wig: An R Package for Wasserstein Index Generation (WIG) Model

Fangzhou Xie

Department of Economics, Rutgers University

August 6, 2024

Abstract

In this paper, I review the basics of Computation Optimal Transport, especially the entropic regularized OT problem, and the Sinkhorn algorithm

TODO:

1. edit all "Sinkhorn"

Contents

1 Introduction

intro, literature review, etc

The remainder of this paper will be organized as follows. Section 2 briefly reviews the Monge-Kantorivich problem, its entropic regularized Sinkhorn problem and solution. Section 3 introduces the Sinkhorn algorithm and its variants to solve the entropic regularized problem. Section 4 derives the gradients and Jacobians of the family of Sinkhorn algorithms. Section 5 discusses the Wasserstein Barycenter and Wasserstein Dictionary Learning problem. Section 6 shows the gradient of Wasserstein Barycenter algorithm and the parameter optimization for Wasserstein Dictionary Learning. Section 7 finally presents the Wasserstein Index Generation model for automatically time-series index generation, based on Wasserstein Dictionary Learning.

some literature review

2 Monge-Kantorovich Problem and Its Entropic Regularization

The Monge-Kantorovich problem[1] states that: given two probability vectors[2] $\mathbf{a} \in \Sigma_M$ and $\mathbf{b} \in \Sigma_N$, how to find a *coupling matrix* $\mathbf{P} \in \mathbb{R}_+^{M \times N}$ where each $\mathbf{P}_{ij}$ describes the flow of masses from bin i to bin j, such that the total cost of moving masses are optimal (minimal). The cost of moving one unit of mass from bin i to bin j is $\mathbf{C}_{ij}$ and $\mathbf{C} \in \mathbb{R}_+^{M \times N}$.

Problem 1 (Monge-Kantorovich)**.** *Let* $\mathbf{a} \in \Sigma_M$ *and* $\mathbf{b} \in \Sigma_N$*. The set of all coupling matrices is:*

$$\mathbf{U}(\mathbf{a}, \mathbf{b}) \equiv \left\{ \mathbf{P} \in \mathbb{R}_+^{M \times N} : \mathbf{P}\mathbb{1}_M = \mathbf{a} \text{ and } \mathbf{P}^\top \mathbb{1}_M = \mathbf{b} \right\}. \tag{1}$$

Given a cost matrix $\mathbf{C} \in \mathbb{R}^{M \times N}$*, the Kantorovich optimal transport problem tries to find the solution of the following*

$$L_{\mathbf{C}}(\mathbf{a}, \mathbf{b}) \equiv \min_{\mathbf{P} \in \mathbf{U}(\mathbf{a}, \mathbf{b})} \langle \mathbf{C}, \mathbf{P} \rangle = \sum_{i,j} \mathbf{C}_{ij} \mathbf{P}_{ij}. \tag{2}$$

To solve this problem in practice, one needs to resort to linear programming[3], which can be very challenging when the problem becomes large enough. Instead of finding the exact

[1] This is also called Kantorovich relaxation of the original Monge problem, as the problem was first proposed by Monge (1781) and then relaxed by Kantorovich (1942). For mathematical foundation of Optimal Transport theory, the classic references are Villani (2003) and Villani (2008); for its application in Economics, see Galichon (2016); for an introduction for applied mathematicians, see Santambrogio (2015); for the computational aspects (algorithms and their properties) and its applications in data science and machine learning, see Peyré and Cuturi (2019).

[2] Also called "histograms" in the Computational Optimal Transport community.

[3] See Chapter 3 of Peyré and Cuturi (2019) for a historical overview and related algorithms.

solution to the above problem, one can actually add an entropy regularization term to make this problem convex and hence greatly ease the computation. This was first proposed in the seminal work of Cuturi (2013) which leverages Sinkhorn-Knopp double scaling algorithm (Sinkhorn, 1964; Sinkhorn and Knopp, 1967; Knight, 2008). [4]

Problem 2 (Cuturi, 2013; Peyré and Cuturi, 2019, Entropic Regularized OT Problem)**.** *Given the coupling matrix* $\mathbf{P} \in \mathbb{R}_+^{M\times N}$*, its discrete entropy is defined as*

$$\mathbf{H}(\mathbf{P}) \equiv -\langle \mathbf{P}, \log(\mathbf{P}) - 1\rangle = -\sum_{i,j} \mathbf{P}_{ij}(\log(\mathbf{P}_{ij}) - 1).$$

With the cost matrix $\mathbf{C}$*, the entropic regularized loss function is*

$$L_{\mathbf{C}}^{\varepsilon}(\mathbf{a}, \mathbf{b}) \equiv \langle \mathbf{P}, \mathbf{C}\rangle - \varepsilon \mathbf{H}(\mathbf{P}), \tag{3}$$

and the entropic regularized OT problem is thus

$$\min_{\mathbf{P}\in\mathbf{U}(\mathbf{a},\mathbf{b})} L_{\mathbf{C}}^{\varepsilon}(\mathbf{a}, \mathbf{b}) \equiv \min_{\mathbf{P}\in\mathbf{U}(\mathbf{a},\mathbf{b})} \langle \mathbf{P}, \mathbf{C}\rangle - \varepsilon \mathbf{H}(\mathbf{P}). \tag{4}$$

The Sinkhorn problem has a unique optimal solution, since it is ε-strongly convex (Peyré and Cuturi, 2019).

Proposition 3 (Peyré and Cuturi, 2019, Proposition 4.3)**.** *The solution to Equation* (4) *is unique and has the form*

$$\mathbf{P}_{ij} = \mathbf{u}_i \mathbf{K}_{ij} \mathbf{v}_j, \tag{5}$$

or in matrix notation

$$\mathbf{P} = \operatorname{diag}(\mathbf{u})\, \mathbf{K}\, \operatorname{diag}(\mathbf{v}) \tag{6}$$

where $i \in \{1, \dots, M\}$ *and* $j \in \{1, \dots, N\}$ *with two (unknown) scaling variables* $\mathbf{u} \in \mathbb{R}^M$ *and* $\mathbf{v} \in \mathbb{R}^N$*.*

Proof. Let $\mathbf{f} \in \mathbb{R}^M$ and $\mathbf{g} \in \mathbb{R}^N$ be two dual variables for the two constraints $\mathbf{P}\mathbb{1}_N = \mathbf{a}$ and $\mathbf{P}^\top \mathbb{1}_M = \mathbf{b}$, respectively. The Lagrangian of Equation (4) becomes

$$\mathscr{L}(\mathbf{P}, \mathbf{f}, \mathbf{g}) = \langle \mathbf{P}, \mathbf{C}\rangle - \varepsilon \mathbf{H}(\mathbf{P}) - \langle \mathbf{f}, \mathbf{P}\mathbb{1}_m - \mathbf{a}\rangle - \langle \mathbf{g}, \mathbf{P}^\top \mathbb{1}_n - \mathbf{b}\rangle. \tag{7}$$

The first-order-condition gives us

$$\frac{\partial \mathscr{L}(\mathbf{P}, \mathbf{f}, \mathbf{g})}{\partial \mathbf{P}_{ij}} = \mathbf{C}_{ij} + \varepsilon \log(\mathbf{P}_{ij}) - \mathbf{f}_i - \mathbf{g}_j = 0$$

[4] Therefore, in Optimal Transport literature, people usually refer to the numeric algorithm to solve the entropic regularized Kantorovich problem as the "Sinkhorn algorithm". Hence, its extensions are also named as "X-khorn algorithms", for example "Greenkhorn algorithm" (Altschuler, Weed, and Rigollet, 2017) or "Screenkhorn" (Alaya et al., 2019). Because of its simplicity, it has been re-discovered multiple times in history and renamed accordingly, for example, it can be linked to Schödinger bridge problem (Schrödinger, 1931) or the RAS method (Bacharach, 1970). See Knight (2008), Léonard (2013), and Modin (2024) for historical accounts.

Therefore we can solve for the optimal coupling matrix $\mathbf{P}$ as:

$$\log(\mathbf{P}_{ij}) = \frac{\mathbf{f}_i}{\varepsilon}\left(-\frac{\mathbf{C}_{ij}}{\varepsilon}\right)\frac{\mathbf{g}_j}{\varepsilon}$$

or

$$\mathbf{P}_{ij} = \exp(\frac{\mathbf{f}_i}{\varepsilon})\exp(-\frac{\mathbf{C}_{ij}}{\varepsilon})\exp(\frac{\mathbf{g}_j}{\varepsilon}) = \mathbf{u}_i\mathbf{C}_{ij}\mathbf{v}_j, \tag{8}$$

where $\mathbf{u}_i = \exp(\frac{\mathbf{f}_i}{\varepsilon})$, $\mathbf{v}_j = \exp(\frac{\mathbf{g}_j}{\varepsilon})$, and $\mathbf{K}_{ij} = \exp(-\frac{\mathbf{C}_{ij}}{\varepsilon})$, respectively. □

Remark. *Note that computation of $\mathbf{P}_{ij}$ are effectively in the exponential domain, if we choose to find optimal solution for $\mathbf{P}$ by updating $\mathbf{f}$ and $\mathbf{g}$. This is indeed the case for the vanilla Sinkhorn Algorithm, and hence it has numeric instability issue which need to be overcome by the "log-stabilization" technique, which moves all computation in the log-domain (Section 3.3).*

3 Sinkhorn Algorithms

In this section, I will discuss three different versions of Sinkhorn algorithms.

3.1 Vanilla Sinkhorn Algorithm

Given the optimal solution to the Sinkhorn problem in Equation (6), with the constraints in Equation (1), we have

$$\operatorname{diag}(\mathbf{u})\,\mathbf{K}\,\operatorname{diag}(\mathbf{v})\mathbb{1}_N = \mathbf{a} \quad \text{and} \quad \operatorname{diag}(\mathbf{v})\,\mathbf{K}^\top\,\operatorname{diag}(\mathbf{u})\mathbb{1}_M = \mathbf{b}. \tag{9}$$

Since we also have $\operatorname{diag}(\mathbf{v})\mathbb{1}_N = \mathbf{v}$ and $\operatorname{diag}(\mathbf{u})\mathbb{1}_M = \mathbf{u}$, we have that

$$\mathbf{u} \odot (\mathbf{K}\mathbf{v}) = \mathbf{a} \quad \text{and} \quad \mathbf{v} \odot (\mathbf{K}^\top\mathbf{u}) = \mathbf{b}. \tag{10}$$

Equivalently, we can also write the solution as

$$\mathbf{u} = \mathbf{a} \oslash (\mathbf{K}\mathbf{v}) \quad \text{and} \quad \mathbf{v} = \mathbf{b} \oslash (\mathbf{K}^\top\mathbf{u}). \tag{11}$$

In the above equations, $\odot$ and $\oslash$ refer to element-wise multiplication and division, respectively. Therefore, we can leverage Equation (11) to update $\mathbf{u}$ and $\mathbf{v}$ till convergence and conclude with optimal coupling $\mathbf{P}$ by Equation (6).

Update 3.1 (Vanilla Sinkhorn Updates)**.** *At any iteration $\ell = 1, \ldots, L$ we can update the scaling variables $\mathbf{u}$ and $\mathbf{v}$ by the following equations:*

$$\mathbf{u}^{(\ell)} \equiv \mathbf{a} \oslash (\mathbf{K}\mathbf{v}^{(\ell-1)}) \quad \textit{and} \quad \mathbf{v}^{(\ell)} \equiv \mathbf{b} \oslash (\mathbf{K}^\top\mathbf{u}^{(\ell)}). \tag{12}$$

Remark. *In Update 3.1, the choice of* $\mathbf{v}^{(0)}$ *is rather arbitrary, as long as it is not* $\mathbb{0}$*. The most common choice is to consider the vectors of ones, i.e.* $\mathbb{1}_N$*. Changing the initialization vector will result in different solutions to* $\mathbf{u}$ *and* $\mathbf{v}$*, since they are only defined up to a multiplicative constant. But* $\mathbf{u}$ *and* $\mathbf{v}$ *will converge regardless of the initialization, and result in the same optimal coupling matrix* $\mathbf{P}$ *upon convergence. See Peyré and Cuturi (2019, p.82) for discussion.*

After L iterations, the scaling variables $\mathbf{u}^{(L)}$ and $\mathbf{v}^{(L)}$ can be used to compute the optimal coupling matrix $\mathbf{P} = \operatorname{diag} \mathbf{u}^{(L)} \mathbf{K} \operatorname{diag} \mathbf{v}^{(L)}$.

From Update 3.1, we can arrive at the vanilla Sinkhorn algorithm (Algorithm 3.1).

Algorithm 3.1 Vanilla Sinkhorn

Input: $\mathbf{a} \in \Sigma_M$, $\mathbf{b} \in \Sigma_N$, $\mathbf{C} \in \mathbb{R}^{M \times N}$, $\varepsilon > 0$.
Initialize: $\mathbf{u} = \mathbb{1}_M$, $\mathbf{v} = \mathbb{1}_N$.
1: $\mathbf{K} = \exp(-\frac{\mathbf{C}}{\varepsilon})$
2: **while** Not Convergence **do**
3: $\quad \mathbf{u} \leftarrow \mathbf{a} \oslash (\mathbf{K}\mathbf{v})$
4: $\quad \mathbf{v} \leftarrow \mathbf{b} \oslash (\mathbf{K}^\top \mathbf{u})$
5: **end while**
6: $\mathbf{P} = \operatorname{diag}(\mathbf{u})\, \mathbf{K}\, \operatorname{diag}(\mathbf{v})$
Output: Optimal Coupling $\mathbf{P}$

Remark (Termination/Convergence Condition)**.** *Since we have Equation* (10)*, we can use those equations to check if the scaling variables* $\mathbf{u}$ *and* $\mathbf{v}$ *at the current iteration could approximate* $\mathbf{a}$ *and* $\mathbf{b}$ *well. That is, for some small* $\rho > 0$*,*

$$\|\mathbf{u}^{(\ell)} \odot \left(\mathbf{K}\mathbf{v}^{(\ell)}\right) - \mathbf{a}\|_2 \leq \rho \quad \textit{and} \quad \|\mathbf{v}^{(\ell)} \odot \left(\mathbf{K}^\top \mathbf{u}^{(\ell)}\right) - \mathbf{b}\|_2 \leq \rho$$

If the norms are smaller than some threshold ρ*, we can then terminate the program.*

3.2 Parallel Sinkhorn Algorithm

Note that in the Vanilla Sinkhorn Algorithm, the computation are carried out by vector-vector element-wise computation, i.e., $\mathbf{u}, \mathbf{a}$, and $\mathbf{K}\mathbf{v} \in \mathbb{R}^M$ and $\mathbf{v}, \mathbf{b}$, and $\mathbf{K}^\top \mathbf{u} \in \mathbb{R}^N$. That is, if we have a single source $\mathbf{a}$ and a single target $\mathbf{b}$, we are computing the "one-to-one" transportation. However, sometimes we need to calculate one-to-many, many-to-one, or many-to-many transport and this can be easily computed in a parallelized fashion. All we need to do is to define the matrices $\mathbf{A}$ and $\mathbf{B}$ instead of the vectors $\mathbf{a}$ and $\mathbf{b}$, and all that remains is to carry out the matrix computation instead of the vector ones.

Definition 4. *For some* $S \in \mathbb{N}_+$*:*

- *Suppose we have* $\mathbf{a} \in \Sigma_M$ *and* $\mathbf{b}_1, \ldots, \mathbf{b}_S \in \Sigma_N$*. Then we can define*

$$\mathbf{A} = [\mathbf{a}, \ldots, \mathbf{a}] \in \mathbb{R}^{M \times S} \quad \textit{and} \quad \mathbf{B} = [\mathbf{b}_1, \ldots, \mathbf{b}_S] \in \mathbb{R}^{N \times S}.$$

- *Suppose we have* $\mathbf{a}_1, \ldots, \mathbf{a}_S \in \Sigma_M$ *and* $\mathbf{b} \in \Sigma_N$. *Then we can define*

$$\mathbf{A} = [\mathbf{a}_1, \ldots, \mathbf{a}_S] \in \mathbb{R}^{M \times S} \quad \textit{and} \quad \mathbf{B} = [\mathbf{b}, \ldots, \mathbf{b}] \in \mathbb{R}^{N \times S}.$$

- *Suppose we have* $\mathbf{a}_1, \ldots, \mathbf{a}_S \in \Sigma_M$ *and* $\mathbf{b}_1, \ldots, \mathbf{b}_S \in \Sigma_N$. *Then we can define*

$$\mathbf{A} = [\mathbf{a}_1, \ldots, \mathbf{a}_S] \in \mathbb{R}^{M \times S} \quad \textit{and} \quad \mathbf{B} = [\mathbf{b}_1, \ldots, \mathbf{b}_S] \in \mathbb{R}^{N \times S}.$$

Remark. *In the many-to-many case, the number of columns in* $\mathbf{A}$ *should be equal to that of* $\mathbf{B}$*; whereas in one-to-many or many-to-one cases, we need to duplicate the single vector* S *times, as if we had a* S*-column matrix to begin with. In many modern linear algebra packages, this can be achieved automatically by "broadcasting,"*[5] *without the need of actually doing the duplication of the column vectors.*

Therefore, we can have the matrix version[6] of Equation (11) with scaling variable $\mathbf{U} \in \mathbb{R}^{M \times S}$ and $\mathbf{V} \in \mathbb{R}^{N \times S}$,

$$\mathbf{U} = \mathbf{A} \oslash (\mathbf{K}\mathbf{V}) \quad \text{and} \quad \mathbf{V} = \mathbf{B} \oslash (\mathbf{K}^\top \mathbf{V}). \tag{13}$$

Hence we could have the matrix version of the updating equations of Sinkhorn algorithm.

Update 3.2. *At any iteration* $\ell = 1, \ldots, L$ *and initialized* $\mathbf{V}^{(0)} \in \mathbb{R}^{N \times S}$, *we have*

$$\mathbf{U}^{(\ell)} \equiv \mathbf{A} \oslash (\mathbf{K}\mathbf{V}^{(\ell-1)}) \quad \textit{and} \quad \mathbf{V}^{(\ell)} \equiv \mathbf{B} \oslash (\mathbf{K}^\top \mathbf{U}^{(\ell)}). \tag{14}$$

Remark. *Since Update 3.2 only involves matrix operations, the Sinkhorn algorithm are intrinsically GPU friendly, as matrix operations are even more efficient on GPU than on CPU.*

Algorithm 3.2 Parallel Sinkhorn Algorithm

Input: $\mathbf{A} \in \Sigma_{M \times S}$, $\mathbf{B} \in \Sigma_{N \times S}$, $\mathbf{C} \in \mathbb{R}^{M \times N}$, $\varepsilon > 0$.
Initialize: $\mathbf{U} = \mathbb{1}_{M \times S}$, $\mathbf{V} = \mathbb{1}_{N \times S}$.
1: $\mathbf{K} = \exp(-\frac{\mathbf{C}}{\varepsilon})$
2: **while** Not Convergence **do**
3: $\quad \mathbf{U} \leftarrow \mathbf{A} \oslash (\mathbf{K}\mathbf{V})$
4: $\quad \mathbf{V} \leftarrow \mathbf{B} \oslash (\mathbf{K}^\top \mathbf{U})$
5: **end while**
6: $\forall s, \mathbf{P}_s = diag(\mathbf{U}_s)\, \mathbf{K}\, diag(\mathbf{V}_s)$
Output: Optimal Coupling $\mathbf{P}_s, \forall s$

[5] For example, Eigen (C++): `https://eigen.tuxfamily.org/dox/group__TutorialReductionsVisitorsBroadcasting.html`, NumPy (Python): `https://numpy.org/doc/stable/user/basics.broadcasting.html`.

[6] Instead of computing the vector-vector computation in Equation (11) S times, we are computing a matrix-matrix product/division once. They are mathematically equivalent, but in practice, the latter would be much faster due to the implementation of linear algebra routines (e.g. BLAS). Hence, this simple extension is considered parallel since we process S columns of data in one batch.

Remark (Convergence Condition)**.** *Similar to the termination condition in Section 3.1, we now have the matrix norms to check convergence. For some $\rho > 0$,*

$$\|\mathbf{U}^{(\ell)} \odot \left(\mathbf{K}\mathbf{V}^{(\ell)}\right) - \mathbf{A}\|_2 \leq \rho, \quad \textit{and} \quad \|\mathbf{V}^{(\ell)} \odot \left(\mathbf{K}^\top \mathbf{U}^{(\ell)}\right) - \mathbf{B}\|_2.$$

3.3 Log-stabilized Sinkhorn

As can be seen in Equations (8) and (11), the updates on $\mathbf{u}$ and $\mathbf{v}$ are in the exponential domain. Therefore, if either $\mathbf{C}_{ij}$ is large or ε is small, the corresponding $\mathbf{K}_{ij}$ would be very large, and the subsequent updates will fail because of the Inf/NaN values introduced by the numeric overflow. This is commonly known as the numeric instability issue of the Sinkhorn algorithm (Peyré and Cuturi, 2019, Chapter 4.4).

To circumvent this instability issue, we could move the computation into the log-domain, therefore without the need for computation of the exponential function. To this end, instead of updating the $\mathbf{u}$ and $\mathbf{v}$ as in Update 3.1, we can consider the dual variables $\mathbf{f}$ and $\mathbf{g}$ in the Lagrangian in Equation (7). Since $\mathbf{u} = \exp\left(\frac{\mathbf{f}}{\varepsilon}\right)$ and $\mathbf{v} = \exp\left(\frac{\mathbf{g}}{\varepsilon}\right)$, we have

$$\mathbf{f} = \varepsilon \log(\mathbf{u}) \quad \text{and} \quad \mathbf{g} = \varepsilon \log(\mathbf{v}). \tag{15}$$

Then, we can also rewrite the elements in the coupling matrix $\mathbf{P}$ as

$$\mathbf{P}_{ij}^{(\ell)} = \mathbf{u}_i^{(\ell)} \mathbf{K}_{ij} \mathbf{v}_j^{(\ell)} = \exp\left(-\frac{\mathbf{C}_{ij} - \mathbf{f}_i^{(\ell)} - \mathbf{g}_j^{(\ell)}}{\varepsilon}\right)$$

and the matrix form of $\mathbf{P}$ expressed in $\mathbf{f}$ and $\mathbf{g}$ is

$$\mathbf{P}^{(\ell)} = \exp\left(-\frac{\mathbf{C} - \mathbf{f}^{(\ell)} \cdot \mathbb{1}_N^\top - \mathbb{1}_M \cdot \mathbf{g}^{(\ell)\top}}{\varepsilon}\right). \tag{16}$$

Let us also denote a function $\mathcal{P}(\mathbf{f}, \mathbf{g})$ with $\mathbf{f} \in \mathbb{R}^M$ and $\mathbf{g} \in \mathbb{R}^N$,

$$\mathcal{P}(\mathbf{f}, \mathbf{g}) = \exp\left(-\frac{\mathbf{C} - \mathbf{f} \cdot \mathbb{1}_N^\top - \mathbb{1}_M \cdot \mathbf{g}^\top}{\varepsilon}\right), \tag{17}$$

and rewrite $\mathbf{P}^{(\ell)} = \mathcal{P}\left(\mathbf{f}^{(\ell)}, \mathbf{g}^{(\ell)}\right)$. Note that $\mathbf{f}$ and $\mathbf{g}$ are dual variables in the Lagrangian in Equation (7). Then we can plug in $\mathbf{u}$ and $\mathbf{v}$ to get the following updating equations

$$\begin{aligned} \mathbf{f}^{(\ell)} &= \varepsilon \log \mathbf{u}^{(\ell)} = \varepsilon \log \mathbf{a} - \varepsilon \log(\mathbf{K}\mathbf{v}^{(\ell-1)}), \\ \mathbf{g}^{(\ell)} &= \varepsilon \log \mathbf{v}^{(\ell)} = \varepsilon \log \mathbf{b} = \varepsilon \log(\mathbf{K}^\top \mathbf{u}^{(\ell)}) \end{aligned} \tag{18}$$

Since $\mathbf{K}\mathbf{v}^{(\ell-1)} \in \mathbb{R}^m$, $\forall i$, the i-th element is

$$\begin{aligned}\left(\mathbf{K}\mathbf{v}^{(\ell-1)}\right)_i &= \sum_j \mathbf{K}_{ij}\mathbf{v}_j^{(\ell-1)} \\ &= \sum_j \exp(-\frac{\mathbf{C}_{ij}}{\varepsilon})\exp(\frac{\mathbf{g}_j^{(\ell-1)}}{\varepsilon}) \\ &= \left[\sum_j \exp(-\frac{\mathbf{C}_{ij}-\mathbf{f}_i^{(\ell-1)}-\mathbf{g}_j^{(\ell-1)}}{\varepsilon})\right]\exp(-\frac{\mathbf{f}_i^{(\ell-1)}}{\varepsilon}),\end{aligned}$$

and

$$\begin{aligned}-\varepsilon\log\left(\mathbf{K}\mathbf{v}^{(\ell-1)}\right)_i &= -\varepsilon\log\sum_j \exp(-\frac{\mathbf{C}_{ij}-\mathbf{f}_i^{(\ell-1)}-\mathbf{g}_j^{(\ell-1)}}{\varepsilon}) + \mathbf{f}_i^{(\ell-1)} \\ &= -\varepsilon\log\left[\exp\left(-\frac{\mathbf{C}-\mathbf{f}^{(\ell-1)}\mathbb{1}_N^\top - \mathbb{1}_M(\mathbf{g}^{(\ell-1)})^\top}{\varepsilon}\right)\cdot\mathbb{1}_N\right]_i + \mathbf{f}_i^{(\ell-1)} \\ &= -\varepsilon\log\left[\mathcal{P}\left(\mathbf{f}^{(\ell-1)},\mathbf{g}^{(\ell-1)}\right)\cdot\mathbb{1}_N\right]_i + \mathbf{f}_i^{(\ell-1)}.\end{aligned}$$

Therefore, we have the updating equation for $\mathbf{f}^{(\ell)}$. If we follow similar steps, we can also get it for $\mathbf{g}^{(\ell)}$. That is,

$$\begin{aligned}\mathbf{f}^{(\ell)} &= \mathbf{f}^{(\ell-1)} + \varepsilon\log\mathbf{a} - \varepsilon\log\left[\mathcal{P}\left(\mathbf{f}^{(\ell-1)},\mathbf{g}^{(\ell-1)}\right)\cdot\mathbb{1}_N\right], \\ \mathbf{g}^{(\ell)} &= \mathbf{g}^{(\ell-1)} + \varepsilon\log\mathbf{b} - \varepsilon\log\left[\mathcal{P}^\top\left(\mathbf{f}^{(\ell)},\mathbf{g}^{(\ell-1)}\right)\cdot\mathbb{1}_M\right].\end{aligned} \tag{19}$$

Note that $\log\left[\mathcal{P}(\mathbf{f},\mathbf{g})\cdot\mathbb{1}_N\right]$ is essentially a "log-sum-exp" form, and we can apply the "log-sum-exp" trick[7] to make this equation numeric stable while computing it in practice. Therefore, let us define the following two functions,

$$\begin{aligned}c(\mathbf{f},\mathbf{g}) &= \min\left\{\mathbf{C}-\mathbf{f}\cdot\mathbb{1}_N^\top - \mathbb{1}_M\cdot\mathbf{g}^\top\right\}, \\ \mathcal{Q}(\mathbf{f},\mathbf{g}) &= \exp\left(-\frac{\mathbf{C}-\mathbf{f}\cdot\mathbb{1}_N^\top-\mathbb{1}_M\cdot\mathbf{g}^\top - c(\mathbf{f},\mathbf{g})}{\varepsilon}\right),\end{aligned} \tag{20}$$

we can again rewrite Equation (19) as

$$\begin{aligned}\mathbf{f}^{(\ell)} &= \mathbf{f}^{(\ell-1)} + \varepsilon\log\mathbf{a} - \varepsilon\log\left[\mathcal{P}\left(\mathbf{f}^{(\ell-1)},\mathbf{g}^{(\ell-1)}\right)\cdot\mathbb{1}_N\right], \\ &= \mathbf{f}^{(\ell-1)} + \varepsilon\log\mathbf{a} - \varepsilon\log\left[\exp\left(-\frac{\mathbf{C}-\mathbf{f}^{(\ell-1)}\cdot\mathbb{1}_N^\top-\mathbb{1}_M\cdot\mathbf{g}^{(\ell-1)\top}}{\varepsilon}\right)\cdot\mathbb{1}_N\right] \\ &= \mathbf{f}^{(\ell-1)} + \varepsilon\log\mathbf{a} - \varepsilon\log\left[\exp\left(-\frac{c\left(\mathbf{f}^{(\ell-1)},\mathbf{g}^{(\ell-1)}\right)}{\varepsilon}\right)\mathcal{Q}\left(\mathbf{f}^{(\ell-1)},\mathbf{g}^{(\ell-1)}\right)\cdot\mathbb{1}_N\right] \\ &= \mathbf{f}^{(\ell-1)} + \varepsilon\log\mathbf{a} + c\left(\mathbf{f}^{(\ell-1)},\mathbf{g}^{(\ell-1)}\right) - \varepsilon\log\left[\mathcal{Q}\left(\mathbf{f}^{(\ell-1)},\mathbf{g}^{(\ell-1)}\right)\cdot\mathbb{1}_N\right], \\ \mathbf{g}^{(\ell)} &= \mathbf{g}^{(\ell-1)} + \varepsilon\log\mathbf{b} + c\left(\mathbf{f}^{(\ell)},\mathbf{g}^{(\ell-1)}\right) - \varepsilon\log\left[\mathcal{Q}^\top\left(\mathbf{f}^{(\ell)},\mathbf{g}^{(\ell-1)}\right)\cdot\mathbb{1}_M\right].\end{aligned}$$

[7] https://en.wikipedia.org/wiki/LogSumExp

Thus, we have obtained the numerically stable version of the updating equations for $\mathbf{f}$ and $\mathbf{g}$.

Update 3.3. *At any iteration $\ell = 1, \ldots, L$ and initialized $\mathbf{f}^{(0)}$ and $\mathbf{g}^{(0)}$, we have*

$$\begin{aligned} \mathbf{f}^{(\ell)} &= \mathbf{f}^{(\ell-1)} + \varepsilon \log \mathbf{a} + c\left(\mathbf{f}^{(\ell-1)}, \mathbf{g}^{(\ell-1)}\right) - \varepsilon \log\left[\mathcal{Q}\left(\mathbf{f}^{(\ell-1)}, \mathbf{g}^{(\ell-1)}\right) \cdot \mathbb{1}_N\right], \\ \mathbf{g}^{(\ell)} &= \mathbf{g}^{(\ell-1)} + \varepsilon \log \mathbf{b} + c\left(\mathbf{f}^{(\ell)}, \mathbf{g}^{(\ell-1)}\right) - \varepsilon \log\left[\mathcal{Q}^\top\left(\mathbf{f}^{(\ell)}, \mathbf{g}^{(\ell-1)}\right) \cdot \mathbb{1}_M\right]. \end{aligned} \tag{21}$$

Remark. *As mentioned in Section 3.1, the initialization for $\mathbf{u}$ and $\mathbf{v}$ are vectors of ones. This is to say that, the initializations of $\mathbf{f}$ and $\mathbf{g}$ are vectors of zeros, i.e. $\mathbf{f}^{(0)} = \mathbb{0}_M$ and $\mathbf{g}^{(0)} = \mathbb{0}_N$.*

Therefore, we can have the Log-stabilized Sinkhorn algorithm.

Algorithm 3.3 Log-Stabilized Sinkhorn Algorithm

Input: $\mathbf{a} \in \Sigma_M$, $\mathbf{b} \in \Sigma_N$, $\mathbf{C} \in \mathbb{R}^{M \times N}$, $\varepsilon > 0$.
Initialize: $\mathbf{f} = \mathbb{0}_M$, $\mathbf{g} = \mathbb{0}_N$.

while Not Convergence **do**
  # update f
  $\mathbf{M} = \mathbf{C} - \mathbf{f} \cdot \mathbb{1}_N^\top - \mathbb{1}_M \cdot \mathbf{g}^\top$
  $c = \min \mathbf{M}$
  $\mathbf{Q} = \exp\left(-\frac{\mathbf{M}-c}{\varepsilon}\right)$
  $\mathbf{f} = \mathbf{f} + \varepsilon \log \mathbf{a} + c - \varepsilon \log\left[\mathbf{Q} \cdot \mathbb{1}_N\right]$
  # update g
  $\mathbf{M} = \mathbf{C} - \mathbf{f} \cdot \mathbb{1}_N^\top - \mathbb{1}_M \cdot \mathbf{g}^\top$
  $c = \min \mathbf{M}$
  $\mathbf{Q} = \exp\left(-\frac{\mathbf{M}-c}{\varepsilon}\right)$
  $\mathbf{g} = \mathbf{g} + \varepsilon \log \mathbf{b} + c - \varepsilon \log\left[\mathbf{Q}^\top \cdot \mathbb{1}_M\right]$
end while
$\mathbf{P} = \exp\left(-\frac{\mathbf{C} - \mathbf{f}\,\mathbb{1}_N^\top - \mathbb{1}_M\,\mathbf{g}^\top}{\varepsilon}\right)$

Output: $\mathbf{P}$

Remark. *The advantage of using log-stabilized Sinkhorn is to make sure the computation is stable for any arbitrary ε. The disadvantage, however, is that we cannot easily parallel its computation over several margins as in Section 3.2. Moreover, the computation of matrix $\mathbf{Q}$ involves* exp *and* log *at every iteration twice for both updating $\mathbf{f}$ and $\mathbf{g}$. Thus, the log-stabilized Sinkhorn is less efficient than Vanilla and Parallel Sinkhorn algorithms. This is the tradeoff we need to make if we need to make sure the computation is stable, especially if we want to use a very small regularization constant ε. As in Peyré and Cuturi (2019, Chapter 4.1), the Sinkhorn problem converges to original Kantorovich problem when ε is small, i.e. $L_{\mathbf{C}}^{\varepsilon}(\mathbf{a}, \mathbf{b}) \to L_{\mathbf{C}}(\mathbf{a}, \mathbf{b})$ as $\varepsilon \to 0$.*

Remark (Convergence Condition)**.** *Recall that in Equation* (17)*, we can recover the optimal coupling matrix from $\mathbf{f}$ and $\mathbf{g}$, then for some $rho > 0$,*

$$\|\mathcal{P}\left(\mathbf{f}^{(\ell)}, \mathbf{g}^{(\ell)}\right) - \mathbf{a}\|_2 \leq \rho \quad \textit{and} \quad \|\mathcal{P}^\top\left(\mathbf{f}^{(\ell)}, \mathbf{g}^{(\ell)}\right) - \mathbf{b}\|_2 \leq \rho.$$

4 Differentiation of Sinkhorn Algorithms

In this section, I will derive the gradients of Sinkhorn loss with respect to its arguments for the algorithms described in Section 3. This is useful when one wishes to use Sinkhorn loss as a loss function for training purpose in machine learning, for example, Wasserstein Dictionary Learning in Sections 5 to 6.

4.1 Vanilla Sinkhorn

The optimal coupling matrix $\mathbf{P}$ from the Vanilla Sinkhorn algorithm is

$$\mathbf{P} = \operatorname{diag}(\mathbf{u}^{(L)})\mathbf{K}\operatorname{diag}(\mathbf{v}^{(L)}), \tag{22}$$

and the loss function evaluated at the solution is $L^{\varepsilon}_{\mathbf{C}}(\mathbf{P}) = \langle \mathbf{P}, \mathbf{C}\rangle + \varepsilon\langle \mathbf{P}, \log \mathbf{P} - 1\rangle$.

Therefore, by Chain's rule, the Jacobian of Sinkhorn loss w.r.t. first argument $\mathbf{a}$ is

$$DL^{\varepsilon}_{\mathbf{C}}(\mathbf{a}) = DL^{\varepsilon}_{\mathbf{C}}(\mathbf{P}) \cdot D\mathbf{P}(\mathbf{a}). \tag{23}$$

Note that $\langle \mathbf{A}, \mathbf{B}\rangle = \mathbb{1}_M^\top (\mathbf{A} \odot \mathbf{B})\, \mathbb{1}_N$, for matrices $\mathbf{A}$ and $\mathbf{B}$ in $\mathbb{R}^{M\times N}$. To get the first term $DL^{\varepsilon}_{\mathbf{C}}(\mathbf{P})$, we need the differential as the following

$$\begin{aligned}
d\operatorname{vec} L^{\varepsilon}_{\mathbf{C}}(\mathbf{P}) &= d\operatorname{vec}\left[\mathbb{1}_M^\top (\mathbf{P}\odot\mathbf{C})\,\mathbb{1}_N\right] + \varepsilon d\operatorname{vec}\left\{\mathbb{1}_M^\top\left[\mathbf{P}\odot[\log\mathbf{P}-1]\right]\mathbb{1}_N\right\} \\
&= d\left[(\mathbb{1}_N\otimes\mathbb{1}_M)^\top(\operatorname{vec}\mathbf{C}\otimes\operatorname{vec}\mathbf{P})\right] + \varepsilon d\left\{(\mathbb{1}_N\otimes\mathbb{1}_M)^\top\left[\operatorname{vec}\mathbf{P}\otimes\operatorname{vec}(\log\mathbf{P}-1)\right]\right\} \\
&= \mathbb{1}_{MN}^\top(\operatorname{vec}\mathbf{C}\otimes d\operatorname{vec}\mathbf{P}) + \varepsilon\mathbb{1}_{MN}^\top\cdot d\left[\operatorname{vec}\mathbf{P}\otimes\operatorname{vec}(\log\mathbf{P}-1)\right] \\
&= \mathbb{1}_{MN}^\top\cdot\operatorname{diag}\operatorname{vec}\mathbf{C}\cdot d\operatorname{vec}\mathbf{P} + \varepsilon\mathbb{1}_{MN}^\top\cdot\operatorname{diag}\operatorname{vec}\log\mathbf{P}\cdot d\operatorname{vec}\mathbf{P} \\
&= \mathbb{1}_{MN}^\top\cdot\operatorname{diag}\operatorname{vec}(\mathbf{C}+\varepsilon\log\mathbf{P})\cdot d\operatorname{vec}\mathbf{P}.
\end{aligned}$$

and the Jacobian is

$$DL^{\varepsilon}_{\mathbf{C}}(\mathbf{P}) = \mathbb{1}_{MN}^\top\operatorname{diag}\operatorname{vec}(\mathbf{C}+\varepsilon\log\mathbf{P}). \tag{24}$$

Note that this Jacobian matrix has only one row, since $(\mathbb{1}_N\otimes\mathbb{1}_M) = \mathbb{1}_{MN}$ has only one column. Thus, the gradient[8] $\nabla_{\mathbf{a}} L^{\varepsilon}_{\mathbf{C}}$, would be column vector of size M.

To get the second term, i.e. $D\mathbf{P}(\mathbf{a})$, we also need the differential of $\mathbf{P}$ in Equation (16) instead of Equation (22), as Equation (16) involves the dual variables $\mathbf{f}$ and $\mathbf{g}$, which will be useful when we derive the Jacobian matrices for the log-stabilized Sinkhorn algorithm in Section 3.3.

The differential is

[8] $\nabla_{\mathbf{a}} L^{\varepsilon}_{\mathbf{C}} = (DL^{\varepsilon}_{\mathbf{C}}(\mathbf{a}))^\top$.

$$
\begin{aligned}
d\operatorname{vec}\mathbf{P} &= -\frac{1}{\varepsilon}\operatorname{diag}\operatorname{vec}\mathbf{P}\, d\operatorname{vec}\left(\mathbf{C} - \mathbf{f}^{(L)}\mathbb{1}_N^\top - \mathbb{1}_M\left(\mathbf{g}^{(L)}\right)^\top\right) \\
&= \frac{1}{\varepsilon}\operatorname{diag}\operatorname{vec}\mathbf{P}\left[d\operatorname{vec}\left(\mathbf{f}^{(L)}\mathbb{1}_N^\top\right) + d\operatorname{vec}\left(\mathbb{1}_m\left(\mathbf{g}^{(L)}\right)^\top\right)\right] \\
&= \frac{1}{\varepsilon}\operatorname{diag}\operatorname{vec}\mathbf{P}\left[(\mathbb{1}_N \otimes I_M)\, d\mathbf{f}^{(L)} + (I_N \otimes \mathbb{1}_M)\, d\mathbf{g}^{(L)}\right] \\
&= \operatorname{diag}\operatorname{vec}\mathbf{P}\left[(\mathbb{1}_N \otimes I_M)\cdot\operatorname{diag}\frac{1}{\mathbf{u}^{(L)}}\cdot d\mathbf{u}^{(L)} + (I_N \otimes \mathbb{1}_M)\cdot\operatorname{diag}\frac{1}{\mathbf{v}^{(L)}}\cdot d\mathbf{v}^{(L)}\right],
\end{aligned}
\tag{25}
$$

since we also have $\mathbf{f} = \varepsilon\log\mathbf{u}$ and $\mathbf{g} = \varepsilon\log\mathbf{v}$, $d\mathbf{f} = \varepsilon\operatorname{diag}\frac{1}{\mathbf{u}}\cdot d\mathbf{u}$ and $d\mathbf{g} = \varepsilon\operatorname{diag}\frac{1}{\mathbf{v}}\cdot d\mathbf{v}$, respectively.

Therefore,

$$
D\mathbf{P}(\mathbf{a}) = \operatorname{diag}\operatorname{vec}\mathbf{P}\left[(\mathbb{1}_N \otimes I_M)\cdot\operatorname{diag}\frac{1}{\mathbf{u}^{(L)}}\cdot D\mathbf{u}^{(L)}(\mathbf{a}) + (I_N \otimes \mathbb{1}_M)\cdot\operatorname{diag}\frac{1}{\mathbf{v}^{(L)}}\cdot D\mathbf{v}^{(L)}(\mathbf{a})\right].
\tag{26}
$$

From Equation (26), all that remains to be shown are the formulae of $D\mathbf{u}^{(L)}(\mathbf{a})$ and $D\mathbf{v}^{(L)}(\mathbf{a})$, which can be obtained iteratively differentiating Update 3.1. Again, we need first to calculate the differentials

$$
\begin{aligned}
d\,\mathbf{u}^{(\ell)} &= d\left[\mathbf{a}\oslash\left(\mathbf{K}\mathbf{v}^{(\ell-1)}\right)\right] = d\left(\mathbf{a}\odot\frac{1}{\mathbf{K}\mathbf{v}^{(\ell-1)}}\right) \\
&= d\,\mathbf{a}\odot\frac{1}{\mathbf{K}\mathbf{v}^{(\ell-1)}} - \mathbf{a}\odot d\left(\frac{1}{\mathbf{K}\mathbf{v}^{(\ell-1)}}\right) \\
&= \operatorname{diag}(\frac{1}{\mathbf{K}\mathbf{v}^{(\ell-1)}})\cdot d\,\mathbf{a} - \operatorname{diag}\frac{\mathbf{a}}{\left(\mathbf{K}\mathbf{v}^{(\ell-1)}\right)^2}\cdot\mathbf{K}\cdot d\,\mathbf{v}^{(l-1)} \\
d\,\mathbf{v}^{(\ell)} &= d\left(\mathbf{b}\odot\frac{1}{\mathbf{K}^\top\mathbf{u}^{(\ell)}}\right) \\
&= d\,\mathbf{b}\odot\frac{1}{\mathbf{K}^\top\mathbf{u}^{(\ell)}} + \mathbf{b}\odot d\left(\frac{1}{\mathbf{K}^\top\mathbf{u}^{(\ell)}}\right) \\
&= -\operatorname{diag}\frac{\mathbf{b}}{\left(\mathbf{K}^\top\mathbf{u}^{(\ell)}\right)^2}\cdot\mathbf{K}^\top\cdot d\,\mathbf{u}^{(\ell)}
\end{aligned}
$$

Therefore, the Jacobian updating equations are

Update 4.1 (Updating Equations for Jacobians of Vanilla Sinkhorn)**.**

$$
\begin{aligned}
D\,\mathbf{u}^{(\ell)}(\mathbf{a}) &= \operatorname{diag}\frac{1}{\mathbf{K}\mathbf{v}^{(\ell-1)}} - \operatorname{diag}\frac{\mathbf{a}}{\left(\mathbf{K}\mathbf{v}^{(\ell-1)}\right)^2}\cdot\mathbf{K}\cdot D\,\mathbf{v}^{(\ell-1)}(\mathbf{a}), \\
D\,\mathbf{v}^{(\ell)}(\mathbf{a}) &= -\operatorname{diag}\frac{\mathbf{b}}{\left(\mathbf{K}^\top\mathbf{u}^{(\ell)}\right)^2}\cdot\mathbf{K}^\top\cdot D\,\mathbf{u}^{(\ell)}(\mathbf{a}).
\end{aligned}
\tag{27}
$$

Therefore, for the initialized $D\mathbf{v}^{(0)}\left(\mathbf{u}^{(0)}\right) = \mathbb{0}_{N\times N}$, we can compute $D\mathbf{u}^{(\ell)}$ and $D\mathbf{v}^{(\ell)}$ for any ℓ, including L. Then we can finally arrive at the formula for the Jacobian

$$\begin{aligned} DL^{\varepsilon}_{\mathbf{C}}(\mathbf{a}) &= DL^{\varepsilon}_{\mathbf{C}}(\mathbf{P}) \cdot D\mathbf{P}(\mathbf{a}) \\ &= \mathbb{1}_{MN}^{\top} \operatorname{diag}\operatorname{vec}\left(\mathbf{C} + \varepsilon \log \mathbf{P}\right) \operatorname{diag}\operatorname{vec} \mathbf{P} \Bigg[(\mathbb{1}_N \otimes I_M) \operatorname{diag} \frac{1}{\mathbf{u}^{(L)}} \cdot D\mathbf{u}^{(L)}(\mathbf{a}) \\ &\qquad + (I_N \otimes \mathbb{1}_M) \operatorname{diag} \frac{1}{\mathbf{v}^{(L)}} \cdot D\mathbf{v}^{(L)}(\mathbf{a}) \Bigg], \end{aligned} \tag{28}$$

and the gradient $\nabla_{\mathbf{a}} L^{\varepsilon}_{\mathbf{C}} = \left[DL^{\varepsilon}_{\mathbf{C}}(\mathbf{a}) \right]^{\top}$.

We can combine the Jacobian updates with the Sinkhorn updates in one loop, hence without the need to save intermediate Jacobian matrices.

Algorithm 4.1 Vanilla Sinkhorn with Gradient

Input: $\mathbf{a} \in \Sigma_M$, $\mathbf{b} \in \Sigma_N$, $\mathbf{C} \in \mathbb{R}^{M\times N}$, $\varepsilon > 0$.
Initialize: $\mathbf{u} = \mathbb{1}_M$, $\mathbf{v} = \mathbb{1}_N$, $\mathbf{J}_{\mathbf{u}} = \mathbb{0}_{M\times M}$, $\mathbf{J}_{\mathbf{v}} = \mathbb{0}_{N\times M}$.

$\mathbf{K} = \exp(-\frac{\mathbf{C}}{\varepsilon})$
while Not Convergence **do**
  # update Ju and u
  $\mathbf{J}_{\mathbf{u}} = \operatorname{diag} \frac{1}{\mathbf{K}\mathbf{v}} - \operatorname{diag} \frac{\mathbf{a}}{(\mathbf{K}\mathbf{v})^2} \cdot \mathbf{K} \cdot \mathbf{J}_{\mathbf{v}}$
  $\mathbf{u} \leftarrow \mathbf{a} \oslash (\mathbf{K}\mathbf{v})$
  # update Jv and v
  $\mathbf{J}_{\mathbf{v}} = -\operatorname{diag} \frac{\mathbf{b}}{\left(\mathbf{K}^{\top}\mathbf{u}^{(\ell)}\right)^2} \cdot \mathbf{K}^{\top} \cdot \mathbf{J}_{\mathbf{u}}$
  $\mathbf{v} \leftarrow \mathbf{b} \oslash (\mathbf{K}^{\top}\mathbf{u})$
end while
# compute Optimal Coupling
$\mathbf{P} = \operatorname{diag}(\mathbf{u})\, \mathbf{K}\, \operatorname{diag}(\mathbf{v})$
compute gradient
$\mathbf{J}_{\mathbf{P}} = \mathbb{1}_{MN}^{\top} \cdot \operatorname{diag}\operatorname{vec}\left(\mathbf{C} + \varepsilon \log \mathbf{P}\right)$
$\mathbf{J}_{\mathbf{a}} = \operatorname{diag}\operatorname{vec} \mathbf{P} \cdot \left[(\mathbb{1}_N \otimes I_M) \operatorname{diag} \frac{1}{\mathbf{u}} \cdot \mathbf{J}_{\mathbf{u}} + (I_N \times \mathbb{1}_m) \operatorname{diag} \frac{1}{\mathbf{v}} \cdot \mathbf{J}_{\mathbf{v}} \right]$
$\nabla = \left(\mathbf{J}_{\mathbf{P}} \cdot \mathbf{J}_{\mathbf{a}}\right)^{\top}$

Output: $\mathbf{P}$, ∇

4.2 Parallel Sinkhorn

For the Parallel Sinkhorn algorithm, since we have S transport maps being computed at the same time, as supposed to the Vanilla Sinkhorn where we only compute one transport map at a time, we would have a loss vector of length S instead of a scalar. Therefore, without a proper way to aggregate the loss vector into a scalar, we cannot derive the gradient of the loss. We can, however, derive the Jacobian updating equations for the Parallel Sinkhorn

algorithm, which is the focus of this subsection. With the Jacobians defined recursively, one can easily apply it to gradient computation, once the (scalar) loss function is properly defined.

A natural way to introduce an aggregation is to define the Wasserstein barycenter problem, where we can take the weighted sum of the Sinkhorn losses as the loss function for the minimization problem. But this discussion is deferred to Section 5.1.

Here, to derive the Jacobian updating equations, we can take the differentials

$$\begin{aligned} d\ \mathrm{vec}\,\mathbf{U}^{(\ell)} &= d\left[\mathrm{vec}\,\mathbf{A}\odot\frac{1}{\mathrm{vec}\,(\mathbf{K}\mathbf{V}^{(\ell-1)})}\right] \\ &= d\ \mathrm{vec}\,\mathbf{A}\odot\frac{1}{\mathrm{vec}\,(\mathbf{K}\mathbf{V}^{(\ell-1)})}+\mathrm{vec}\,\mathbf{A}\odot d\,\frac{1}{\mathrm{vec}\,(\mathbf{K}\mathbf{V}^{(\ell-1)})} \\ &= \mathrm{diag}\,\frac{1}{\mathrm{vec}\,(\mathbf{K}\mathbf{V}^{(\ell-1)})}\cdot d\ \mathrm{vec}\,\mathbf{A}-\mathrm{diag}\,\mathrm{vec}\,\mathbf{A}\cdot\mathrm{diag}\,\mathrm{vec}\,\frac{1}{\left(\mathbf{K}\mathbf{V}^{(\ell-1)}\right)^2}\cdot d\ \mathrm{vec}\left(\mathbf{K}\mathbf{V}^{(\ell-1)}\right) \\ &= \mathrm{diag}\,\frac{1}{\mathrm{vec}\,(\mathbf{K}\mathbf{V}^{(\ell-1)})}\cdot d\ \mathrm{vec}\,\mathbf{A}-\mathrm{diag}\,\mathrm{vec}\,\frac{\mathbf{A}}{\left(\mathbf{K}\mathbf{V}^{(\ell-1)}\right)^2}\cdot(I_S\otimes\mathbf{K})\cdot d\ \mathrm{vec}\,\mathbf{V}^{(\ell-1)}, \end{aligned} \tag{29}$$

and

$$\begin{aligned} d\ \mathrm{vec}\,\mathbf{V}^{(\ell)} &= d\left[\mathrm{vec}\,\mathbf{B}\odot\frac{1}{\mathrm{vec}\,(\mathbf{K}^\top\mathbf{U}^{(\ell)})}\right] \\ &= \mathrm{vec}\,\mathbf{B}\odot d\,\frac{1}{\mathrm{vec}\,(\mathbf{K}^\top\mathbf{U}^{(\ell)})} \\ &= -\mathrm{diag}\,\mathrm{vec}\,\mathbf{B}\cdot\mathrm{diag}\,\mathrm{vec}\,\frac{1}{\left(\mathbf{K}^\top\mathbf{U}^{(\ell)}\right)^2}\cdot d\ \mathrm{vec}\left(\mathbf{K}^\top\mathbf{U}^{(\ell)}\right) \\ &= -\mathrm{diag}\,\mathrm{vec}\,\frac{\mathbf{B}}{\left(\mathbf{K}^\top\mathbf{U}^{(\ell)}\right)^2}\cdot(I_S\otimes\mathbf{K})^\top\cdot d\ \mathrm{vec}\,\mathbf{U}^{(\ell)}. \end{aligned} \tag{30}$$

Therefore, we obtain the Jacobian updating equations.

Update 4.2 (Updating Equations for Jacobians of Parallel Sinkhorn)**.**

$$\begin{aligned} D\,\mathbf{U}^{(\ell)}(\mathbf{A}) &= \mathrm{diag}\,\frac{1}{\mathrm{vec}\,(\mathbf{K}\mathbf{V}^{(\ell-1)})}-\mathrm{diag}\,\mathrm{vec}\,\frac{\mathbf{A}}{\left(\mathbf{K}\mathbf{V}^{(\ell-1)}\right)^2}\cdot(I_S\otimes\mathbf{K})\cdot D\,\mathbf{V}^{(\ell-1)}(\mathbf{A}), \\ D\,\mathbf{V}^{(\ell)}(\mathbf{A}) &= -\mathrm{diag}\,\mathrm{vec}\,\frac{\mathbf{B}}{\left(\mathbf{K}^\top\mathbf{U}^{(\ell)}\right)^2}\cdot(I_S\otimes\mathbf{K})^\top\cdot D\,\mathbf{U}^{(\ell)}(\mathbf{A}). \end{aligned} \tag{31}$$

Algorithm 4.2 Parallel Sinkhorn Algorithm with Jacobians

Input: $\mathbf{A} \in \Sigma_{M\times S}$, $\mathbf{B} \in \Sigma_{N\times S}$, $\mathbf{C} \in \mathbb{R}^{M\times N}$, $\varepsilon > 0$.
Initialize: $\mathbf{U} = \mathbb{1}_{M\times S}$, $\mathbf{V} = \mathbb{1}_{N\times S}$, $\mathbf{J_U} = \mathbb{0}_{MS\times MS}$, $\mathbf{J_V} = \mathbb{0}_{NS\times MS}$.
$\mathbf{K} = \exp(-\frac{\mathbf{C}}{\varepsilon})$
while Not Convergence **do**
  $\mathbf{J_U} = \operatorname{diag}\frac{1}{\operatorname{vec}(\mathbf{KV})} - \operatorname{diag}\operatorname{vec}\frac{\mathbf{A}}{(\mathbf{KV})^2}\cdot(I_S\otimes\mathbf{K})\cdot\mathbf{J_V}$ # update JU
  $\mathbf{U} \leftarrow \mathbf{A} \oslash (\mathbf{KV})$
  $\mathbf{J_V} = -\operatorname{diag}\operatorname{vec}\frac{\mathbf{B}}{\left(\mathbf{K}^\top\mathbf{U}\right)^2}\cdot(I_S\otimes\mathbf{K})^\top\cdot\mathbf{J_U}$ # update JV
  $\mathbf{V} \leftarrow \mathbf{B} \oslash (\mathbf{K}^\top\mathbf{U})$
end while
$\forall d, \mathbf{P}_d = diag(\mathbf{U}_d)\,\mathbf{K}\,diag(\mathbf{V}_d)$
Output: $\mathbf{P}_d$ for all d, $\mathbf{J_U}$, $\mathbf{J_V}$

4.3 Log-Stabilized Sinkhorn

To derive the gradient of the log-stabilized Sinkhorn algorithm, we again use the Chain's rule to compute the Jacobian first as in Equation (23). We would thus have the same $D\,L^{\varepsilon}_{\mathbf{C}}(\mathbf{P}) = \mathbb{1}_{MN}^\top\cdot\operatorname{diag}\operatorname{vec}(\mathbf{C}+\varepsilon\log\mathbf{P})$, where $\mathbf{P} = \mathcal{P}\left(\mathbf{f}^{(L)},\mathbf{g}^{(L)}\right)$.

Then all we need is $D\,\mathbf{P}(\mathbf{a})$. To get this term, we will take the differential $\mathcal{P}$ defined as Equation (17) in Section 3.3, and $\mathbf{f} = \mathbf{f}(\mathbf{a})$ and $\mathbf{g} = \mathbf{g}(\mathbf{a})$,

$$\begin{aligned} d\,\operatorname{vec}\mathcal{P}(\mathbf{f},\mathbf{g}) &= d\,\operatorname{vec}\exp\left(-\frac{\mathbf{C}-\mathbf{f}\cdot\mathbb{1}_N^\top-\mathbb{1}_M\cdot\mathbf{g}^\top}{\varepsilon}\right)\\ &= \operatorname{diag}\operatorname{vec}\mathcal{P}(\mathbf{f},\mathbf{g})\cdot d\left[-\frac{\mathbf{C}-\mathbf{f}\cdot\mathbb{1}_N^\top-\mathbb{1}_M\cdot\mathbf{g}^\top}{\varepsilon}\right]\\ &= \frac{1}{\varepsilon}\operatorname{diag}\operatorname{vec}\mathcal{P}(\mathbf{f},\mathbf{g})\cdot d\,\operatorname{vec}\left(\mathbf{f}\cdot\mathbb{1}_N^\top+\mathbb{1}_M\cdot\mathbf{g}^\top\right)\\ &= \frac{1}{\varepsilon}\operatorname{diag}\operatorname{vec}\mathcal{P}(\mathbf{f},\mathbf{g})\cdot\Big[(\mathbb{1}_N\otimes I_M)\,d\,\mathbf{f}+(I_N\otimes\mathbb{1}_M)\,d\,\mathbf{g}\Big]. \end{aligned} \tag{32}$$

Therefore, the Jacobian of $\mathbf{P} = \mathcal{P}\left(\mathbf{f}^{(L)},\mathbf{g}^{(L)}\right)$ is

$$\begin{aligned} D\,\mathbf{P}(\mathbf{a}) &= D\,\mathcal{P}(\mathbf{f}^{(L)}(\mathbf{a}),\mathbf{g}^{(L)}(\mathbf{a}))\\ &= \frac{1}{\varepsilon}\operatorname{diag}\operatorname{vec}\mathbf{P}\cdot\Big[(\mathbb{1}_N\otimes I_M)\,D\,\mathbf{f}^{(L)}(\mathbf{a})+(I_N\otimes\mathbb{1}_M)\,D\,\mathbf{g}^{(L)}(\mathbf{a})\Big]. \end{aligned} \tag{33}$$

Note that for $\mathcal{P}$, $\mathcal{Q}$, and c defined in Equations (17) and (20),

$$\mathcal{P}(\mathbf{f},\mathbf{g}) = \exp\left(-\frac{\mathbf{C}-\mathbf{f}\cdot\mathbb{1}_N^\top-\mathbb{1}_M\cdot\mathbf{g}^\top}{\varepsilon}\right) = \exp\left(-\frac{c(\mathbf{f},\mathbf{g})}{\varepsilon}\right)\cdot\mathcal{Q}(\mathbf{f},\mathbf{g}), \tag{34}$$

and thus

$$\operatorname{diag}\operatorname{vec}\frac{1}{\mathcal{P}(\mathbf{f},\mathbf{g})\cdot\mathbb{1}_N}=\frac{1}{\exp\left(-\frac{c(\mathbf{f},\mathbf{g})}{\varepsilon}\right)}\cdot\operatorname{diag}\operatorname{vec}\frac{1}{\mathcal{Q}(\mathbf{f},\mathbf{g})\cdot\mathbb{1}_N},$$
$$\operatorname{diag}\operatorname{vec}\mathcal{P}(\mathbf{f},\mathbf{g})=\exp\left(-\frac{c(\mathbf{f},\mathbf{g})}{\varepsilon}\right)\cdot\operatorname{diag}\operatorname{vec}\mathcal{Q}(\mathbf{f},\mathbf{g}).$$

Then we need to differentiate $\mathbf{f}^{(L)}$ and $\mathbf{g}^{(L)}$ to obtain $D\mathbf{f}^{(L)}(\mathbf{a})$ and $D\mathbf{g}^{(L)}(\mathbf{a})$ to obtain the formulae for the computation of $D\,\mathbf{P}(\mathbf{a})$. Instead of directly differentiating from Equation (21), we will take the differentials from Equation (19), as this is equivalent to the form in Equation (21), but without the need for differentiation on the min function:

$$\begin{aligned}
d\ \operatorname{vec}\mathbf{f}^{(\ell)} &= d\,\mathbf{f}^{(\ell-1)}+\varepsilon d\ \log\mathbf{a}-\varepsilon d\ \operatorname{vec}\log\left[\mathcal{P}\left(\mathbf{f}^{(\ell-1)},\mathbf{g}^{(\ell-1)}\right)\cdot\mathbb{1}_N\right]\\
&= d\,\mathbf{f}^{(\ell-1)}+\varepsilon\operatorname{diag}\frac{1}{\mathbf{a}}\cdot d\,\mathbf{a}-\varepsilon\operatorname{diag}\operatorname{vec}\frac{1}{\mathcal{P}\left(\mathbf{f}^{(\ell-1)},\mathbf{g}^{(\ell-1)}\right)\cdot\mathbb{1}_N}\\
&\quad\cdot d\ \operatorname{vec}\left[\mathcal{P}\left(\mathbf{f}^{(\ell-1)},\mathbf{g}^{(\ell-1)}\right)\cdot\mathbb{1}_N\right]\\
&= d\,\mathbf{f}^{(\ell-1)}+\varepsilon\operatorname{diag}\frac{1}{\mathbf{a}}\cdot d\,\mathbf{a}-\frac{\varepsilon}{\exp\left(-\frac{c\left(\mathbf{f}^{(\ell)},\mathbf{g}^{(\ell)}\right)}{\varepsilon}\right)}\\
&\quad\cdot\operatorname{diag}\operatorname{vec}\frac{1}{\mathcal{Q}\left(\mathbf{f}^{(\ell-1)},\mathbf{g}^{(\ell-1)}\right)\cdot\mathbb{1}_N}\cdot\left(\mathbb{1}_N^\top\otimes I_M\right)\cdot d\ \operatorname{vec}\mathcal{P}\left(\mathbf{f}^{(\ell-1)},\mathbf{g}^{(\ell-1)}\right)\\
&= d\,\mathbf{f}^{(\ell-1)}+\varepsilon\operatorname{diag}\frac{1}{\mathbf{a}}\cdot d\,\mathbf{a}-\operatorname{diag}\operatorname{vec}\frac{1}{\mathcal{Q}\left(\mathbf{f}^{(\ell-1)},\mathbf{g}^{(\ell-1)}\right)\cdot\mathbb{1}_N}\cdot\left(\mathbb{1}_N\otimes I_M\right)^\top\\
&\quad\cdot\operatorname{diag}\operatorname{vec}\mathcal{Q}\left(\mathbf{f}^{(\ell-1)},\mathbf{g}^{(\ell-1)}\right)\cdot\left[\left(\mathbb{1}_N\otimes I_M\right)\cdot d\,\mathbf{f}^{(\ell-1)}+\left(I_N\otimes\mathbb{1}_M\right)\cdot d\,\mathbf{g}^{(\ell-1)}\right].
\end{aligned}\tag{35}$$

Similarly, we can have the differential for $\mathbf{g}^{\ell}$. Note that a commutation matrix[9] $\mathcal{K}^{(M,N)}$ is one such that $\mathcal{K}^{(M,N)}\cdot\operatorname{vec}\mathbf{A}=\operatorname{vec}\left(\mathbf{A}^\top\right)$ for any matrix $\mathbf{A}\in\mathbb{R}^{M\times N}$. Then,

[9] `https://en.wikipedia.org/wiki/Commutation_matrix`

$$
\begin{aligned}
d\ \mathrm{vec}\,\mathbf{g}^{(\ell)} &= d\mathbf{g}^{(\ell-1)} - \varepsilon\,\mathrm{diag}\,\mathrm{vec}\,\frac{1}{\mathcal{P}^\top\left(\mathbf{f}^{(\ell)},\mathbf{g}^{(\ell-1)}\right)\cdot\mathbb{1}_M}\cdot d\ \mathrm{vec}\left[\mathcal{P}^\top\left(\mathbf{f}^{(\ell)},\mathbf{g}^{(\ell-1)}\right)\cdot\mathbb{1}_M\right] \\
&= d\mathbf{g}^{(\ell-1)} - \varepsilon\,\mathrm{diag}\,\mathrm{vec}\,\frac{1}{\mathcal{P}^\top\left(\mathbf{f}^{(\ell)},\mathbf{g}^{(\ell-1)}\right)\cdot\mathbb{1}_M}\cdot d\ \left[\left(\mathbb{1}_M^\top\otimes I_N\right)\cdot\mathcal{K}^{(M,N)}\cdot\mathcal{P}\left(\mathbf{f}^{(\ell)},\mathbf{g}^{(\ell-1)}\right)\right] \\
&= d\mathbf{g}^{(\ell-1)} - \varepsilon\,\mathrm{diag}\,\mathrm{vec}\,\frac{1}{\mathcal{P}^\top\left(\mathbf{f}^{(\ell)},\mathbf{g}^{(\ell-1)}\right)\cdot\mathbb{1}_M}\cdot\left(\mathbb{1}_M^\top\otimes I_N\right)\cdot\mathcal{K}^{(M,N)}\cdot d\ \mathrm{vec}\,\mathcal{P}\left(\mathbf{f}^{(\ell)},\mathbf{g}^{(\ell-1)}\right) \\
&= d\mathbf{g}^{(\ell-1)} - \frac{\varepsilon}{\exp\left(-\frac{c\left(\mathbf{f}^{(\ell)},\mathbf{g}^{(\ell-1)}\right)}{\varepsilon}\right)}\cdot\mathrm{diag}\,\mathrm{vec}\,\frac{1}{\mathcal{Q}^\top\left(\mathbf{f}^{(\ell)},\mathbf{g}^{(\ell-1)}\right)\cdot\mathbb{1}_M} \\
&\quad\cdot\left(\mathbb{1}_M\otimes I_N\right)^\top\cdot\mathcal{K}^{(M,N)}\cdot\frac{\exp\left(-\frac{c\left(\mathbf{f}^{(\ell)},\mathbf{g}^{(\ell-1)}\right)}{\varepsilon}\right)}{\varepsilon} \\
&\quad\cdot\mathrm{diag}\,\mathrm{vec}\,\mathcal{Q}\left(\mathbf{f}^{(\ell)},\mathbf{g}^{(\ell-1)}\right)\cdot\left[\left(\mathbb{1}_N\otimes I_M\right)\cdot d\,\mathbf{f}^{(\ell)} + \left(I_N\otimes\mathbb{1}_M\right)\cdot d\,\mathbf{g}^{(\ell-1)}\right] \\
&= d\mathbf{g}^{(\ell-1)} - \mathrm{diag}\,\mathrm{vec}\,\frac{1}{\mathcal{Q}^\top\left(\mathbf{f}^{(\ell)},\mathbf{g}^{(\ell-1)}\right)\cdot\mathbb{1}_M}\cdot\left(\mathbb{1}_M\otimes I_N\right)^\top\cdot\mathcal{K}^{(M,N)} \\
&\quad\cdot\mathrm{diag}\,\mathrm{vec}\,\mathcal{Q}\left(\mathbf{f}^{(\ell)},\mathbf{g}^{(\ell-1)}\right)\cdot\left[\left(\mathbb{1}_N\otimes I_M\right)\cdot d\,\mathbf{f}^{(\ell)} + \left(I_N\otimes\mathbb{1}_M\right)\cdot d\,\mathbf{g}^{(\ell-1)}\right].
\end{aligned}
\tag{36}
$$

Therefore, from Equations (35) and (36), we have obtained the updating equations for the Jacobians with the numerically stable $\mathcal{Q}$ instead of $\mathcal{P}$.

Update 4.3 (Updating Equations for Jacobians of Log-Stabilized Sinkhorn)**.**

$$
\begin{aligned}
D\,\mathbf{f}^{(\ell)}(\mathbf{a}) &= D\,\mathbf{f}^{(\ell-1)}(\mathbf{a}) + \varepsilon\,\mathrm{diag}\,\frac{1}{\mathbf{a}} - \mathrm{diag}\,\mathrm{vec}\,\frac{1}{\mathcal{Q}\left(\mathbf{f}^{(\ell-1)},\mathbf{g}^{(\ell-1)}\right)\cdot\mathbb{1}_N}\cdot\left(\mathbb{1}_N\otimes I_M\right)^\top \\
&\quad\cdot\mathrm{diag}\,\mathrm{vec}\,\mathcal{Q}\left(\mathbf{f}^{(\ell-1)},\mathbf{g}^{(\ell-1)}\right)\cdot\left[\left(\mathbb{1}_N\otimes I_M\right)\cdot D\,\mathbf{f}^{(\ell-1)}(\mathbf{a}) + \left(I_N\otimes\mathbb{1}_M\right)\cdot D\,\mathbf{g}^{(\ell-1)}(\mathbf{a})\right],
\end{aligned}
\tag{37}
$$

$$
\begin{aligned}
D\,\mathbf{g}^{(\ell)}(\mathbf{a}) &= D\,\mathbf{g}^{(\ell-1)}(\mathbf{a}) - \mathrm{diag}\,\mathrm{vec}\,\frac{1}{\mathcal{Q}^\top\left(\mathbf{f}^{(\ell)},\mathbf{g}^{(\ell-1)}\right)\cdot\mathbb{1}_M}\cdot\left(\mathbb{1}_M\otimes I_N\right)^\top\cdot\mathcal{K}^{(M,N)} \\
&\quad\cdot\mathrm{diag}\,\mathrm{vec}\,\mathcal{Q}\left(\mathbf{f}^{(\ell)},\mathbf{g}^{(\ell-1)}\right)\cdot\left[\left(\mathbb{1}_N\otimes I_M\right)\cdot D\,\mathbf{f}^{(\ell)}(\mathbf{a}) + \left(I_N\otimes\mathbb{1}_M\right)\cdot D\,\mathbf{g}^{(\ell-1)}(\mathbf{a})\right].
\end{aligned}
\tag{38}
$$

Therefore, from Update 4.3, we could iteratively obtain $D\,\mathbf{f}^{(L)}(\mathbf{a})$ and $D\,\mathbf{g}^{(L)}(\mathbf{a})$. And thus obtain $D\,\mathbf{P}(\mathbf{a})$ from Equation (33). Then with the Jacobian in Equation (24), we finally have the Jacobian of the loss with respect to argument $\mathbf{a}$, and its transpose would be the gradient.

Algorithm 4.3 Log-Stabilized Sinkhorn with Gradient

Input: $\mathbf{a} \in \Sigma_M$, $\mathbf{b} \in \Sigma_N$, $\mathbf{C} \in \mathbb{R}^{M \times N}$, $\varepsilon > 0$.
Initialize: $\mathbf{f} = \mathbb{0}_M$, $\mathbf{g} = \mathbb{0}_N$, $\mathbf{J_f} = \mathbb{0}_{M \times M}$, $\mathbf{J_g} = \mathbb{0}_{N \times M}$.

$\mathbf{K} = Commutation(M, N)$ # commutation matrix K
while Not Convergence **do**
  # update f and Jf
  $\mathbf{S} = \mathbf{C} - \mathbf{f} \cdot \mathbb{1}_N^\top - \mathbb{1}_M \cdot \mathbf{g}^\top$
  $c = \min \mathbf{S}$
  $\mathbf{Q} = \exp\left(-\frac{\mathbf{S}-c}{\varepsilon}\right)$
  $\mathbf{J} = (\mathbb{1}_N \otimes I_M) \cdot \mathbf{J_f} + (I_N \otimes \mathbb{1}_M) \cdot \mathbf{J_g}$
  $\mathbf{J_f} = \mathbf{J_f} + \varepsilon \operatorname{diag} \frac{1}{\mathbf{a}} - \operatorname{diag vec} \frac{1}{\mathbf{Q} \cdot \mathbb{1}_N} \cdot (\mathbb{1}_N \otimes I_M)^\top \cdot \operatorname{diag vec} \mathbf{Q} \cdot \mathbf{J}$
  $\mathbf{f} = \mathbf{f} + \varepsilon \log \mathbf{a} + c - \varepsilon \log [\mathbf{Q} \cdot \mathbb{1}_N]$
  # update g and Jg
  $\mathbf{S} = \mathbf{C} - \mathbf{f} \cdot \mathbb{1}_N^\top - \mathbb{1}_M \cdot \mathbf{g}^\top$
  $c = \min \mathbf{S}$
  $\mathbf{Q} = \exp\left(-\frac{\mathbf{S}-c}{\varepsilon}\right)$
  $\mathbf{J} = (\mathbb{1}_N \otimes I_M) \cdot \mathbf{J_f} + (I_N \otimes \mathbb{1}_M) \cdot \mathbf{J_g}$
  $\mathbf{J_g} = \mathbf{J_g} - \operatorname{diag vec} \frac{1}{\mathbf{Q}^\top \cdot \mathbb{1}_M} \cdot (\mathbb{1}_M \otimes I_N)^\top \cdot \mathbf{K} \cdot \operatorname{diag vec} \mathbf{Q} \cdot \mathbf{J}$
  $\mathbf{g} = \mathbf{g} + \varepsilon \log \mathbf{b} + c - \varepsilon \log \left[\mathbf{Q}^\top \cdot \mathbb{1}_M\right]$
end while
$\mathbf{S} = \mathbf{C} - \mathbf{f} \cdot \mathbb{1}_N^\top - \mathbb{1}_M \cdot \mathbf{g}^\top$
$\mathbf{P} = \exp\left(-\frac{\mathbf{S}}{\varepsilon}\right)$ # optimal coupling
$\mathbf{J_{P(a)}} = \frac{1}{\varepsilon} \cdot \operatorname{diag vec} \mathbf{P} \cdot [(\mathbb{1}_N \otimes I_M) \cdot \mathbf{J_f} + (I_N \otimes \mathbb{1}_M) \cdot \mathbf{J_g}]$
$\mathbf{J_{L(P)}} = \mathbb{1}_{MN}^\top \cdot \operatorname{diag vec} (\mathbf{C} + \varepsilon \log \mathbf{P})$
$\nabla = \left(\mathbf{J_{L(P)}} \cdot \mathbf{J_{P(a)}}\right)^\top$

Output: $\mathbf{P}$, ∇.

5 Wasserstein Barycenter and Dictionary Learning

5.1 Fréchet Mean and Wasserstein Barycenter

As is often in the case in practice, one wants to compute the "mean" or "center" given a sequence of points. This is useful when we want to find a unknown center as an estimate to the group of points (usually data). Suppose we have $(x_s)_{s=1}^S \in \mathcal{X}^S$ in a metric space $(\mathcal{X}, d)$, where $d(\cdot)$ is a metric defined on $\mathcal{X}$, then we have the following problem

$$\min_{x \in \mathcal{X}} \sum_s^S \lambda_s d(x, x_s)^p,$$

for a weight vector $\boldsymbol{\lambda} \in \Sigma_S$ and some power p but is often set to 2. This is often called

"Fréchet" or "Karcher" mean[10].

We can then state the Wasserstein Barycenter problem, similar to the idea of "Fréchet mean" as a weighted sum of the individual distances/losses. Consider the loss function in Equation (2), a weight vector $\boldsymbol{\lambda} \in \Sigma_S$, and a sequence of discrete densities $\{\mathbf{a}_s\}_{s=1}^S$, where $a_s \in \Sigma_{M_s}$ and M_s is the length of a_s, a Wasserstein barycenter is computed by the following minimization problem

$$\arg\min_{\mathbf{b}\in\Sigma_N} \sum_{s=1}^{S} \lambda_s L_{\mathbf{C}_s}\left(\mathbf{a}_s, \mathbf{b}\right),$$

where the cost matrix $\mathbf{C}_s \in \mathbb{R}^{M_s \times N}$ specifies the cost for transporting between the probability vectors $\mathbf{a}_s$ and $\mathbf{b}$. Of course, this problem can be approximated by the entropic regularized version by simply swapping the loss function by Equation (3). Thus, we have the entropic regularized Wasserstein barycenter problem

$$\arg\min_{\mathbf{b}\in\Sigma_N} \sum_{s=1}^{S} \lambda_s L^{\varepsilon}_{\mathbf{C}_s}\left(\mathbf{a}_s, \mathbf{b}\right). \tag{39}$$

$\mathbf{b}$ is therefore the unknown computed barycenter as the "mean" of $\mathbf{a}_1, \ldots, \mathbf{a}_S$. Again, we will arrive at the Sinkhorn-like updates (Benamou et al., 2015; Schmitz et al., 2018):

$$\begin{aligned}
\mathbf{u}_s^{(\ell)} &= \frac{\mathbf{a}_s}{\mathbf{K}_s \mathbf{v}_s^{(\ell-1)}}, \\
\mathbf{b}^{(\ell)} &= \mathbf{\Pi}_{s=1}^{S} \left(\mathbf{K}_s^{\top} \mathbf{u}_s^{(\ell)}\right)^{\lambda_s}, \\
\mathbf{v}_s^{(\ell)} &= \frac{\mathbf{b}^{(\ell)}}{\mathbf{K}_s^{\top} \mathbf{u}_s^{(\ell)}},
\end{aligned} \tag{40}$$

where $\mathbf{K}_s = \exp\left(-\frac{\mathbf{C}_s}{\varepsilon}\right)$, and $\mathbf{u}_s$, $\mathbf{v}_s$ are scaling vectors for $s = 1, \ldots, S$. Also, $\frac{\cdot}{\cdot}$ and $(\cdot)^{(\cdot)}$ are to be understood as element-wise operations.

In practical applications, however, we usually have $M_1 = \ldots = M_s = N$ and $\mathbf{K} = \mathbf{K}_1 = \mathbf{K}_2 = \cdots = \mathbf{K}_S$,

we can thus rewrite Equation (40) as the following updating equations

Update 5.1.

$$\begin{aligned}
\mathbf{U}^{(\ell)} &= \frac{\mathbf{A}}{\mathbf{K}\mathbf{V}^{(\ell-1)}}, \\
\mathbf{b}^{(\ell)} &= \mathbf{\Pi}_{row} \left(\mathbf{K}^{\top} \mathbf{U}^{(\ell)}\right)^{\boldsymbol{\lambda}^{\top} \otimes \mathbb{1}_N}, \\
\mathbf{V}^{(\ell)} &= \frac{\mathbf{B}^{(\ell)}}{\mathbf{K}^{\top}\mathbf{U}^{(\ell)}},
\end{aligned} \tag{41}$$

where $\mathbf{B}^{(\ell)} = \left[\mathbf{b}^{(\ell)}, \ldots, \mathbf{b}^{(\ell)}\right] \in \mathbb{R}^{N \times S}$*, and the product operator* $\mathbf{\Pi}_{row}$ *refers to the row-wise product on the matrix.*

[10] Or Riemannian center of mass. See Karcher (2014) for a historical account on the naming.

The row-wise product will reduce the matrix into a length N vector, and the power operation in the second line refers to the element-wise power operation. The matrix $\boldsymbol{\lambda}^\top \otimes \mathbb{1}_N$ simply populates the length S vector $\boldsymbol{\lambda}$ into an $N \times S$ matrix, which can also be seen as applying each element of the $\boldsymbol{\lambda}$ vector to the base matrix column-wise.

Algorithm 5.1 Wasserstein Barycenter Algorithm (Parallel)

Input: $\mathbf{A} \in \Sigma_{N\times S}$, $\mathbf{C} \in \mathbb{R}^{N\times N}$, $\varepsilon > 0$.
Initialize: $\mathbf{U} = \mathbb{1}_{N\times S}$, $\mathbf{V}_{N\times S} = \mathbb{1}_{N\times S}$, $\mathbf{b} = \mathbb{0}_N$.
1: $\mathbf{K} = \exp(-\frac{\mathbf{C}}{\varepsilon})$
2: **while** Not Convergence **do**
3: $\quad \mathbf{U} \leftarrow \mathbf{A} \oslash (\mathbf{KV})$
4: $\quad \mathbf{b} = \boldsymbol{\Pi}_{row} \left(\mathbf{K}^\top \mathbf{U}\right)^{\boldsymbol{\lambda}^\top \otimes \mathbb{1}_N}$
5: $\quad \mathbf{V} \leftarrow \mathbf{b} \oslash (\mathbf{K}^\top \mathbf{U})$
6: **end while**
Output: $\mathbf{b}$

Remark. *In Line 4 of Algorithm 5.1, the notation $\boldsymbol{\Pi}_s$ refers to row-wise product reduction as in Equation* (41)*. In Line 5, the numerator $\mathbf{b}$ can be broadcast to be the same numerator of all columns of the denominator matrix as discussed in Section 3.2.*

Similar to the discussion of Section 3.3, numerical instability is still an issue for Algorithm 5.1 due the computation of the exponential function and its potential numeric overflow. Therefore, we would also need to stabilize the computation in the log-domain.

TODO: regular (parallel) and log domain algo

5.2 Wasserstein Dictionary Learning

In the Wasserstein barycenter problem, we have the data as sequence of probability vectors $\{\mathbf{a}_s\}_{s=1}^S$, and would like to obtain a "centroid" of the data $\mathbf{b}$. Here, $\mathbf{a}_s$'s are known as data, but $\mathbf{b}$ is unknown and thus need to be estimated or computed by optimization. The so-called Wasserstein Dictionary Learning (Schmitz et al., 2018) is the almost converse problem: suppose we have the observed $\mathbf{b}$'s, how to find the latent and thus unknown $\mathbf{a}_s$ such that the barycenter from which would reconstruct the data $\mathbf{b}$. In this case, it is the data $\mathbf{b}$ that is known, but not $\mathbf{a}_s$'s.

The Wasserstein Dictionary Learning can be considered broadly as one of the (nonlinear) topic models (Blei and Lafferty, 2009), the most famous of which would probably be Latent Dirichlet Allocation model (Blei, Ng, and Jordan, 2003). As is the case with all other topic models, we have a sequence of data and we need to discover the common "factors" or "topics" among them, either in Natural Language Processing (Xu et al., 2018) or Image Processing (Schmitz et al., 2018) settings.

To state the problem, let $\mathbf{B} \in \mathbb{R}_+^{N\times M}$ be the matrix containing the data, i.e. M probability vectors each of size N, and thus for each column $\mathbf{b}_m$ we have $\mathbf{b}_m \in \Sigma_N$ for $m = 1, \ldots, M$.

For the latent topic[11] matrix $\mathbf{A} \in \mathbb{R}_+^{N \times S}$ where each column $\mathbf{a}_s \in \Sigma_N$, and the weight matrix $\mathbf{W} \in \mathbb{R}_+^{S \times M}$ where each column $\mathbf{w}_m \in \Sigma_S$, we have

$$\begin{aligned} &\min_{\mathbf{A},\mathbf{W}} \sum_{m=1}^{M} \mathcal{L}\left(\widehat{\mathbf{b}}_m, \mathbf{b}_m\right), \\ &\text{s.t.} \widehat{\mathbf{b}}_m = \arg\min_{\mathbf{b} \in \Sigma_N} \sum_{s=1}^{S} \mathbf{W}_{sm} L_{\mathbf{C}}^{\varepsilon}\left(\mathbf{a}_s, \mathbf{b}\right), \\ &\quad \mathbf{A} = [\mathbf{a}_1, \ldots, \mathbf{a}_S] \in \mathbb{R}_+^{N \times S}, \text{where } \mathbf{a}_s \in \Sigma_N, \\ &\quad \mathbf{W} = [\mathbf{w}_1, \ldots, \mathbf{w}_M] \in \mathbb{R}_+^{S \times M}, \text{where } \mathbf{w}_m \in \Sigma_S. \end{aligned} \tag{42}$$

Essentially, we are considering the $\widehat{\mathbf{b}}_m$, the computed barycenter from the current $\mathbf{A}$, as the reconstruction for the true $\mathbf{b}_m$ that we observe in the data. Then we will need to optimize $\mathbf{A}$ (and also implicitly $\mathbf{W}$) such that the reconstructions approximate the original data well. This problem, however, is not guaranteed to be convex (Schmitz et al., 2018), whether jointly in $\mathbf{A}$ and $\mathbf{W}$ or for each separately, unlike the original Sinkhorn problem discussed in Section 2. Thus, we aim to solve this problem by a gradient descent approach to find a local minimum, as is the usual case in machine learning literature. In order to do so, we will take the gradient of the loss with respect to the parameters (here $\mathbf{A}$ and $\mathbf{W}$), and then optimize them by some learning rate, and then repeat until some stopping criterion. The gradient computation can be easily carried out by any modern Automatic Differentiation system[12], but in this note, I manually derive the gradients and Jacobians of all algorithms (in Sections 4 and 6) to achieve memory efficient and faster code.

Note, however, we cannot directly optimize the parameters from Equation (42), since the problem is a constrained optimization problem with two constraints on the $\mathbf{a}_s$ and $\mathbf{w}_m$ being in their respective simplices. Therefore, we need to convert this constrained optimization problem into an unconstrained one (Schmitz et al., 2018; Xu et al., 2018): that is, to let $\mathbf{a}_s = \text{softmax}\left(\boldsymbol{\alpha}_s\right)$ and $\mathbf{w}_m = \text{softmax}\left(\boldsymbol{\lambda}_s\right)$, as the softmax function will ensure its output to be a probability vector, i.e. $\mathbf{a}_s \in \Sigma_N$ and $\mathbf{w}_m \in \Sigma_S$. Thus, we have transformed the problem as an optimization problem with parameters $\boldsymbol{\alpha}$ and $\boldsymbol{\lambda}$. Now we need to define the softmax function.

[11] Sometimes also called atom or factor.

[12] They are usually embedded within neural network libraries, e.g. PyTorch (Paszke et al., 2017), TensorFlow (Abadi et al., 2016), etc., but can also be found as stand-alone libraries, e.g. autodiff in C++ (Leal, 2018) or Zygote in Julia (Innes, 2019). Unlike Symbolic or Numeric Differentiation, Automatic Differentiation (AD) essentially relies on chain's rule and could provide accurate gradient computation. There are usually two modes for AD systems: Forward mode or Reverse mode, and each has their own advantages. To see this, let a function $f : \mathbb{R}^n \to \mathbb{R}^m$, when $n \ll m$, then Forward mode is more efficient; if, however, $n \gg m$, then Reverse mode is more efficient. Neural Network libraries are usually implemented in Reverse mode, since there is usually a scalar loss but with many parameters of the neural network model. Interested readers could refer to Nocedal and Wright (2006, Chapter 8).

Definition 5 (Softmax function)**.** *Let* $\mathbf{x} \in \mathbb{R}^N$*, and the softmax function is*

$$softmax\left(\mathbf{x}_i\right) = \frac{\exp \mathbf{x}_i}{\sum_j \exp \mathbf{x}_j}.$$

Then the softmax function for a vector $\mathbf{x}$ *is*

$$softmax\left(\mathbf{x}\right) = \frac{\exp \mathbf{x}}{\mathbb{1}_N^\top \cdot \exp \mathbf{x}} = \frac{\exp \mathbf{x}}{\mathbb{1}_N^\top \cdot \exp \mathbf{x} \cdot \mathbb{1}_N}.$$

Then for a matrix $\mathbf{X} \in \mathbb{R}^{N \times S}$*, we have softmax for a matrix*

$$\begin{aligned} softmax\left(\mathbf{X}\right) &= \left[softmax\left(\mathbf{x}_1\right), \ldots, softmax\left(\mathbf{x}_S\right)\right] \\ &= \left[\frac{\exp \mathbf{x}_1}{\mathbb{1}_N^\top \cdot \exp \mathbf{x}_1 \cdot \mathbb{1}_N}, \ldots, \frac{\exp \mathbf{x}_S}{\mathbb{1}_N^\top \cdot \exp \mathbf{x}_S \cdot \mathbb{1}_N}\right], \\ &= \frac{\exp \mathbf{X}}{\left[\mathbb{1}_N^\top \cdot \exp \mathbf{x}_1, \ldots, \mathbb{1}_N^\top \cdot \exp \mathbf{x}_S\right] \otimes \mathbb{1}_N} \\ &= \frac{\exp \mathbf{X}}{\left(\mathbb{1}_N^\top \cdot \exp \mathbf{X}\right) \otimes \mathbb{1}_N}. \end{aligned}$$

The softmax of a matrix is equivalent to softmax of its column vectors' separately.

Therefore, we can enforce the constraints $\mathbf{a}_s \in \Sigma_N$ and $\mathbf{w}_m \in \Sigma_S$ by the following change of variables

$$\mathbf{A} = \operatorname{softmax}\left(\boldsymbol{\alpha}\right), \quad \text{and} \quad \mathbf{W} = \operatorname{softmax}\left(\boldsymbol{\lambda}\right), \tag{43}$$

where $\boldsymbol{\alpha} = [\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_S] \in \mathbb{R}^{N \times S}$, and $\boldsymbol{\lambda} = [\boldsymbol{\lambda}_1, \ldots, \boldsymbol{\lambda}_M] \in \mathbb{R}^{S \times M}$. Let us slightly abuse the notation to denote the vector of losses across the topics

$$L_{\mathbf{C}}^{\varepsilon}\left(\mathbf{A}, \mathbf{b}\right) = \left[L_{\mathbf{C}}^{\varepsilon}\left(\mathbf{a}_1, \mathbf{b}\right), \ldots, L_{\mathbf{C}}^{\varepsilon}\left(\mathbf{a}_S, \mathbf{b}\right)\right]^\top \in \mathbb{R}^N.$$

Note that $L_{\mathbf{C}}^{\varepsilon}\left(\mathbf{A}, \mathbf{b}\right)$ is essentially the same as $L_{\mathbf{C}}^{\varepsilon}\left(\mathbf{A}, \mathbf{B}\right)$, where $\mathbf{B} = [\mathbf{b}, \ldots, \mathbf{b}] \in \mathbb{R}^{N \times S}$ as the "many-to-one" transport problem in Section 3.2. Therefore, we can rewrite the problem in Equation (42) as an unconstrained one with respect to parameters $\boldsymbol{\alpha}$ and $\boldsymbol{\lambda}$

$$\begin{aligned} &\min_{\boldsymbol{\alpha}, \boldsymbol{\lambda}} \sum_{m=1}^{M} \mathcal{L}\left(\widehat{\mathbf{b}}_m, \mathbf{b}_m\right), \\ &\text{s.t.} \widehat{\mathbf{b}}_m = \operatorname*{arg\,min}_{\mathbf{b} \in \Sigma_N} \mathbf{w}_m^\top \cdot L_{\mathbf{C}}^{\varepsilon}\left(\mathbf{A}, \mathbf{b}\right), \\ &\quad \mathbf{A} = \operatorname{softmax}\left(\boldsymbol{\alpha}\right), \\ &\quad \mathbf{w}_m = \operatorname{softmax}\left(\boldsymbol{\lambda}_m\right). \end{aligned} \tag{44}$$

Essentially, we will take some random initialized parameters $\mathbf{A}$ and $\mathbf{W}$, calculate their optimal barycenter, calculate the loss between the barycenter and the original data, take the

gradients of the loss with respect to the parameters, and then update the parameters based on the gradients. The computation of the optimal barycenter is based on the Wasserstein barycenter problem, detailed in Section 5.1, and the parameter-updating routine can be chosen to be any optimization technique, for example Schmitz et al. (2018) chooses L-BFGS[13] (Liu and Nocedal, 1989), and Xie (2020b) uses Adam (Kingma and Ba, 2015) optimizer as it performs well in practice, especially in large-scale machine learning applications.

Here I list the Stochastic Gradient Descent, Adam, and AdamW optimizers' algorithms in Algorithms 5.2 to 5.4, and they are all implemented in the ***wig*** package, which will be discussed in details in Section 8. Instead of providing the full algorithms as in Kingma and Ba (2015, Algorithm 1) or Loshchilov and Hutter (2019, Algorithm 2), I provide the algorithms as the parameter-updating steps to disentangle them from the gradient calculation.

Definition 6 (**dekel2012**). *For some batch size B and objective function to be minimized $f(\mathbf{x};\boldsymbol{\theta}_t)$ with data $\mathbf{x}$ and parameters $\boldsymbol{\theta}_t$ at timestamp t, define the mini-batch gradient as*

$$\mathbf{g}_t = \frac{1}{B}\sum_{b=1}^{B}\nabla_\theta f_t(\mathbf{x}_b;\boldsymbol{\theta}_t), \tag{45}$$

where $\{\mathbf{x}_b\}_{b=1}^{B}$ denote the sequence of data.

Update 5.2 (Stochastic Gradient Descent). *Given a function to be minimized $f(\mathbf{x};\boldsymbol{\theta}_t)$ and a mini-batched gradient $\mathbf{g}_t$ with current parameters $\boldsymbol{\theta}_t$ and learning rate η,*

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta\cdot\mathbf{g}_t.$$

Update 5.3 (Adam Optimizer (Kingma and Ba, 2015)). *Given a function to be minimized $f(\mathbf{x};\boldsymbol{\theta}_t)$ and a mini-batched gradient $\mathbf{g}_t$ with current parameters $\boldsymbol{\theta}_t$ and learning rate η,*

TODO: re-write the optimizers as updating-steps, instead of the loop

Algorithm 5.2 Stochastic Gradient Descent

Input: $f(\mathbf{x};\boldsymbol{\theta})$: function to be optimized (minimized) with parameters $\boldsymbol{\theta}$ and data $\mathbf{x}$
Input: Parameters: learning rate η, batch size B
Input: Initialize:
1: $\boldsymbol{\theta}_0$ # initialized parameters
2: $t = 0$ # initialized time stamp
3: **while** Not Convergence **do**
4: $\quad t = t + 1$
5: $\quad \mathbf{g}_t = \frac{1}{B}\sum_{b=1}^{B}\nabla_\theta f_t(\mathbf{x}_b;\boldsymbol{\theta}_{t-1})$ # mini-batched gradient with current parameters $\boldsymbol{\theta}_{t-1}$
6: $\quad \boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \eta\cdot\mathbf{g}_t$
7: **end while**
Output: θ_t

[13] A limited memory version of the classic quasi-Newton optimization method BFGS algorithm, named after the authors of Broyden (1970), Fletcher (1970), Goldfarb (1970), and Shanno (1970).

Algorithm 5.3 Adam Optimizer (Kingma and Ba, 2015, Algorithm 1)

Input: $f(\mathbf{x};\boldsymbol{\theta})$: function to be optimized (minimized) with parameters $\boldsymbol{\theta}$ and data $\mathbf{x}$
Input: Parameters: learning rate $\eta = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$, batch size B
Input: Initialize:
θ_0 # initialized parameters
$m_0 = 0$ # initialized first moment
$v_0 = 0$ # initialized second moment
while Not Convergence **do**
  $t = t + 1$
  $\mathbf{g}_t = \frac{1}{B}\sum_{b=1}^{B} \nabla_\theta f_t(\mathbf{x}_b; \boldsymbol{\theta}_{t-1})$ # mini-batch gradient same as SGD
  $\mathbf{m}_t = \beta_1 \cdot \mathbf{m}_{t-1} + (1-\beta_1) \cdot \mathbf{g}_t$ # first moment estimate
  $\mathbf{v}_t = \beta_2 \cdot \mathbf{v}_{t-1} + (1-\beta_2) \cdot \mathbf{g}_t^2$ # second moment estimate
  $\widehat{\mathbf{m}}_t = \frac{\mathbf{m}_t}{1-\beta_1^t}$
  $\widehat{\mathbf{v}}_t = \frac{\mathbf{v}_t}{1-\beta_2^t}$
  $\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \eta \cdot \frac{\widehat{\mathbf{m}}_t}{\sqrt{\widehat{\mathbf{v}}_t} + \epsilon}$
end while
Output: θ_t

Algorithm 5.4 AdamW Optimizer (Loshchilov and Hutter, 2019, Algorithm 2)

Input: $f(\mathbf{x};\boldsymbol{\theta})$: function to be optimized (minimized) with parameters $\boldsymbol{\theta}$ and data $\mathbf{x}$
Input: Parameters: learning rate $\eta = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$, batch size B
Input: Decay parameter: τ
Input: Initialize:
θ_0 # initialized parameters
$m_0 = 0$ # initialized first moment
$v_0 = 0$ # initialized second moment
$\eta_0 \in \mathbb{R}$ # schedule multiplier
while Not Convergence **do**
  $t = t + 1$
  $\mathbf{g}_t = \frac{1}{B}\sum_{b=1}^{B} \nabla_\theta f_t(\mathbf{x}_b; \boldsymbol{\theta}_{t-1}) + \tau \cdot \boldsymbol{\theta}_{t-1}$ # mini-batch gradient same as SGD
  $\mathbf{m}_t = \beta_1 \cdot \mathbf{m}_{t-1} + (1-\beta_1) \cdot \mathbf{g}_t$ # first moment estimate
  $\mathbf{v}_t = \beta_2 \cdot \mathbf{v}_{t-1} + (1-\beta_2) \cdot \mathbf{g}_t^2$ # second moment estimate
  $\widehat{\mathbf{m}}_t = \frac{\mathbf{m}_t}{1-\beta_1^t}$
  $\widehat{\mathbf{v}}_t = \frac{\mathbf{v}_t}{1-\beta_2^t}$
  $\eta_t = \text{Schedule}(t)$
  $\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \eta_t \left(\eta \cdot \frac{\widehat{\mathbf{m}}_t}{\sqrt{\widehat{\mathbf{v}}_t} + \epsilon} + \tau \cdot \boldsymbol{\theta}_{t-1} \right)$
end while
Output: θ_t

6 Wasserstein Dictionary Learning with Closed-Form Gradients

From the Wasserstein Dictionary Learning problem described as Equation (44) in Section 5.2, we can describe the solution algorithm and its gradient calculation. Therefore, we could write the parameter updating algorithm as well. The routine has been described in Section 5.2, but now I give all the details in this subsection.

The loss function $\mathcal{L}$ here under consideration in this paper would be quadratic loss[14], i.e. $\mathcal{L}(\mathbf{p}, \mathbf{q}) = \|\mathbf{p} - \mathbf{q}\|_2^2$.

7 Wasserstein Index Generation Model

Xie (2020b)

8 *wig* Package in R for a WIG Implementation

Xie (2020a)

[14]Though other losses can also be used, for example see Schmitz et al. (2018, Table 1) for different loss functions and their gradients.

References

Abadi, Martin et al. (2016). "TensorFlow: A System for Large-Scale Machine Learning." In: *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pp. 265–283.

Alaya, Mokhtar Z. et al. (2019). "Screening Sinkhorn Algorithm for Regularized Optimal Transport."

Altschuler, Jason, Jonathan Weed, and Philippe Rigollet (2017). "Near-Linear Time Approximation Algorithms for Optimal Transport via Sinkhorn Iteration." In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon et al. Curran Associates, Inc., pp. 1964–1974.

Bacharach, Michael (1970). *Biproportional Matrices and Input-Output Change*. CUP Archive. 192 pp.

Benamou, Jean-David et al. (2015). "Iterative Bregman Projections for Regularized Transportation Problems." In: *SIAM Journal on Scientific Computing* 37.2, A1111–A1138. DOI: 10.1137/141000439.

Blei, David M. and John D. Lafferty (2009). "Topic Models." In: *Text Mining*. Chapman and Hall/CRC.

Blei, David M., Andrew Y. Ng, and Michael I. Jordan (2003). "Latent Dirichlet Allocation." In: *Journal of Machine Learning Research* 3 (Jan), pp. 993–1022.

Broyden, C. G. (1970). "The Convergence of a Class of Double-Rank Minimization Algorithms 1. General Considerations." In: *IMA Journal of Applied Mathematics* 6.1, pp. 76–90. DOI: 10.1093/imamat/6.1.76.

Cuturi, Marco (2013). "Sinkhorn Distances: Lightspeed Computation of Optimal Transport." In: *Advances in Neural Information Processing Systems 26*. Ed. by C. J. C. Burges et al. Curran Associates, Inc., pp. 2292–2300.

Fletcher, R. (1970). "A New Approach to Variable Metric Algorithms." In: *The Computer Journal* 13.3, pp. 317–322. DOI: 10.1093/comjnl/13.3.317.

Galichon, Alfred (2016). *Optimal Transport Methods in Economics*. Princeton University Press. 185 pp. DOI: 10.23943/princeton/9780691172767.001.0001.

Goldfarb, Donald (1970). "A Family of Variable-Metric Methods Derived by Variational Means." In: *Mathematics of Computation* 24.109, pp. 23–26. DOI: 10.2307/2004873.

Innes, Michael (2019). *Don't Unroll Adjoint: Differentiating SSA-form Programs*. DOI: 10.48550/arXiv.1810.07951. URL: http://arxiv.org/abs/1810.07951 (visited on 07/27/2022). Pre-published.

Kantorovich, L. V. (1942). "On the Translocation of Masses." In: *Dokl. Akad. Nauk. USSR (N.S.)* 37, pp. 199–201.

Karcher, Hermann (2014). *Riemannian Center of Mass and so Called Karcher Mean*. DOI: 10.48550/arXiv.1407.2087. URL: http://arxiv.org/abs/1407.2087 (visited on 07/28/2024). Pre-published.

Kingma, Diederick P and Jimmy Ba (2015). "Adam: A Method for Stochastic Optimization." In: *International Conference on Learning Representations (ICLR)*.

Knight, P. (2008). "The Sinkhorn-Knopp Algorithm: Convergence and Applications." In: *SIAM Journal on Matrix Analysis and Applications* 30.1, pp. 261–275. DOI: 10.1137/060659624.

Leal, Allan M. M. (2018). *Autodiff, a Modern, Fast and Expressive C++ Library for Automatic Differentiation*. https://autodiff.github.io.

Léonard, Christian (2013). "A Survey of the Schr\"odinger Problem and Some of Its Connections with Optimal Transport."

Liu, Dong C. and Jorge Nocedal (1989). "On the Limited Memory BFGS Method for Large Scale Optimization." In: *Mathematical Programming* 45.1, pp. 503–528. DOI: 10.1007/BF01589116.

Loshchilov, Ilya and Frank Hutter (2019). *Decoupled Weight Decay Regularization*. DOI: 10.48550/arXiv.1711.05101. URL: http://arxiv.org/abs/1711.05101 (visited on 08/01/2024). Pre-published.

Modin, Klas (2024). *On the Geometry and Dynamical Formulation of the Sinkhorn Algorithm for Optimal Transport*. DOI: 10.48550/arXiv.2309.09089. URL: http://arxiv.org/abs/2309.09089 (visited on 08/03/2024). Pre-published.

Monge, Gaspard (1781). "Memoire Sur La Theorie Des Deblais et Des Remblais." In: *Histoire de l'Academie Royale des Sciences de Paris*.

Nocedal, Jorge and Stephen Wright (2006). *Numerical Optimization*. 2nd edition. New York: Springer. 686 pp.

Paszke, Adam et al. (2017). "Automatic Differentiation in PyTorch." In: *NIPS-W*.

Peyré, Gabriel and Marco Cuturi (2019). "Computational Optimal Transport: With Applications to Data Science." In: *Foundations and Trends® in Machine Learning* 11.5-6, pp. 355–607. DOI: 10.1561/2200000073.

Santambrogio, Filippo (2015). *Optimal Transport for Applied Mathematicians: Calculus of Variations, PDEs, and Modeling*. Birkhäuser. 376 pp.

Schmitz, Morgan A. et al. (2018). "Wasserstein Dictionary Learning: Optimal Transport-Based Unsupervised Nonlinear Dictionary Learning." In: *SIAM Journal on Imaging Sciences* 11.1, pp. 643–678. DOI: 10.1137/17M1140431.

Schrödinger, Erwin (1931). *Über Die Umkehrung Der Naturgesetze*. Verlag der Akademie der Wissenschaften in Kommission bei Walter De Gruyter u ...

Shanno, D. F. (1970). "Conditioning of Quasi-Newton Methods for Function Minimization." In: *Mathematics of Computation* 24.111, pp. 647–656. DOI: 10.1090/S0025-5718-1970-0274029-X.

Sinkhorn, Richard (1964). "A Relationship Between Arbitrary Positive Matrices and Doubly Stochastic Matrices." In: *The Annals of Mathematical Statistics* 35.2, pp. 876–879. DOI: 10.1214/aoms/1177703591.

Sinkhorn, Richard and Paul Knopp (1967). "Concerning Nonnegative Matrices and Doubly Stochastic Matrices." In: *Pacific Journal of Mathematics* 21.2, pp. 343–348.

Villani, Cédric (2003). *Topics in Optimal Transportation*. Vol. 58. Graduate Studies in Mathematics. American Mathematical Society. DOI: 10.1090/gsm/058.

Villani, Cédric (2008). *Optimal Transport: Old and New*. Springer Science & Business Media. 970 pp.

Xie, Fangzhou (2020a). "Pruned Wasserstein Index Generation Model and Wigpy Package." In: *CARMA 2020 - 3rd International Conference on Advanced Research Methods and Analytics*. DOI: 10.4995/CARMA2020.2020.11557.

Xie, Fangzhou (2020b). "Wasserstein Index Generation Model: Automatic Generation of Time-Series Index with Application to Economic Policy Uncertainty." In: *Economics Letters* 186, p. 108874. DOI: 10.1016/j.econlet.2019.108874.

Xu, Hongteng et al. (2018). "Distilled Wasserstein Learning for Word Embedding and Topic Modeling." In: *Advances in Neural Information Processing Systems 31*. Ed. by S. Bengio et al. Curran Associates, Inc., pp. 1723–1732.

Appendix

Mathematical Notations

Mathematical notations used in this note:

- $\mathbf{a}$: lower case letters in bold denote vectors
- $\mathbf{A}$: upper case letters in bold denote matrices
- $\mathbb{1}_n$, $\mathbb{0}_n$: a vector of ones (or zeros) of length n
- Σ_n: probability simplex $\Sigma_n \equiv \left\{\mathbf{a} \in \mathbb{R}_+^n : \sum_{i=1}^n \mathbf{a}_i = 1\right\}$.
- $\langle \cdot, \cdot \rangle$: inner (dot) product of two matrices of same size
- $\odot$: element-wise (Hadamard) multiplication
- $\oslash$ or $\frac{1}{\mathbf{X}}$: element-wise (Hadamard) division
- $\log(\cdot)$, $\exp(\cdot)$: element-wise logarithm and exponential functions
- $\operatorname{diag}(\mathbf{x})$: create a diagonal matrix $n \times n$ from vector $\mathbf{x} \in \mathbb{R}^n$
- $\operatorname{vec} \mathbf{AB} = (I_m \otimes \mathbf{A}) \operatorname{vec} \mathbf{B} = \left(\mathbf{B}^\top \otimes I_k\right) \operatorname{vec} \mathbf{A}$: for $\mathbf{A} : k \times l$ and $\mathbf{B} : l \times m$
- $\mathcal{K}^{(m,n)}$: Commutation matrix such that $\mathcal{K}^{(m,n)} \operatorname{vec} \mathbf{A} = \operatorname{vec}\left(\mathbf{A}^\top\right)$ for a matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$