorn algorithm, on the GPU.

$\implies$ Generalized **QuickSort** algorithm.
Scaling up optimal transport to anatomical data

Progresses of the last decade add up to a $\times 100 - \times 1000$ acceleration:

Sinkhorn GPU $\rightarrow$ + KeOps $\rightarrow$ + Annealing $\rightarrow$ + Multi-scale

With a precision of 1%, on a modern gaming GPU:

```
pip install geomloss +
modern GPU (1 000 €)
```

10k points in 30-50ms

100k points in 100-200ms
3. Geometric deep learning
Design task-specific trainable models

**Context.** Trainable models on non-Euclidean domains (point clouds, surfaces, graphs, etc.), beyond 2D/3D images.

**Challenge.** In spite of growing interest in the industry, these models still lack support on the numerical side. C++/CUDA is (often) required to reach top performance.

**Solution.** Using KeOps, with a few lines of Python:
- **Local** interactions: K-nearest neighbors.
- **Global** interactions: generalized convolutions.

Modelling freedom

$\implies$ Domain-specific priors.
Lung registration “Exhale – Inhale”

Complex deformations, high resolution (50k–300k points), high accuracy (< 1mm).
Three-steps registration

1. Affine-RobOT pre-alignment.
2.a. Deep prediction network.
2.b. Smooth deformation model.

This **pragmatic** method:

- Is **easy to train** on synthetic data.
- Scales up to high-resolution: 100k points in 1s.
- Excellent results: **KITTI** (outdoors scans) and **DirLab** (lungs).

**Accurate point cloud registration with robust optimal transport**, Shen, Feydy et al., NeurIPS 2021, already on ArXiv.
Three-steps registration

0. Input data
1. Pre-alignment
2. Deep registration
3. Fine-tuning
Conclusion
Key points

- **Symbolic** matrices are to **geometric** ML what **sparse** matrices are to **graph** processing:
  - KeOps: **x30 speed-up** vs. PyTorch, TF et JAX.
  - Useful in a wide range of settings.

- Optimal Transport = **generalized sorting**:
  - Geometric gradients.
  - Super-fast $O(N \log N)$ solvers.

- These tools open **new paths** for geometers and statisticians:
  - GPUs are more **versatile** than you think.
  - Ongoing work to provide **fast GPU backends** to researchers, going beyond what Google and Facebook are ready to pay for.
Summary: a long-term investment that is starting to bear fruits

Two major evolutions:

• “Big” geometric problem: $N > 10k \quad \rightarrow \quad N > 1M$.
• Optimal transport: linear problem + generalized quicksort.
Genuine team work

Alain Trouvé  Thibault Séjourné  F.-X. Vialard  Gabriel Peyré

Benjamin Charlier  Joan Glaunès  Freyr Sverrisson  Shen Zhengyang

+ Marc Niethammer, Bruno Correia, Michael Bronstein...
Our contribution to the community

KeOps and GeomLoss are:

- **Fast**: 10 to 1,000 speedup vs. standard GPU implementations.
- **Memory-efficient**: $O(N)$, not $O(N^2)$.
- **Versatile**, with a **transparent** interface: freedom!
- **Powerful and well-documented**: research-friendly.

- **Slow** with large vectors of dimension $D > 100$.

Coming soon:

- → **Approximation strategies** (Nyström, etc.) in KeOps.
- → **Wasserstein barycenters** and **grid images** in GeomLoss.
Documentation and tutorials are available online

www.kernel-operations.io

www.jeanfeydy.com/geometric_data_analysis.pdf
References
M. Agueh and G. Carlier.

**Barycenters in the Wasserstein space.**


Dimitri P Bertsekas.

**A distributed algorithm for the assignment problem.**

Christophe Chnafa, Simon Mendez, and Franck Nicoud.

Image-based large-eddy simulation in a realistic left heart.


Haili Chui and Anand Rangarajan.

A new algorithm for non-rigid point matching.

Adam Conner-Simons and Rachel Gordon.

**Using ai to predict breast cancer and personalize care.**


MIT CSAIL.

Marco Cuturi.

**Sinkhorn distances: Lightspeed computation of optimal transport.**

Steven Gold, Anand Rangarajan, Chien-Ping Lu, Suguna Pappu, and Eric Mjolsness.  

New algorithms for 2d and 3d point matching: Pose estimation and correspondence.  


Leonid V Kantorovich.  

On the translocation of masses.  

Harold W Kuhn.

The Hungarian method for the assignment problem.


Jeffrey J Kosowsky and Alan L Yuille.

The invisible hand algorithm: Solving the assignment problem with statistical physics.

Florent Leclercq.

Bayesian optimization for likelihood-free cosmological inference.


Bruno Lévy.

A numerical algorithm for l2 semi-discrete optimal transport in 3d.

Christian Ledig, Andreas Schuh, Ricardo Guerrero, Rolf A Heckemann, and Daniel Rueckert.


Quentin Mérigot.

A multiscale approach to optimal transport.

Bernhard Schmitzer.

Stabilized sparse scaling algorithms for entropy regularized transport problems.