Optimal Transport in Python: A Practical Introduction with POT

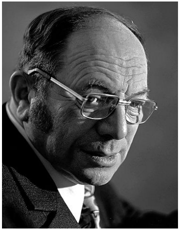
Rémi Flamary, École polytechnique

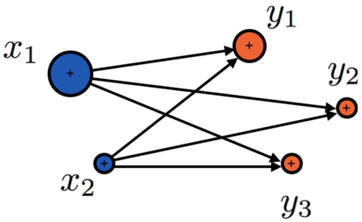
October 30, 2025

PyData Paris 2025, Cité des Sciences et de l'Industrie, France

Table of content

Optimal transport

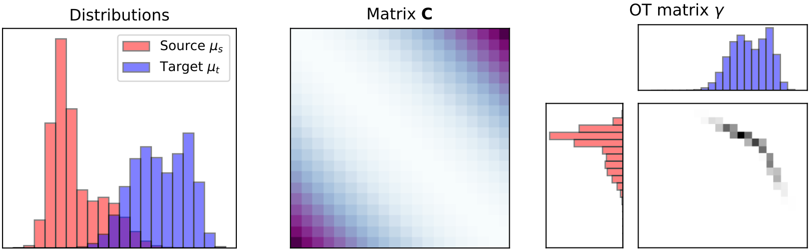




Principle: Move mass in the most efficient way

- Problem introduced by Gaspard Monge in his memoire [Monge, 1781].
- Monge formulation seeks for a mapping between two mass distribution.
- Reformulated by Leonid Kantorovich to allow splitting [Kantorovich, 1942].
- Applications originally for resource allocation problems

Discrete Optimal Transport



Kantorovitch formulation : OT Linear Program

When $\mu_s = \sum_{i=1}^{n_s} a_i \delta_{\mathbf{x}_i^s}$ and $\mu_t = \sum_{i=1}^{n_t} b_i \delta_{\mathbf{x}_i^t}$

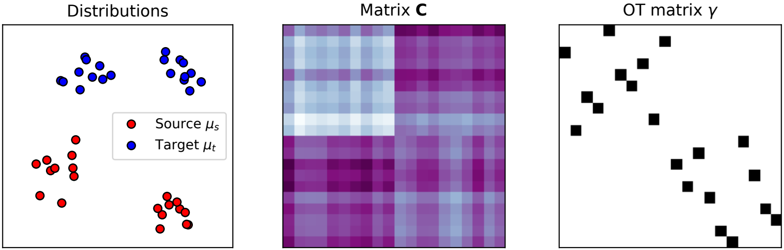
$$W_p^p(\mu_s, \mu_t) = \min_{\boldsymbol{T} \geq \mathbf{0}} \sum_{i,j} T_{i,j} C_{i,j}$$

$$\text{s.t.} \quad \sum_j T_{i,j} = a_i, \forall i \quad \text{and} \quad \sum_i T_{i,j} = b_j, \forall j$$

where $\mathbf{C}$ is a cost matrix with $C_{i,j} = c(\mathbf{x}_i^s, \mathbf{x}_j^t) = \|\mathbf{x}_i^s - \mathbf{x}_j^t\|^p$

- Solving the OT problem with network simplex is $O(n^3 \log(n))$ for $n = n_s = n_t$.
- $W_p(\mu_s, \mu_t)$ is called the Wasserstein distance (EMD for $p = 1$).

Discrete Optimal Transport



Kantorovitch formulation : OT Linear Program

When $\mu_s = \sum_{i=1}^{n_s} a_i \delta_{\mathbf{x}_i^s}$ and $\mu_t = \sum_{i=1}^{n_t} b_i \delta_{\mathbf{x}_i^t}$

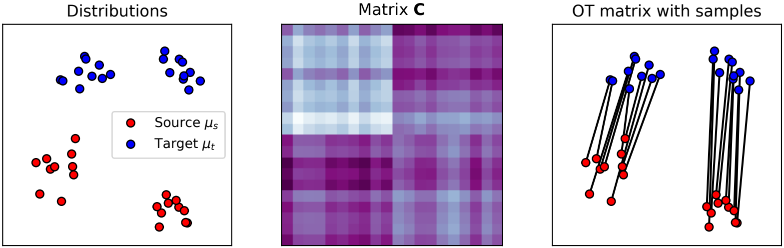
$$W_p^p(\mu_s, \mu_t) = \min_{\boldsymbol{T} \geq \mathbf{0}} \sum_{i,j} T_{i,j} C_{i,j}$$

$$\text{s.t.} \quad \sum_j T_{i,j} = a_i, \forall i \quad \text{and} \quad \sum_i T_{i,j} = b_j, \forall j$$

where $\mathbf{C}$ is a cost matrix with $C_{i,j} = c(\mathbf{x}_i^s, \mathbf{x}_j^t) = \|\mathbf{x}_i^s - \mathbf{x}_j^t\|^p$

- Solving the OT problem with network simplex is $O(n^3 \log(n))$ for $n = n_s = n_t$.
- $W_p(\mu_s, \mu_t)$ is called the Wasserstein distance (EMD for $p = 1$).

Discrete Optimal Transport



Kantorovitch formulation : OT Linear Program

When $\mu_s = \sum_{i=1}^{n_s} a_i \delta_{\mathbf{x}_i^s}$ and $\mu_t = \sum_{i=1}^{n_t} b_i \delta_{\mathbf{x}_i^t}$

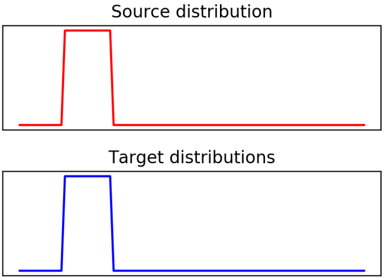
$$W_p^p(\mu_s, \mu_t) \quad = \quad \min_{\boldsymbol{T} \geq \mathbf{0}} \quad \sum_{i,j} T_{i,j} C_{i,j}$$

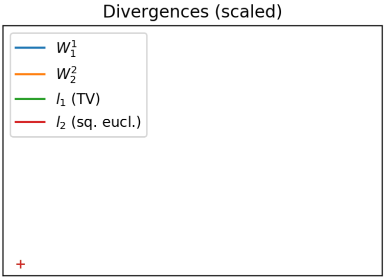
$$\text{s.t.} \quad \sum_j T_{i,j} = a_i, \forall i \quad \text{and} \quad \sum_i T_{i,j} = b_j, \forall j$$

where $\mathbf{C}$ is a cost matrix with $C_{i,j} = c(\mathbf{x}_i^s, \mathbf{x}_j^t) = \|\mathbf{x}_i^s - \mathbf{x}_j^t\|^p$

- Solving the OT problem with network simplex is $O(n^3 \log(n))$ for $n = n_s = n_t$.
- $W_p(\mu_s, \mu_t)$ is called the Wasserstein distance (EMD for $p = 1$).

Wasserstein distance





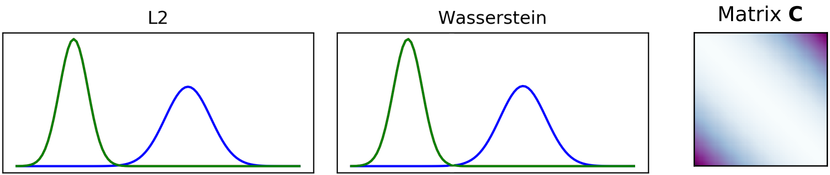
Wasserstein distance

$$W_p^p(\mu_s, \mu_t) = \min_{\mathbf{T} \geq \mathbf{0}} \sum_{i,j} T_{i,j} \|\mathbf{x}_i^s - \mathbf{x}_j^t\|^p \quad \text{s.t.} \quad \sum_j T_{i,j} = a_i, \forall i \quad \text{and} \quad \sum_i T_{i,j} = b_j, \forall j$$

In this case we have $c(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|^p$

- A.K.A. Earth Mover's Distance (W_1^1) [Rubner et al., 2000].
- Useful between discrete distribution even without overlapping support.
- Smooth approximation can be computed with Sinkhorn [Cuturi, 2013].
- **Wasserstein barycenter**: $\overline{\mu} = \arg\min_\mu \sum_i w_i W_2^2(\mu, \mu_i)$

Wasserstein distance



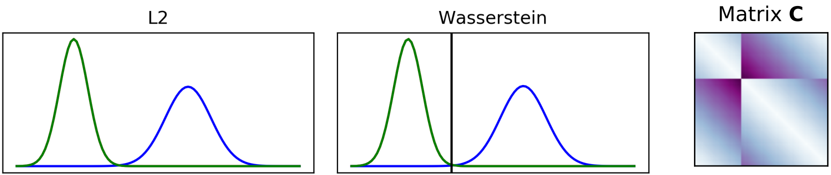
Wasserstein distance

$$W_p^p(\mu_s, \mu_t) = \min_{\boldsymbol{T} \geq \mathbf{0}} \sum_{i,j} T_{i,j} \|\mathbf{x}_i^s - \mathbf{x}_j^t\|^p \quad \text{s.t.} \quad \sum_j T_{i,j} = a_i, \forall i \quad \text{and} \quad \sum_i T_{i,j} = b_j, \forall j$$

In this case we have $c(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|^p$

- A.K.A. Earth Mover's Distance (W_1^1) [Rubner et al., 2000].
- Useful between discrete distribution even without overlapping support.
- Smooth approximation can be computed with Sinkhorn [Cuturi, 2013].
- **Wasserstein barycenter**: $\overline{\mu} = \arg\min_\mu \sum_i w_i W_2^2(\mu, \mu_i)$

Wasserstein distance



Wasserstein distance

$$W_p^p(\mu_s, \mu_t) = \min_{\boldsymbol{T} \geq \boldsymbol{0}} \sum_{i,j} T_{i,j} \|\mathbf{x}_i^s - \mathbf{x}_j^t\|^p \quad \text{s.t.} \quad \sum_j T_{i,j} = a_i, \forall i \quad \text{and} \quad \sum_i T_{i,j} = b_j, \forall j$$

In this case we have $c(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|^p$

- A.K.A. Earth Mover's Distance (W_1^1) [Rubner et al., 2000].
- Useful between discrete distribution even without overlapping support.
- Smooth approximation can be computed with Sinkhorn [Cuturi, 2013].
- **Wasserstein barycenter**: $\overline{\mu} = \arg\min_\mu \sum_i w_i W_2^2(\mu, \mu_i)$

Outline

OT for Machine Learning and Data Science

Distributions are everywhere in machine learning

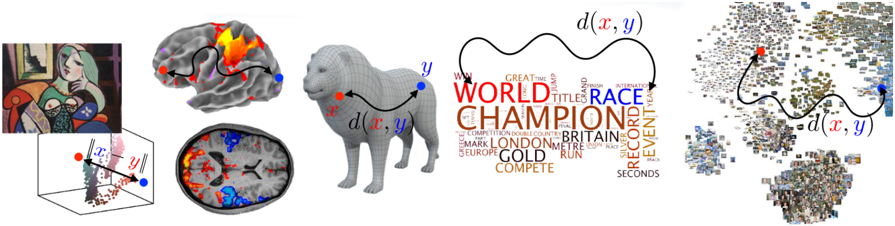
- Images, vision, graphics, Time series, text, genes, proteins.
- Many datum and datasets can be seen as distributions.
- Optimal transport provides tools for comparing them with a meaningful geometry.

How to use OT for ML?

- As a loss to compare distributions (Wasserstein distance).
- To learn a mapping between distributions (OT mapping).
- To learn on non-standard data (structured data, graphs).

Illustration from the slides of Gabriel Peyré.

OT for Machine Learning and Data Science



Distributions are everywhere in machine learning

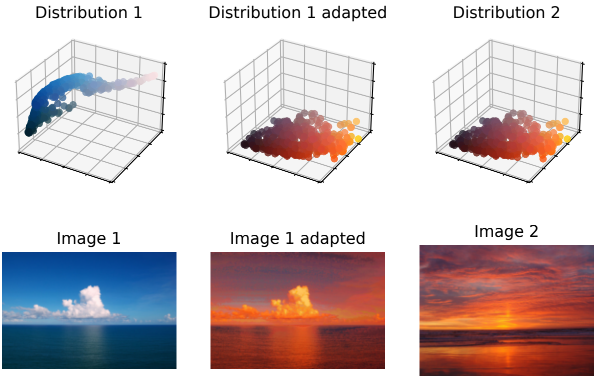
- Images, vision, graphics, Time series, text, genes, proteins.
- Many datum and datasets can be seen as distributions.
- Optimal transport provides tools for comparing them with a meaningful geometry.

How to use OT for ML?

- As a loss to compare distributions (Wasserstein distance).
- To learn a mapping between distributions (OT mapping).
- To learn on non-standard data (structured data, graphs).

Illustration from the slides of Gabriel Peyré.

OT for images processing and graphics



Applications of OT for images processing and graphics

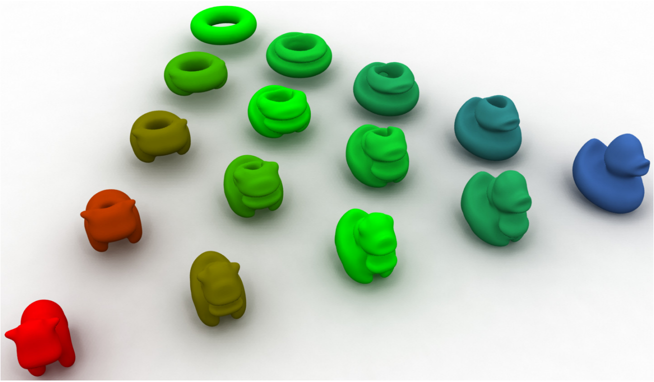
- Transporting pixels for color transfer [Ferradans et al., 2014].
- Transporting image patches for style transfer [Mroueh, 2019].
- Shape interpolation with OT barycenters [Solomon et al., 2015].

OT for images processing and graphics

Applications of OT for images processing and graphics

- Transporting pixels for color transfer [Ferradans et al., 2014].
- Transporting image patches for style transfer [Mroueh, 2019].
- Shape interpolation with OT barycenters [Solomon et al., 2015].

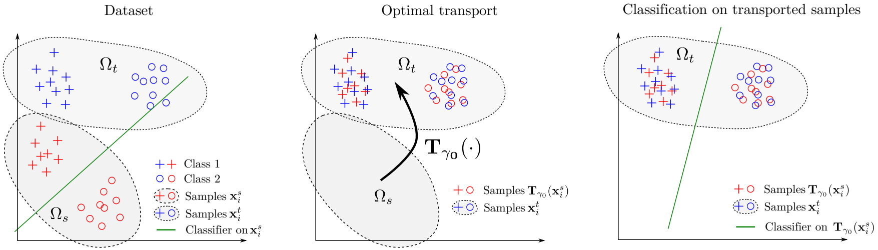
OT for images processing and graphics



Applications of OT for images processing and graphics

- Transporting pixels for color transfer [Ferradans et al., 2014].
- Transporting image patches for style transfer [Mroueh, 2019].
- Shape interpolation with OT barycenters [Solomon et al., 2015].

Optimal Transport for Domain Adaptation



Transport the data [Courty et al., 2016]

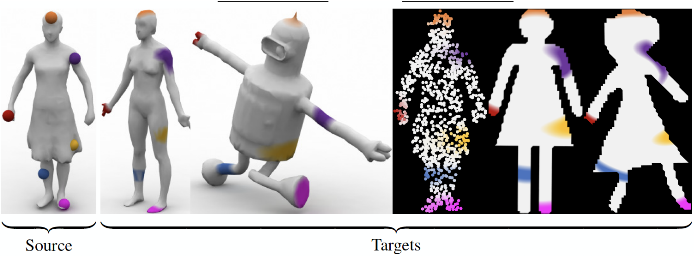
1. Estimate optimal transport between distributions.
2. Transport the training samples on target domain.
3. Learn a classifier on the transported training samples.

Can also be used to compensate for biased datasets [Gordaliza et al., 2019]

Transport the labels

- Label Propagation using OT matrix [Solomon et al., 2014, Redko et al., 2019].
- Optimize the target classifier [Courty et al., 2017, Damodaran et al., 2018].
- Change in proportion of classes [Redko et al., 2019, Rakotomamonjy et al., 2020].

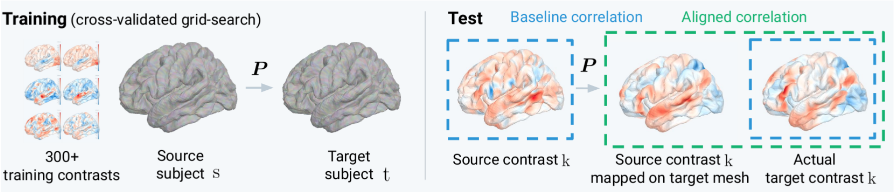
OT between graphs



Gromov-Wasserstein OT distance [Memoli, 2011]

- OT plan is alignements between nodes of two graphs.
- Minimize changes in pairwise relationships between nodes (preserve structure).
- OT between surfaces, shapes, graphs [Solomon et al., 2016, Vayer et al., 2018].
- Applications on:
  - Brain MRI alignment [Thual et al., 2022]
  - Single cell data [Demetci et al., 2022, Tran et al., 2023]
- Barycenter (denoising or compression) of graphs [Vayer et al., 2018].

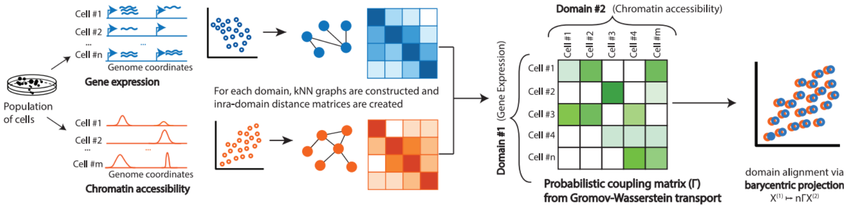
OT between graphs



Gromov-Wasserstein OT distance [Memoli, 2011]

- OT plan is alignements between nodes of two graphs.
- Minimize changes in pairwise relationships between nodes (preserve structure).
- OT between surfaces, shapes, graphs [Solomon et al., 2016, Vayer et al., 2018].
- Applications on:
  - Brain MRI alignment [Thual et al., 2022]
  - Single cell data [Demetci et al., 2022, Tran et al., 2023]
- Barycenter (denoising or compression) of graphs [Vayer et al., 2018].

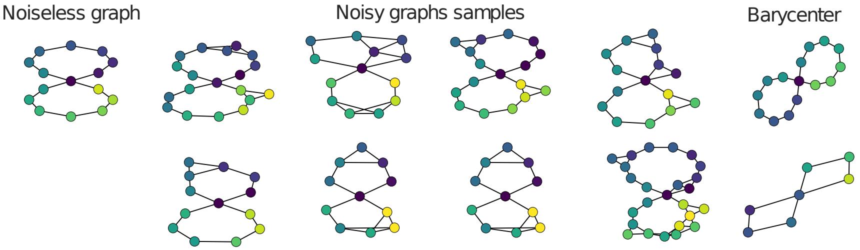
OT between graphs



Gromov-Wasserstein OT distance [Memoli, 2011]

- OT plan is alignements between nodes of two graphs.
- Minimize changes in pairwise relationships between nodes (preserve structure).
- OT between surfaces, shapes, graphs [Solomon et al., 2016, Vayer et al., 2018].
- Applications on:
  - Brain MRI alignment [Thual et al., 2022]
  - Single cell data [Demetci et al., 2022, Tran et al., 2023]
- Barycenter (denoising or compression) of graphs [Vayer et al., 2018].

OT between graphs



Gromov-Wasserstein OT distance [Memoli, 2011]

- OT plan is alignements between nodes of two graphs.
- Minimize changes in pairwise relationships between nodes (preserve structure).
- OT between surfaces, shapes, graphs [Solomon et al., 2016, Vayer et al., 2018].
- Applications on:
    - Brain MRI alignment [Thual et al., 2022]
    - Single cell data [Demetci et al., 2022, Tran et al., 2023]
- Barycenter (denoising or compression) of graphs [Vayer et al., 2018].

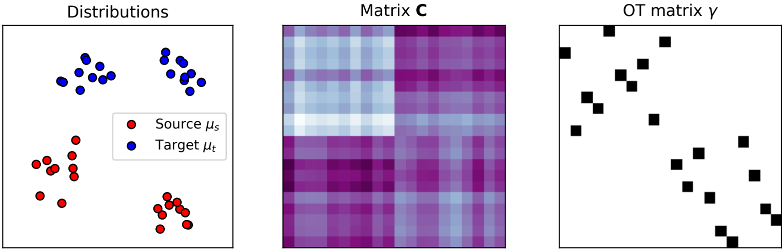
Outline

Python Optimal Transport (POT)

The toolbox

- Website/documentation: `https://pythonot.github.io/`
- Github: `https://github.com/PythonOT/POT`
- Activity: 79 contributors, 2.6k stars, 5.5 M PyPI downloads, 1300 citations.
- Features: OT solvers from 81 papers, 58 examples in gallery.
- CI-CD: 97% test coverage, 100% PEP8 compliant with pre-commit.
- Maintained since 2017: 2 releases/year, 1.5k commits.
- Packages in PyPI (POT), Conda forge, Debian, Ubuntu.

How to solve OT in Python?

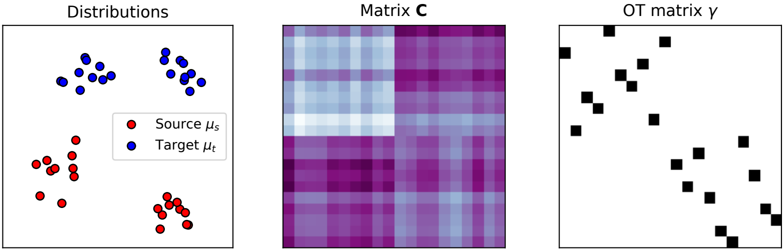


Solving discrete OT with POT

```python
import ot # import POT

# Xs and Xt are positions of source and target samples
C = ot.dist(Xs, Xt, metric = 'euclidean')  # ground cost matrix
result = ot.solve(C, a=a, b=b) # returns an OTResult object

T = result.plan  # get the OT plan
cost = result.value  # get the OT cost
```

Gallery : `https://pythonot.github.io/auto_examples/plot_OT_2D_samples.html`

How to solve OT in Python?

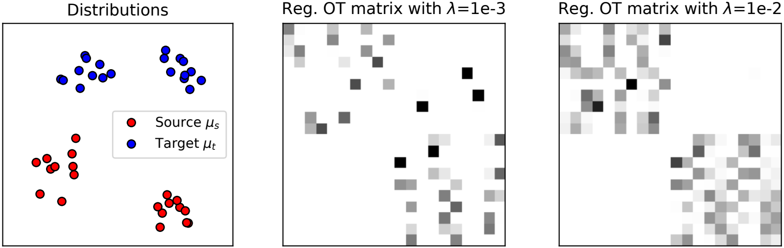


Solving empirical discrete OT with POT

```python
import ot # import POT

# Xs and Xt are positions of source and target samples
# default values uniform weights for a,b
result = ot.solve_sample(Xs, Xt, metric='euclidean')

T = result.plan  # get the OT plan
cost = result.value  # get the OT cost
```

Gallery : https://pythonot.github.io/auto_examples/plot_OT_2D_samples.html

How to solve OT in Python?

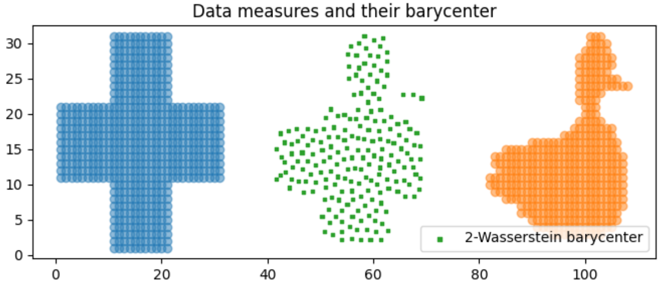


Solving regularized discrete OT with POT (sinkhorn)

```python
import ot # import POT

# Xs and Xt are positions of source and target samples
# reg is entropic regularization strength
result = ot.solve_sample(Xs, Xt, reg=0.1, metric='euclidean')

T = result.plan  # get the OT plan
cost = result.value  # get the OT cost
```

Gallery : https://pythonot.github.io/auto_examples/plot_OT_2D_samples.html

OT barycenter between empirical distributions



Wasserstein (free support) barycenter with POT

```
from ot.lp import free_support_barycenter

X_list = [X1, X2] # list of locations of the measures
a_list = [a1, a2] # list of weights of the measures
w = [0.5, 0.5] # barycenter weights
X_init = np.random.randn(k, d) # initial barycenter locations
Xbary = free_support_barycenter(X_list, a_list, X_init,
    weights=w)
```

Gallery : Free support barycenter example

Advanced feature : POT backends

POT Backends

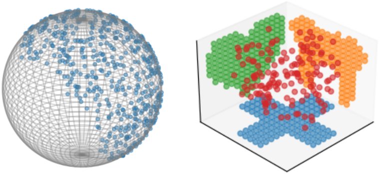
- Automatic detection of type of inputs (Numpy, Pytorch, Tensorflow, Jax, Cupy).
- Coded in functions with the backend : `nx = get_backend(C,a,b, ...)` .
- Differentiation through the OT solvers (automatic or manual definition).
- Works with CPU and GPU tensors (similar to `array-api`)

Example in Pytorch

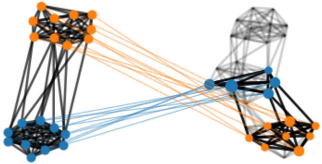
```python
import ot
import torch

# differentiable loss (or OT plan)
Xs = torch.randn((100,2), requires_grad=True).cuda()
Xt = torch.randn((80,2)).cuda()
loss = ot.solve_sample(Xs, Xt, reg=0.1).value # runs on GPU
loss.backward() # gradients on Xs

# batched with C_batch a (batch, n, n) tensor of cost matrices
loss_batch = ot.solve_batch(C_batch, reg=0.1).value
loss_batch.mean().backward() # grads backprop. through C_batch
```

Advanced features : other solvers in POT

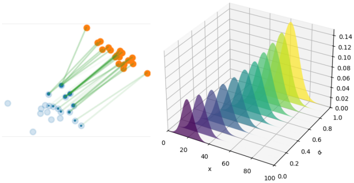
Sliced OT (line, sphere, subspace)



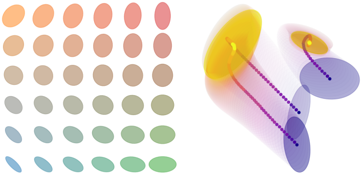
(Fused) Gromov-Wasserstein OT



Unbalanced and partial OT



OT on Gaussian and Gaussian mixtures



Example Gallery: `https://pythonot.github.io/auto_examples/index.html`

Outline

Acknowledgements

POT contributors

Fundings

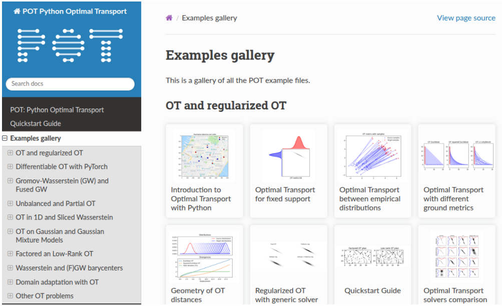
ANR

3iA Côte d'Azur
Institut interdisciplinaire d'intelligence artificielle

cnrs

Hi! PARIS
HEC PARIS
INSTITUT POLYTECHNIQUE DE PARIS
PARIS ARTIFICIAL INTELLIGENCE FOR SOCIETY

Conclusions and Q&A



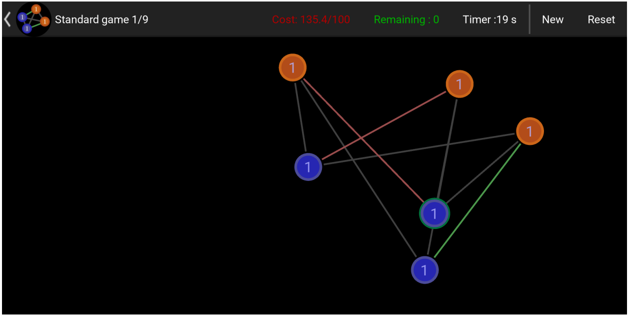
Link to slides

Optimal Transport in Python

- POT is a well established library for optimal transport in Python.
- Both basic and more advanced OT solvers from the literature implemented.
- Backends for Numpy/Scipy, Pytorch, Cupy, Tensorflow and Jax.
- Many other examples in the gallery: `https://pythonot.github.io/`
- Other open source libraries: GeomLoss (GPU, wrapper in POT), OTT-JAX.

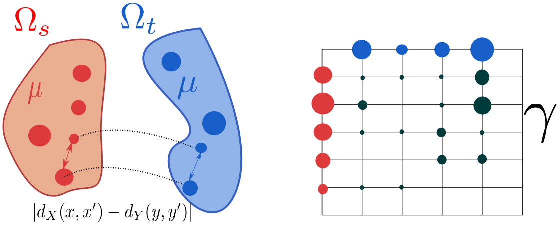
OTGame (OT Puzzle game on android)

OTGame



https://play.google.com/store/apps/details?id=com.flamary.otgame

Gromov-Wasserstein and extensions



Inspired from Gabriel Peyré

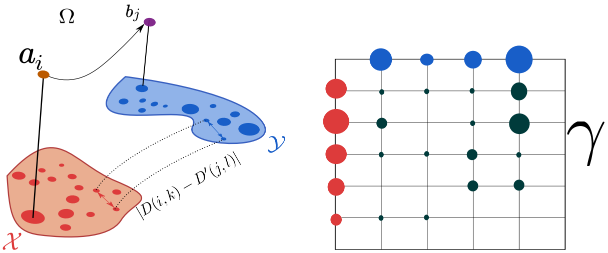
GW for discrete distributions [Memoli, 2011]

$$\mathcal{GW}_p^p(\mu_s, \mu_t) = \min_{T \in \Pi(\mu_s, \mu_t)} \sum_{i,j,k,l} |D_{i,k} - D'_{j,l}|^p T_{i,j} T_{k,l}$$

with $\mu_s = \sum_i a_i \delta_{\mathbf{x}_i^s}$ and $\mu_t = \sum_j b_j \delta_{x_j^t}$ and $D_{i,k} = \|\mathbf{x}_i^s - \mathbf{x}_k^s\|$, $D'_{j,l} = \|\mathbf{x}_j^t - \mathbf{x}_l^t\|$

- Distance between metric measured spaces : across different spaces.
- Search for an OT plan that preserve the pairwise relationships between samples.
- Entropy regularized GW proposed in [Peyré et al., 2016].
- Fused GW interpolates between Wass. and GW [Vayer et al., 2018].

Gromov-Wasserstein and extensions



FGW for discrete distributions **[Vayer et al., 2018]**

$$\mathcal{FGW}_p^p(\mu_s, \mu_t) = \min_{T \in \Pi(\mu_s, \mu_t)} \sum_{i,j,k,l} \left((1-\alpha)C_{i,j}^q + \alpha|D_{i,k} - D'_{j,l}|^q\right)^p T_{i,j}\, T_{k,l}$$

with $\mu_s = \sum_i a_i \delta_{\mathbf{x}_i^s}$ and $\mu_t = \sum_j b_j \delta_{x_j^t}$ and $D_{i,k} = \|\mathbf{x}_i^s - \mathbf{x}_k^s\|$, $D'_{j,l} = \|\mathbf{x}_j^t - \mathbf{x}_l^t\|$

- Distance between metric measured spaces : across different spaces.
- Search for an OT plan that preserve the pairwise relationships between samples.
- Entropy regularized GW proposed in [Peyré et al., 2016].
- Fused GW interpolates between Wass. and GW [Vayer et al., 2018].

References I

[Courty et al., 2016] Courty, N., Flamary, R., Tuia, D., and Rakotomamonjy, A. (2016).
Optimal transport for domain adaptation.
Pattern Analysis and Machine Intelligence, IEEE Transactions on.

[Courty et al., 2017] Courty, N., Flamary, R., Tuia, D., and Rakotomamonjy, A. (2017).
Optimal transport for domain adaptation.
IEEETPAMI, 39(9):1853–1865.

[Cuturi, 2013] Cuturi, M. (2013).
Sinkhorn distances: Lightspeed computation of optimal transport.
In *NIPS*, pages 2292–2300.

[Damodaran et al., 2018] Damodaran, B. B., Kellenberger, B., Flamary, R., Tuia, D., and Courty, N. (2018).
Deepjdot: Deep joint distribution optimal transport for unsupervised domain adaptation.

[Demetci et al., 2022] Demetci, P., Santorella, R., Chakravarthy, M., Sandstede, B., and Singh, R. (2022).
Scotv2: Single-cell multiomic alignment with disproportionate cell-type representation.
Journal of Computational Biology, 29(11):1213–1228.

References II

[Ferradans et al., 2014] Ferradans, S., Papadakis, N., Peyré, G., and Aujol, J.-F. (2014).
Regularized discrete optimal transport.
SIAM Journal on Imaging Sciences, 7(3):1853–1882.

[Gordaliza et al., 2019] Gordaliza, P., Del Barrio, E., Fabrice, G., and Loubes, J.-M. (2019).
Obtaining fairness using optimal transport theory.
In *International Conference on Machine Learning*, pages 2357–2365. PMLR.

[Kantorovich, 1942] Kantorovich, L. (1942).
On the translocation of masses.
C.R. (Doklady) Acad. Sci. URSS (N.S.), 37:199–201.

[Memoli, 2011] Memoli, F. (2011).
Gromov wasserstein distances and the metric approach to object matching.
Foundations of Computational Mathematics, pages 1–71.

[Monge, 1781] Monge, G. (1781).
Mémoire sur la théorie des déblais et des remblais.
De l'Imprimerie Royale.

References III

[Mroueh, 2019] Mroueh, Y. (2019).
Wasserstein style transfer.
arXiv preprint arXiv:1905.12828.

[Peyré et al., 2016] Peyré, G., Cuturi, M., and Solomon, J. (2016).
Gromov-wasserstein averaging of kernel and distance matrices.
In *ICML*, pages 2664–2672.

[Rakotomamonjy et al., 2020] Rakotomamonjy, A., Flamary, R., Gasso, G., Alaya, M., Berar, M., and Courty, N. (2020).
Match and reweight strategy for generalized target shift.

[Redko et al., 2019] Redko, I., Courty, N., Flamary, R., and Tuia, D. (2019).
Optimal transport for multi-source domain adaptation under target shift.
In *International Conference on Artificial Intelligence and Statistics (AISTAT)*.

[Rubner et al., 2000] Rubner, Y., Tomasi, C., and Guibas, L. J. (2000).
The earth mover's distance as a metric for image retrieval.
International journal of computer vision, 40(2):99–121.

References IV

[Solomon et al., 2015] Solomon, J., De Goes, F., Peyré, G., Cuturi, M., Butscher, A., Nguyen, A., Du, T., and Guibas, L. (2015).
Convolutional wasserstein distances: Efficient optimal transportation on geometric domains.
ACM Transactions on Graphics (TOG), 34(4):66.

[Solomon et al., 2016] Solomon, J., Peyré, G., Kim, V. G., and Sra, S. (2016).
Entropic metric alignment for correspondence problems.
ACM Transactions on Graphics (TOG), 35(4):72.

[Solomon et al., 2014] Solomon, J., Rustamov, R., Guibas, L., and Butscher, A. (2014).
Wasserstein propagation for semi-supervised learning.
In *International Conference on Machine Learning*, pages 306–314. PMLR.

[Thual et al., 2022] Thual, A., Tran, H., Zemskova, T., Courty, N., Flamary, R., Dehaene, S., and Thirion, B. (2022).
Aligning individual brains with fused unbalanced gromov-wasserstein.
In *Neural Information Processing Systems (NeurIPS)*.

References V

[Tran et al., 2023] Tran, Q. H., Janati, H., Courty, N., Flamary, R., Redko, I., Demetci, P., and Singh, R. (2023).
Unbalanced co-optimal transport.
In *Thirty-Seventh AAAI Conference on Artificial Intelligence (AAAI)*.

[Vayer et al., 2018] Vayer, T., Chapel, L., Flamary, R., Tavenard, R., and Courty, N. (2018).
Fused gromov-wasserstein distance for structured objects: theoretical foundations and mathematical properties.