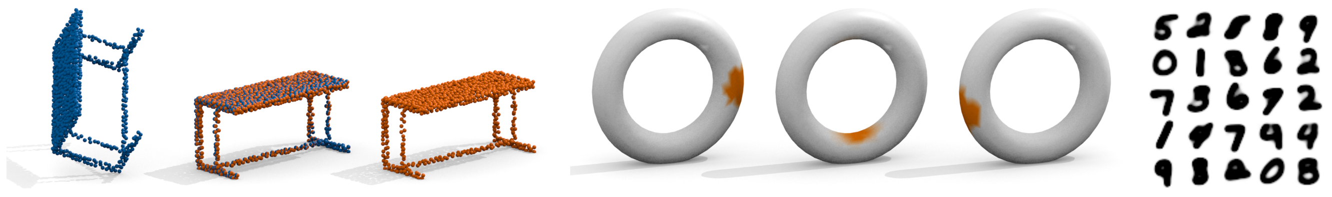
A Unified Framework for Implicit Sinkhorn Differentiation

Marvin Eisenberger*, Aysim Toker*, Laura Leal-Taixé*, Florian Bernard†, Daniel Cremers*
Technical University of Munich*, University of Bonn†



(a) Registration of two pointclouds (b) Barycentric interpolation of distributions on a torus (c) MNIST clustering

Figure 1. The Sinkhorn operator is becoming a fundamental building block for various computer vision algorithms. Relevant applications include (a) point cloud registration, (b) interpolation on manifolds, (c) image clustering, and many more [13, 25, 39, 46, 47]. A recent trend to training respective neural networks efficiently is implicit differentiation [7, 11, 17, 22, 26]. In this work, we provide a framework of implicit Sinkhorn differentiation that generalizes existing methods. It is the first to derive analytical gradients for the Sinkhorn operator in its most general form, covering all the applications (a)-(c) shown above.

Abstract

The Sinkhorn operator has recently experienced a surge of popularity in computer vision and related fields. One major reason is its ease of integration into deep learning frameworks. To allow for an efficient training of respective neural networks, we propose an algorithm that obtains analytical gradients of a Sinkhorn layer via implicit differentiation. In comparison to prior work, our framework is based on the most general formulation of the Sinkhorn operator. It allows for any type of loss function, while both the target capacities and cost matrices are differentiated jointly. We further construct error bounds of the resulting algorithm for approximate inputs. Finally, we demonstrate that for a number of applications, simply replacing automatic differentiation with our algorithm directly improves the stability and accuracy of the obtained gradients. Moreover, we show that it is computationally more efficient, particularly when resources like GPU memory are scarce.[1]

1. Introduction

Computing matchings and permutations is a fundamental problem at the heart of many computer vision and machine learning algorithms. Common applications include pose estimation, 3D reconstruction, localization, information transfer, ranking, and sorting, with data domains ranging from images, voxel grids, point clouds, 3D surface meshes to generic Euclidean features. A popular tool to address this is the Sinkhorn operator, which has its roots in the theory of entropy regularized optimal transport [9]. The Sinkhorn operator can be computed efficiently via a simple iterative matrix scaling approach. Furthermore, the resulting operator is differentiable, and can therefore be readily integrated into deep learning frameworks.

A key question is how to compute the first-order derivative of a respective Sinkhorn layer in practice. The standard approach is automatic differentiation of Sinkhorn's algorithm. Yet, this comes with a considerable computational burden because the runtime of the resulting backward pass scales linearly with the number of forward iterations. More importantly, since the computation graph needs to be maintained for all unrolled matrix-scaling steps, the memory demand is often prohibitively high for GPU processing.

A number of recent works leverage implicit gradients as an alternative to automatic differentiation [7, 11, 17, 22, 26] to backpropagate through a Sinkhorn layer. Although such approaches prove to be computationally inexpensive, a downside is that corresponding algorithms are less straightforward to derive and implement. Hence, many application works still rely on automatic differentiation [13, 25, 39, 46, 47]. Yet, the computational burden of automatic differentiation might drive practitioners to opt for an insufficiently small number of Sinkhorn iterations which in turn impairs the performance as we experimentally verify in Sec. 5.

To date, existing work on implicit differentiation of Sinkhorn layers suffers from two major limitations: (i) Most approaches derive gradients only for very specific settings, *i.e.* specific loss functions, structured inputs, or only a subset of all inputs. Algorithms are therefore often not transferable to similar but distinct settings. (ii) Secondly, beyond their empirical success, there is a lack of an in-depth theoretical analysis that supports the use of implicit gradients.

Our work provides a unified framework of implicit dif-

[1]Our implementation is available under the following link: https://github.com/marvin-eisenberger/implicit-sinkhorn

ferentiation techniques for Sinkhorn layers. To encourage practical adaptation, we provide a simple module that works out-of-the-box for the most general formulation, see Fig. 2. We can thus recover existing methods as special cases of our framework, see Tab. 1 for an overview. Our contribution can be summarized as follows:

1. From first principles we derive an efficient algorithm for computing gradients of a generic Sinkhorn layer.
2. We provide theoretical guarantees for the accuracy of the resulting gradients as a function of the approximation error in the forward pass (Theorem 5).
3. Our PyTorch module can be applied in an out-of-the-box manner to existing approaches based on automatic differentiation. This often improves the quantitative results while using significantly less GPU memory.

2. Related work

There is a vast literature on computational optimal transport (OT) [33,43]. In the following, we provide an overview of related machine learning applications. Our approach is based on entropy regularized optimal transport pioneered by [9]. The resulting differentiable Sinkhorn divergence can be used as a loss function for training machine learning models [8, 16, 18]. To allow for first-order optimization, two common approaches for computing gradients are implicit differentiation [11, 22, 26] and automatic differentiation [1, 19]. Relevant applications of the Sinkhorn divergence include computing Wasserstein barycenters [10, 27, 41], dictionary learning [40], as well as using a geometrically meaningful loss function for autoencoders [31] or generative adversarial networks (GAN) [19, 37].

More recently, several approaches emerged that use the Sinkhorn operator as a differentiable transportation layer in a neural network. Potential applications include permutation learning [28, 38], ranking [2, 12], sorting via reinforcement learning [14], discriminant analysis [17] and computing matchings between images [39], point clouds [25,46,47] or triangle meshes [13, 29]. Most of these approaches rely on automatic differentiation of the Sinkhorn algorithm to address the resulting bilevel optimization problem. In our work, we follow the recent trend of using implicit differentiation for the inner optimization layer [3, 5, 20]. Other approaches compute the input cost matrix via Bayesian inverse modeling [42] or smooth the OT linear assignment problem (LAP) directly [34].

There are a number of methods that compute analytical gradients of a Sinkhorn layer, see Tab. 1 for an overview. The idea of our work is to provide a unifying framework that generalizes specific methods, as well as providing additional theoretical insights. The pioneering work of Luise *et al.* [26] computes gradients for the Sinkhorn divergence loss, while optimizing for the marginals. [1] and [22] provide further theoretical analysis. Flamary *et al.* [17] compute explicit gradients for the application of discriminant analysis. However, they directly solve the linear system specified by the implicit function theorem which leads to an algorithmic complexity of $\mathcal{O}(n^6)$. Similar to ours, [7] and [45] compute gradients of the cost matrix $\boldsymbol{C}$, but they assume constant marginals. The recent approach by Cuturi *et al.* [11] derives implicit gradients from the dual objective for the special case of low rank cost matrices $\boldsymbol{C}(\boldsymbol{x}, \boldsymbol{y})$.

Method	$\nabla_{[\boldsymbol{a};\boldsymbol{b}]}\ell$	$\nabla_{\boldsymbol{x}}\ell$	$\nabla_{\boldsymbol{C}}\ell$	Loss	Obj.
Luise *et al.* [26]	✓	✗	✗	Wasserstein	dual
Klatt *et al.* [22]	✓	✗	✗	Wasserstein	primal
Ablin *et al.* [1]	✓	✗	✗	Wasserstein	dual
Flamary *et al.* [17]	✗	✓	✗	Discr. analysis	primal
Campbell *et al.* [7]	✗	✓	✓	any	primal
Xie *et al.* [45]	✗	✓	✓	any	dual
Cuturi *et al.* [11]	✓	✓	✗	any	dual
Ours	✓	✓	✓	any	primal

Table 1. **Overview of prior work.** We provide an overview of related approaches that, like ours, derive implicit gradients of a Sinkhorn layer. For each method, we denote admissible inputs, *i.e.* which inputs are differentiated. In the most general case, we want to optimize both the marginals $\boldsymbol{a}$ and $\boldsymbol{b}$ and the cost matrix $\boldsymbol{C}$ defined in Sec. 3. As a special case, [11, 17] provide gradients $\nabla_{\boldsymbol{x}}\ell$ for low rank cost matrices of the form $\boldsymbol{C}_{i,j} := \|\boldsymbol{x}_i - \boldsymbol{y}_j\|_2^p$. We furthermore denote which types of loss functions are permitted and whether gradients are derived via the primal or dual objective.

3. Background

Optimal transport. Optimal transport enables us to compute the distance between two probability measures on the same domain $\Omega \subset \mathbb{R}^d$. In this work, we consider discrete probability measures $\mu := \sum_{i=1}^m a_i \delta_{\boldsymbol{x}_i}$ and $\nu := \sum_{j=1}^n b_j \delta_{\boldsymbol{y}_j}$, defined over the sets of points $\{\boldsymbol{x}_1, \dots, \boldsymbol{x}_m\}$ and $\{\boldsymbol{y}_1, \dots, \boldsymbol{y}_n\}$, where $\delta_{\boldsymbol{x}_i}$ is the Dirac measure at $\boldsymbol{x}_i$. Such measures are fully characterized by the probability mass vectors $\boldsymbol{a} \in \Delta_m$ and $\boldsymbol{b} \in \Delta_n$ that lie on the probability simplex

$$\Delta_m = \left\{\boldsymbol{a} \in \mathbb{R}^m | a_i \geq 0, \boldsymbol{a}^\top \mathbb{1}_m = 1\right\}, \tag{1}$$

where $\mathbb{1}_m \in \mathbb{R}^m$ is the vector of all ones. We can then define the distance between μ and ν as

$$d(\mu, \nu) := \min_{\boldsymbol{P} \in \Pi(\boldsymbol{a},\boldsymbol{b})} \langle \boldsymbol{P}, \boldsymbol{C} \rangle_F. \tag{2}$$

The transportation plan $\boldsymbol{P} \in \Pi(\boldsymbol{a}, \boldsymbol{b})$ determines a discrete probability measure on the product space $\{\boldsymbol{x}_1, \dots, \boldsymbol{x}_m\} \times \{\boldsymbol{y}_1, \dots, \boldsymbol{y}_n\}$, whose marginal distributions coincide with μ and ν. Consequently, $\boldsymbol{P}$ is contained in the transportation polytope $\Pi(\boldsymbol{a}, \boldsymbol{b})$ defined as

$$\Pi(\boldsymbol{a}, \boldsymbol{b}) := \{\boldsymbol{P} \in \mathbb{R}_+^{m \times n} | \boldsymbol{P}\mathbb{1}_n = \boldsymbol{a}, \boldsymbol{P}^\top \mathbb{1}_m = \boldsymbol{b}\}. \tag{3}$$

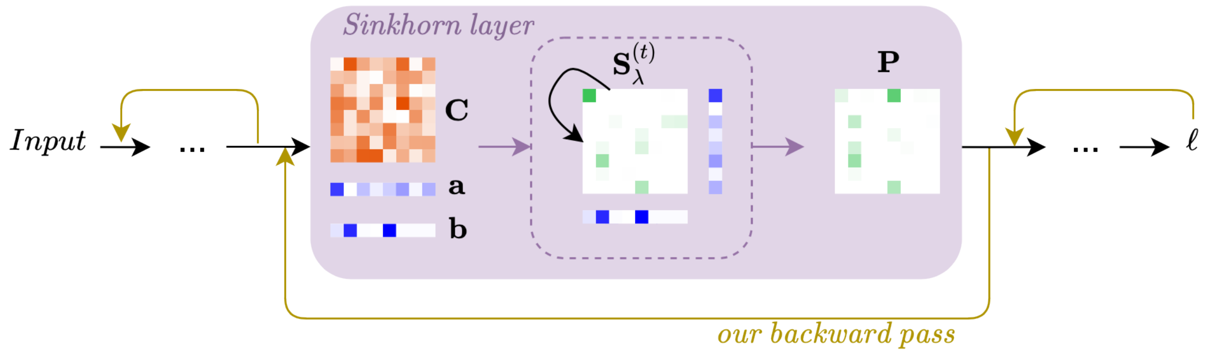


Figure 2. **Overview of a typical workflow with an embedded Sinkhorn layer.** We consider a neural network whose inputs are e.g. images, 3D point clouds, voxel grids, surface meshes, etc. The Sinkhorn layer maps the cost matrix $\boldsymbol{C}$ and marginals $\boldsymbol{a}, \boldsymbol{b}$ to the transportation plan $\boldsymbol{P}$ via iterative matrix scaling. During training, we compute respective gradients $(\nabla_{\boldsymbol{C}}\ell, \nabla_{\boldsymbol{a}}\ell, \nabla_{\boldsymbol{b}}\ell)$ in closed form via implicit differentiation. Our algorithm applies to the most general formulation of the Sinkhorn operator: Both the cost matrix $\boldsymbol{C}$ and marginals $\boldsymbol{a}, \boldsymbol{b}$ are learnable and the whole network potentially contains learnable weights before and after the Sinkhorn layer.

The cost matrix $\boldsymbol{C} \in \mathbb{R}^{m\times n}$ specifies the transportation cost from individual points $\boldsymbol{x}_i$ to $\boldsymbol{y}_j$. Choosing

$$\boldsymbol{C}_{i,j} := \|\boldsymbol{x}_i - \boldsymbol{y}_j\|_2^p$$

for $p \geq 1$, e.g. yields the so-called Wasserstein distance $d(\cdot,\cdot) = W_p^p(\cdot,\cdot)$, see [43].

Entropy regularization. Evaluating the distance $d(\mu,\nu)$ in practice requires solving the linear assignment problem (LAP) from Eq. (2). This can be done via specialized algorithms like the Hungarian algorithm [23] or the Auction algorithm [4], as well as recent solvers [32, 36]. However, most approaches are computationally heavy and slow in practice [9]. A popular alternative is augmenting the LAP objective in Eq. (2) with an additional entropy regularizer, giving rise to the *Sinkhorn operator*

$$S_\lambda(\boldsymbol{C},\boldsymbol{a},\boldsymbol{b}) := \underset{\boldsymbol{P}\in\Pi(\boldsymbol{a},\boldsymbol{b})}{\arg\min}\, \langle \boldsymbol{P}, \boldsymbol{C}\rangle_F - \lambda h(\boldsymbol{P}), \tag{4}$$

where $\lambda > 0$ weights the regularization. The seminal work of Cuturi *et al.* [9] shows that the additional entropy regularization term $h(\boldsymbol{P}) = -\sum_{i,j} P_{i,j}(\log P_{i,j} - 1)$ allows for an efficient minimization of Eq. (4). Specifically, this can be done via a scheme of alternating Sinkhorn projections

$$\boldsymbol{S}_\lambda^{(0)} := \exp\left(-\frac{1}{\lambda}\boldsymbol{C}\right), \quad \text{and}$$
$$\boldsymbol{S}_\lambda^{(t+1)} := \mathcal{T}_c\big(\mathcal{T}_r\big(\boldsymbol{S}_\lambda^{(t)}\big)\big). \tag{5}$$

The operators $\mathcal{T}_c(\boldsymbol{S}) := \boldsymbol{S} \oslash (\mathbb{1}_m\mathbb{1}_m^\top \boldsymbol{S}) \odot (\mathbb{1}_m\boldsymbol{b}^\top)$ and $\mathcal{T}_r(\boldsymbol{S}) := \boldsymbol{S} \oslash (\boldsymbol{S}\mathbb{1}_n\mathbb{1}_n^\top) \odot (\boldsymbol{a}\mathbb{1}_n^\top)$ correspond to renormalizations of the columns and rows of $\boldsymbol{S}_\lambda^{(t)}$, where $\odot$ denotes the Hadamard product and $\oslash$ denotes element-wise division. As shown by [9], in the limit this scheme converges to a minimizer $\boldsymbol{S}_\lambda^{(t)} \xrightarrow{t\to\infty} \boldsymbol{S}_\lambda$ of Eq. (4). In practice, we can use a finite number of iterations $\tau \in \mathbb{N}$ to achieve a sufficiently small residual.

4. Method

4.1. Problem formulation

Integrating the Sinkhorn operator from Eq. (4) into deep neural networks has become a popular tool for a wide range of practical tasks, see our discussion in Sec. 2. A major contributing factor is that the entropy regularization makes the mapping $S_\lambda : \mathbb{R}^{m\times n} \times \mathbb{R}^m \times \mathbb{R}^n \to \mathbb{R}^{m\times n}$ differentiable. To allow for first-order-optimization, we need to compute

$$(\boldsymbol{C},\boldsymbol{a},\boldsymbol{b}) \quad\mapsto\quad \boldsymbol{P}^* := S_\lambda(\boldsymbol{C},\boldsymbol{a},\boldsymbol{b}) \quad \text{and} \tag{6}$$
$$\nabla_{\boldsymbol{P}}\ell \quad\mapsto\quad (\nabla_{\boldsymbol{C}}\ell, \nabla_{\boldsymbol{a}}\ell, \nabla_{\boldsymbol{b}}\ell), \tag{7}$$

which denote the forward pass and the backpropagation of gradients, respectively. Those expressions arise in the context of a typical workflow within a deep neural network with a scalar loss ℓ and learnable parameters before and/or after the Sinkhorn operator S_λ, see Fig. 2 for an overview.

A common strategy is to replace the exact forward pass $S_\lambda(\boldsymbol{C},\boldsymbol{a},\boldsymbol{b})$ in Eq. (6) by the approximate solution $\boldsymbol{S}_\lambda^{(\tau)}$ from Eq. (5). Like the original solution in Eq. (4), $\boldsymbol{S}_\lambda^{(\tau)}$ is differentiable w.r.t. $(\boldsymbol{C},\boldsymbol{a},\boldsymbol{b})$. Moreover, the mapping $(\boldsymbol{C},\boldsymbol{a},\boldsymbol{b}) \mapsto \boldsymbol{S}_\lambda^{(\tau)}$ consists of a small number of matrix scaling operations that can be implemented in a few lines of code, see Eq. (5).

4.2. Backward pass via implicit differentiation

The goal of this section is to derive the main result stated in Theorem 3, which is the key motivation of our algorithm in Sec. 4.3. To this end, we start by reframing the optimization problem in Eq. (4) in terms of its Karush–Kuhn–Tucker (KKT) conditions, see Appendix C.1 for a proof:

Lemma 1. *The transportation plan $\boldsymbol{P}^*$ is a global minimum of Eq. (4) iff $\mathcal{K}(\boldsymbol{c}, \boldsymbol{a}, \boldsymbol{b}, \boldsymbol{p}^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*) = \mathbf{0}_l$, with*

$$\mathcal{K}(\cdot) := \begin{bmatrix} \boldsymbol{c} + \lambda \log(\boldsymbol{p}^*) + \mathbb{1}_n \otimes \boldsymbol{\alpha}^* + \boldsymbol{\beta}^* \otimes \mathbb{1}_m \\ (\mathbb{1}_n^\top \otimes \boldsymbol{I}_m)\boldsymbol{p}^* - \boldsymbol{a} \\ (\boldsymbol{I}_n \otimes \mathbb{1}_m^\top)\boldsymbol{p}^* - \boldsymbol{b} \end{bmatrix} \tag{8}$$

where $l := mn + m + n$. Here, $\boldsymbol{\alpha}^* \in \mathbb{R}^m$ and $\boldsymbol{\beta}^* \in \mathbb{R}^n$ are the dual variables corresponding to the two equality contraints in Eq. (3). We further define $\boldsymbol{c}, \boldsymbol{p}^* \in \mathbb{R}^{mn}$ as the vectorized versions of $\boldsymbol{C}, \boldsymbol{P}^* \in \mathbb{R}^{m \times n}$, respectively, and assume $\log(p) := -\infty, p \leq 0$.

Establishing this identity is an important first step towards computing a closed-form gradient for the backward pass in Eq. (7). It reframes the optimization problem in Eq. (4) as a root-finding problem $\mathcal{K}(\cdot) = \mathbf{0}$. In the next step, this then allows us to explicitly construct the derivative of the Sinkhorn operator $S_\lambda(\cdot)$ via implicit differentiation, see Appendix C.2 for a proof:

Lemma 2. *The KKT conditions in Eq. (8) implicitly define a continuously differentiable function $(\boldsymbol{c}, \boldsymbol{a}, \tilde{\boldsymbol{b}}) \mapsto (\boldsymbol{p}, \boldsymbol{\alpha}, \tilde{\boldsymbol{\beta}})$ with the Jacobian $\boldsymbol{J} \in \mathbb{R}^{(l-1)\times(l-1)}$ being*

$$\boldsymbol{J} := \frac{\partial \left[\boldsymbol{p}; \boldsymbol{\alpha}; \tilde{\boldsymbol{\beta}}\right]}{\partial \left[\boldsymbol{c}; -\boldsymbol{a}; -\tilde{\boldsymbol{b}}\right]} = -\underbrace{\begin{bmatrix} \lambda \operatorname{diag}(\boldsymbol{p})^{-1} & \tilde{\boldsymbol{E}} \\ \tilde{\boldsymbol{E}}^\top & \mathbf{0} \end{bmatrix}}_{:=\boldsymbol{K}}{}^{-1}. \tag{9}$$

For brevity we use the short hand notation $[\boldsymbol{v}; \boldsymbol{u}] := [\boldsymbol{v}^\top, \boldsymbol{u}^\top]^\top$ for stacking vectors $\boldsymbol{v}, \boldsymbol{u}$ vertically. Note that the last entry of $\tilde{\boldsymbol{b}} := \boldsymbol{b}_{-n}$ and $\tilde{\boldsymbol{\beta}} := \boldsymbol{\beta}_{-n}$ is removed. This is due to a surplus degree of freedom in the equality conditions from Eq. (3), see part (b) of the proof. Likewise, for

$$\boldsymbol{E} = \begin{bmatrix} \mathbb{1}_n \otimes \boldsymbol{I}_m & \boldsymbol{I}_n \otimes \mathbb{1}_m \end{bmatrix} \in \mathbb{R}^{mn \times (m+n)}, \tag{10}$$

the last column is removed $\tilde{\boldsymbol{E}} := \boldsymbol{E}_{:,-(m+n)}$.

In principle, we can use Lemma 2 directly to solve Eq. (7). However, the computational cost of inverting the matrix $\boldsymbol{K}$ in Eq. (9) is prohibitive. In fact, even storing the Jacobian $\boldsymbol{J}$ in the working memory of a typical machine is problematic, since it is a dense matrix with $\mathcal{O}(mn)$ rows and columns, where $m, n > 1000$ in practice. Instead, we observe that computing Eq. (7) merely requires us to compute vector-Jacobian products (VJP) of the form $\boldsymbol{v}^\top \boldsymbol{J}$. The main results from this section can therefore be summarized as follows, see Appendix C.3 for a proof:

Theorem 3 (Backward pass)**.** *For $\boldsymbol{P} = \boldsymbol{P}^*$, the backward pass in Eq. (7) can be computed in closed form by solving the following linear system:*

$$\begin{bmatrix} \lambda \operatorname{diag}(\boldsymbol{p})^{-1} & \tilde{\boldsymbol{E}} \\ \tilde{\boldsymbol{E}}^\top & \mathbf{0} \end{bmatrix} \begin{bmatrix} \nabla_{\boldsymbol{c}} \ell \\ -\nabla_{[\boldsymbol{a}; \tilde{\boldsymbol{b}}]} \ell \end{bmatrix} = \begin{bmatrix} -\nabla_{\boldsymbol{p}} \ell \\ \mathbf{0} \end{bmatrix}. \tag{11}$$

4.3. Algorithm

In the previous section, we derived a closed-form expression of the Sinkhorn backward pass in Theorem 3. This requires solving the sparse linear system in Eq. (11), which has $\mathcal{O}(mn)$ rows and columns, and thus amounts to a worst-case complexity of $\mathcal{O}(m^3 n^3)$ [17]. We can further reduce the computation cost by exploiting the specific block structure of $\boldsymbol{K}$, which leads to the following algorithm:

Algorithm 1: *Sinkhorn operator backward*

```
Input  : $\nabla_{\boldsymbol{P}} \ell, \boldsymbol{P}, \boldsymbol{a}, \boldsymbol{b}$
Output: $\nabla_{\boldsymbol{C}} \ell, \nabla_{\boldsymbol{a}} \ell, \nabla_{\boldsymbol{b}} \ell$
1 $\boldsymbol{T} \leftarrow \boldsymbol{P} \odot \nabla_{\boldsymbol{P}} \ell$.
2 $\tilde{\boldsymbol{T}} \leftarrow \boldsymbol{T}_{:,-n}, \tilde{\boldsymbol{P}} \leftarrow \boldsymbol{P}_{:,-n} \in \mathbb{R}^{m \times n-1}$.
3 $\boldsymbol{t}^{(a)} \leftarrow \boldsymbol{T}\mathbb{1}_n, \tilde{\boldsymbol{t}}^{(b)} \leftarrow \tilde{\boldsymbol{T}}^\top \mathbb{1}_m$.
4 $\begin{bmatrix} \nabla_{\boldsymbol{a}} \ell \\ \nabla_{\tilde{\boldsymbol{b}}} \ell \end{bmatrix} \leftarrow \begin{bmatrix} \operatorname{diag}(\boldsymbol{a}) & \tilde{\boldsymbol{P}} \\ \tilde{\boldsymbol{P}}^\top & \operatorname{diag}(\tilde{\boldsymbol{b}}) \end{bmatrix}^{-1} \begin{bmatrix} \boldsymbol{t}^{(a)} \\ \tilde{\boldsymbol{t}}^{(b)} \end{bmatrix}$.
5 $\nabla_{\boldsymbol{b}} \ell \leftarrow \left[\nabla_{\tilde{\boldsymbol{b}}} \ell; 0\right]$.
6 $\boldsymbol{U} \leftarrow \nabla_{\boldsymbol{a}} \ell \mathbb{1}_n^\top + \mathbb{1}_m \nabla_{\boldsymbol{b}} \ell^\top$.
7 $\nabla_{\boldsymbol{C}} \ell \leftarrow -\lambda^{-1}(\boldsymbol{T} - \boldsymbol{P} \odot \boldsymbol{U})$.
```

See Appendix A for a PyTorch implementation of this algorithm. Most methods listed in Tab. 1 consider a special case of the functional specified in Eq. (4). The resulting gradients of Algorithm 1 are thereby, for the most part, consistent with such specialized approaches. We now show that the resulting gradients $\nabla_{\boldsymbol{C}} \ell, \nabla_{\boldsymbol{a}} \ell, \nabla_{\boldsymbol{b}} \ell$ from Algorithm 1 are indeed solutions of the linear system in Theorem 3.

Theorem 4. *Let $\boldsymbol{a}, \boldsymbol{b}$ be two input marginals and $\boldsymbol{P} = \boldsymbol{P}^*$ the transportation plan resulting from the forward pass in Eq. (6), then Algorithm 1 solves the backward pass Eq. (7).*

Sketch of the proof. The main idea of this proof is showing that Algorithm 1 yields a solution $\nabla_{[\boldsymbol{c}; \boldsymbol{a}; \tilde{\boldsymbol{b}}]} \ell$ of the linear system from Eq. (11). To that end, we leverage the Schur complement trick which yields the following two expressions:

$$\nabla_{[\boldsymbol{a}; \tilde{\boldsymbol{b}}]} \ell = \left(\tilde{\boldsymbol{E}}^\top \operatorname{diag}(\boldsymbol{p}) \tilde{\boldsymbol{E}}\right)^{-1} \tilde{\boldsymbol{E}}^\top \operatorname{diag}(\boldsymbol{p}) \nabla_{\boldsymbol{p}} \ell. \tag{12a}$$

$$\nabla_{\boldsymbol{c}} \ell = -\lambda^{-1} \left(\operatorname{diag}(\boldsymbol{p}) \nabla_{\boldsymbol{p}} \ell - \operatorname{diag}(\boldsymbol{p}) \tilde{\boldsymbol{E}} \nabla_{[\boldsymbol{a}; \tilde{\boldsymbol{b}}]} \ell\right). \tag{12b}$$

In Appendix C.4 we further show that these two identities in their vectorized form are equivalent to Algorithm 1. □

4.4. Practical considerations

Error bounds. Theorem 4 proves that Algorithm 1 computes the exact gradients $\nabla_{\boldsymbol{C}} \ell, \nabla_{\boldsymbol{a}} \ell, \nabla_{\boldsymbol{b}} \ell$, given that $\boldsymbol{P} = \boldsymbol{P}^*$ is the exact solution of Eq. (4). In practice, the operator S_λ in Eq. (6) is replaced by the Sinkhorn approximation

$\boldsymbol{S}_\lambda^{(\tau)}$ from Eq. (5) for a fixed, finite $\tau \in \mathbb{N}$. This small discrepancy in the approximation $\boldsymbol{P} = \boldsymbol{S}_\lambda^{(\tau)} \approx \boldsymbol{P}^*$ propagates to the backward pass as follows:

Theorem 5 (Error bounds)**.** *Let $\boldsymbol{P}^* := S_\lambda(\boldsymbol{C}, \boldsymbol{a}, \boldsymbol{b})$ be the exact solution of Eq. (4) and let $\boldsymbol{P}^{(\tau)} := \boldsymbol{S}_\lambda^{(\tau)}$ be the Sinkhorn estimate from Eq. (5). Further, let $\sigma_+, \sigma_-, C_1, C_2, \epsilon > 0$, s.t. $\left\|\boldsymbol{P}^* - \boldsymbol{P}^{(\tau)}\right\|_F < \epsilon$ and that for all $\boldsymbol{P}$ for which $\|\boldsymbol{P} - \boldsymbol{P}^*\|_F < \epsilon$ we have $\min_{i,j} \boldsymbol{P}_{i,j} \geq \sigma_-$, $\max_{i,j} \boldsymbol{P}_{i,j} \leq \sigma_+$ and the loss ℓ has bounded derivatives $\left\|\nabla_{\boldsymbol{p}}\ell\right\|_2 \leq C_1$ and $\left\|\nabla_{\boldsymbol{p}}^2\ell\right\|_F \leq C_2$. For $\kappa = \|\tilde{\boldsymbol{E}}^\dagger\|_2$, where $\tilde{\boldsymbol{E}}^\dagger$ indicates the Moore-Penrose inverse of $\tilde{\boldsymbol{E}}$, the difference between the gradients $\nabla_{\boldsymbol{C}}\ell^*, \nabla_{\boldsymbol{a}}\ell^*, \nabla_{\boldsymbol{b}}\ell^*$ of the exact $\boldsymbol{P}^*$ and the gradients $\nabla_{\boldsymbol{C}}\ell^{(\tau)}, \nabla_{\boldsymbol{a}}\ell^{(\tau)}, \nabla_{\boldsymbol{b}}\ell^{(\tau)}$ of the approximate $\boldsymbol{P}^{(\tau)}$, obtained via Algorithm 1, satisfy*

$$\left\|\nabla_{[\boldsymbol{a};\boldsymbol{b}]}\ell^* - \nabla_{[\boldsymbol{a};\boldsymbol{b}]}\ell^{(\tau)}\right\|_F \leq \kappa\sqrt{\frac{\sigma_+}{\sigma_-}}\left(\frac{1}{\sigma_-}C_1 + C_2\right)\left\|\boldsymbol{P}^* - \boldsymbol{P}^{(\tau)}\right\|_F \tag{13a}$$

$$\left\|\nabla_{\boldsymbol{C}}\ell^* - \nabla_{\boldsymbol{C}}\ell^{(\tau)}\right\|_F \leq \lambda^{-1}\sigma_+\left(\frac{1}{\sigma_-}C_1 + C_2\right)\left\|\boldsymbol{P}^* - \boldsymbol{P}^{(\tau)}\right\|_F. \tag{13b}$$

We provide a proof in Appendix C.5, as well as an empirical evaluation in Appendix B.1.

Computation cost. In comparison to automatic differentiation (AD), the computation cost of Algorithm 1 is independent of the number of Sinkhorn iterations τ. For square matrices, $m = n$, the runtime and memory complexities of AD are $\mathcal{O}(\tau n^2)$. On the other hand, our approach has a runtime and memory complexity of $\mathcal{O}(n^3)$ and $\mathcal{O}(n^2)$ respectively. We show empirical comparisons between the two approaches in Sec. 5.1. Another compelling feature of our approach is that none of the operations in Algorithm 1 explicitly convert the matrices $\boldsymbol{P}, \nabla_{\boldsymbol{P}}\ell, \nabla_{\boldsymbol{C}}\ell, \dots \in \mathbb{R}^{m \times n}$ into their vectorized form $\boldsymbol{p}, \nabla_{\boldsymbol{p}}\ell, \nabla_{\boldsymbol{c}}\ell, \dots \in \mathbb{R}^{mn}$. This makes it computationally more efficient since GPU processing favors small, dense matrix operations over the large, sparse linear system in Eq. (11).

Marginal probability invariance. As discussed in Lemma 2, the last element of $\tilde{\boldsymbol{b}}$ needs to be removed to make $\boldsymbol{K}$ invertible. However, setting the last entry of the gradient $\nabla_{b_n}\ell = 0$ to zero still yields exact gradients: By definition, the full marginal $\boldsymbol{b}$ is constrained to the probability simplex Δ_n, see Eq. (1). In practice, we apply an a priori $\mathrm{softmax}$ to $\boldsymbol{b}$ (and analogously $\boldsymbol{a}$). For some applications, $\boldsymbol{b}$ can be assumed to be immutable, if we only want to learn the cost matrix $\boldsymbol{C}$ and not the marginals $\boldsymbol{a}$ and $\boldsymbol{b}$. Overall, this means that the gradient of $\boldsymbol{b}$ is effectively indifferent to constant offsets of all entries, and setting $\nabla_{b_n}\ell = 0$ does not contradict the statement of Theorem 3.

5. Experiments

In Sec. 5.1, we empirically compare the computation cost of Algorithm 1 to automatic differentiation (AD). In Sec. 5.2 and Sec. 5.3, we show results on two common classes of applications where we want to learn the marginals $\boldsymbol{a}$ and the cost matrix $\boldsymbol{C}$ respectively. We assume a fixed GPU memory (VRAM) budget of 24GB – any setting that exceeds this limit is deemed out of memory (OOM).

5.1. Computation cost

We empirically compare the computation cost of our algorithm with the standard automatic differentiation approach, see Fig. 3. All results were computed on a single NVIDIA Quadro RTX 8000 graphics card. In practice, the computation cost of both approaches primarily depends on the parameters m, n, τ. It is for the most part indifferent to other hyperparameters and the actual values of $\boldsymbol{C}, \boldsymbol{a}, \boldsymbol{b}$. We therefore use random (log normal distributed) cost matrices $\ln \boldsymbol{C}_{i,j} \sim \mathcal{N}(0,1)$ and uniform marginals $\boldsymbol{a} = \boldsymbol{b} = \frac{1}{n}\mathbb{1}_n$ with $m = n \in \{10, 100, 1000\}$. For each setting, we report the cost of the forward and backward pass averaged over 1k iterations. Depending on m, n, our approach is faster for $\tau \gtrsim 40, 50, 90$ iterations. Note that our backward pass is independent of the number of forward iterations τ. Finally, the memory requirements are dramatically higher for AD, since it needs to maintain the computation graph of all τ forward iterations. In practice, this often limits the admissible batch size or input resolution, see Sec. 5.2 and Sec. 5.3.

5.2. Wasserstein barycenters

The main idea of Barycenter computation is to interpolate between a collection of objects $\{\boldsymbol{b}_1, \dots, \boldsymbol{b}_k\} \subset \mathbb{R}^n$ as a convex combination with weights that lie on the probability simplex $\boldsymbol{w} \in \Delta_k$, see Eq. (1). Specifically, we optimize

$$\boldsymbol{a}^* := \arg\min_{\boldsymbol{a} \in \Delta_n} \sum_{i=1}^k w_i d(\boldsymbol{a}, \boldsymbol{b}_i) \qquad \text{with} \tag{14}$$

$$d(\boldsymbol{a}, \boldsymbol{b}) := \min_{\boldsymbol{P} \in \Pi(\boldsymbol{a}, \boldsymbol{b})} \langle \boldsymbol{P}, \boldsymbol{D} \rangle_F - \lambda h(\boldsymbol{P}), \tag{15}$$

where $\boldsymbol{D} \in \mathbb{R}^{n \times n}$ denotes the squared pairwise distance matrix between the domains of $\boldsymbol{a}$ and $\boldsymbol{b}$. We use the Adam optimizer [21] for the outer optimization in Eq. (14). The inner optimization Eq. (15) is a special case of Eq. (4). Overall, Eq. (14) allows us to compute geometrically meaningful interpolations in arbitrary metric spaces. We consider the explicit tasks of interpolating between images in Fig. 4 and functions on manifolds in Fig. 5. Note that there are a number of specialized algorithms that minimize Eq. (14) in a highly efficient manner [10, 27, 41]. In Appendix B.2, we further show how to apply the barycenter technique to image clustering on the MNIST dataset.

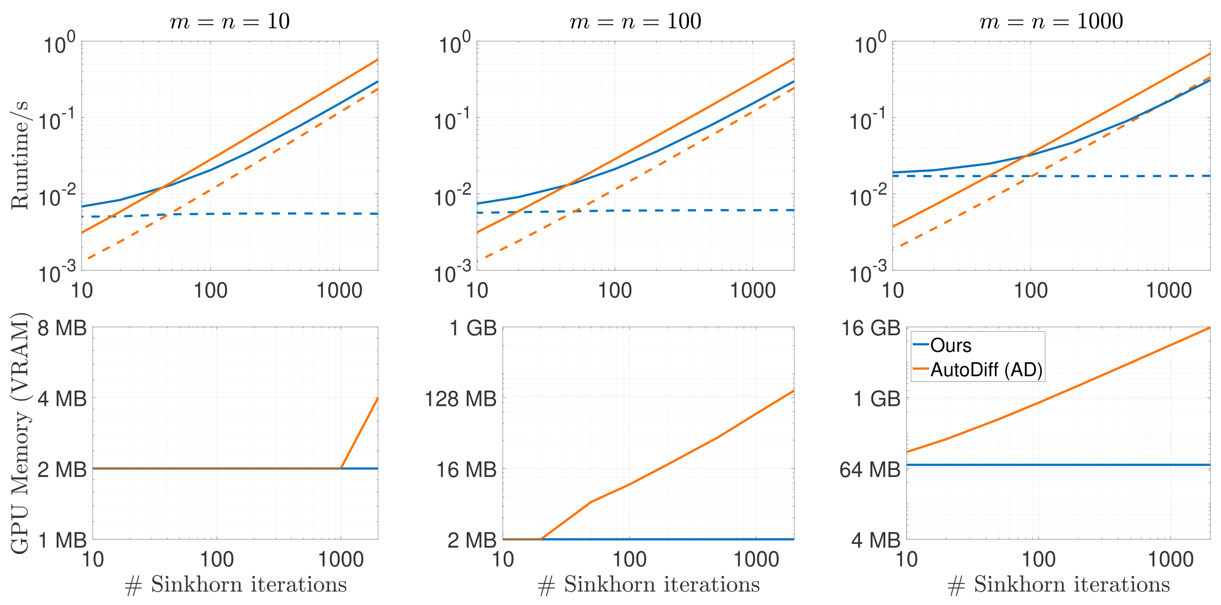


Figure 3. **Computational complexity.** We compare the runtime per iteration (top row) and GPU memory requirements (bottom row) of our approach (blue) and automatic differentiation (orange). We consider a broad range of settings with quadratic cost matrices of size $m = n \in \{10, 100, 1000\}$ and $\tau \in [10, 2000]$ Sinkhorn iterations. For the runtime, we show both the total time (solid lines) and the time of only the backward pass (dashed lines). Both ours and AD were implemented in the PyTorch [30] framework, where memory is allocated in discrete units, which leads to a large overlap for the minimum allocation size of 2MB (bottom row, left plot).

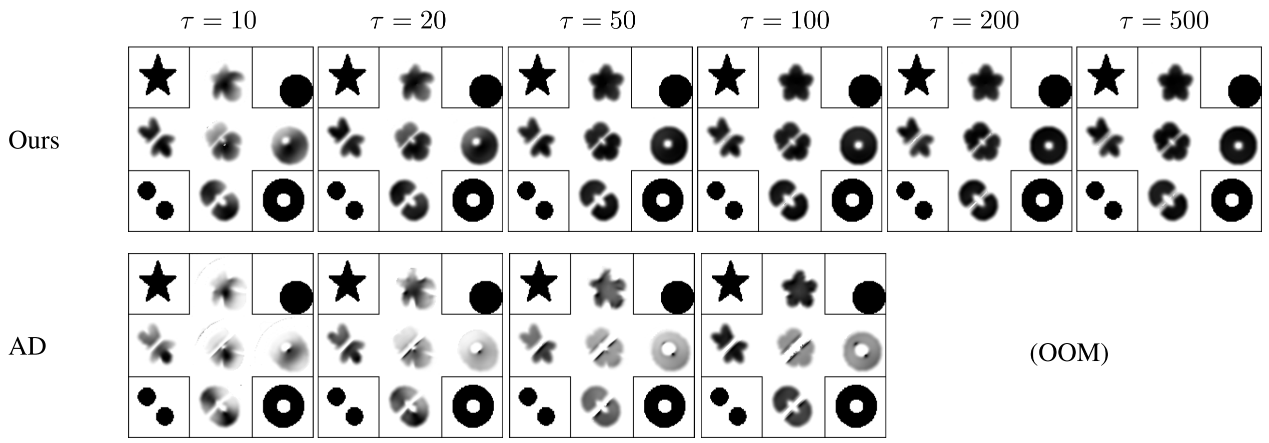


Figure 4. **Wasserstein barycenter.** A comparison between our method (top row) and AD (bottom row) on the application of image barycenter computation. In each cell, we show 5 centroids of 4 input images (corners) with bilinear interpolation weights. The predictions based on the proposed implicit gradients are more stable (providing more crisp interpolations), even for very few Sinkhorn iterations τ. Moreover, AD is out of memory for $\tau \geq 200$. Here, the input images have a resolution of $n = 64^2$ and we set $\lambda = 0.002$.

5.3. Permutation learning and matching

Number sorting. The Sinkhorn operator is nowadays a standard tool to parameterize approximate permutations within a neural network. One work that clearly demonstrates the effectiveness of this approach is the Gumbel-Sinkhorn (GS) method [28]. The main idea is to learn the natural ordering of sets of input elements $\{x_1, \dots, x_n\}$, see Appendix B.3 for more details. Here, we consider the concrete example of learning to sort real numbers from the unit interval $x_i \in [0, 1]$ for $n \in \{200, 500, 1000\}$ numbers. We compare the implicit Sinkhorn module to the vanilla GS method in Fig. 6. Without further modifications, our method significantly decreases the error at test time, defined as the proportion of incorrectly sorted elements.

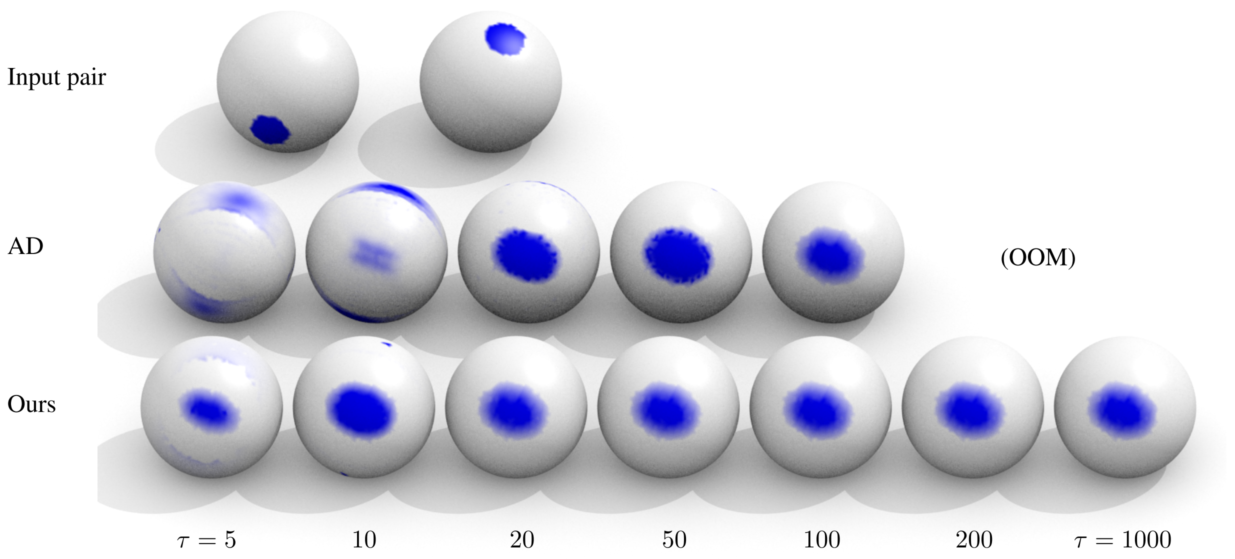


Figure 5. **Manifold barycenter.** We compute barycenters of two circular input distributions on the surface of a sphere (first row). Specifically, we compare the results of minimizing Eq. (14) with AD (second row) and implicit gradients (third row). The sphere is discretized as a triangular mesh with 5000 vertices. On this resolution, AD is out of memory for $\tau \geq 200$ Sinkhorn iterations whereas ours is still feasible for $\tau = 1000$. The obtained interpolations produce the slightly elongated shape of an ellipse since the surface of the sphere has a constant positive Gaussian curvature.

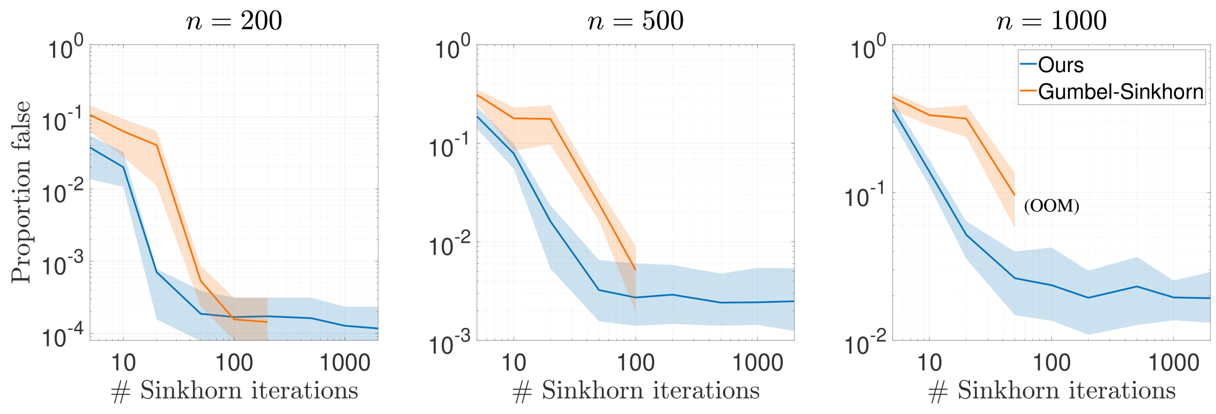


Figure 6. **Number sorting.** We show that we can improve the Gumbel-Sinkhorn method [28] directly with Algorithm 1. Specifically, we consider the task of permutation learning to sort random number sequences of length $n \in \{200, 500, 1000\}$, see [28, Sec 5.1] for more details. We replace AD in the GS network with implicit differentiation (blue curves) and compare the obtained results to the vanilla GS architecture (orange curves). Our approach yields more accurate permutations while using much less computational resources – GS is out of memory for $\tau > 200, 100, 50$ forward iterations, respectively. For all settings, we show the mean proportion of correct test set predictions (solid lines), as well as the 10 and 90 percentiles (filled areas). The curves are to some degree noisy, since individual results depend on a finite number of (random) test samples. Also, notice that the log-scale of the y-axis exaggerates small fluctuations for $\tau \geq 100$.

Point cloud registration. Several recent methods use the Sinkhorn operator as a differentiable, bijective matching layer for deep learning [13, 25, 39, 46, 47]. Here, we consider the concrete application of rigid point cloud registration [47] and show that we can improve the performance with implicit differentiation, see Tab. 2. While our results on the clean test data are comparable but slightly worse than the vanilla RPM-Net [47], our module generalizes more robustly to partial and noisy observations. This indicates that, since computing gradients with our method is less noisy than AD, it helps to learn a robust matching policy that is overall more consistent, see Fig. 7 for qualitative comparisons. We provide further details on the RPM-Net baseline and more qualitative results in Appendix B.3.

		clean data	partial			noisy		
			90%	80%	70%	$\sigma = 0.001$	$\sigma = 0.01$	$\sigma = 0.1$
Rot. MAE (↓)	RPM	0.0299	41.1427	47.1848	52.5945	18.5886	28.1436	43.1884
	Ours	0.1371	4.4955	11.0519	20.9274	1.0238	1.2548	2.2272
Trans. MAE (↓)	RPM	0.0002	0.1743	0.2126	0.2490	0.0848	0.1187	0.1770
	Ours	0.0015	0.0484	0.0995	0.1578	0.0096	0.0113	0.0171
Chamf. dist. (↓)	RPM	0.0005	4.3413	4.6829	4.9581	2.2077	3.0492	4.6935
	Ours	0.0054	0.5498	1.4291	2.2080	0.0783	0.1237	0.4562

Table 2. **Point cloud registration.** We compare the quantitative performance of RPM-Net [47] and implicit differentiation on ModelNet40 [44]. The two architectures are identical except for the altered Sinkhorn module. For all results, we follow the training protocol described in [47, Sec. 6]. Moreover, we assess the ability of the obtained networks to generalize to partial and noisy inputs at test time. For the former, we follow [47, Sec. 6.6] and remove up to 70% of the input point clouds from a random half-space. For the noisy test set, we add Gaussian white noise $\mathcal{N}(0, \sigma)$ with different variances $\sigma \in \{0.001, 0.01, 0.1\}$. For all settings, we report the rotation and translation errors, as well as the Chamfer distance to the reference surface. The latter is scaled by a factor of $1e2$ for readability.

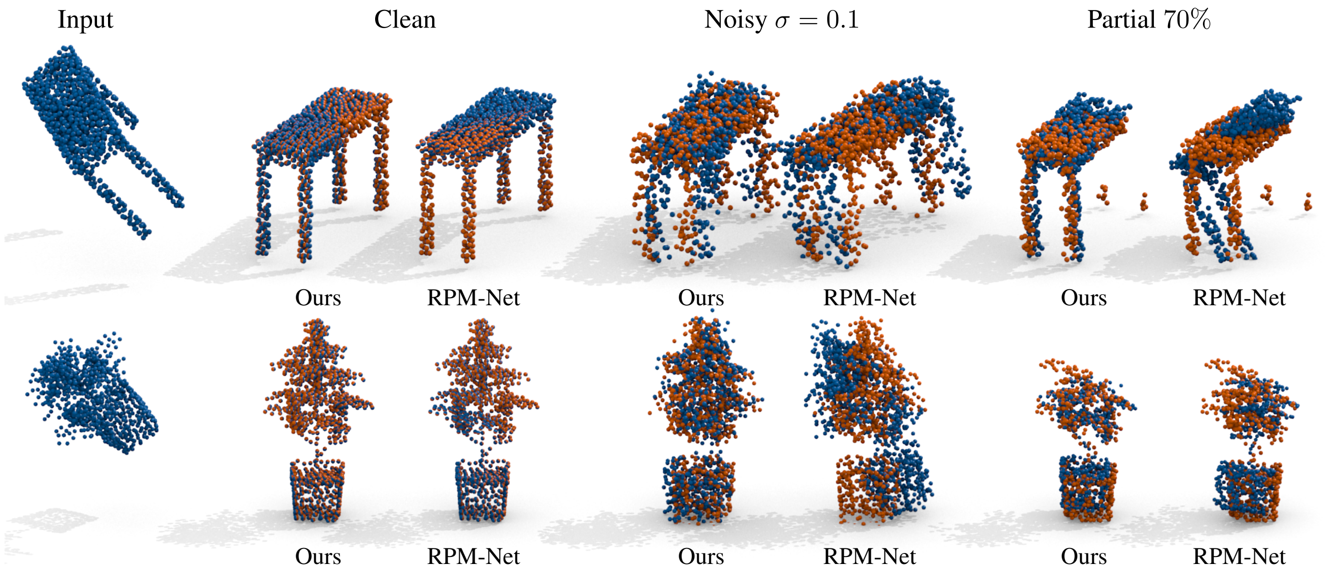


Figure 7. **Point cloud registration.** Qualitative comparisons of RPM-Net [47] and the improved version based on implicit differentiation. In each row, we show a different test pair with the input pose X (1st column, blue), as well as the overlap of the reference pose Y (orange) and the predicted pose (blue) for the clean, noisy, and partial settings. Both approaches work well for the clean data, but ours generalizes more robustly to noisy and partial pairs.

6. Conclusion

We presented a unifying framework that provides analytical gradients of the Sinkhorn operator in its most general form. In contrast to more specialized approaches [7, 11, 17, 22, 26], our algorithm can be deployed in a broad range of applications in a straightforward manner. Choosing the number of Sinkhorn iterations $\tau \in \mathbb{N}$ is generally subject to a trade-off between the computation cost and accuracy. The main advantage of implicit differentiation is that it proves to be much more scalable than AD, since the backward pass is independent of τ. Our experiments demonstrate that combining the implicit Sinkhorn module with existing approaches often improves the performance. We further provide theoretical insights and error bounds that quantify the accuracy of Algorithm 1 for noisy inputs.

Limitations & societal impact In our view, one of the main limitations of Algorithm 1 is that AD results in a faster training time for very few iterations $\tau \approx 10$. Whether this is offset by the empirically more stable training (see Sec. 5.2 and Sec. 5.3) has to be judged on a case-by-case basis. In terms of the societal impact, one of the major advantages of our method is that it reduces computation time and GPU memory demand of Sinkhorn layers within neural networks. It thereby has the potential to make such techniques more accessible to individuals and organizations with limited access to computational resources.

Acknowledgements

This work was supported by the ERC Advanced Grant SIMULACRON and the Munich School for Data Science.

References

[1] Pierre Ablin, Gabriel Peyré, and Thomas Moreau. Superefficiency of automatic differentiation for functions defined as a minimum. In *International Conference on Machine Learning*, pages 32–41. PMLR, 2020. 2

[2] Ryan Prescott Adams and Richard S Zemel. Ranking via Sinkhorn propagation. *arXiv preprint arXiv:1106.1925*, 2011. 2

[3] Brandon Amos and J Zico Kolter. Optnet: Differentiable optimization as a layer in neural networks. In *International Conference on Machine Learning*, pages 136–145. PMLR, 2017. 2

[4] Dimitri P Bertsekas. A distributed algorithm for the assignment problem. *Lab. for Information and Decision Systems Working Paper, MIT*, 1979. 3

[5] Mathieu Blondel, Quentin Berthet, Marco Cuturi, Roy Frostig, Stephan Hoyer, Felipe Llinares-López, Fabian Pedregosa, and Jean-Philippe Vert. Efficient and modular implicit differentiation. *arXiv preprint arXiv:2105.15183*, 2021. 2

[6] Ethan D Bolker. Transportation polytopes. *Journal of Combinatorial Theory, Series B*, 13(3):251–262, 1972. 16

[7] Dylan Campbell, Liu Liu, and Stephen Gould. Solving the blind perspective-n-point problem end-to-end with robust differentiable geometric optimization. In *European Conference on Computer Vision*, pages 244–261. Springer, 2020. 1, 2, 8

[8] Lenaic Chizat, Pierre Roussillon, Flavien Léger, François-Xavier Vialard, and Gabriel Peyré. Faster Wasserstein distance estimation with the Sinkhorn divergence. *Advances in Neural Information Processing Systems*, 33, 2020. 2

[9] Marco Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. In *Advances in neural information processing systems*, pages 2292–2300, 2013. 1, 2, 3, 11

[10] Marco Cuturi and Arnaud Doucet. Fast computation of Wasserstein barycenters. In *International conference on machine learning*, pages 685–693. PMLR, 2014. 2, 5

[11] Marco Cuturi, Olivier Teboul, Jonathan Niles-Weed, and Jean-Philippe Vert. Supervised quantile normalization for low-rank matrix approximation. *arXiv preprint arXiv:2002.03229*, 2020. 1, 2, 8

[12] Marco Cuturi, Olivier Teboul, and Jean-Philippe Vert. Differentiable ranking and sorting using optimal transport. *Advances in Neural Information Processing Systems*, 32, 2019. 2

[13] Marvin Eisenberger, Aysim Toker, Laura Leal-Taixé, and Daniel Cremers. Deep shells: Unsupervised shape correspondence with optimal transport. In *Advances in Neural Information Processing Systems*, volume 33, pages 10491–10502. Curran Associates, Inc., 2020. 1, 2, 7, 11

[14] Patrick Emami and Sanjay Ranka. Learning permutations with Sinkhorn policy gradient. *arXiv preprint arXiv:1805.07010*, 2018. 2

[15] Sven Erlander and Neil F Stewart. *The gravity model in transportation analysis: theory and extensions*, volume 3. Vsp, 1990. 16

[16] Jean Feydy, Thibault Séjourné, François-Xavier Vialard, Shun-ichi Amari, Alain Trouvé, and Gabriel Peyré. Interpolating between optimal transport and MMD using Sinkhorn divergences. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 2681–2690. PMLR, 2019. 2

[17] Rémi Flamary, Marco Cuturi, Nicolas Courty, and Alain Rakotomamonjy. Wasserstein discriminant analysis. *Machine Learning*, 107(12):1923–1945, 2018. 1, 2, 4, 8

[18] Charlie Frogner, Chiyuan Zhang, Hossein Mobahi, Mauricio Araya-Polo, and Tomaso Poggio. Learning with a Wasserstein loss. In *Proceedings of the 28th International Conference on Neural Information Processing Systems-Volume 2*, pages 2053–2061, 2015. 2

[19] Aude Genevay, Gabriel Peyré, and Marco Cuturi. Learning generative models with Sinkhorn divergences. In *International Conference on Artificial Intelligence and Statistics*, pages 1608–1617. PMLR, 2018. 2

[20] Stephen Gould, Richard Hartley, and Dylan Campbell. Deep declarative networks: A new hope. *arXiv preprint arXiv:1909.04866*, 2019. 2

[21] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 5

[22] Marcel Klatt, Carla Tameling, and Axel Munk. Empirical regularized optimal transport: Statistical theory and applications. *SIAM Journal on Mathematics of Data Science*, 2(2):419–443, 2020. 1, 2, 8

[23] Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955. 3, 11

[24] Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998. 11

[25] Liu Liu, Dylan Campbell, Hongdong Li, Dingfu Zhou, Xibin Song, and Ruigang Yang. Learning 2d-3d correspondences to solve the blind perspective-n-point problem. *arXiv preprint arXiv:2003.06752*, 2020. 1, 2, 7, 11

[26] Giulia Luise, Alessandro Rudi, Massimiliano Pontil, and Carlo Ciliberto. Differential properties of Sinkhorn approximation for learning with Wasserstein distance. *Advances in Neural Information Processing Systems*, 31:5859–5870, 2018. 1, 2, 8

[27] Giulia Luise, Saverio Salzo, Massimiliano Pontil, and Carlo Ciliberto. Sinkhorn barycenters with free support via Frank-Wolfe algorithm. *Advances in Neural Information Processing Systems*, 32:9322–9333, 2019. 2, 5

[28] G Mena, J Snoek, S Linderman, and D Belanger. Learning latent permutations with Gumbel-Sinkhorn networks. In *ICLR 2018 Conference Track*, volume 2018. OpenReview, 2018. 2, 6, 7, 11, 12

[29] Gautam Pai, Jing Ren, Simone Melzi, Peter Wonka, and Maks Ovsjanikov. Fast sinkhorn filters: Using matrix scaling for non-rigid shape correspondence with functional maps. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 384–393, 2021. 2

[30] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming

Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32:8026–8037, 2019. 6, 11

[31] Giorgio Patrini, Rianne van den Berg, Patrick Forre, Marcello Carioni, Samarth Bhargav, Max Welling, Tim Genewein, and Frank Nielsen. Sinkhorn autoencoders. In *Uncertainty in Artificial Intelligence*, pages 733–743. PMLR, 2020. 2

[32] Ofir Pele and Michael Werman. Fast and robust earth mover's distances. In *2009 IEEE 12th international conference on computer vision*, pages 460–467. IEEE, 2009. 3

[33] Gabriel Peyré, Marco Cuturi, et al. Computational optimal transport: With applications to data science. *Foundations and Trends® in Machine Learning*, 11(5-6):355–607, 2019. 2, 16

[34] Marin Vlastelica Pogan, Anselm Paulus, Vit Musil, Georg Martius, and Michal Rolinek. Differentiation of blackbox combinatorial solvers. In *International Conference on Learning Representations*, 2019. 2

[35] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017. 11

[36] Yossi Rubner, Leonidas J Guibas, and Carlo Tomasi. The earth mover's distance, multi-dimensional scaling, and color-based image retrieval. In *Proceedings of the ARPA image understanding workshop*, volume 661, page 668, 1997. 3

[37] Tim Salimans, Dimitris Metaxas, Han Zhang, and Alec Radford. Improving gans using optimal transport. In *6th International Conference on Learning Representations, ICLR 2018*, 2018. 2

[38] Rodrigo Santa Cruz, Basura Fernando, Anoop Cherian, and Stephen Gould. Deeppermnet: Visual permutation learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3949–3957, 2017. 2

[39] Paul-Edouard Sarlin, Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. Superglue: Learning feature matching with graph neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4938–4947, 2020. 1, 2, 7, 11

[40] Morgan A Schmitz, Matthieu Heitz, Nicolas Bonneel, Fred Ngole, David Coeurjolly, Marco Cuturi, Gabriel Peyré, and Jean-Luc Starck. Wasserstein dictionary learning: Optimal transport-based unsupervised nonlinear dictionary learning. *SIAM Journal on Imaging Sciences*, 11(1):643–678, 2018. 2

[41] Justin Solomon, Fernando De Goes, Gabriel Peyré, Marco Cuturi, Adrian Butscher, Andy Nguyen, Tao Du, and Leonidas Guibas. Convolutional Wasserstein distances: Efficient optimal transportation on geometric domains. *ACM Transactions on Graphics (TOG)*, 34(4):1–11, 2015. 2, 5

[42] Andrew M Stuart and Marie-Therese Wolfram. Inverse optimal transport. *SIAM Journal on Applied Mathematics*, 80(1):599–619, 2020. 2

[43] Cédric Villani. *Topics in optimal transportation*. Number 58. American Mathematical Soc., 2003. 2, 3

[44] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1912–1920, 2015. 8, 13, 15

[45] Yujia Xie, Hanjun Dai, Minshuo Chen, Bo Dai, Tuo Zhao, Hongyuan Zha, Wei Wei, and Tomas Pfister. Differentiable top-k operator with optimal transport. *arXiv preprint arXiv:2002.06504*, 2020. 2

[46] Lei Yang, Wenxi Liu, Zhiming Cui, Nenglun Chen, and Wenping Wang. Mapping in a cycle: Sinkhorn regularized unsupervised learning for point cloud shapes. In *European Conference on Computer Vision*, pages 455–472. Springer, 2020. 1, 2, 7, 11

[47] Zi Jian Yew and Gim Hee Lee. Rpm-net: Robust point matching using learned features. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11824–11833, 2020. 1, 2, 7, 8, 11, 13, 15