Optimal Transport Report

An implementation of RGAS and RAGAS[1] algorithms for the computation of the Projection Robust Wasserstein distance

Emilien Jemelen†, Christophe Morau†, Théo Morvan†

ENSAE & IPP M2 Data Science
May 2023

[1] *RGAS* : Riemannian Gradient Ascent with Sinkhorn Iteration ; *RAGAS* : Riemannian Adaptive Gradient Ascent with Sinkhorn Iteration

†With equal contribution

Abstract

This report describes our implementation of the RGAS and RAGAS algorithms for computing the Projection Robust Wasserstein (PRW) distance. We first provide an overview of the theoretical foundations of the algorithms as presented in Lin et al. [2020]. We then describe our implementation process, including the challenges we faced and how we addressed them. Finally, we present our numerical results using both simulated distributions (a fragmented hypercube) and real distributions (with the MNIST figures dataset). Our implementation was done in Python via JAX OTT toolbox, a Deep Learning framework. The results presented in this paper tend to confirm that the PRW distance behaves consistently in high-dimension and is therefore an alternative to the Wasserstein distance or the subspace robust Wasserstein (SRW). Further contributions to our implementation would demonstrate how it is also a faster alternative in such a setting.

Contents

1 Introduction

This report describes our work for the course Optimal Transport. We decided to implement different algorithms described in Lin et al. [2020], namely *RGAS* and *RAGAS*.

Being able to compute the distance between two distributions lies at the heart of Optimal Transport (OT), and the Wasserstein distance can appear as a natural function for it, as highlighted by Kantorovitch's Duality and his relaxation of Monge's problem (Beiglbock et al. [2010]). However, a significant barrier to the direct application of OT in Machine Learning lies in its statistical limitations: the complexity of approximating the Wasserstein distance between densities using samples of these densities grows exponentially in dimension. There have been many attempts to solve this issue, usually through regularization or simplification of the dual problem.

For instance, the Projection Robust Wasserstein (PRW) distance aims at looking for a k-dimensional subspace that maximizes the distance between the projected measures. However, two elements have prevented the PRW distance from being broadly adopted : its non-convexity and its non-smoothness. A convex relaxation of the PRW - the subspace robust Wasserstein (SRW) - was therefore introduced, still with better performances than the Wasserstein distance. However, Lin et al. [2020] prove that an efficient computation of the PRW with higher performances than the SRW distance is still possible using Riemannian optimization and that the relaxation is not necessarily needed in some cases.

We will first briefly describe the theoretical results of the paper, then highlight some pitfalls we had to overcome when implementing the algorithms, and present our numerical results. We first tested RGAS an RAGAS on a setting similar to the one used in Lin et al. [2020] (a fragmented hypercube), and then used the Mixed National Institute of Standards and Technology (MNIST) dataset for tests on real data. Both algorithm have been implemented in Python via JAX, a Deep Learning framework.

2 Theoretical foundations of RGAS and RAGAS algorithms

2.1 Rewritting of the optimization problem

Let μ and ν be two probability measures on R^d with a second order moment, $\Pi(\mu, \nu)$ the set of couplings between the two measures. The 2-Wasserstein distance can then be defined as follows :

$$W_2(\mu, \nu) := \left(\inf_{\pi \in \Pi(\mu,\nu)} \int \|x - y\|^2 d\pi(x, y) \right)^{1/2} \tag{1}$$

While the k-dimensional PRW distance is defined as follows :

$$P_k(\mu, \nu) := \sup_{E \in G_k} W_2(P_{E\#\mu}, P_{E\#\nu}) \tag{2}$$

where P_E is the orthogonal projection on E, $G_k := \{E \subseteq R \, | dim(E) = k)$ and $T_{\#\mu}$ is the push-forward operator. This means that it's the maximum Wasserstein distance one can achieve between the "pushed" measures when projected on a subspace of dimension k. However, as shown in Paty and Cuturi [2019], our problem can be reformulated as a max-min optimization problem:

$$P_k(\mu, \nu) = \max_{E \in G_k} \min_{\pi \in \Pi(\mu,\nu)} \left(\int \|P_E(x - y)\|^2 d\pi(x, y) \right)^{1/2} \tag{3}$$

In particular, for discrete measures, these previous equations can rewritten as follows :

$$\max_{U \in St(d,k)} \left(f(U) := \min_{\pi \in \Pi(\mu,\nu)} \sum_{i=1}^{n} \sum_{j=1}^{n} \pi_{i,j} \|U^T x_i - U^T y_j\|^2 \right) \tag{4}$$

where $St(d,k) := \{U \in R^{d \times k} | U^T U = I\}$ is the Stiefel manifold, $\Pi(\mu,\nu) := \{\pi \in R_+^{n \times n}, r(\pi) = r, c(\pi) = c\}$ is the transportation polytope and $c, r \in \Delta^n$ are the respective weights of μ and ν.

However, this formulation of the problem is not computationally solvable. Indeed, it is neither convex nor quasi-convex-concave, and both the orthogonality constraint (it requires computing a SVD decomposition) and the projection on the transportation polytope are computationally costly. Furthermore, the solution (the set of minimizers over the transportation polytope) might be composed of multiple elements, preventing the function $f(U)$ from being differentiable.

To solve this computation issue, an entropic regularization $H(\pi) := -\langle \pi, log(\pi) - \mathbf{1}_n \mathbf{1}_n \rangle$ can be added to our problem :

$$\max_{U \in St(d,k)} \left(f_\eta(U) := \min_{\pi \in \Pi(\mu,\nu)} \sum_{i=1}^{n} \sum_{j=1}^{n} \pi_{i,j} \|U^T x_i - U^T y_j\|^2 - \eta H(\pi) \right) \tag{5}$$

The function $f_\eta(U)$ admits now a unique minimum. However, η has to be chosen carefully : when it is too large, f_η becomes a poor approximation of our original function f.

2.2 Solving the problem using Riemannian optimization

For a given U, $f_\eta(U)$ is the solution of the regularized OT problem and can be solved using the Sinkhorn's algorithm. The maximization problem is trickier, considering that it is constrained over the Stiefel manifold. By endowing the Stiefel manifold with a Riemannian metric, it will however be possible to introduce the Riemann sub-gradient and then leverage it to perform a Riemannian gradient ascent to find the optimal U.

First, Lin et al. [2020] compute the differential of the function we seek to maximize f_η. To do so, the authors introduce the correlation matrix, defined as $V_\pi = \sum_{i=1}^{n} \sum_{j=1}^{n} \pi_{i,j}(x_i - y_j)(x_i - y_j)^T$. Then $f_\eta(U) = \min_{\pi \in \Pi(\mu,\nu)} \{\langle UU^T, V_\pi \rangle - \eta H(\pi)\}$. By Danskin's theorem, the smoothness of f_η and symmetry of V_π yield the following formula: $\nabla f_\eta(U) = 2V_{\pi^*(U)} U \; \forall U \in R^{d \times k}$.

The Riemannian subdifferential is the projection of ∇f_η over $T_U St = \{\xi : \xi^T U + U^T \xi = 0\}$, the tangent space to the Stiefel manifold at point U. By denoting $P_{T_U St}$ this projection operator, the Riemannian subdifferential is then defined as $f(U) = P_{T_U St}(\nabla f_\eta)$.

As such, the Riemannian subdifferential is a point belonging to the tangent space to the Stiefel manifold. Lin et al. [2020] finally use a retraction, a smooth mapping from the tangent space to the Stiefel manifold itself to get the update given by the Riemannian gradient ascent procedure.

Computing the differential of f_η in the first place requires to know the optimal transportation plan π^*, which is unknown. The idea of the RGAS algorithm is then to perform sequentially updates of the transportation plan π (using Sinkhorn for a given U) and of the subspace projection U (using Riemannian gradient ascent for a given π, that is used in place of π^*). In short, the RGAS algorithm does the following:

Overview of a RGAS iteration :

1. Update of π_{t+1} by solving the regularized OT with fixed U_t : this classic problem of Optimal Transport can be solved thanks to Sinkhorn's algorithm
2. Computation of V using π_{t+1} and then by multiplying it with U_t we obtain the differential of f. Then we project this on the constraint tangent (Riemannian differentiation)
3. We then apply our retractation to obtrain our new U_{t+1}

The RAGAS algorithm repeats the same procedure but additionally applies adaptive weight vectors to the differential to improve the search direction.

3 Numerical results

Our code and the results are fully available on our Colab workspace[2]. As described in the guidelines, we implemented the different algorithms through the JAX-OTT toolbox for more efficient computations.

RGAS and RAGAS algorithms require a cautious choice of hyper-parameters ϵ, L_1, L_2, θ, α, β and the scaling method for matrix C.

We chose to follow the guidelines of Lin et al. [2020] for the choice of the values of ϵ ($= 0.001$), α ($= 10^{-6}$) and β ($= 0.8$). After noticing the moderate impact a variation in θ had on the results of RGAS and RAGAS, we decided to let θ set to 1.

Having too high coefficients in the matrix C could cause numerical issues. We chose to scale the matrix by a constant factor (a multiplication of each coefficient by a float) determined empirically to optimize convergence in each scenario.

As for L_1 and L_2, we implemented an algorithm to determine their values after Proposition 2.1 of Lin et al. [2020].

3.1 Choice of L_1 and L_2

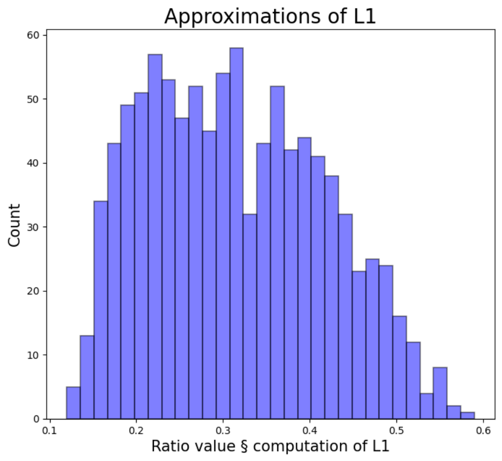
L_1 and L_2 are defined in Proposition 2.1 of Lin et al. [2020]. Since L1 and L2 are defined as upper bounds of functions which fundamentally take as input elements of the Stiefel manifold, we simulated Stiefel elements and took the empirical max of the functions to retrieve L1 and L2. This method yielded results (see *Figure 1* below).

This method was used with RGAS algorithm, and seemed to yield good results in terms of stability and speed of convergence. Yet, given the formula of γ in the RAGAS iterations and the impossibility to stabilize the denominator (and the convergence behaviour of the algorithm) by only playing with matrix C, we had to choose manually L_1 and L_2 (for RAGAS only).

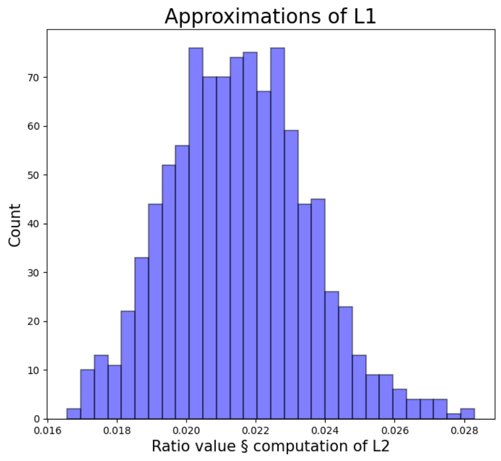
3.2 Tests of the algorithms on simulated data

To ensure that our implementations were working properly, we first performed tests on a fragmented uniform hypercube, following the same procedure as the one described in Lin et al. [2020]. We defined $\mu = U([-1,1]^d)$ and $\nu = T_{\#\mu}$ the push-forward measure of μ under the map T defined as $T(x) = x + 2sign(x) \odot (\sum_{k=1}^{k^*} e_k)$. As detailed in Lin et al. [2020], T is the subgradient of f a convex

[2]https://colab.research.google.com/drive/1jK9SOXBqIRUJZ9ovyyaM8yvWuSPbo-wx?usp=sharing



(a) Approximation of L1



(b) Approximation of L2

Figure 1: L_1 and L_2 with d = 15, n = 100, 1000 simulations

function and thus - by Brenier's theorem - an optimal transport map between μ and ν.

Our goal was to find the set of hyper-parameters such that both RGAS and RAGAS would converge to $P_k^2(\mu, \nu) = 4k^* = (W_2(\mu, \nu))^2$, the squared Wasserstein-2 distance between the distributions μ and ν. We tried out different set of hyper-parameters for both algorithms, as described in the following subsections.

3.2.1 Run of RGAS on fragmented hypercube settings

Setting of the hypercube and choice of specific hyper-parameters

We ran RGAS on a fixed simulation setting of n = 100 samples and d = 15 features uniformly distributed, but with different values of k^* : 2, 4, 7, 8. According to Lin et al. [2020], we expected post-convergence PRW values of 8, 16, 28 and 32 respectively. Hyper-parameters L_1 and L_2 were determined automatically by the method described in section 3.1 above.

Regarding the scaling factor applied to matrix C, which quickly appeared to have a great impact on the speed of convergence of RGAS, it was determined manually : a lower scaling factor would decrease $||C||_\infty$ and add more speed but more variability (with the risk of oscillating around the target value $4k^*$) to the iterations of RGAS, whereas a greater scaling factor would provide more stability to the evolution of the RGAS iterations, at the expense of the overall speed. So we seeked the lowest scaling factor of C such that RGAS would converge.

A scaling factor of 2.5e-5 quickly appeared to be a good compromise between a fast-but-possibly-oscillating convergence towards the target and a very-smooth-but-slow evolution. That would be our best scaling factor value for RAGAS algorithm too.

Definition of an alternative stopping criterion for RGAS and RAGAS

In addition, due to the infinite oscillation of RGAS around $4k^*$ for $k^* = 2$, with any scaling factor for C, and with values such that it was quite obvious that the algorithm had converged despite the 0.001 epsilon threshold not being precisely reached (for instance, epsilon values could oscillate around 0.0011 for thousands of iterations, with P_k values oscillating between 7.9 and 8.1), we defined an alternative stopping criterion, based on the moving average of the last 30 P_k values and their standard deviation.

Concretely, given a target value for P_k ($4k^*$ in this case) plus two additional hyper-parameters called *neighbourhood control level* and *std control level* (with default values set to 1), the algorithm would stop if the l_1-norm between P_k and the target were to be lower than *neighbourhood control level* and if the standard deviation of last 30 P_k values returned by RGAS were lower than *std control level.* The same alternative criterion is included in our definition of RAGAS algorithm. By default, the algorithms convergence was assessed with primary epsilon stopping criterion, but in some specific cases with established target closeliness and infinite oscillation around it we made use of the alternative stopping criterion.

Our results are illustrated in *figure 2* and in table 1.

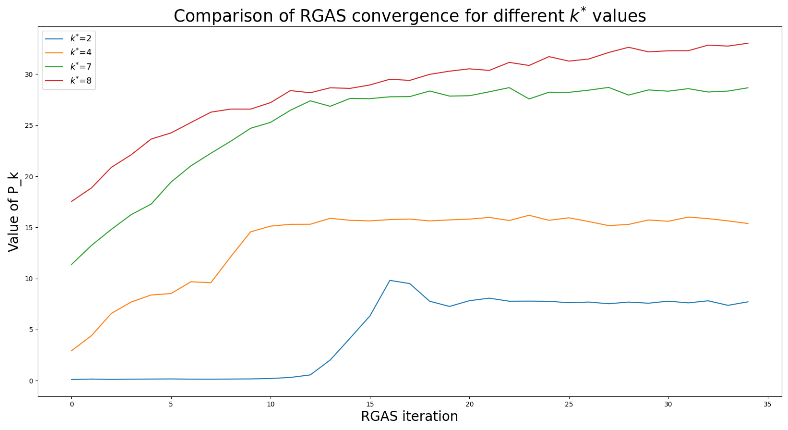


Figure 2: RGAS algorithm convergence for $k^* \in \{2, 4, 7, 8\}$ and C scaling factor = 2.5e-5

	PRW distance	Real value
2	7.74	8
4	15.82	16
7	28.21	28
8	32.66	32

Table 1: PRW distance with RGAS algorithm for $k^* \in \{2, 4, 7, 8\}$ vs real values

3.2.2 Run of RAGAS on fragmented hypercube settings

We performed similar tests for RAGAS, with some minor changes in the setting. Due to the unresolved instability of the algorithm with the values of L_1 and L_2 determined by our aforementioned method, we had to find manually adequate values for L_1 and L_2. These 2 hyper-parameters were highly sensitive, given the formula of γ. A search for the values providing the quickest stable convergence resulted in $L_1 = L_2 = 2.5\text{e-}8$, which appeared to be the most adequate value such that RAGAS is stable even for small k^* values.

Our results are illustrated by *figure 3* and in table 2. Similarly to RGAS, the number of iterations required to achieve convergences depends on the the value of k^*.

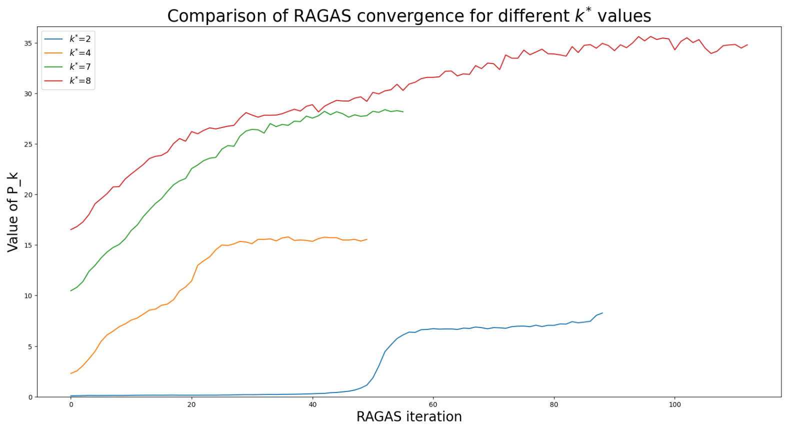


Figure 3: RAGAS algorithm convergence for $k^* \in \{2, 4, 7, 8\}$ and C scaling factor $= 2.5\text{e-}5$

	PRW distance	Real value
2	8.26	8
4	15.54	16
7	28.17	28
8	34.79	32

Table 2: PRW distance with RAGAS algorithm for $k^* \in \{2, 4, 7, 8\}$ vs real values

3.2.3 Comparison between different distances

After implementing RGAS and RAGAS on the fragmented hypercube setting, we compared the results obtained when using the Wasserstein distance, the SRW and the PRW computed both by RGAS and by RAGAS. As illustrated by *figure 4*, we can draw the following conclusions :

- We can empirically validate the fact that SRW and W2 are equivalent, as the blue and orange points coincide, irrespective of the value of k^*.
- We see that for k^* lower or equal than 7, PRW from RGAS and RAGAS are consistent with W2 and SRW. Yet, it seems that when k^* gets greater than 8, PRW from RGAS and RAGAS diverges from SRW and W2: this may be evidence that PRW is better suited for high-dimensional settings.

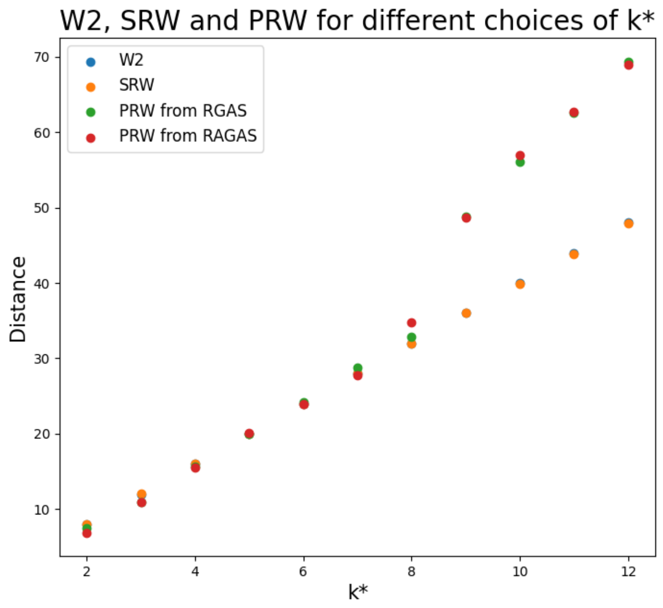


Figure 4: Comparison of distances for different values of k^*

3.3 Tests of RGAS and RAGAS on MNIST data

As practical implementation of the algorithms, we chose to test and measure the optimal way to transport the shape of a figure into the shape of another figure in two simple scenarii. Concretely, we tried to see if there were any differences between the transport of a 0 into a 1 and the transport of a 0 into an 8. We compared the PRW results returned by RGAS and RAGAS to W2 and SRW distances.

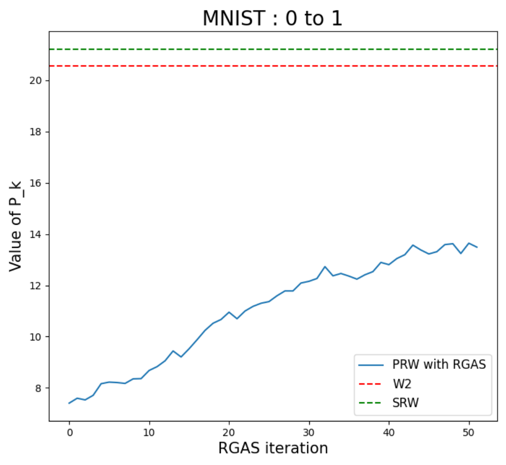
In terms of data, we chose the MNIST data first because of an intuition we wanted to measure: transporting a 0 into a 1 should be less direct than transporting a 0 into an 8 (so we expected the PRW distance between 0 and 1 to be higher than the PRW between 0 and 8). Second, we chose the MNIST data because images are by definition high-dimensional settings, and Lin et al. [2020] suggested that in some high-dimensional settings PRW might perform at least as well as SRW and W2 distances.

To capture the figure signal in any MNIST image and express the signal as a size-128 embedding, we trained a convolutional neural network which reached more than 99% of accuracy for figures classification after 3 epochs of training.

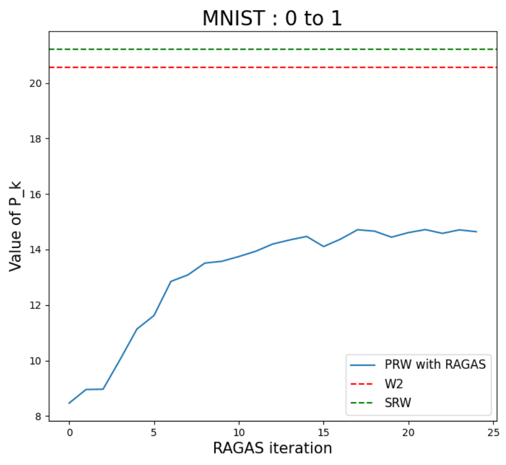
3.3.1 Transport from 0 to 1

As illustrated by *Figure 5*, we first observe that PRW from RGAS and RAGAS coincide more or less after convergence. This is a sanity check, since theoretically they should be equal.

Furthemore, the post-convergence PRW distances returned by both algorithms are lower than W2 and SRW in this high-dimensional setting.



(a) Convergence of RGAS algorithm



(b) Convergence of RAGAS algorithm

Figure 5: Convergences of RGAS and RAGAS ; transportation cost from 0 to 1, $k^* = 50$

3.3.2 Transport from 0 to 8

As illustrated by *Figure 6*, we observe again that PRW from RGAS and RAGAS coincide - broadly - after convergence.

Furthemore, the post-convergence PRW distances returned by both algorithms are once again lower than W2 and SRW in this high-dimensional setting.

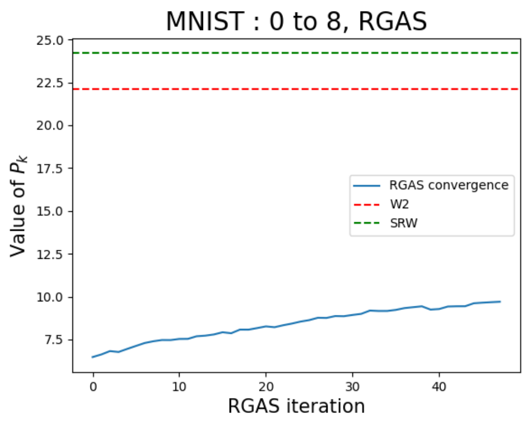
Overall, we observe that the post-convergence PRW distance between a 0 and a 1 (around 14) is greater than the post-convergence PRW distance between a 0 and an 8 (around 10). This result comforts the initial intuition that the shape of a 0 should be easier to *transport* into the shape of an 8 than into the shape of a 1.

4 Conclusion/discussion

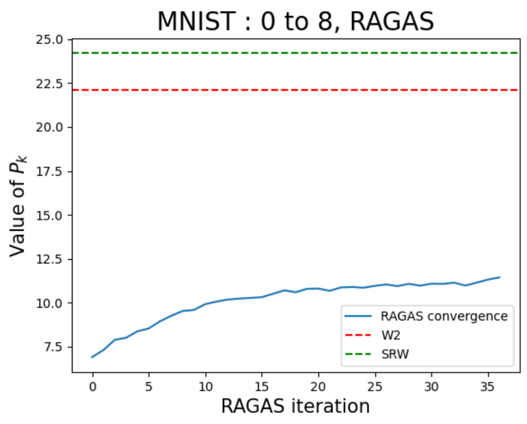
In this paper, we presented an implementation and evaluation of two algorithms, RGAS and RAGAS, for solving optimal transport problems. We tested these algorithms on simulated discrete distributions, and examined the effect of different hyper-parameters on their convergence properties. Our results show that both algorithms can converge to the optimal solution, although the choice of hyper-parameters is crucial to achieving convergence in a timely manner.

We found that the choice of scaling factor for matrix C is particularly important for RGAS, as it strongly affects the speed of convergence. Meanwhile, the choice of hyper-parameters for RAGAS is more intricate, since the value of γ in the RAGAS iterations cannot be stabilized by adjusting C alone. In this case, we had to choose values of L_1 and L_2 manually. Our results suggest that further research is needed to develop more automated methods for choosing hyper-parameters, especially for RAGAS.

Our implementation of the algorithms was based on the JAX-OTT library, which provided a convenient framework for performing the necessary computations efficiently. However, our code was designed



(a) Convergence of RGAS algorithm



(b) Convergence of RAGAS algorithm

Figure 6: Convergences of RGAS and RAGAS regarding transportation cost from 0 to 8, $k^* = 50$

as an illustrative tutorial on how RGAS and RAGAS work and not as a fully efficient implementation ready to be integrated to a dedicated package. As a consequence, it would have been unfair to compare the computational performances of our algorithms to the performances of the already optimized packages we used for SRW and W2. Further work would need to focus on improving the computation time of our implementation since this should be the comparative advantage of PRW over SRW and W2 in high-dimension.

Overall, our findings demonstrate the potential of these algorithms for solving optimal transport problems, and that the use of the Projection Robust Wasserstein distance can be relevant in some cases. The fragmented hypercube settings demonstrate that the PRW distance is consistent with SRW and W2 in high-dimension, while the MNIST example proves that RGAS and RAGAS can be used to perform distance comparisons on real-world high-dimensional data. Further contributions to our tutorial could seek to improve the computation time of our algorithms in order to examine whether PRW can be faster than SRW and W2 or not. Also, implementing tests to assess the robustness of the PRW distance to statistical noise could contribute to building a more precise report on the potential advantages of the PRW distance.

References

Mathias Beiglbock, Christian Leonard, and Walter Schachermayer. A general duality theorem for the monge-kantorovich transport problem. *arXiv:0911.4347*, 2010.

Tianyi Lin, Chenyou Fan, Nhat Ho, Marco Cuturi, and Michael I. Jordan. Projection robust wasserstein distance and riemannian optimization. *arXiv:2006.07458*, 2020.

F.-P. Paty and M. Cuturi. Subspace robust wasserstein distances. In *Proceedings of the 36th International Conference on Machine Learning*, pages 5072–5081, 2019.