SCALABLE SINKHORN BACKPROPAGATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Optimal transport has recently gained increasing attention in the context of deep learning. A major contributing factor is the line of work on smooth relaxations that make the classical optimal transport problem differentiable. The most prominent example is entropy regularized optimal transport which can be optimized efficiently via an alternating scheme of Sinkhorn projections. We thus experienced a surge of deep learning techniques that use the Sinkhorn operator to learn matchings, permutations, sorting and ranking, or to construct a geometrically motivated loss function for generative models. The prevalent approach to training such a neural network is first-order optimization by algorithmic unrolling of the forward pass. Hence, the runtime and memory complexity of the backward pass increase linearly with the number of Sinkhorn iterations. This often makes it impractical when computational resources like GPU memory are scarce. A more efficient alternative is computing the derivative of a Sinkhorn layer via implicit differentiation. Our main contribution is deriving a simple and efficient algorithm that performs this backward pass in closed form. It is based on the Sinkhorn operator in its most general form – with learnable cost matrices and target capacities. We further provide a theoretical analysis with error bounds for approximate inputs. Finally, we demonstrate that, for a number of applications, replacing automatic differentiation with our module often improves the stability and accuracy of the obtained gradients while drastically reducing the computation cost.

1 INTRODUCTION

Computing similarities between probability distributions is a fundamental problem in machine learning. In the most general case, tools like the Kullback–Leibler (KL) divergence (Kullback & Leibler, 1951) establish a notion of distance between such measures μ and ν. In practice, μ and ν are often themselves embedded in a metric space $\big(\Omega, d_\Omega\big)$. Common examples include spaces of images, voxel grids, point clouds, 3D surface meshes, or generic Euclidean features. For this class of problems, optimal transport (OT) proves to be a powerful formalism (Villani, 2003; Peyré et al., 2019). The Kantorovich formulation (Kantorovich, 1942) of OT yields a well-defined distance function over probability measures on Ω that respects the geometric structure induced by d_Ω.

The pioneering work of Cuturi (2013) further proposes to add an entropy regularization term to the standard Kantorovich objective. This relaxation dramatically reduces the cost of computing the OT distance between discrete measures μ, ν from a super-cubic to a quadratic runtime complexity. A crucial contributing factor that popularized entropy regularized OT in the context of deep learning is that the resulting Sinkhorn divergence is differentiable. As a result, a number of works show how the Sinkhorn divergence can be used as a loss function to train a machine learning model (Luise et al., 2018; Genevay et al., 2018; Klatt et al., 2020; Ablin et al., 2020). In this work, we consider an extension of this idea via the more general *Sinkhorn operator*. This operator is defined as the mapping $(\boldsymbol{C}, \boldsymbol{a}, \boldsymbol{b}) \mapsto \boldsymbol{P}$ of a cost matrix $\boldsymbol{C} \in \mathbb{R}^{m \times n}$ and the source and target capacities $\boldsymbol{a} \in \mathbb{R}^m, \boldsymbol{b} \in \mathbb{R}^n$ to the transportation plan $\boldsymbol{P} \in \mathbb{R}_+^{m \times n}$. In comparison, the Sinkhorn divergence $(\boldsymbol{a}, \boldsymbol{b}) \mapsto \langle \boldsymbol{C}, \boldsymbol{P} \rangle_F$ is a special case of this operator (Cuturi, 2013, Eq. (2)).

It is nowadays common practice to design a deep neural network as the composition of various mathematical functions or "layers". Some of those layers may themselves be defined as the solution of an (inner) optimization problem, as long as the mapping induced by the inner optimization is well defined and differentiable. In order to train such a neural network, we need to solve a bilevel optimization problem. The most common approaches to that end are algorithmic unrolling of the inner

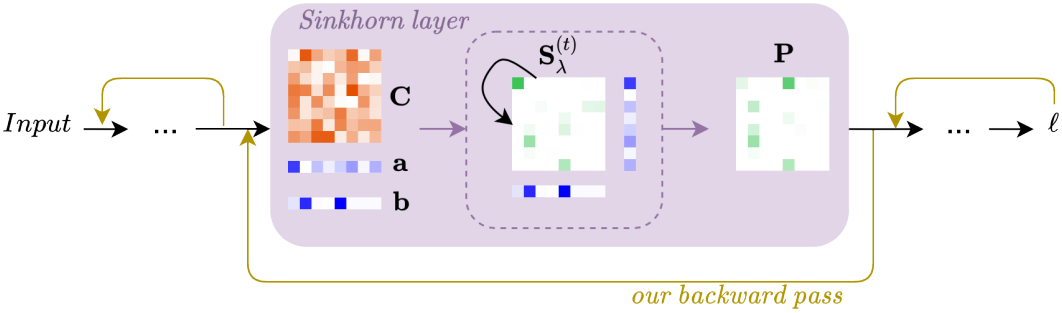

Figure 1: Overview of a typical workflow with an embedded Sinkhorn layer. We consider a neural network whose input are e.g. images, 3D point clouds, voxel grids, surface meshes, etc. The Sinkhorn layer maps the cost matrix $\boldsymbol{C}$ and marginals $\boldsymbol{a}$, $\boldsymbol{b}$ to the transportation plan $\boldsymbol{P}$ via iterative matrix scaling. In general, the whole network potentially contains learnable weights before and after the Sinkhorn layer. To compute the gradients $(\nabla_{\boldsymbol{C}}\ell, \nabla_{\boldsymbol{a}}\ell, \nabla_{\boldsymbol{b}}\ell)$ during training, instead of using naive automatic differentiation, we propose a novel customised backward pass which computes gradients efficiently and in closed form, see Algorithm 1.

optimization loop and implicit differentiation (Amos & Kolter, 2017; Gould et al., 2019; Bai et al., 2019; Blondel et al., 2021). There is a vast literature on applications of combining the Sinkhorn operator with deep learning, see our related work discussion in Section 4. The majority of approaches unroll the forward optimization steps and apply automatic differentiation to compute gradients of the resulting iterative scheme. Since the computation graph for all iterations needs to be maintained, the resulting computation cost is often prohibitively high for GPU processing. To alleviate this cost, there are a number of approaches related to our work that apply implicit differentiation to specific subclasses of problems, like learning with a fixed cost matrix or immutable input capacities (Luise et al., 2018; Flamary et al., 2018; Campbell et al., 2020; Cuturi et al., 2020; Klatt et al., 2020). We can recover most such methods as special cases of our framework. In our work, we derive, from first principles, an efficient algorithm to compute gradients of a generic Sinkhorn layer see Fig. 1. Our contribution can be summarized as follows:

1. We leverage the implicit function theorem to compute gradients of the *Sinkhorn operator* in its most general form. We further use the Schur complement of the corresponding vector-Jacobian product (VJP) to construct an efficient backpropagation algorithm (Algorithm 1).
2. We provide theoretical guarantees for the accuracy of the resulting gradients, in dependency of the approximation error in the forward pass (Theorem 5).
3. Finally, we show that our algorithm can be used out of the box to enhance existing approaches that use a differentiable Sinkhorn layer. Plugging in our module often improves the quantitative results while using significantly less GPU memory.

2 Background

Optimal transport Optimal transport enables us to compute the distance of two probability measures on the same domain $\Omega \subset \mathbb{R}^d$. In this work, we focus on the specific case of 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

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

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 Villani (2003).

Entropy regularization Evaluating the distance $d(\mu, \nu)$ in practice requires solving the linear assignment problem (LAP) from Equation 2. This can be done via specialized algorithms like the Hungarian algorithm (Kuhn, 1955) or the Auction algorithm (Bertsekas, 1979), as well as recent solvers (Rubner et al., 1997; Pele & Werman, 2009). However, most approaches are computationally heavy and slow in practice (Cuturi, 2013). A popular alternative is augmenting the LAP objective in Equation 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 $\lambda > 0$. The seminal work of Cuturi (2013) showed 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 Equation 4. Specifically, this can be done via a scheme of alternating Sinkhorn projections

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

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 Cuturi (2013), in the limit this scheme converges to a minimizer $\boldsymbol{S}_\lambda^{(t)} \xrightarrow{t \to \infty} \boldsymbol{S}_\lambda$ of Equation 4. In practice, we can use a finite number of iterations $\tau \in \mathbb{N}$ to achieve a sufficiently small residual.

3 METHOD

3.1 PROBLEM FORMULATION

The integration of the Sinkhorn operator from Equation 4 into deep neural networks has become popular for solving a wide range of practical tasks, see our discussion in Section 4. 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. 1 for an overview.

A common strategy is to replace the exact forward pass $S_\lambda(\boldsymbol{C}, \boldsymbol{a}, \boldsymbol{b})$ in Equation 6 by the approximate solution $\boldsymbol{S}_\lambda^{(\tau)}$ from Equation 5. Like the original solution in Equation 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 Equation 5.

For the backward pass in Equation 7, an overwhelming number of applications nowadays rely on automatic differentiation in environments like PyTorch (Paszke et al., 2019) or TensorFlow (Abadi et al., 2016). While this solution is straightforward and convenient, it has fundamental shortcomings: First of all, the computation time for the backward pass increases with the number of Sinkhorn iterations τ. Moreover, the computation graph needs to be maintained for all τ matrix scaling iterations. For many applications, this makes it infeasible or limits the resolution since the GPU memory is often the computational bottleneck in practice. Instead, to address these shortcomings, in the next section we utilize implicit differentiation in order to introduce a closed-form expression for computing the gradients of ℓ w.r.t both the marginals $\boldsymbol{a}, \boldsymbol{b}$ and the cost matrix $\boldsymbol{C}$.

3.2 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. 3.3. To this end, we start by reframing the optimization problem in Equation 4 in terms of its Karush–Kuhn–Tucker (KKT) conditions:

Lemma 1. *The transportation plan $\boldsymbol{P}^*$ is a global minimum of Equation 4 iff*

$$\mathcal{K}(\boldsymbol{c}, \boldsymbol{a}, \boldsymbol{b}, \boldsymbol{p}^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*) := \begin{bmatrix} \boldsymbol{c} + \lambda \log(\boldsymbol{p}^*) + \mathbb{1}_n \otimes \boldsymbol{\alpha}^* + \boldsymbol{\beta}^* \otimes \mathbb{1}_m \\ (\mathbb{1}_n^\top \otimes \boldsymbol{I}_m)\boldsymbol{p}^* - \boldsymbol{a} \\ (\boldsymbol{I}_n \otimes \mathbb{1}_m^\top)\boldsymbol{p}^* - \boldsymbol{b} \end{bmatrix} = \mathbf{0}_l, \tag{8}$$

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 Equation 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 $\log(p) := -\infty, p \leq 0$.

Proof. The function $\mathcal{K}$ contains the KKT conditions corresponding to the optimization problem in Equation 4. The proposed identity therefore follows directly from the (strict) convexity of Equation 4, see Erlander & Stewart (1990). Apart from the two equality constraints, Equation 3 contains additional inequality constraints $P_{i,j} \geq 0$. Those are however inactive and can be dropped, because the entropy term in Equation 4 invariably yields transportation plans in the interior of the positive orthant $P_{i,j} > 0$, see Peyré et al. (2019, p. 68). □

Establishing this identity is an important first step towards computing a closed-form gradient for the backward pass in Equation 7. It reframes the optimization problem in Equation 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:

Lemma 2. *The KKT conditions in Equation 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 matrix*[1]

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

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 Equation 3, see part (b) of the proof. Likewise, for

$$\boldsymbol{E} = [\mathbb{1}_n \otimes \boldsymbol{I}_m \quad \boldsymbol{I}_n \otimes \mathbb{1}_m] \in \mathbb{R}^{mn \times (m+n)}, \tag{10}$$

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

Sketch of the proof. The basis for this proof is leveraging the implicit function theorem (IFT), which we show in two parts, (a) and (b). See Appendix C.1 for the full proof.

(a) We first establish that the matrix $\boldsymbol{K}$ is invertible. We can then prove by direct computation that $\boldsymbol{K}$ corresponds to the partial derivative of $\mathcal{K}$ from Equation 8 wrt. $[\boldsymbol{p}; \boldsymbol{\alpha}; \tilde{\boldsymbol{\beta}}]$. The identity from Equation 9 then follows directly from the IFT, since the partial derivative of $\mathcal{K}$ w.r.t. $[\boldsymbol{c}; -\boldsymbol{a}; -\boldsymbol{b}]$ is simply $\frac{\partial \mathcal{K}}{\partial [\boldsymbol{c}; -\boldsymbol{a}; -\boldsymbol{b}]} = \boldsymbol{I}_l$.

(b) To show why the last condition from $\boldsymbol{E}$ is removed, we first assert that the kernel $\ker(\boldsymbol{E}^\top)$ is $(m-1)(n-1)$ dimensional. The rank–nullity theorem then yields that the subspace spanned by the columns of $\boldsymbol{E}$ is $mn-(m-1)(n-1) = m+n-1$ dimensional. Removing the last condition $\tilde{\boldsymbol{E}} := \boldsymbol{E}_{:,-(n+m)}$ ensures that the $m+n-1$ columns of $\tilde{\boldsymbol{E}}$ are linearly independent and $\boldsymbol{K}$ is invertible.

□

[1] 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.

In principle, we can use Lemma 2 directly to compute the backward pass from Equation 7. However, the computational cost of inverting the matrix $\boldsymbol{K}$ in Equation 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 Equation 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:

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

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

Proof. This identity follows trivially from Lemma 2 by applying the chain rule

$$\nabla_{[\boldsymbol{c};-\boldsymbol{a};-\tilde{\boldsymbol{b}}]} \ell = \left(\frac{\partial \left[\boldsymbol{p}; \boldsymbol{\alpha}; \tilde{\boldsymbol{\beta}}\right]}{\partial \left[\boldsymbol{c}; -\boldsymbol{a}; -\tilde{\boldsymbol{b}}\right]} \right)^\top \nabla_{[\boldsymbol{p};\boldsymbol{\alpha};\tilde{\boldsymbol{\beta}}]} \ell = -\boldsymbol{K}^{-1} \nabla_{[\boldsymbol{p};\boldsymbol{\alpha};\tilde{\boldsymbol{\beta}}]} \ell = \boldsymbol{K}^{-1} \begin{bmatrix} -\nabla_{\boldsymbol{p}} \ell \\ \boldsymbol{0} \end{bmatrix}. \tag{12}$$

The last equality holds, since the loss ℓ does not depend on the dual variables $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$, see Fig. 1. □

3.3 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 Equation 11, which has $\mathcal{O}(mn)$ rows and columns, and thus amounts to a worst-case complexity of $\mathcal{O}(m^3 n^3)$. We can further reduce the computation cost by exploiting the specific block structure of $\boldsymbol{K}$, which leads to our algorithm:

Algorithm 1: *Sinkhorn operator backward*

Input : $\nabla_{\boldsymbol{P}} \ell, \boldsymbol{P}, \boldsymbol{a}, \boldsymbol{b}$
Output: $\nabla_{\boldsymbol{C}} \ell, \nabla_{\boldsymbol{a}} \ell, \nabla_{\boldsymbol{b}} \ell$

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

See Appendix A for a PyTorch implementation of this algorithm. 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 Equation 6, then Algorithm 1 solves the backward pass defined in Equation 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 Equation 11. To that end, we leverage the Schur complement trick which yields the following two expressions:

$$\nabla_{[\boldsymbol{a};\tilde{\boldsymbol{b}}]} \ell = \left(\tilde{\boldsymbol{E}}^\top \operatorname{diag}(\boldsymbol{p}) \tilde{\boldsymbol{E}} \right)^{-1} \tilde{\boldsymbol{E}}^\top \operatorname{diag}(\boldsymbol{p}) \nabla_{\boldsymbol{p}} \ell. \tag{13a}$$

$$\nabla_{\boldsymbol{c}} \ell = -\lambda^{-1} \left(\operatorname{diag}(\boldsymbol{p}) \nabla_{\boldsymbol{p}} \ell - \operatorname{diag}(\boldsymbol{p}) \tilde{\boldsymbol{E}} \nabla_{[\boldsymbol{a};\tilde{\boldsymbol{b}}]} \ell \right). \tag{13b}$$

In Appendix C.2 we further show that these two identities in their vectorized form are equivalent to Algorithm 1 in matrix notation. □

3.4 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 Equation 4. In practice, the operator S_λ in Equation 6 is replaced by the Sinkhorn approximation $\boldsymbol{S}_\lambda^{(\tau)}$ from Equation 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 Equation 4 and let $\boldsymbol{P}^{(\tau)} := \boldsymbol{S}_\lambda^{(\tau)}$ be the Sinkhorn estimate from Equation 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 $\left\|\nabla^2_{\boldsymbol{p}}\ell\right\|_F \leq C_2$. 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*

$$\left\|\nabla_{[\boldsymbol{a};\boldsymbol{b}]}\ell^* - \nabla_{[\boldsymbol{a};\boldsymbol{b}]}\ell^{(\tau)}\right\|_F \leq \kappa\sqrt{\frac{\sigma_+}{\sigma_-}}\left(\frac{1}{\sigma_-}C_1 + C_2\right)\left\|\boldsymbol{P}^* - \boldsymbol{P}^{(\tau)}\right\|_F, \quad \textit{and} \tag{14a}$$

$$\left\|\nabla_{\boldsymbol{C}}\ell^* - \nabla_{\boldsymbol{C}}\ell^{(\tau)}\right\|_F \leq \lambda^{-1}\sigma_+\left(\frac{1}{\sigma_-}C_1 + C_2\right)\left\|\boldsymbol{P}^* - \boldsymbol{P}^{(\tau)}\right\|_F. \tag{14b}$$

We provide a proof in Appendix C.3, 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 Equation 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 Equation 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.

4 RELATED WORK

There is a vast literature on computational optimal transport (OT) (Villani, 2003; Peyré et al., 2019). In the following, we provide an overview of related machine learning applications. Our approach is based on entropy regularized optimal transport pioneered by Cuturi (2013). The resulting differentiable Sinkhorn divergence can be used as a loss function for training machine learning models (Frogner et al., 2015; Feydy et al., 2019; Chizat et al., 2020). To allow for first-order optimization, two common approaches for computing gradients are implicit differentiation (Luise et al., 2018; Cuturi et al., 2020; Klatt et al., 2020) and automatic differentiation (Genevay et al., 2018; Ablin et al., 2020). Relevant applications of the Sinkhorn divergence include computing Wasserstein barycenters (Cuturi & Doucet, 2014; Solomon et al., 2015; Luise et al., 2019), dictionary learning (Schmitz et al., 2018), as well as using a geometrically meaningful loss function for autoencoders (Patrini et al., 2020) or generative adversarial networks (GAN) (Genevay et al., 2018; Salimans et al., 2018).

More recently, several approaches emerged that use the Sinkhorn operator as a differentiable transportation layer in a neural network. Potential applications include permutation learning (Santa Cruz et al., 2017; Mena et al., 2018), ranking (Adams & Zemel, 2011; Cuturi et al., 2019), sorting via reinforcement learning (Emami & Ranka, 2018), discriminant analysis (Flamary et al., 2018) and

computing matchings between images (Sarlin et al., 2020), point clouds (Yew & Lee, 2020; Yang et al., 2020; Liu et al., 2020) or triangle meshes (Eisenberger et al., 2020; Pai et al., 2021). 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 (Amos & Kolter, 2017; Gould et al., 2019; Blondel et al., 2021). Similar ideas are pursued in (Luise et al., 2018; Klatt et al., 2020; Campbell et al., 2020) for the special subcases of learning with loss functions that are independent of the cost matrix or the marginals. Other approaches compute the input cost matrix via Bayesian inverse modeling (Stuart & Wolfram, 2020) or smooth the OT linear assignment problem (LAP) directly (Pogan et al., 2019).

5 Experiments

We now provide practical evidence for the merits of our Algorithm 1. In Sec. 5.1, we compare its computation cost 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. For all experiments, 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. 2. 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.

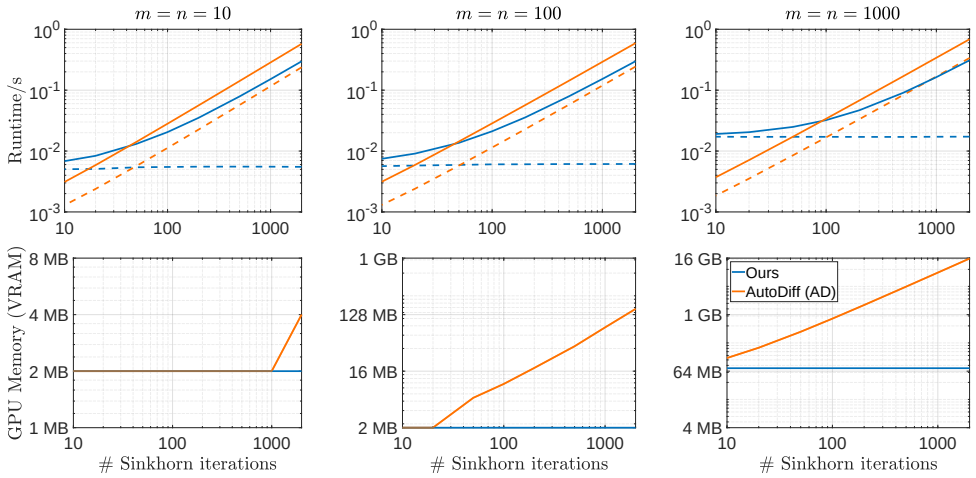

Figure 2: **Computational complexity.** We compare the runtime per iteration (top row) and GPU memory requirements (bottom row) of our approach (blue) and automatic differentiation (orange). We consider a broad range of settings with quadratic cost matrices of size $m = n \in \{10, 100, 1000\}$ and $\tau \in [10, 2000]$ 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 (Paszke et al., 2019) 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).

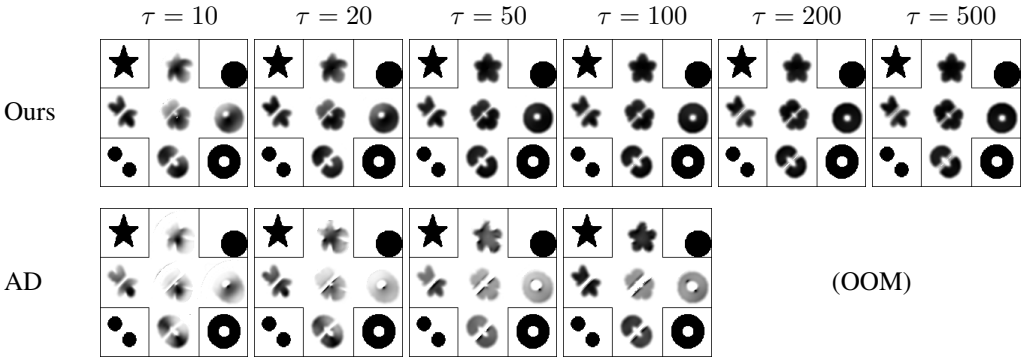

Figure 3: **Wasserstein barycenter.** A comparison between our method (top row) and automatic differentiation (bottom row) on the application of image barycenter computation. In each cell, we show 5 centroids of 4 input images (corners) with bilinear interpolation weights. The predictions of our approach are more stable, 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 $\lambda = 0.002$.

5.2 Wasserstein barycenters

Barycenter computation is a standard example of an inverse problem based on the Sinkhorn operator in Equation 4. The main idea 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 $\boldsymbol{w} \in \Delta_n$, see Equation 1. Specifically, the goal is to optimize for the marginal $\boldsymbol{a}^*$ via the expression

$$\boldsymbol{a}^* := \operatorname*{arg\,min}_{\boldsymbol{a} \in \Delta_n} \sum_{i=1}^{k} w_i d(\boldsymbol{a}, \boldsymbol{b}_i) \;\text{ with }\; d(\boldsymbol{a}, \boldsymbol{b}) := \min_{\boldsymbol{P} \in \Pi(\boldsymbol{a}, \boldsymbol{b})} \langle \boldsymbol{P}, \boldsymbol{D} \rangle_F - \lambda h(\boldsymbol{P}), \tag{15}$$

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 (Kingma & Ba, 2014) for the outer optimization in Equation 15. The inner optimization is a special case of Equation 4. Overall, Equation 15 allows us to compute geometrically meaningful interpolations in arbitrary metric spaces. We consider the explicit tasks of interpolating between images in Fig. 3 and functions on manifolds in Appendix B.2. Note that there are a number of specialized algorithms that minimize Equation 15 in a highly efficient manner (Cuturi & Doucet, 2014; Solomon et al., 2015; Luise et al., 2019). We mainly consider the barycenter problem a useful toy example to assess the stability of our algorithm in comparison to AD. On the other hand, the interpretation as a generic optimization problem is overall more flexible and allows us to trivially extend it to related tasks like image clustering, see Appendix B.2.

5.3 Permutation learning and matching

Number sorting The Sinkhorn operator is nowadays a standard tool to parameterize approximate, learnable permutations within a neural network, see the second paragraph of our related work discussion in Sec. 4. One work that clearly demonstrates the effectiveness of this approach is the Gumbel-Sinkhorn (GS) method (Mena et al., 2018). 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 insert our Sinkhorn module in the GS network and compare the performance with the vanilla GS method in Fig. 4. Without further modifications, our method significantly decreases the error at test time, defined as the proportion of incorrectly sorted elements.

Point cloud registration A number of recent methods use the Sinkhorn operator as a differentiable, bijective matching layer for deep learning (Sarlin et al., 2020; Yew & Lee, 2020; Yang et al., 2020; Liu et al., 2020; Eisenberger et al., 2020). Here, we consider the concrete application of rigid point cloud registration (Yew & Lee, 2020) and show that we can improve the performance with our backward algorithm, see Table 1. While our results on the clean test data are comparable but

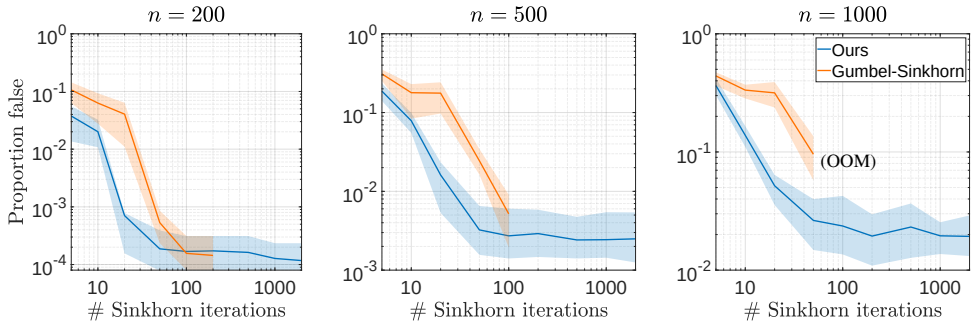

Figure 4: **Number sorting.** We show that we can improve the Gumbel-Sinkhorn method (Mena et al., 2018) directly with our Sinkhorn module. Specifically, we consider the task of permutation learning to sort random number sequences of length $n \in \{200, 500, 1000\}$, see Mena et al. (2018, Sec 5.1) for more details. We replace AD in the GS network with our module (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).

slightly worse than the vanilla RPM-Net (Yew & Lee, 2020), 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 with less overfitting. We provide more details on the RPM-Net baseline and qualitative comparisons in Appendix B.3.

6 Discussion

Our experiments clearly demonstrate that, for a broad range of applications, combining our Sinkhorn module with existing approaches often improves the performance. On the other hand, AD has an empirically faster runtime for very few Sinkhorn iterations $\tau \approx 10$ and high $n \geq 1000$. In our experiments, however, we see a clear trend that, to a certain point, increasing τ leads to more accurate results. Choosing τ 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. For the same reason, it also has a dramatically lower GPU memory demand, see Fig. 2. We therefore believe that the theoretical and empirical insights we provide have the potential to open up new avenues for future research, for which naive backpropagation schemes are computationally intractable or not sufficiently accurate.

		clean data	partial			noisy		
			90%	80%	70%	$\sigma = 0.001$	$\sigma = 0.01$	$\sigma = 0.1$
Rot. MSE	RPM	0.2329	63.7670	69.6706	74.4227	37.2575	50.3618	66.1996
	Ours	0.3231	10.3078	24.5810	40.7793	1.7912	2.2110	4.0208
Trans. MSE	RPM	0.0014	0.2659	0.3079	0.3462	0.1603	0.2100	0.2740
	Ours	0.0033	0.0797	0.1582	0.2420	0.0149	0.0181	0.0303
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 1: **Point cloud registration.** We compare the quantitative performance of RPM-Net (Yew & Lee, 2020) and our approach on ModelNet40 (Wu et al., 2015). The two architectures are identical except for the altered Sinkhorn module. For all results, we follow the training protocol described in Yew & Lee (2020, 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 Yew & Lee (2020, 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.

REFERENCES

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pp. 265–283, 2016.

Pierre Ablin, Gabriel Peyré, and Thomas Moreau. Super-efficiency of automatic differentiation for functions defined as a minimum. In *International Conference on Machine Learning*, pp. 32–41. PMLR, 2020.

Ryan Prescott Adams and Richard S Zemel. Ranking via sinkhorn propagation. *arXiv preprint arXiv:1106.1925*, 2011.

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

Shaojie Bai, J Zico Kolter, and Vladlen Koltun. Deep equilibrium models. *Advances in Neural Information Processing Systems*, 32:690–701, 2019.

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

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.

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

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*, pp. 244–261. Springer, 2020.

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.

Marco Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. In *Advances in neural information processing systems*, pp. 2292–2300, 2013.

Marco Cuturi and Arnaud Doucet. Fast computation of wasserstein barycenters. In *International conference on machine learning*, pp. 685–693. PMLR, 2014.

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

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.

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, pp. 10491–10502. Curran Associates, Inc., 2020.

Patrick Emami and Sanjay Ranka. Learning permutations with sinkhorn policy gradient. *arXiv preprint arXiv:1805.07010*, 2018.

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

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*, pp. 2681–2690. PMLR, 2019.

Rémi Flamary, Marco Cuturi, Nicolas Courty, and Alain Rakotomamonjy. Wasserstein discriminant analysis. *Machine Learning*, 107(12):1923–1945, 2018.

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*, pp. 2053–2061, 2015.

Aude Genevay, Gabriel Peyré, and Marco Cuturi. Learning generative models with sinkhorn divergences. In *International Conference on Artificial Intelligence and Statistics*, pp. 1608–1617. PMLR, 2018.

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

L Kantorovich. On the transfer of masses: Doklady akademii nauk ussr. 1942.

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

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.

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

Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

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

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.

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.

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.

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.

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*, pp. 384–393, 2021.

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.

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*, pp. 733–743. PMLR, 2020.

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

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.

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

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*, pp. 652–660, 2017.

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, pp. 668, 1997.

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

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*, pp. 3949–3957, 2017.

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*, pp. 4938–4947, 2020.

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.

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.

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

Cédric Villani. *Topics in optimal transportation*. Number 58. American Mathematical Soc., 2003.

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*, pp. 1912–1920, 2015.

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*, pp. 455–472. Springer, 2020.

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*, pp. 11824–11833, 2020.

A PyTorch implementation

Our proposed algorithm can be easily integrated into existing deep learning architectures. In the following, we provide our PyTorch (Paszke et al., 2019) implementation of this module. The forward pass is the standard Sinkhorn matrix scaling algorithm (Cuturi, 2013) with a robust log-space implementation. The backward function contains an implementation of our Algorithm 1.

```
import torch

class Sinkhorn(torch.autograd.Function):
   @staticmethod
   def forward(ctx, c, a, b, num_sink, lambd_sink):
      log_p = -c / lambd_sink
      log_a = torch.log(a).unsqueeze(dim=1)
      log_b = torch.log(b).unsqueeze(dim=0)
      for _ in range(num_sink):
         log_p -= (torch.logsumexp(log_p, dim=0, keepdim=True) - log_b)
         log_p -= (torch.logsumexp(log_p, dim=1, keepdim=True) - log_a)
      p = torch.exp(log_p)

      ctx.save_for_backward(p, torch.sum(p, dim=1), torch.sum(p, dim=0))
      ctx.lambd_sink = lambd_sink
      return p

   @staticmethod
   def backward(ctx, grad_p):
      p, a, b = ctx.saved_tensors
      m, n = p.shape

      grad_p *= -1 / ctx.lambd_sink * p
      K = torch.cat((
         torch.cat((torch.diag(a), p), dim=1),
         torch.cat((p.T, torch.diag(b)), dim=1)),
         dim=0)[:-1, :-1]
      t = torch.cat((
         grad_p.sum(dim=1),
         grad_p[:, :-1].sum(dim=0)),
         dim=0).unsqueeze(1)
      grad_ab, _ = torch.solve(t, K)
      grad_a = grad_ab[:m, :]
      grad_b = torch.cat((grad_ab[m:, :], torch.zeros([1, 1],
          device=device, dtype=torch.float32)), dim=0)
      U = grad_a + grad_b.T
      grad_p -= p * U
      grad_a = -ctx.lambd_sink * grad_a.squeeze(dim=1)
      grad_b = -ctx.lambd_sink * grad_b.squeeze(dim=1)

      return grad_p, grad_a, grad_b, None, None
```

B Additional experiments

B.1 Gradient accuracy

Theorem 5 states that the error of our backward pass in Algorithm 1 can be bounded by the error of the forward pass in Equation 6. We now assess the magnitude of this error in practice by revisiting the experiments on image barycenter computation and number sorting from Sec. 5. Since the ground truth gradients are unknown in general, we define the (approximate) ground truth gradients as $\nabla_{\boldsymbol{C}} \ell^* := \nabla_{\boldsymbol{C}} \ell^{(\tau_{\max})}, \nabla_{\boldsymbol{a}} \ell^* := \nabla_{\boldsymbol{a}} \ell^{(\tau_{\max})}$ for $\tau_{\max} := 10,000$. In Fig. 5, we compare the error of our approach to AD, averaged over all settings and iterations from the experiments in Fig. 3 and Fig. 4 respectively. These results show clearly that, on average, our approach produces

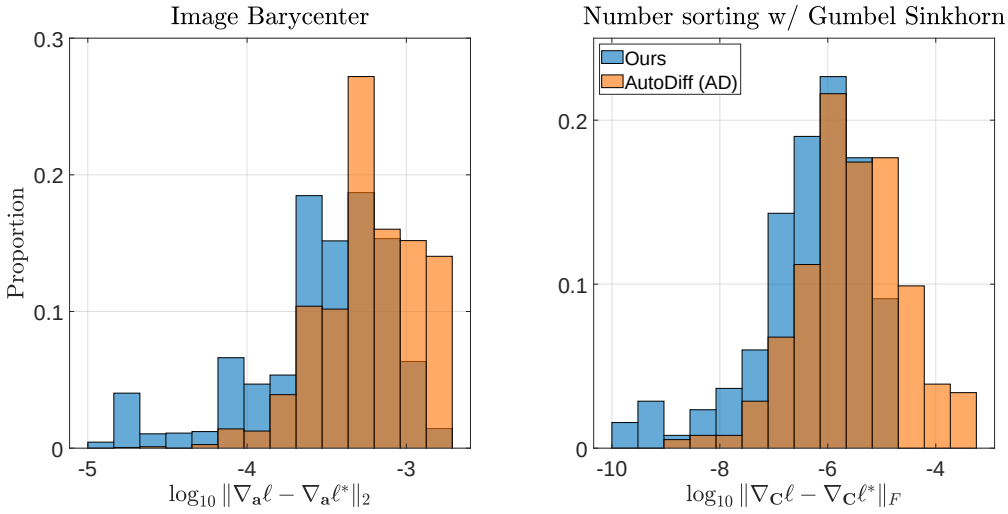

Figure 5: **Gradient accuracy.** We empirically assess the accuracy of the error bound discussed in Theorem 5. Specifically, we show the accuracy of the gradients $\nabla_{\boldsymbol{a}}\ell$ for the image barycenter experiment in Sec. 5.2 and $\nabla_{\boldsymbol{C}}\ell$ for the number sorting experiment in Sec. 5.3. While both distributions have a large overlap, the gradients from our approach are noticeably more accurate on average. Note that both comparisons show histograms on a log scale.

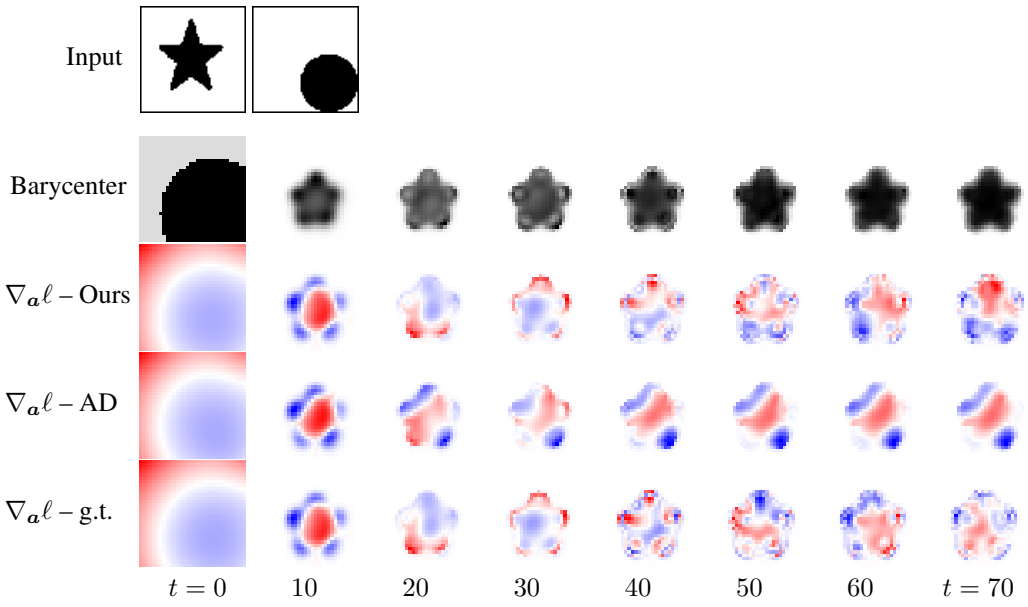

Figure 6: **Image barycenter gradients.** A qualitative comparison of our gradients (3rd row), the AD gradients (4th row), and the ground truth gradients (last row) for the image barycenter experiment from Sec. 5.2. Specifically, we consider the task of interpolating between two input images (1st row) with uniform interpolation weights $w_1 = w_2 = 0.5$. We show intermediate snapshots of the obtained barycenter image (2nd row) for different numbers of gradient descent iterations $t \in \{0, \dots, 70\}$ that result from minimizing the energy in Equation 15.

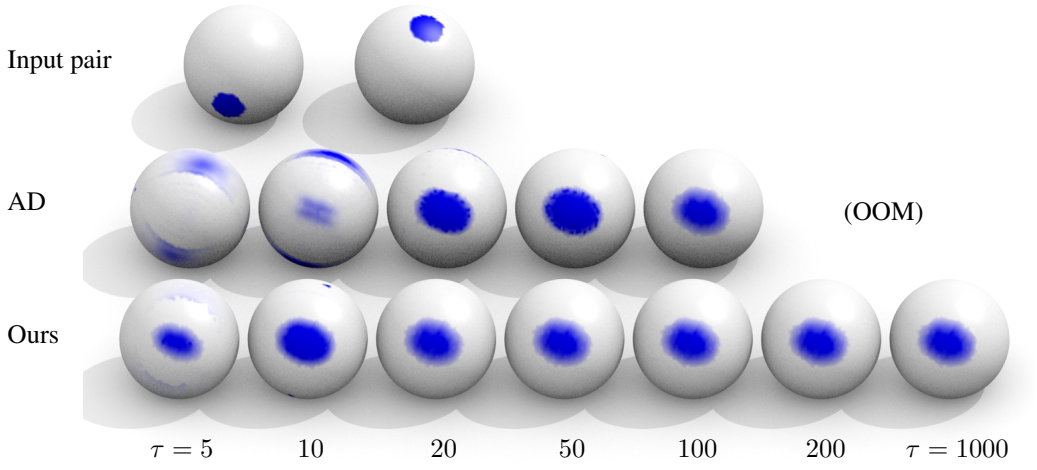

Figure 7: **Manifold barycenter.** We compute barycenters of two circular input distributions on the surface of a sphere (first row). Specifically, we compare the results of minimizing Equation 15 with AD (second row) and our gradients (third row). The sphere is discretized as a triangular mesh with 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.

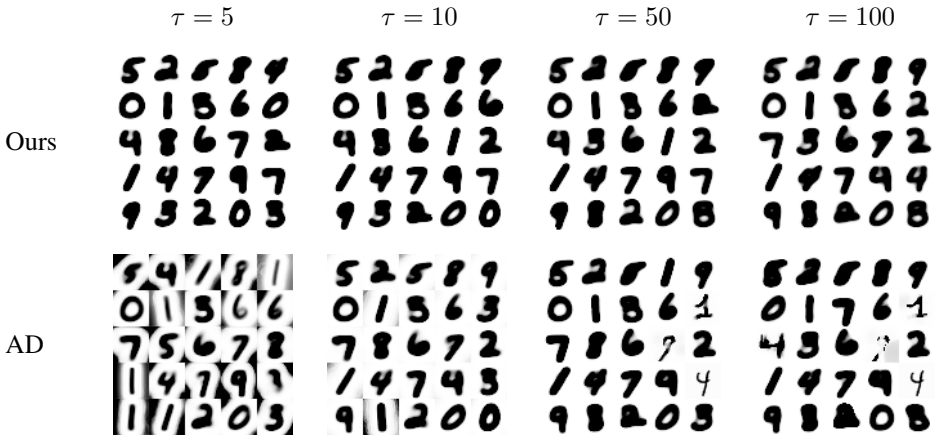

Figure 8: **MNIST k-means clustering**. A comparison of our approach and automatic differentiation on the task of k-means image clustering. For both approaches, we show the predicted $k = 25$ clusters for $\tau \in \{5, 10, 50, 100\}$ Sinkhorn iterations. We choose more than 10 clusters to capture several different appearances and styles for each digit. To make individual results more comparable, we use an identical random initialization of the cluster centroids for all settings. For AD, the maximum permissible batch sizes for the 4 settings are $512, 256, 64, 32$, whereas ours consistently allows for a batch size of 1024.

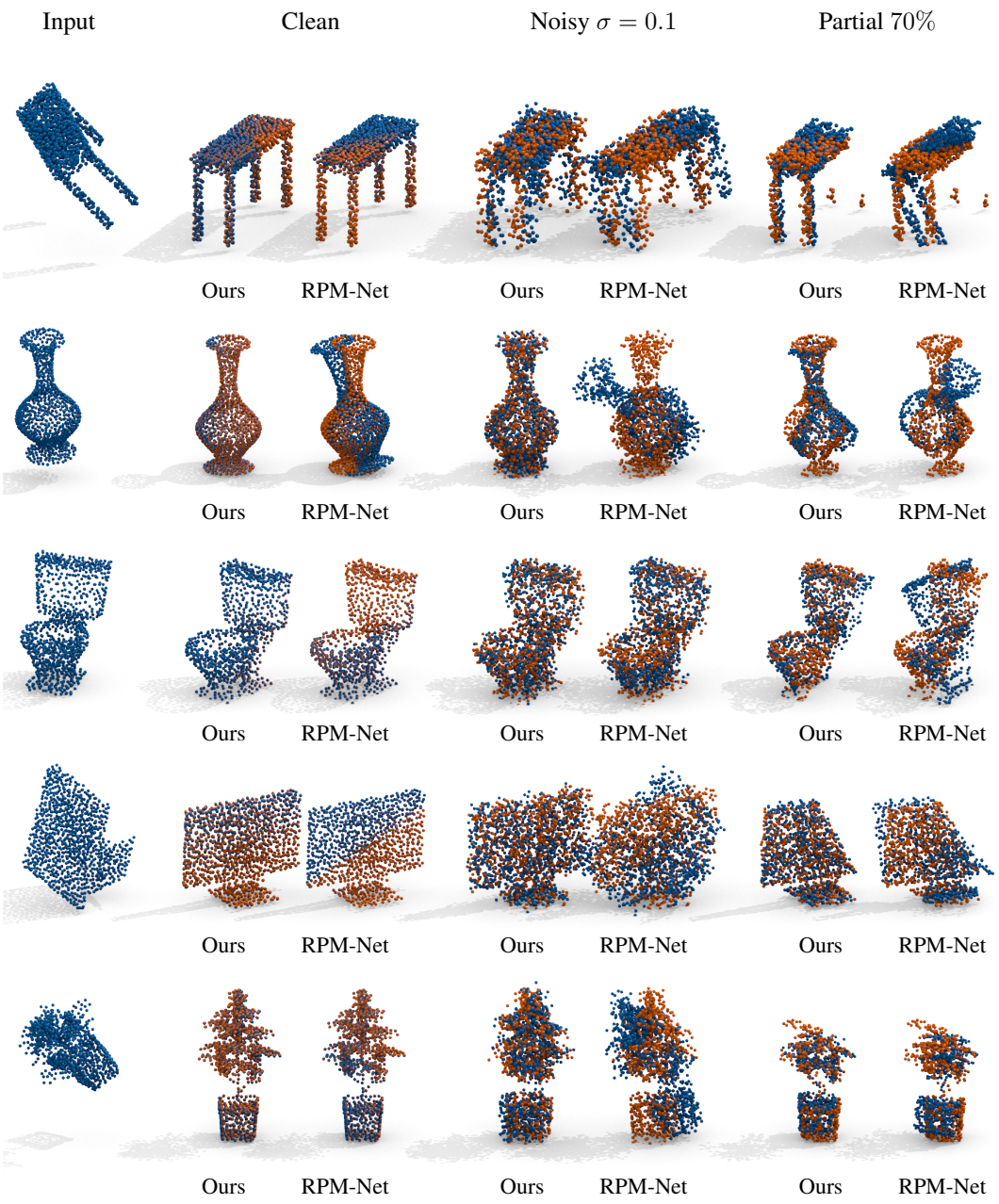

Figure 9: **Point cloud registration.** A qualitative comparison of RPM-Net (Yew & Lee, 2020) and the augmented version with our custom backward pass, see Alg. 1. The increased stability of the gradients predicted by our algorithm directly translates into more robust generalization results: Both methods are trained and tested on separate subsets of the 40 object categories in ModelNet40 (Wu et al., 2015), see Yew & Lee (2020, Section 6) for more details. Both methods yield accurate predictions for the clean test data, as indicated by the corresponding quantitative results in Table 1. On the other hand, our approach shows significant improvements when generalizing to noisy test data and partial inputs. 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.

more accurate gradients than AD. Intuitively, ours yields optimal gradients for a suboptimal forward pass $\boldsymbol{P}^* \approx \boldsymbol{S}_\lambda^{(\tau)}$, see Theorem 5, whereas AD yields approximate gradients for an approximate forward pass. To further illustrate this point, we show a qualitative comparison of the gradients of both approaches on the task of image barycenter computation in Fig. 6. These results demonstrate the usefulness of our custom backward pass. Our gradients and the ground truth start disagreeing in certain regions for $t \geq 50$, but only after the barycenter optimization converges. Compared to the vanilla AD approach, the gradients are much closer to the ground truth and therefore more useful.

B.2 Manifold barycenters and image clustering

In Section 5.2, we show that our approach can be used to compute Wasserstein barycenters of images. We can leverage the same approach to interpolate between distributions on more general manifold domains. Specifically, we can minimize Equation 15 for a set of input marginals $\{\boldsymbol{b}_1, \dots, \boldsymbol{b}_k\}$ and use squared pairwise geodesic distances $\boldsymbol{D} \in \mathbb{R}^{m \times n}$ as the input cost matrix. In Fig. 7, we show the predicted barycenters for our method and AD with two circular input distributions on a sphere. Moreover, we can use a similar approach and leverage Equation 15 to predict a clustering of a set of input images. To that end, we apply the k-means algorithm which alternates between computing cluster centroids (by minimizing Equation 15) and assigning all images $\boldsymbol{b}_i$ to their respective corresponding centroid (defined as the centroid with minimum distance d, right side of Equation 15). We show results on the 60k images from the MNIST dataset (LeCun, 1998) in Fig. 8. While our algorithm is robust to varying Sinkhorn iterations τ, AD's results vary significantly. For small $\tau \leq 10$, the Sinkhorn approximation is not sufficiently exact. For higher τ, memory constraints force AD to use a smaller batch size which again leads to instabilities.

B.3 Details on permutation baselines

Gumbel-Sinkhorn networks As outlined in Sec. 5.3, the goal of the Gumbel-Sinkhorn method (Mena et al., 2018) is to learn how to sort a set of input elements $\{x_1, \dots, x_n\}$. To this end, the cost matrix $\boldsymbol{C}$ is parameterized via a permutation-invariant neural network architecture (set encoder), conditioned on the input elements $\{x_1, \dots, x_n\}$. The matrix $\boldsymbol{C}$, together with marginals $\boldsymbol{a} = \boldsymbol{b} = \mathbb{1}_n$ are then passed through a differentiable Sinkhorn layer.[2] The final output $\boldsymbol{P}$ is a bistochatic matrix which encodes an approximate permutation. The training objective is a geometric loss, minimizing the distance of the sorted elements x_i to their natural ground-truth ordering. At test time, the Hungarian algorithm (Kuhn, 1955) is applied to the learned cost matrix $\boldsymbol{C}$ to obtain an exact, hard permutation. In Sec. 5.3, we show the concrete application of sorting n real numbers, sampled randomly from the uniform distribution $x_i \sim \mathcal{U}(0, 1)$. More specifically, we consider separate training and test sets of 4096 and 128 random sequences each and report the error, defined as the proportion of incorrectly placed numbers in the test set, see Fig. 4. To provide a more complete picture, we report quantitative results on the task of generalizing to different test sets here. Specifically, we train the vanilla GS method and our approach for $\tau = 200$ (the maximum number for which GS is below the GPU memory limit) for $n = 200$ numbers sampled from $\mathcal{U}(0, 1)$. We then investigate the error on test sets sampled from different distributions $x_i \sim \mathcal{U}(s, t)$ with $s < t$ in Table 2. Even though the variance is quite high, these results indicate that our method evidently helps to reduce overfitting. Note, that the improved generalization is observed for the setting $\tau = 200$, $n = 200$ where the performance of both methods on the standard test set is almost identical, see Fig. 4.

RPM-Net A number of recent works use the Sinkhorn operator as a differentiable matching module for deep learning (Sarlin et al., 2020; Yew & Lee, 2020; Yang et al., 2020; Liu et al., 2020; Eisenberger et al., 2020). The standard strategy of such methods is to use a learnable feature extractor to obtain descriptors $\boldsymbol{F}^X \in \mathbb{R}^{m \times p}, \boldsymbol{F}^Y \in \mathbb{R}^{n \times p}$ on pairs of input objects X and Y in a siamese manner. We can then define the cost matrix $\boldsymbol{C} := \boldsymbol{D}$ as the squared pairwise distance matrix $\boldsymbol{D}_{i,j} = \left\| \boldsymbol{F}^X_{:,i} - \boldsymbol{F}^Y_{:,j} \right\|_2^2$. For fixed, uniform marginals $\boldsymbol{a}$ and $\boldsymbol{b}$, the Sinkhorn operator then yields a soft matching $\boldsymbol{P} = S_\lambda(\boldsymbol{C}, \boldsymbol{a}, \boldsymbol{b}) \in \mathbb{R}^{m \times n}$ between the two input objects. The baseline method RPM-Net we consider in Sec. 5.3 uses this methodology to obtain a matching between pairs of input

[2] Strictly speaking, the choice of marginals $\boldsymbol{a} = \boldsymbol{b} = \mathbb{1}_n$ does not fit in our framework, since we require $\boldsymbol{a}, \boldsymbol{b} \in \Delta_n$, see Equation 1. However, we can simply use $\boldsymbol{a} = \boldsymbol{b} = \frac{1}{n}\mathbb{1}_n$ and scale the resulting transportation plan $\boldsymbol{P}$ by a constant factor of n.

	$\mathcal{U}(1,2)$	$\mathcal{U}(10,11)$	$\mathcal{U}(100,101)$	$\mathcal{U}(-1,0)$
Ours	0.2964 ± 0.0744	0.3340 ± 0.3059	0.3531 ± 0.1380	0.3552 ± 0.2116
Gumbel-Sinkhorn	0.3163 ± 0.0976	0.3620 ± 0.3179	0.3955 ± 0.1478	0.4678 ± 0.2526

Table 2: **Number sorting generalization.** We assess the capability of our approach (first row) and the vanilla Gumbel-Sinkhorn method (second row) (Mena et al., 2018) to generalize to various test sets $\mathcal{U}(s,t)$ with $s < t$. We train both methods to sort sets of $n = 200$ numbers x_i from the uniform distribution on the unit interval $\mathcal{U}(0,1)$ with $\tau = 200$ Sinkhorn iterations. For each test set, we show the mean proportion of false predictions, as well as the empirical standard deviation, obtained from 50 test runs per setting.

point clouds. As a feature extractor, it uses PointNet (Qi et al., 2017) with a custom input task point cloud, see Yew & Lee (2020, Sec. 5.1) for more details. Using the obtained soft correspondences, a simple SVD transformation then yields the optimal rigid transformation between the two input point clouds. In order to train their model, RPM-Net uses automatic differentiation of the Sinkhorn layer. We can now demonstrate that replacing AD with our backward algorithm improves the performance. To that end, we revisit the experiments from Yew & Lee (2020, Sec. 6): We use 20 separate object identities of the ModelNet40 dataset (Wu et al., 2015) as our train and test set respectively. In Fig 9, we now show a qualitative comparison corresponding to the results in Table 1 in the main paper. On the standard test set, both approaches produce comparable, high-quality results. On the other hand, our method significantly improves the robustness when generalizing to noisy data or partial views.

C PROOFS

In the following, we provide proofs of Lemma 2, Theorem 4 and Theorem 5 from the main paper.

C.1 PROOF OF LEMMA 2

Proof. As mentioned above, the key for proving this statement is applying the implicit function theorem. We start by (a) showing that this indeed yields the identity from Equation 9, and then (b) justify why removing the last $(m+n)$-th equality condition from $\boldsymbol{E}$ is necessary.

(a) First of all, we can verify by direct computation that the matrix $\boldsymbol{K}$ from Equation 9 corresponds to the partial derivatives of the KKT conditions from Equation 8

$$\boldsymbol{K} = \left(\frac{\partial \mathcal{K}(\boldsymbol{c}, \boldsymbol{a}, \boldsymbol{b}, \boldsymbol{p}, \boldsymbol{\alpha}, \boldsymbol{\beta})}{\partial \left[\boldsymbol{p}; \boldsymbol{\alpha}; \boldsymbol{\beta}\right]} \right)_{-l,-l}, \tag{16}$$

where the notation $(\cdot)_{-l,-l}$ means that the last row and column is removed and where $l = mn + m + n$. Furthermore, $\boldsymbol{K}$ is invertible (since the solution lies in the interior $\boldsymbol{P}_{i,j} > 0$, see Peyré et al. (2019, p. 68)), and the $m + n - 1$ columns of $\tilde{\boldsymbol{E}}$ are linearly dependent, see (b). Consequently, the implicit function theorem states that $\mathcal{K}$ implicitly defines a mapping $(\boldsymbol{c}, \boldsymbol{a}, \boldsymbol{b}) \mapsto (\boldsymbol{p}, \boldsymbol{\alpha}, \boldsymbol{\beta})$ whose Jacobian is

$$\boldsymbol{J} = \frac{\partial \left[\boldsymbol{p}; \boldsymbol{\alpha}; \tilde{\boldsymbol{\beta}}\right]}{\partial \left[\boldsymbol{c}; -\boldsymbol{a}; -\tilde{\boldsymbol{b}}\right]} = -\left(\frac{\partial \mathcal{K}}{\partial \left[\boldsymbol{p}; \boldsymbol{\alpha}; \boldsymbol{\beta}\right]} \right)^{-1}_{-l,-l} \underbrace{\left(\frac{\partial \mathcal{K}}{\partial \left[\boldsymbol{c}; -\boldsymbol{a}; -\boldsymbol{b}\right]} \right)_{-l,-l}}_{=\boldsymbol{I}_{l-1}} = -\boldsymbol{K}^{-1}. \tag{17}$$

(b) As part of the proof in (a), we use the fact that the columns of $\tilde{\boldsymbol{E}}$ are linearly independent. Verifying this statement also provides insight as to why removing the last row of $\tilde{\boldsymbol{b}}, \tilde{\boldsymbol{\beta}}$ and the last column of $\tilde{\boldsymbol{E}}$ is necessary. Intuitively, the columns of $\boldsymbol{E}$ contain one redundant condition: The identity

$$\sum_{i=1}^{m} \boldsymbol{P}_{i,n} = 0 \iff \boldsymbol{E}_{:,m+n}^{\top} \boldsymbol{p} = 0, \tag{18}$$

follows directly from the other $m+n-1$ conditions. More formally, we can take the kernel

$$\ker(\boldsymbol{E}^\top) = \big\{\boldsymbol{P} \in \mathbb{R}^{m\times n} | \boldsymbol{E}^\top \boldsymbol{p} = 0\big\}, \tag{19}$$

of $\boldsymbol{E}^\top \in \mathbb{R}^{m+n\times mn}$ and observe that $\dim\big(\ker(\boldsymbol{E}^\top)\big) = (m-1)(n-1)$, see Bolker (1972, Sec. 1). The rank–nullity theorem then implies that the dimension of the subspace spanned by the columns of $\boldsymbol{E}$ is of dimension $mn - (m-1)(n-1) = m+n-1$. Consequently, removing the redundant condition in Equation 19 from $\boldsymbol{E}$ yields the reduced $\tilde{\boldsymbol{E}} \in \mathbb{R}^{mn\times m+n-1}$ with $= m+n-1$ linearly independent columns.

□

C.2 Proof of Theorem 4

Proof. We want to show that the gradients $\nabla_{\boldsymbol{C}}\ell, \nabla_{\boldsymbol{a}}\ell, \nabla_{\boldsymbol{b}}\ell$ obtained with Algorithm 1 are equivalent to the solution of Equation 11. To that end, we start by applying the Schur complement trick to the block matrix $\boldsymbol{K}$. This yields the expression

$$\big(\boldsymbol{K}^{-1}\big)_{:,1:mn} = \begin{bmatrix} \lambda^{-1}\operatorname{diag}(\boldsymbol{p})\big(\boldsymbol{I}_{mn} + \tilde{\boldsymbol{E}}\big(\tilde{\boldsymbol{E}}^\top \operatorname{diag}(\boldsymbol{p})\tilde{\boldsymbol{E}}\big)^{-1}\tilde{\boldsymbol{E}}^\top \operatorname{diag}(\boldsymbol{p})\big) \\ \big(\tilde{\boldsymbol{E}}^\top \operatorname{diag}(\boldsymbol{p})\tilde{\boldsymbol{E}}\big)^{-1}\tilde{\boldsymbol{E}}^\top \operatorname{diag}(\boldsymbol{p}) \end{bmatrix}, \tag{20}$$

for the first mn columns of its inverse. In the next step, we can insert this expression in Equation 11 and invert the linear system of equations

$$\begin{bmatrix} \nabla_{\boldsymbol{c}}\ell \\ -\nabla_{[\boldsymbol{a};\tilde{\boldsymbol{b}}]}\ell \end{bmatrix} = \boldsymbol{K}^{-1}\begin{bmatrix} -\nabla_{\boldsymbol{p}}\ell \\ \boldsymbol{0} \end{bmatrix} = -\big(\boldsymbol{K}^{-1}\big)_{:,1:mn}\nabla_{\boldsymbol{p}}\ell. \tag{21}$$

Further simplification yields the following identities for the gradients of $\boldsymbol{C}, \boldsymbol{a}$ and $\boldsymbol{b}$

$$\begin{aligned} \begin{bmatrix} \nabla_{\boldsymbol{c}}\ell \\ \nabla_{[\boldsymbol{a};\tilde{\boldsymbol{b}}]}\ell \end{bmatrix} &= \begin{bmatrix} -\lambda^{-1}\operatorname{diag}(\boldsymbol{p})\big(\boldsymbol{I}_{mn} - \tilde{\boldsymbol{E}}\big(\tilde{\boldsymbol{E}}^\top \operatorname{diag}(\boldsymbol{p})\tilde{\boldsymbol{E}}\big)^{-1}\tilde{\boldsymbol{E}}^\top \operatorname{diag}(\boldsymbol{p})\big)\nabla_{\boldsymbol{p}}\ell \\ \big(\tilde{\boldsymbol{E}}^\top \operatorname{diag}(\boldsymbol{p})\tilde{\boldsymbol{E}}\big)^{-1}\tilde{\boldsymbol{E}}^\top \operatorname{diag}(\boldsymbol{p})\nabla_{\boldsymbol{p}}\ell \end{bmatrix} \\ &= \begin{bmatrix} -\lambda^{-1}\big(\operatorname{diag}(\boldsymbol{p})\nabla_{\boldsymbol{p}}\ell - \operatorname{diag}(\boldsymbol{p})\tilde{\boldsymbol{E}}\nabla_{[\boldsymbol{a};\tilde{\boldsymbol{b}}]}\ell\big) \\ \big(\tilde{\boldsymbol{E}}^\top \operatorname{diag}(\boldsymbol{p})\tilde{\boldsymbol{E}}\big)^{-1}\tilde{\boldsymbol{E}}^\top \operatorname{diag}(\boldsymbol{p})\nabla_{\boldsymbol{p}}\ell \end{bmatrix}, \end{aligned} \tag{22}$$

where the latter equality results from substituting the obtained expression for $\nabla_{[\boldsymbol{a};\tilde{\boldsymbol{b}}]}\ell$ in the first block row. In the remainder of this proof, we can show line by line that these expressions yield Algorithm 1. The main idea is to first compute the second block row identity in Equation 22 and then use the result to eventually obtain $\nabla_{\boldsymbol{c}}\ell$ from the first block row:

l. 1 The first line defines the matrix $\boldsymbol{T} := \boldsymbol{P} \odot \nabla_{\boldsymbol{P}}\ell$ via the Hadamard product $\odot$. In vectorized form it corresponds to the expression

$$\boldsymbol{t} = \operatorname{diag}(\boldsymbol{p})\nabla_{\boldsymbol{p}}\ell. \tag{23}$$

l. 2 As detailed in Lemma 2, we remove the last equality condition from $\boldsymbol{E}$ to obtain $\tilde{\boldsymbol{E}}$. Equivalent considerations require us to introduce the truncated versions $\tilde{\boldsymbol{T}}, \tilde{\boldsymbol{P}}$ of $\boldsymbol{T}, \boldsymbol{P}$.

l. 3 The operator $\boldsymbol{E}^\top$ then maps $\boldsymbol{t}$ to the vector

$$\boldsymbol{E}^\top\boldsymbol{t} = \begin{bmatrix}\mathbb{1}_n \otimes \boldsymbol{I}_m & \boldsymbol{I}_n \otimes \mathbb{1}_m\end{bmatrix}^\top \boldsymbol{t} = \begin{bmatrix}(\mathbb{1}_n^\top \otimes \boldsymbol{I}_m)\boldsymbol{t} \\ (\boldsymbol{I}_n \otimes \mathbb{1}_m^\top)\boldsymbol{t}\end{bmatrix} = \begin{bmatrix}\boldsymbol{T}\mathbb{1}_n \\ \boldsymbol{T}^\top\mathbb{1}_m\end{bmatrix}, \tag{24}$$

that contains its row and column sums. In terms of the truncated $\tilde{\boldsymbol{E}}$, the last row of Equation 24 gets removed, thus

$$\tilde{\boldsymbol{E}}^\top\boldsymbol{t} = \begin{bmatrix}\boldsymbol{T}\mathbb{1}_n \\ \tilde{\boldsymbol{T}}^\top\mathbb{1}_m\end{bmatrix} = \begin{bmatrix}\boldsymbol{t}^{(a)} \\ \tilde{\boldsymbol{t}}^{(b)}\end{bmatrix}. \tag{25}$$

l. 4 A direction computation reveals that

$$\begin{aligned}\boldsymbol{E}^\top \operatorname{diag}(\boldsymbol{p})\boldsymbol{E} &= \begin{bmatrix}(\mathbb{1}_n^\top \otimes \boldsymbol{I}_m)\operatorname{diag}(\boldsymbol{p})(\mathbb{1}_n \otimes \boldsymbol{I}_m) & (\mathbb{1}_n^\top \otimes \boldsymbol{I}_m)\operatorname{diag}(\boldsymbol{p})(\boldsymbol{I}_n \otimes \mathbb{1}_m)\\ (\boldsymbol{I}_n \otimes \mathbb{1}_m^\top)\operatorname{diag}(\boldsymbol{p})(\mathbb{1}_n \otimes \boldsymbol{I}_m) & (\boldsymbol{I}_n \otimes \mathbb{1}_m^\top)\operatorname{diag}(\boldsymbol{p})(\boldsymbol{I}_n \otimes \mathbb{1}_m)\end{bmatrix}\\ &= \begin{bmatrix}\operatorname{diag}(\boldsymbol{P}\mathbb{1}_n) & \boldsymbol{P}\\ \boldsymbol{P}^\top & \operatorname{diag}(\boldsymbol{P}^\top\mathbb{1}_m)\end{bmatrix} = \begin{bmatrix}\operatorname{diag}(\boldsymbol{a}) & \boldsymbol{P}\\ \boldsymbol{P}^\top & \operatorname{diag}(\boldsymbol{b})\end{bmatrix}. \end{aligned} \tag{26}$$

The linear system in l. 4 of Algorithm 1 therefore yields the gradients $\nabla_{[\boldsymbol{a};\tilde{\boldsymbol{b}}]}\ell$ by inserting Equation 23, Equation 25 and (the reduced version of) Equation 26 into the second block row identity from Equation 22.

l. 5 We can expand $\nabla_{\tilde{\boldsymbol{b}}}\ell$ to $\nabla_{\boldsymbol{b}}\ell$ by appending zero $\nabla_{b_n}\ell = 0$ as the last entry. Since $\boldsymbol{b}$ is constrained to the probability simplex Δ_n this gradient is exact for all entries, see the discussion in Section 3.4.

l. 6 Having computed the gradients $\nabla_{[\boldsymbol{a};\boldsymbol{b}]}\ell$, we can now insert them in the first row of Equation 22. Here, the reduced and the original expressions are equivalent

$$\tilde{\boldsymbol{E}}\nabla_{[\boldsymbol{a};\tilde{\boldsymbol{b}}]}\ell = \boldsymbol{E}\nabla_{[\boldsymbol{a};\boldsymbol{b}]}\ell \tag{27}$$

because l. 5 specifies $\nabla_{b_n}\ell = 0$. Thus,

$$\tilde{\boldsymbol{E}}\nabla_{[\boldsymbol{a};\tilde{\boldsymbol{b}}]}\ell = \begin{bmatrix}\mathbb{1}_n \otimes \boldsymbol{I}_m & \boldsymbol{I}_n \otimes \mathbb{1}_m\end{bmatrix}\nabla_{[\boldsymbol{a};\boldsymbol{b}]}\ell = \mathbb{1}_n \otimes \nabla_{\boldsymbol{a}}\ell + \nabla_{\boldsymbol{b}}\ell \otimes \mathbb{1}_m =: \boldsymbol{u}, \tag{28}$$

defines the vectorized version of $\boldsymbol{U}$ from l. 6.

l. 7 Putting everything together, we insert the identities from Equation 23 and Equation 27 into the first block row of Equation 22

$$\nabla_{\boldsymbol{c}}\ell = -\lambda^{-1}\big(\operatorname{diag}(\boldsymbol{p})\nabla_{\boldsymbol{p}}\ell - \operatorname{diag}(\boldsymbol{p})\tilde{\boldsymbol{E}}\nabla_{[\boldsymbol{a};\tilde{\boldsymbol{b}}]}\ell\big) = -\lambda^{-1}\big(\boldsymbol{t} - \operatorname{diag}(\boldsymbol{p})\boldsymbol{u}\big), \tag{29}$$

which is equivalent to the matrix-valued expression in l. 7.

□

C.3 Proof of Theorem 5

Proof. The key for constructing the error bounds in Equation 14a and Equation 14b is finding a bound for the first-order derivatives $\frac{\partial\nabla_{\boldsymbol{c}}\ell}{\partial\boldsymbol{p}}$ and $\frac{\partial\nabla_{[\boldsymbol{a};\boldsymbol{b}]}\ell}{\partial\boldsymbol{p}}$. For brevity, we introduce the short-hand notation $\bar{\boldsymbol{P}} := \operatorname{diag}(\boldsymbol{p})$. Furthermore, we define the projection of $\boldsymbol{x}$ onto the column space of $\tilde{\boldsymbol{E}}$ as

$$\boldsymbol{\Pi}_{\boldsymbol{E}}\boldsymbol{x} := \operatorname*{arg\,min}_{\boldsymbol{y}\in\operatorname{span}(\tilde{\boldsymbol{E}})} \|\boldsymbol{x}-\boldsymbol{y}\|_{\bar{\boldsymbol{P}}}^2 = \tilde{\boldsymbol{E}}\operatorname*{arg\,min}_{\boldsymbol{z}\in\mathbb{R}^{m+n-1}} \|\boldsymbol{x}-\tilde{\boldsymbol{E}}\boldsymbol{z}\|_{\bar{\boldsymbol{P}}}^2, \tag{30}$$

where $\langle\cdot,\cdot\rangle_{\bar{\boldsymbol{P}}} := \langle\bar{\boldsymbol{P}}^{\frac{1}{2}}\cdot,\bar{\boldsymbol{P}}^{\frac{1}{2}}\cdot\rangle_2$. In matrix notation, Equation 30 reads

$$\boldsymbol{\Pi}_{\boldsymbol{E}} = \tilde{\boldsymbol{E}}\big(\tilde{\boldsymbol{E}}^\top\bar{\boldsymbol{P}}\tilde{\boldsymbol{E}}\big)^{-1}\tilde{\boldsymbol{E}}^\top\bar{\boldsymbol{P}}. \tag{31}$$

Using Equation 13a and Equation 13b from the proof of Theorem 4, we can then rewrite the backward pass compactly as

$$\nabla_{[\boldsymbol{a};\tilde{\boldsymbol{b}}]}\ell = \tilde{\boldsymbol{E}}^\dagger\boldsymbol{\Pi}_{\boldsymbol{E}}\nabla_{\boldsymbol{p}}\ell, \quad \text{and} \tag{32a}$$

$$\nabla_{\boldsymbol{c}}\ell = -\lambda^{-1}\big(\bar{\boldsymbol{P}}(\boldsymbol{I}-\boldsymbol{\Pi}_{\boldsymbol{E}})\nabla_{\boldsymbol{p}}\ell\big), \tag{32b}$$

The first identity follows from $\tilde{\boldsymbol{E}}^\dagger\tilde{\boldsymbol{E}} = \boldsymbol{I}$, since the columns of $\tilde{\boldsymbol{E}}$ are linearly independent (see part (b) of the proof of Lemma 2 in Appendix C.1). Direct substitution of Equation 13a into Equation 13b immediately yields Equation 32b. To differentiate $\nabla_{[\boldsymbol{a};\tilde{\boldsymbol{b}}]}\ell$ and $\nabla_{\boldsymbol{c}}\ell$, we apply the chain rule which in turn requires a closed-form expression for the derivative of the projection operator $\boldsymbol{\Pi}_{\boldsymbol{E}}$. Since it is defined as the solution of an optimization problem, we apply the implicit function theorem to the gradient of the objective in Equation 30, i.e.

$$\nabla_{\boldsymbol{z}}\left(\frac{1}{2}\|\boldsymbol{x}-\tilde{\boldsymbol{E}}\boldsymbol{z}\|_{\bar{\boldsymbol{P}}}^2\right) = \tilde{\boldsymbol{E}}^\top\bar{\boldsymbol{P}}\tilde{\boldsymbol{E}}\boldsymbol{z} - \tilde{\boldsymbol{E}}^\top\bar{\boldsymbol{P}}\boldsymbol{x} = \boldsymbol{0}. \tag{33}$$

The Jacobian of the mapping $\boldsymbol{p} \mapsto \boldsymbol{\Pi}_{\boldsymbol{E}}\boldsymbol{x}$ can therefore be written in terms of the IFT as

$$\frac{\partial \boldsymbol{\Pi}_{\boldsymbol{E}}\boldsymbol{x}}{\partial \boldsymbol{p}} = \tilde{\boldsymbol{E}}(\tilde{\boldsymbol{E}}^\top \bar{\boldsymbol{P}} \tilde{\boldsymbol{E}})^{-1}\tilde{\boldsymbol{E}}^\top \operatorname{diag}\big(\boldsymbol{x} - \boldsymbol{\Pi}_{\boldsymbol{E}}\boldsymbol{x}\big), \tag{34}$$

This auxiliary result implies that the Jacobians of the mappings $\boldsymbol{p} \mapsto \nabla_{[\boldsymbol{a};\tilde{\boldsymbol{b}}]}\ell$ and $\boldsymbol{p} \mapsto \nabla_{\boldsymbol{c}}\ell$ defined in Equation 32a and Equation 32b are

$$\frac{\partial \nabla_{[\boldsymbol{a};\tilde{\boldsymbol{b}}]}\ell}{\partial \boldsymbol{p}} = \tilde{\boldsymbol{E}}^\dagger \frac{\partial}{\partial \boldsymbol{p}} \boldsymbol{\Pi}_{\boldsymbol{E}} \nabla_{\boldsymbol{p}}\ell = (\tilde{\boldsymbol{E}}^\top \bar{\boldsymbol{P}} \tilde{\boldsymbol{E}})^{-1}\tilde{\boldsymbol{E}}^\top \operatorname{diag}\big((\boldsymbol{I} - \boldsymbol{\Pi}_{\boldsymbol{E}})\nabla_{\boldsymbol{p}}\ell\big) + \tilde{\boldsymbol{E}}^\dagger \boldsymbol{\Pi}_{\boldsymbol{E}} \nabla^2_{\boldsymbol{p}}\ell, \quad \text{and} \tag{35a}$$

$$\begin{aligned}\frac{\partial \nabla_{\boldsymbol{c}}\ell}{\partial \boldsymbol{p}} &= -\lambda^{-1}\bigg(\operatorname{diag}\big((\boldsymbol{I} - \boldsymbol{\Pi}_{\boldsymbol{E}})\nabla_{\boldsymbol{p}}\ell\big) - \boldsymbol{\Pi}_{\boldsymbol{E}}^\top \operatorname{diag}\big((\boldsymbol{I} - \boldsymbol{\Pi}_{\boldsymbol{E}})\nabla_{\boldsymbol{p}}\ell\big) + \bar{\boldsymbol{P}}(\boldsymbol{I} - \boldsymbol{\Pi}_{\boldsymbol{E}})\nabla^2_{\boldsymbol{p}}\ell\bigg) \\ &= -\lambda^{-1}\bigg(\big(\boldsymbol{I} - \boldsymbol{\Pi}_{\boldsymbol{E}}^\top\big) \operatorname{diag}\big((\boldsymbol{I} - \boldsymbol{\Pi}_{\boldsymbol{E}})\nabla_{\boldsymbol{p}}\ell\big) + \bar{\boldsymbol{P}}(\boldsymbol{I} - \boldsymbol{\Pi}_{\boldsymbol{E}})\nabla^2_{\boldsymbol{p}}\ell\bigg).\end{aligned} \tag{35b}$$

In order to bound the errors of these two gradients, we first derive an upper bound for the norm of the operator $\boldsymbol{\Pi}_{\boldsymbol{E}}$. An important insight is that we can precondition $\boldsymbol{\Pi}_{\boldsymbol{E}}$ via $\bar{\boldsymbol{P}}^{\frac{1}{2}}$

$$\bar{\boldsymbol{P}}^{\frac{1}{2}} \boldsymbol{\Pi}_{\boldsymbol{E}} \bar{\boldsymbol{P}}^{-\frac{1}{2}} \boldsymbol{x} = \underset{\boldsymbol{y} \in \operatorname{span}(\bar{\boldsymbol{P}}^{\frac{1}{2}}\tilde{\boldsymbol{E}})}{\arg\min} \|\boldsymbol{x} - \boldsymbol{y}\|_2^2, \tag{36}$$

which results in an orthogonal projection $\bar{\boldsymbol{P}}^{\frac{1}{2}} \boldsymbol{\Pi}_{\boldsymbol{E}} \bar{\boldsymbol{P}}^{-\frac{1}{2}}$. Since such projections have a spectral radius of at most 1, we can bound the norm of $\boldsymbol{\Pi}_{\boldsymbol{E}}$ as

$$\big\|\boldsymbol{\Pi}_{\boldsymbol{E}}\big\|_2 \leq \big\|\bar{\boldsymbol{P}}^{-\frac{1}{2}}\big\|_2 \big\|\bar{\boldsymbol{P}}^{\frac{1}{2}} \boldsymbol{\Pi}_{\boldsymbol{E}} \bar{\boldsymbol{P}}^{-\frac{1}{2}}\big\|_2 \big\|\bar{\boldsymbol{P}}^{\frac{1}{2}}\big\|_2 \leq \big\|\bar{\boldsymbol{P}}^{-\frac{1}{2}}\big\|_2 \big\|\bar{\boldsymbol{P}}^{\frac{1}{2}}\big\|_2, \tag{37}$$

and equivalently show for the complementary projector $\boldsymbol{I} - \boldsymbol{\Pi}_{\boldsymbol{E}}$ that

$$\big\|\boldsymbol{I} - \boldsymbol{\Pi}_{\boldsymbol{E}}\big\|_2 \leq \big\|\bar{\boldsymbol{P}}^{-\frac{1}{2}}\big\|_2 \big\|\bar{\boldsymbol{P}}^{\frac{1}{2}}(\boldsymbol{I} - \boldsymbol{\Pi}_{\boldsymbol{E}})\bar{\boldsymbol{P}}^{-\frac{1}{2}}\big\|_2 \big\|\bar{\boldsymbol{P}}^{\frac{1}{2}}\big\|_2 \leq \big\|\bar{\boldsymbol{P}}^{-\frac{1}{2}}\big\|_2 \big\|\bar{\boldsymbol{P}}^{\frac{1}{2}}\big\|_2. \tag{38}$$

The Jacobians of the backward pass can then be bounded as

$$\begin{aligned}\left\|\frac{\partial \nabla_{[\boldsymbol{a};\tilde{\boldsymbol{b}}]}\ell}{\partial \boldsymbol{p}}\right\|_F \leq& \big\|(\tilde{\boldsymbol{E}}^\top \bar{\boldsymbol{P}} \tilde{\boldsymbol{E}})^{-1}\tilde{\boldsymbol{E}}^\top\big\|_2 \big\|(\boldsymbol{I} - \boldsymbol{\Pi}_{\boldsymbol{E}})\nabla_{\boldsymbol{p}}\ell\big\|_2 + \big\|\tilde{\boldsymbol{E}}^\dagger \boldsymbol{\Pi}_{\boldsymbol{E}} \nabla^2_{\boldsymbol{p}}\ell\big\|_F \\ =& \big\|\tilde{\boldsymbol{E}}^\dagger \boldsymbol{\Pi}_{\boldsymbol{E}} \bar{\boldsymbol{P}}^{-1}\big\|_2 \big\|(\boldsymbol{I} - \boldsymbol{\Pi}_{\boldsymbol{E}})\nabla_{\boldsymbol{p}}\ell\big\|_2 + \big\|\tilde{\boldsymbol{E}}^\dagger \boldsymbol{\Pi}_{\boldsymbol{E}} \nabla^2_{\boldsymbol{p}}\ell\big\|_F \\ \leq& \big\|\tilde{\boldsymbol{E}}^\dagger\big\|_2 \big\|\bar{\boldsymbol{P}}^{-\frac{1}{2}}\big\|_2^2 \big\|\boldsymbol{I} - \boldsymbol{\Pi}_{\boldsymbol{E}}\big\|_2 \big\|\nabla_{\boldsymbol{p}}\ell\big\|_2 + \big\|\tilde{\boldsymbol{E}}^\dagger\big\|_2 \big\|\boldsymbol{\Pi}_{\boldsymbol{E}}\big\|_2 \big\|\nabla^2_{\boldsymbol{p}}\ell\big\|_F \\ \leq& \big\|\tilde{\boldsymbol{E}}^\dagger\big\|_2 \big\|\bar{\boldsymbol{P}}^{-\frac{1}{2}}\big\|_2 \big\|\bar{\boldsymbol{P}}^{\frac{1}{2}}\big\|_2 \bigg(\big\|\bar{\boldsymbol{P}}^{-\frac{1}{2}}\big\|_2^2 \big\|\nabla_{\boldsymbol{p}}\ell\big\|_2 + \big\|\nabla^2_{\boldsymbol{p}}\ell\big\|_F\bigg) \\ \leq& \kappa\sqrt{\frac{\sigma_+}{\sigma_-}}\bigg(\frac{1}{\sigma_-}C_1 + C_2\bigg), \quad \text{and}\end{aligned} \tag{39a}$$

$$\begin{aligned}\left\|\frac{\partial \nabla_{\boldsymbol{c}}\ell}{\partial \boldsymbol{p}}\right\|_F \leq& \lambda^{-1}\big\|\boldsymbol{I} - \boldsymbol{\Pi}_{\boldsymbol{E}}^\top\big\|_2 \big\|\boldsymbol{I} - \boldsymbol{\Pi}_{\boldsymbol{E}}\big\|_2 \big\|\nabla_{\boldsymbol{p}}\ell\big\|_2 + \lambda^{-1}\big\|\bar{\boldsymbol{P}}(\boldsymbol{I} - \boldsymbol{\Pi}_{\boldsymbol{E}})\big\|_2 \big\|\nabla^2_{\boldsymbol{p}}\ell\big\|_F \\ \leq& \lambda^{-1}\big\|\bar{\boldsymbol{P}}^{-\frac{1}{2}}\big\|_2^2 \big\|\bar{\boldsymbol{P}}^{\frac{1}{2}}\big\|_2^2 \big\|\nabla_{\boldsymbol{p}}\ell\big\|_2 + \lambda^{-1}\big\|\bar{\boldsymbol{P}}^{\frac{1}{2}}\big\|_2^2 \big\|\nabla^2_{\boldsymbol{p}}\ell\big\|_F \\ \leq& \lambda^{-1}\sigma_+\bigg(\frac{1}{\sigma_-}C_1 + C_2\bigg),\end{aligned} \tag{39b}$$

where the constants $\sigma_-, \sigma_+, C_1, C_2 > 0$ are as defined in Theorem 5, and where we use the identity

$$\big\|\boldsymbol{A}\operatorname{diag}(\boldsymbol{b})\big\|_F \leq \big\|\boldsymbol{A}\big\|_2 \big\|\operatorname{diag}(\boldsymbol{b})\big\|_F = \big\|\boldsymbol{A}\big\|_2 \big\|\boldsymbol{b}\big\|_2. \tag{40}$$

As a direct consequence, we obtain the bounds from Equation 14a and Equation 14b, since the bounded derivatives imply the Lipschitz continuity of the differentiable map $\boldsymbol{p} \mapsto \nabla_{[\boldsymbol{c};\boldsymbol{a};\boldsymbol{b}]}\ell$.

□