# Fast Differentiable Sorting and Ranking

**Mathieu Blondel** [1] **Olivier Teboul** [1] **Quentin Berthet** [1] **Josip Djolonga** [1]

## Abstract

The sorting operation is one of the most commonly used building blocks in computer programming. In machine learning, it is often used for robust statistics. However, seen as a function, it is piecewise linear and as a result includes many kinks where it is non-differentiable. More problematic is the related ranking operator, often used for order statistics and ranking metrics. It is a piecewise constant function, meaning that its derivatives are null or undefined. While numerous works have proposed differentiable proxies to sorting and ranking, they do not achieve the $O(n \log n)$ time complexity one would expect from sorting and ranking operations. In this paper, we propose the first differentiable sorting and ranking operators with $O(n \log n)$ time and $O(n)$ space complexity. Our proposal in addition enjoys exact computation and differentiation. We achieve this feat by constructing differentiable operators as projections onto the permutahedron, the convex hull of permutations, and using a reduction to isotonic optimization. Empirically, we confirm that our approach is an order of magnitude faster than existing approaches and showcase two novel applications: differentiable Spearman's rank correlation coefficient and least trimmed squares.

## 1. Introduction

Modern deep learning architectures are built by composing parameterized functional blocks (including loops and conditionals) and are trained end-to-end using gradient backpropagation. This has motivated the term **differentiable programming**, recently popularized, among others, by LeCun (2018). Despite great empirical successes, many operations commonly used in computer programming remain poorly differentiable or downright pathological, limiting the set of architectures for which a gradient can be computed.

We focus in this paper on two such operations: **sorting** and **ranking**. Sorting returns the given input vector with its values re-arranged in monotonic order. It plays a key role to handle outliers in robust statistics, as in least-quantile (Rousseeuw, 1984) or trimmed (Rousseeuw & Leroy, 2005) regression. As a piecewise linear function, however, the sorted vector contains many kinks where it is non-differentiable. In addition, when used in composition with other functions, sorting often induces non-convexity, thus rendering model parameter optimization difficult.

The ranking operation, on the other hand, outputs the positions, or ranks, of the input values in the sorted vector. A workhorse of order statistics (David & Nagaraja, 2004), ranks are used in several metrics, including Spearman's rank correlation coefficient (Spearman, 1904), top-$k$ accuracy and normalized discounted cumulative gain (NDCG). As piecewise constant functions, ranks are unfortunately much more problematic than sorting: their derivatives are null or undefined, preventing gradient backpropagation. For this reason, a large body of work has studied differentiable proxies to ranking. While several works opt to approximate ranking metrics directly (Chapelle & Wu, 2010; Adams & Zemel, 2011; Lapin et al., 2016; Rolínek et al., 2020), others introduce "soft" ranks, which can then be plugged into any differentiable loss function. Taylor et al. (2008) use a random perturbation technique to compute expected ranks in $O(n^3)$ time, where $n$ is the dimensionality of the vector to rank. Qin et al. (2010) propose a simple method based on comparing pairwise distances between values, thereby taking $O(n^2)$ time. This method is refined by Grover et al. (2019) using unimodal row-stochastic matrices. Lastly, Cuturi et al. (2019) adopt an optimal transport viewpoint of sorting and ranking. Their method is based on differentiating through the iterates of the Sinkhorn algorithm (Sinkhorn & Knopp, 1967) and costs $O(Tmn)$ time, where $T$ is the number of Sinkhorn iterations and $m \in \mathbb{N}$ is a hyper-parameter which trades computational cost and precision (convergence to "hard" sort and ranks is only guaranteed if $m = n$).

In this paper, we propose the first differentiable sorting and ranking operators with $O(n \log n)$ time and $O(n)$ memory complexity. Our proposals enjoy **exact** computation and differentiation (i.e., they do not involve differentiating through

[1]Google Research, Brain team. Correspondence to: Mathieu Blondel <mblondel@google.com>, Olivier Teboul <oliviert@google.com>, Quentin Berthet <qberthet@google.com>, Josip Djolonga <josipd@google.com>.

*Proceedings of the $37^{th}$ International Conference on Machine Learning*, PMLR 119, 2020. 

the iterates of an approximate algorithm). We achieve this feat by casting differentiable sorting and ranking as projections onto the permutahedron, the convex hull of all permutations, and using a reduction to isotonic optimization. While the permutahedron had been used for learning before (Yasutake et al., 2011; Ailon et al., 2016; Blondel, 2019), it had not been used to define fast differentiable operators. The rest of the paper is organized as follows.

- We review the necessary background (§2) and show how to cast sorting and ranking as linear programs over the permutahedron, the convex hull of all permutations (§3).
- We introduce regularization in these linear programs, which turns them into projections onto the permutahedron and allows us to define differentiable sorting and ranking operators. We analyze the properties of these operators, such as their asymptotic behavior (§4).
- Using a reduction to isotonic optimization, we achieve $O(n \log n)$ computation and $O(n)$ differentiation of our operators, a key technical contribution of this paper (§5).
- We show that our approach is an order of magnitude faster than existing approaches and showcase two novel applications: differentiable Spearman's rank coefficient and soft least trimmed squares (§6).

## 2. Preliminaries

In this section, we define the notation that will be used throughout this paper. Let $\boldsymbol{\theta} \coloneqq (\theta_1, \dots, \theta_n) \in \mathbb{R}^n$. We will think of $\boldsymbol{\theta}$ as a vector of scores or "logits" produced by a model, i.e., $\boldsymbol{\theta} \coloneqq g(\boldsymbol{x})$ for some $g \colon \mathcal{X} \to \mathbb{R}^n$ and some $\boldsymbol{x} \in \mathcal{X}$. For instance, in a label ranking setting, $\boldsymbol{\theta}$ may contain the score of each of $n$ labels for the features $\boldsymbol{x}$.

We denote a **permutation** of $[n]$ by $\sigma = (\sigma_1, \dots, \sigma_n)$ and its inverse by $\sigma^{-1}$. For convenience, we will sometimes use $\pi \coloneqq \sigma^{-1}$. If a permutation $\sigma$ is seen as a vector, we denote it with bold, $\boldsymbol{\sigma} \in [n]^n$. We denote the set of $n!$ permutations of $[n]$ by $\Sigma$. Given a permutation $\sigma \in \Sigma$, we denote the version of $\boldsymbol{\theta} = (\theta_1, \dots, \theta_n) \in \mathbb{R}^n$ permuted according to $\sigma$ by $\boldsymbol{\theta}_\sigma \coloneqq (\theta_{\sigma_1}, \dots, \theta_{\sigma_n}) \in \mathbb{R}^n$. We define the reversing permutation by $\rho \coloneqq (n, n-1, \dots, 1)$ or $\boldsymbol{\rho}$ in vector form. Given a set $\mathcal{S} \subseteq [n]$ and a vector $\boldsymbol{v} \in \mathbb{R}^n$, we denote the restriction of $\boldsymbol{v}$ to $\mathcal{S}$ by $\boldsymbol{v}_\mathcal{S} \coloneqq (v_i \colon i \in \mathcal{S}) \in \mathbb{R}^{|\mathcal{S}|}$.

We define the **argsort** of $\boldsymbol{\theta}$ as the indices sorting $\boldsymbol{\theta}$, i.e.,

$$\sigma(\boldsymbol{\theta}) \coloneqq (\sigma_1(\boldsymbol{\theta}), \dots, \sigma_n(\boldsymbol{\theta})),$$

where $\theta_{\sigma_1(\boldsymbol{\theta})} \geq \dots \geq \theta_{\sigma_n(\boldsymbol{\theta})}$. If some of the coordinates of $\boldsymbol{\theta}$ are equal, we break ties arbitrarily. We define the **sort** of $\boldsymbol{\theta}$ as the values of $\boldsymbol{\theta}$ in descending order, i.e.,

$$s(\boldsymbol{\theta}) \coloneqq \boldsymbol{\theta}_{\sigma(\boldsymbol{\theta})}.$$

We define the **rank** of $\boldsymbol{\theta}$ as the function evaluating at coordinate $j$ to the position of $\theta_j$ in the descending sort (smaller rank $r_j(\boldsymbol{\theta})$ means that $\theta_j$ has higher value). It is formally equal to the argsort's inverse permutation, i.e.,

$$r(\boldsymbol{\theta}) \coloneqq \sigma^{-1}(\boldsymbol{\theta}).$$

For instance, if $\theta_3 \geq \theta_1 \geq \theta_2$, then $\sigma(\boldsymbol{\theta}) = (3, 1, 2)$, $s(\boldsymbol{\theta}) = (\theta_3, \theta_1, \theta_2)$ and $r(\boldsymbol{\theta}) = (2, 3, 1)$. All three operations can be computed in $O(n \log n)$ time. Note that throughout this paper, we use descending order for convenience. The ascending order counterparts are easily obtained by $\sigma(-\boldsymbol{\theta})$, $-s(-\boldsymbol{\theta})$ and $r(-\boldsymbol{\theta})$, respectively.

## 3. Sorting and ranking as linear programs

We show in this section how to cast sorting and ranking operations as linear programs over the permutahedron. To that end, we first formulate the argsort and ranking operations as optimization problems over the set of permutations $\Sigma$.

**Lemma 1.** *Discrete optimization formulations*

*For all $\boldsymbol{\theta} \in \mathbb{R}^n$ and $\boldsymbol{\rho} \coloneqq (n, n-1, \dots, 1)$, we have*

$$\sigma(\boldsymbol{\theta}) = \operatorname*{argmax}_{\sigma \in \Sigma} \langle \boldsymbol{\theta}_\sigma, \boldsymbol{\rho} \rangle, \text{ and} \tag{1}$$

$$r(\boldsymbol{\theta}) = \operatorname*{argmax}_{\pi \in \Sigma} \langle \boldsymbol{\theta}, \boldsymbol{\rho}_\pi \rangle. \tag{2}$$

A proof is provided in §B.1. To obtain continuous optimization problems, we introduce the permutahedron induced by a vector $\boldsymbol{w} \in \mathbb{R}^n$, the convex hull of permutations of $\boldsymbol{w}$:

$$\mathcal{P}(\boldsymbol{w}) \coloneqq \operatorname{conv}(\{\boldsymbol{w}_\sigma \colon \sigma \in \Sigma\}) \subset \mathbb{R}^n.$$

A well-known object in combinatorics (Bowman, 1972; Ziegler, 2012), the permutahedron of $\boldsymbol{w}$ is a convex polytope, whose vertices correspond to permutations of $\boldsymbol{w}$. It is illustrated in Figure 1. In particular, when $\boldsymbol{w} = \boldsymbol{\rho}$, $\mathcal{P}(\boldsymbol{w}) = \operatorname{conv}(\Sigma)$. With this defined, we can now derive linear programming formulations of sort and ranks.

**Proposition 1.** *Linear programming formulations*

*For all $\boldsymbol{\theta} \in \mathbb{R}^n$ and $\boldsymbol{\rho} \coloneqq (n, n-1, \dots, 1)$, we have*

$$s(\boldsymbol{\theta}) = \operatorname*{argmax}_{\boldsymbol{y} \in \mathcal{P}(\boldsymbol{\theta})} \langle \boldsymbol{y}, \boldsymbol{\rho} \rangle, \text{ and} \tag{3}$$

$$r(\boldsymbol{\theta}) = \operatorname*{argmax}_{\boldsymbol{y} \in \mathcal{P}(\boldsymbol{\rho})} \langle \boldsymbol{y}, -\boldsymbol{\theta} \rangle. \tag{4}$$

A proof is provided in §B.2. The key idea is to perform a change of variable to "absorb" the permutation in (1) and (2) into a permutahedron. From the fundamental theorem of linear programming (Dantzig et al., 1955, Theorem 6), an optimal solution of a linear program is almost surely achieved at a vertex of the convex polytope, a permutation in the case of the permutahedron. Interestingly, $\boldsymbol{\theta}$ appears in the constraints and $\boldsymbol{\rho}$ appears in the objective for sorting, while this is the opposite for ranking.

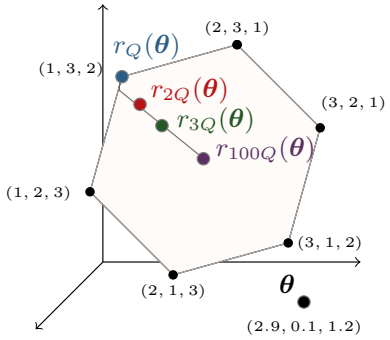


*Figure 1.* **Illustration of the permutahedron** $\mathcal{P}(\boldsymbol{\rho})$, whose vertices are permutations of $\boldsymbol{\rho} = (3, 2, 1)$. In this example, the ranks of $\boldsymbol{\theta} = (2.9, 0.1, 1.2)$ are $r(\boldsymbol{\theta}) = (1, 3, 2)$. In this case, our proposed soft rank $r_{\varepsilon Q}(\boldsymbol{\theta})$ with $\varepsilon = 1$ is exactly equal to $r(\boldsymbol{\theta})$. When $\varepsilon \to \infty$, $r_{\varepsilon Q}(\boldsymbol{\theta})$ converges towards the centroid of the permutahedron. The gray line indicates the regularization path of $r_{\varepsilon Q}(\boldsymbol{\theta})$ between these two regimes, when varying $\varepsilon$.

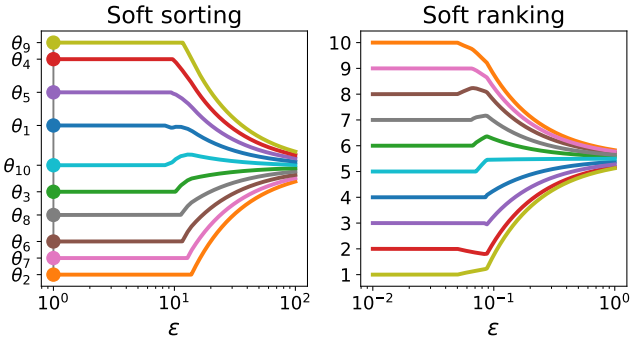


*Figure 2.* Illustration of the soft sorting and ranking operators, $s_{\varepsilon\Psi}(\boldsymbol{\theta})$ and $r_{\varepsilon\Psi}(\boldsymbol{\theta})$ for $\Psi = Q$; the results with $\Psi = E$ are similar. When $\varepsilon \to 0$, they converge to their "hard" counterpart. When $\varepsilon \to \infty$, they collapse into a constant, as proven in Prop.2.

**Differentiability a.e. of sorting.** For $s(\boldsymbol{\theta})$, the fact that $\boldsymbol{\theta}$ appears in the linear program constraints makes $s(\boldsymbol{\theta})$ piecewise linear and thus differentiable almost everywhere. When $\sigma(\boldsymbol{\theta})$ is unique at $\boldsymbol{\theta}$, $s(\boldsymbol{\theta}) = \boldsymbol{\theta}_{\sigma(\boldsymbol{\theta})}$ is differentiable at $\boldsymbol{\theta}$ and its Jacobian is the permutation matrix associated with $\sigma(\boldsymbol{\theta})$. When $\sigma(\boldsymbol{\theta})$ is not unique, we can choose any matrix in Clarke's generalized Jacobian, i.e., any convex combination of the permutation matrices associated with $\sigma(\boldsymbol{\theta})$.

**Lack of useful Jacobian of ranking.** On the other hand, for $r(\boldsymbol{\theta})$, since $\boldsymbol{\theta}$ appears in the objective, a small perturbation to $\boldsymbol{\theta}$ may cause the solution of the linear program to jump to another permutation of $\boldsymbol{\rho}$. This makes $r(\boldsymbol{\theta})$ a discontinuous, piecewise constant function. This means that $r(\boldsymbol{\theta})$ has null or undefined partial derivatives, preventing its use within a neural network trained with backpropagation.

## 4. Differentiable sorting and ranking

As we have already motivated, our primary goal is the design of efficiently computable approximations to the sorting and ranking operators, that would smoothen the numerous kinks of the former, and provide useful derivatives for the latter. We achieve this by introducing strongly convex regularization in our linear programming formulations. This turns them into efficiently computable projection operators, which are differentiable and amenable to formal analysis.

**Projection onto the permutahedron.** Let $\boldsymbol{z}, \boldsymbol{w} \in \mathbb{R}^n$ and consider the linear program $\operatorname{argmax}_{\boldsymbol{\mu} \in \mathcal{P}(\boldsymbol{w})} \langle \boldsymbol{\mu}, \boldsymbol{z} \rangle$. Clearly, we can express $s(\boldsymbol{\theta})$ by setting $(\boldsymbol{z}, \boldsymbol{w}) = (\boldsymbol{\rho}, \boldsymbol{\theta})$ and $r(\boldsymbol{\theta})$ by setting $(\boldsymbol{z}, \boldsymbol{w}) = (-\boldsymbol{\theta}, \boldsymbol{\rho})$. Introducing quadratic regularization $Q(\boldsymbol{\mu}) \coloneqq \frac{1}{2}\|\boldsymbol{\mu}\|^2$ is considered by Martins & Astudillo (2016) over the unit simplex and by Niculae et al. (2018) over marginal polytopes. Similarly, adding $Q$ to our linear program over the permutahedron gives

$$P_Q(\boldsymbol{z}, \boldsymbol{w}) \coloneqq \operatorname*{argmax}_{\boldsymbol{\mu} \in \mathcal{P}(\boldsymbol{w})} \langle \boldsymbol{z}, \boldsymbol{\mu} \rangle - Q(\boldsymbol{\mu}) = \operatorname*{argmin}_{\boldsymbol{\mu} \in \mathcal{P}(\boldsymbol{w})} \frac{1}{2}\|\boldsymbol{\mu} - \boldsymbol{z}\|^2,$$

i.e., the Euclidean projection of $\boldsymbol{z}$ onto $\mathcal{P}(\boldsymbol{w})$. We also consider entropic regularization $E(\boldsymbol{\mu}) \coloneqq \langle \boldsymbol{\mu}, \log \boldsymbol{\mu} - \mathbf{1} \rangle$, popularized in the optimal transport literature (Cuturi, 2013; Peyré & Cuturi, 2017). Subtly, we define

$$\begin{aligned} P_E(\boldsymbol{z}, \boldsymbol{w}) &\coloneqq \log \operatorname*{argmax}_{\boldsymbol{\mu} \in \mathcal{P}(e^{\boldsymbol{w}})} \langle \boldsymbol{z}, \boldsymbol{\mu} \rangle - E(\boldsymbol{\mu}) \\ &= \log \operatorname*{argmin}_{\boldsymbol{\mu} \in \mathcal{P}(e^{\boldsymbol{w}})} \mathrm{KL}(\boldsymbol{\mu}, e^{\boldsymbol{z}}), \end{aligned}$$

where $\mathrm{KL}(\boldsymbol{a}, \boldsymbol{b}) \coloneqq \sum_i a_i \log \frac{a_i}{b_i} - \sum_i a_i + \sum_i b_i$ is the Kullback-Leibler (KL) divergence between two positive measures $\boldsymbol{a} \in \mathbb{R}^n_+$ and $\boldsymbol{b} \in \mathbb{R}^n_+$. $P_E(\boldsymbol{z}, \boldsymbol{w})$ is therefore the *log* KL projection of $e^{\boldsymbol{z}}$ onto $\mathcal{P}(e^{\boldsymbol{w}})$. The purpose of $e^{\boldsymbol{w}}$ is to ensure that $\boldsymbol{\mu}$ always belongs to $\operatorname{dom}(E) = \mathbb{R}^n_+$ (since $\boldsymbol{\mu}$ is a convex combination of the permutations of $e^{\boldsymbol{w}}$) and that of the logarithm is to map $\boldsymbol{\mu}^\star$ back to $\mathbb{R}^n$.

More generally, we can use any strongly convex regularization $\Psi$ under mild conditions. For concreteness, we focus our exposition in the main text on $\Psi \in \{Q, E\}$. We state all our propositions for these two cases and postpone a more general treatment to the appendix.

**Soft operators.** We now build upon these projections to define soft sorting and ranking operators. To control the regularization strength, we introduce a parameter $\varepsilon > 0$ which we multiply $\Psi$ by (equivalently, divide $\boldsymbol{z}$ by).

For sorting, we choose $(\boldsymbol{z}, \boldsymbol{w}) = (\boldsymbol{\rho}, \boldsymbol{\theta})$ and therefore define the $\Psi$-regularized soft sort as

$$s_{\varepsilon\Psi}(\boldsymbol{\theta}) \coloneqq P_{\varepsilon\Psi}(\boldsymbol{\rho}, \boldsymbol{\theta}) = P_\Psi(\boldsymbol{\rho}/\varepsilon, \boldsymbol{\theta}). \tag{5}$$

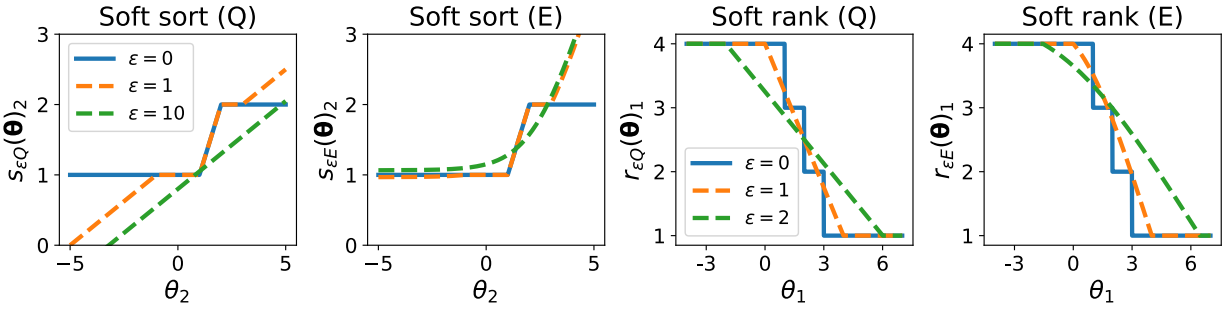


*Figure 3.* **Effect of the regularization parameter $\varepsilon$.** We take the vector $\boldsymbol{\theta} := (0, 3, 1, 2)$, vary one of its coordinates $\theta_i$ and look at how $[s_{\varepsilon\Psi}(\boldsymbol{\theta})]_i$ and $[r_{\varepsilon\Psi}(\boldsymbol{\theta})]_i$ change in response. For soft sorting with $\Psi = Q$, the function is still piecewise linear, like sorting. However, by increasing $\varepsilon$ we reduce the number of kinks, and the function eventually converges to a mean (Proposition 2). With $\Psi = E$, the function tends to be even smoother. For soft ranking with $\Psi = Q$, the function is piecewise linear instead of piecewise constant for the "hard" ranks. With $\Psi = E$, the function again tends to be smoother though it may contain kinks.

For ranking, we choose $(\boldsymbol{z}, \boldsymbol{w}) = (-\boldsymbol{\theta}, \boldsymbol{\rho})$ and therefore define the $\Psi$-regularized soft rank as

$$r_{\varepsilon\Psi}(\boldsymbol{\theta}) := P_{\varepsilon\Psi}(-\boldsymbol{\theta}, \boldsymbol{\rho}) = P_{\Psi}(-\boldsymbol{\theta}/\varepsilon, \boldsymbol{\rho}). \tag{6}$$

We illustrate the behavior of both of these soft operations as we vary $\varepsilon$ in Figures 2 and 3. As for the hard versions, the ascending-order soft sorting and ranking are obtained by negating the input as $-s_{\varepsilon\Psi}(-\boldsymbol{\theta})$ and $r_{\varepsilon\Psi}(-\boldsymbol{\theta})$, respectively.

**Properties.** We can further characterize these approximations. Namely, as we now formalize, they are differentiable a.e., and not only converge to the their "hard" counterparts, but also satisfy some of their properties for all $\varepsilon$.

**Proposition 2.** *Properties of $s_{\varepsilon\Psi}(\boldsymbol{\theta})$ and $r_{\varepsilon\Psi}(\boldsymbol{\theta})$*

1. **Differentiability.** *For all $\varepsilon > 0$, $s_{\varepsilon\Psi}(\boldsymbol{\theta})$ and $r_{\varepsilon\Psi}(\boldsymbol{\theta})$ are differentiable (a.e.) w.r.t. $\boldsymbol{\theta}$.*
2. **Order preservation.** *Let $\boldsymbol{s} := s_{\varepsilon\Psi}(\boldsymbol{\theta})$, $\boldsymbol{r} := r_{\varepsilon\Psi}(\boldsymbol{\theta})$ and $\sigma := \sigma(\boldsymbol{\theta})$. For all $\boldsymbol{\theta} \in \mathbb{R}^n$ and $0 < \varepsilon < \infty$, we have $s_1 \geq s_2 \geq \cdots \geq s_n$ and $r_{\sigma_1} \leq r_{\sigma_2} \leq \cdots \leq r_{\sigma_n}$.*
3. **Asymptotics.** *For all $\boldsymbol{\theta} \in \mathbb{R}^n$ without ties:*

$$s_{\varepsilon\Psi}(\boldsymbol{\theta}) \xrightarrow[\varepsilon\to 0]{} s(\boldsymbol{\theta}) \qquad r_{\varepsilon\Psi}(\boldsymbol{\theta}) \xrightarrow[\varepsilon\to 0]{} r(\boldsymbol{\theta})$$
$$\xrightarrow[\varepsilon\to\infty]{} f_\Psi(\boldsymbol{\theta})\mathbf{1} \qquad \xrightarrow[\varepsilon\to\infty]{} f_\Psi(\boldsymbol{\rho})\mathbf{1},$$

*where $f_Q(\boldsymbol{u}) := mean(\boldsymbol{u})$, $f_E(\boldsymbol{u}) := \log f_Q(\boldsymbol{u})$.*

The last property describes the behavior as $\varepsilon \to 0$ and $\varepsilon \to \infty$. Together with the proof of Proposition 2, we include in §B.3 a slightly stronger result. Namely, we derive an explicit value of $\varepsilon$ below which our operators are exactly equal to their hard counterpart, and a value of $\varepsilon$ above which our operators can be computed in closed form.

**Convexification effect.** Proposition 2 shows that $[s_{\varepsilon\Psi}(\boldsymbol{\theta})]_i$ and $[r_{\varepsilon\Psi}(\boldsymbol{\theta})]_i$ for all $i \in [n]$ converge to convex functions of $\boldsymbol{\theta}$ as $\varepsilon \to \infty$. This suggests that larger $\varepsilon$ make the objective function increasingly easy to optimize (at the cost of departing from "hard" sorting or ranking). This behavior is also visible in Figure 3, where $[s_{\varepsilon Q}(\boldsymbol{\theta})]_2$ converges towards the mean $f_Q$, depicted by a straight line.

**On tuning $\varepsilon$ (or not).** The parameter $\varepsilon > 0$ controls the trade-off between approximation of the original operator and "smoothness". When the model $g(\boldsymbol{x})$ producing the scores or "logits" $\boldsymbol{\theta}$ to be sorted/ranked is a homogeneous function, from (5) and (6), $\varepsilon$ can be absorbed into the model. In our label ranking experiment, we find that indeed tuning $\varepsilon$ is not necessary to achieve excellent accuracy. On the other hand, for top-$k$ classification, we find that applying a logistic map to squash $\boldsymbol{\theta}$ to $[0, 1]^n$ and tuning $\varepsilon$ is important, confirming the empirical finding of Cuturi et al. (2019).

**Relation to linear assignment formulation.** We now discuss the relation between our proposal and a formulation based on the Birkhoff polytope $\mathcal{B} \subset \mathbb{R}^{n\times n}$, the convex hull of permutation matrices. Our exposition corresponds to the method of Cuturi et al. (2019) with $m = n$. Note that using the change of variable $\boldsymbol{y} = \boldsymbol{P}\boldsymbol{\rho}$ and $\mathcal{P}(\boldsymbol{\rho}) = \mathcal{B}\boldsymbol{\rho}$, we can rewrite (4) as $r(\boldsymbol{\theta}) = \boldsymbol{P}(\boldsymbol{\theta})\boldsymbol{\rho}$, where

$$\boldsymbol{P}(\boldsymbol{\theta}) := \operatorname*{argmax}_{\boldsymbol{P}\in\mathcal{B}} \langle \boldsymbol{P}\boldsymbol{\rho}, -\boldsymbol{\theta}\rangle.$$

Let $D(\boldsymbol{a}, \boldsymbol{b}) \in \mathbb{R}^{n\times n}$ be a distance matrix. Simple calculations show that if $[D(\boldsymbol{a}, \boldsymbol{b})]_{i,j} := \frac{1}{2}(a_i - b_j)^2$, then

$$\boldsymbol{P}(\boldsymbol{\theta}) = \operatorname*{argmin}_{\boldsymbol{P}\in\mathcal{B}} \langle \boldsymbol{P}, D(-\boldsymbol{\theta}, \boldsymbol{\rho})\rangle.$$

Similarly, we can rewrite (3) as $s(\boldsymbol{\theta}) = \boldsymbol{P}(\boldsymbol{\theta})^\top\boldsymbol{\theta}$. To obtain a differentiable operator, Cuturi et al. (2019) (see also (Adams & Zemel, 2011)) propose to replace the permutation matrix $\boldsymbol{P}(\boldsymbol{\theta})$ by a doubly stochastic matrix $\boldsymbol{P}_{\varepsilon E}(\boldsymbol{\theta}) :=$

$\operatorname{argmin}_{\boldsymbol{P}\in\mathcal{B}} \langle \boldsymbol{P}, D(-\boldsymbol{\theta}, \boldsymbol{\rho})\rangle + \varepsilon E(\boldsymbol{P})$, which is computed approximately in $O(Tn^2)$ using Sinkhorn (1967). In comparison, our approach is based on regularizing $\boldsymbol{y} = \boldsymbol{P}\boldsymbol{\rho}$ with $\Psi \in \{Q, E\}$ directly, the key to achieve $O(n \log n)$ time and $O(n)$ space complexity, as we now show.

## 5. Fast computation and differentiation

As shown in the previous section, computing our soft sorting and ranking operators boils down to projecting onto a permutahedron. Our key contribution in this section is the derivation of an $O(n \log n)$ forward pass and an $O(n)$ backward pass (multiplication with the Jacobian) for these projections. Beyond soft sorting and ranking, this is an important sensitivity analysis question in its own right.

**Reduction to isotonic optimization.** We now show how to reduce the projections to isotonic optimization, i.e., with simple chain constraints, which is the key to fast computation and differentiation. We will w.l.o.g. assume that $\boldsymbol{w}$ is sorted in descending order (if not the case, we sort it first).

**Proposition 3.** *Reduction to isotonic optimization*

*For all $\boldsymbol{z} \in \mathbb{R}^n$ and sorted $\boldsymbol{w} \in \mathbb{R}^n$ we have*

$$P_\Psi(\boldsymbol{z}, \boldsymbol{w}) = \boldsymbol{z} - \boldsymbol{v}_\Psi(\boldsymbol{z}_{\sigma(\boldsymbol{z})}, \boldsymbol{w})_{\sigma^{-1}(\boldsymbol{z})}$$

*where*

$$\boldsymbol{v}_Q(\boldsymbol{s}, \boldsymbol{w}) \coloneqq \operatorname*{argmin}_{v_1 \geq \cdots \geq v_n} \frac{1}{2}\|\boldsymbol{v} - (\boldsymbol{s} - \boldsymbol{w})\|^2, \text{ and}$$
$$\boldsymbol{v}_E(\boldsymbol{s}, \boldsymbol{w}) \coloneqq \operatorname*{argmin}_{v_1 \geq \cdots \geq v_n} \langle e^{\boldsymbol{s}-\boldsymbol{v}}, \boldsymbol{1}\rangle + \langle e^{\boldsymbol{w}}, \boldsymbol{v}\rangle.$$

The function $\boldsymbol{v}_Q$ is classically known as isotonic regression. The fact that it can be used to solve the Euclidean projection onto $\mathcal{P}(\boldsymbol{w})$ has been noted several times (Negrinho & Martins, 2014; Zeng & Figueiredo, 2015). The reduction of Bregman projections, which we use here, to isotonic optimization was shown by Lim & Wright (2016). Unlike that study, we use the KL projection of $e^{\boldsymbol{z}}$ onto $\mathcal{P}(e^{\boldsymbol{w}})$, and not of $\boldsymbol{z}$ onto $\mathcal{P}(\boldsymbol{w})$, which simplifies many expressions. We include in §B.4 a simple unified proof of Proposition 3 based on Fenchel duality and tools from submodular optimization. We also discuss an interpretation of adding regularization to the primal linear program as relaxing the equality constraints of the dual linear program in §B.5.

**Computation.** As shown by Best et al. (2000), the classical pool adjacent violators (PAV) algorithm for isotonic regression can be extended to minimize any per-coordinate decomposable convex function $f(\boldsymbol{v}) = \sum_{i=1}^n f_i(v_i)$ subject to monotonicity constraints, which is exactly the form of the problems in Proposition 3. The algorithm repeatedly splits the coordinates into a set of contiguous blocks $\mathcal{B}_1, \ldots, \mathcal{B}_m$ that partition $[n]$ (their union is $[n]$ and $\max \mathcal{B}_j + 1 = \min \mathcal{B}_{j+1}$). It only requires access to an oracle that solves for each block $\mathcal{B}_j$ the sub-problem $\gamma(\mathcal{B}_j) = \operatorname{argmin}_{\gamma\in\mathbb{R}} \sum_{i\in\mathcal{B}_j} f_i(\gamma)$, and runs in **linear** time. Further, the solution has a clean block-wise constant structure, namely it is equal to $\gamma(\mathcal{B}_j)$ within block $\mathcal{B}_j$. Fortunately, in our case, as shown in §B.6, the function $\gamma$ can be analytically computed, as

$$\gamma_Q(\mathcal{B}_j; \boldsymbol{s}, \boldsymbol{w}) \coloneqq \frac{1}{|\mathcal{B}_j|} \sum_{i\in\mathcal{B}_j} s_i - w_i, \text{ and} \tag{7}$$

$$\gamma_E(\mathcal{B}_j; \boldsymbol{s}, \boldsymbol{w}) \coloneqq \log \sum_{i\in\mathcal{B}_j} e^{s_i} - \log \sum_{i\in\mathcal{B}_j} e^{w_i}. \tag{8}$$

Hence, PAV returns an **exact** solution of both $\boldsymbol{v}_Q(\boldsymbol{s}, \boldsymbol{w})$ and $\boldsymbol{v}_E(\boldsymbol{s}, \boldsymbol{w})$ in $O(n)$ time (Best et al., 2000). This means that we do not need to choose a number of iterations or a level of precision, unlike with Sinkhorn. Since computing $P_Q(\boldsymbol{z}, \boldsymbol{w})$ and $P_E(\boldsymbol{z}, \boldsymbol{w})$ requires obtaining $\boldsymbol{s} = \boldsymbol{z}_{\sigma(\boldsymbol{\theta})}$ beforehand, the total computational complexity is $O(n \log n)$.

**Differentiating isotonic optimization.** The block-wise structure of the solution also makes its derivatives easy to analyze, despite the fact that we are differentiating the *solution* of an optimization problem. Since the coordinates of the solution in block $\mathcal{B}_j$ are all equal to $\gamma(\mathcal{B}_j)$, which in turn depends only on a subset of the parameters, the Jacobian has a simple block-wise form, which we now formalize.

**Lemma 2.** *Jacobian of isotonic optimization*

*Let $\mathcal{B}_1, \ldots, \mathcal{B}_m$ be the ordered partition of $[n]$ induced by $\boldsymbol{v}_\Psi(\boldsymbol{s}, \boldsymbol{w})$ from Proposition 3. Then,*

$$\frac{\partial \boldsymbol{v}_\Psi(\boldsymbol{s}, \boldsymbol{w})}{\partial \boldsymbol{s}} = \begin{bmatrix} \mathbf{B}_1^\Psi & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \ddots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{B}_m^\Psi \end{bmatrix} \in \mathbb{R}^{n\times n},$$

*where $\mathbf{B}_j^\Psi \coloneqq \partial\gamma_\Psi(\mathcal{B}_j; \boldsymbol{s}, \boldsymbol{w})/\partial \boldsymbol{s} \in \mathbb{R}^{|\mathcal{B}_j|\times|\mathcal{B}_j|}$.*

A proof is given in §B.7. The Jacobians w.r.t. $\boldsymbol{w}$ are entirely similar, thanks to the symmetry of (7) and (8).

In the quadratic regularization case, it was already derived by Djolonga & Krause (2017) that $\mathbf{B}_j^Q \coloneqq \mathbf{1}/|\mathcal{B}_j|$. The multiplication with the Jacobian, $\boldsymbol{\nu} \coloneqq \frac{\partial \boldsymbol{v}_Q(\boldsymbol{s},\boldsymbol{w})}{\partial \boldsymbol{s}}\boldsymbol{u}$ for some vector $\boldsymbol{u}$, can be computed as $\boldsymbol{\nu} = (\boldsymbol{\nu}_1, \ldots, \boldsymbol{\nu}_m)$, where $\boldsymbol{\nu}_j = \operatorname{mean}(\boldsymbol{u}_{\mathcal{B}_j})\mathbf{1} \in \mathbb{R}^{|\mathcal{B}_j|}$. In the entropic regularization case, novel to our knowledge, we have $\mathbf{B}_j^E = \mathbf{1} \otimes \operatorname{softmax}(\boldsymbol{s}_{\mathcal{B}_j})$. Note that $\mathbf{B}_j^E$ is column-wise constant, so that the multiplication with the Jacobian $\boldsymbol{\nu} \coloneqq \frac{\partial \boldsymbol{v}_E(\boldsymbol{s},\boldsymbol{w})}{\partial \boldsymbol{s}}\boldsymbol{u}$, can be computed as $\boldsymbol{\nu}_j = \langle \operatorname{softmax}(\boldsymbol{s}_{\mathcal{B}_j}), \boldsymbol{u}_{\mathcal{B}_j}\rangle \mathbf{1} \in \mathbb{R}^{|\mathcal{B}_j|}$. In both cases, the multiplication with the Jacobian therefore takes $O(n)$ time.

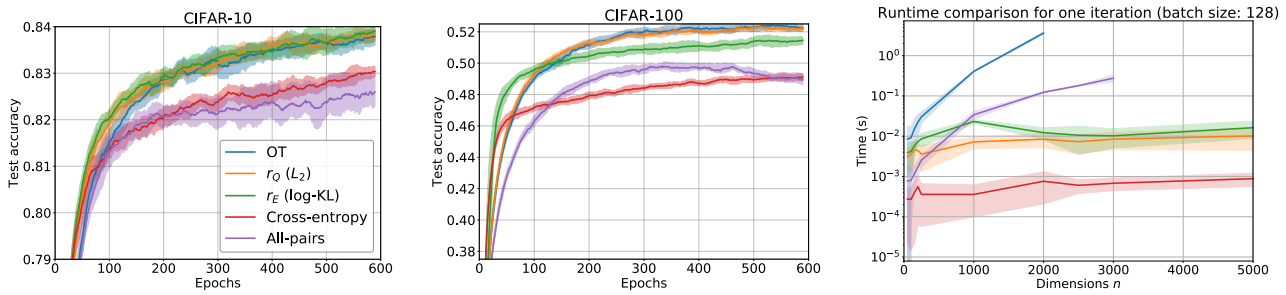


*Figure 4.* **Left, center:** Accuracy comparison on CIFAR-10, CIFAR-100 ($n = 10$, $n = 100$). **Right:** Runtime comparison for one batch computation with backpropagation disabled. OT and All-pairs go out-of-memory starting from $n = 2000$ and $n = 3000$, respectively. With backpropagation enabled, the runtimes are similar but OT and All-pairs go out-of-memory at $n = 1000$ and $n = 2500$, respectively.

There are interesting differences between the two forms of regularization. For quadratic regularization, the Jacobian only depends on the partition $\mathcal{B}_1, \ldots, \mathcal{B}_m$ (not on $\boldsymbol{s}$) and the blocks have constant value. For entropic regularization, the Jacobian does depend on $\boldsymbol{s}$ and the blocks are constant column by column. Both formulations are averaging the incoming gradients, one uniformly and the other weighted.

**Differentiating the projections.** We now combine Proposition 3 with Lemma 2 to characterize the Jacobians of the projections onto the permutahedron and show how to multiply arbitrary vectors with them in linear time.

**Proposition 4.** *Jacobian of the projections*

*Let $P_\Psi(\boldsymbol{z}, \boldsymbol{w})$ be defined in Proposition 3. Then,*

$$\frac{\partial P_\Psi(\boldsymbol{z}, \boldsymbol{w})}{\partial \boldsymbol{z}} = \boldsymbol{J}_\Psi(\boldsymbol{z}_{\sigma(\boldsymbol{z})}, \boldsymbol{w})_{\sigma^{-1}(\boldsymbol{z})},$$

*where $\boldsymbol{J}_\pi$ is the matrix obtained by permuting the rows and columns of $\boldsymbol{J}$ according to $\pi$, and where*

$$\boldsymbol{J}_\Psi(\boldsymbol{s}, \boldsymbol{w}) := \boldsymbol{I} - \frac{\partial \boldsymbol{v}_\Psi(\boldsymbol{s}, \boldsymbol{w})}{\partial \boldsymbol{s}}.$$

Again, the Jacobian w.r.t. $\boldsymbol{w}$ is entirely symmetric. Unlike the Jacobian of isotonic optimization, the Jacobian of the projection is not block diagonal, as we need to permute its rows and columns. We can nonetheless multiply with it in linear time by using the simple identity $(\boldsymbol{J}_\pi)\boldsymbol{z} = (\boldsymbol{J}\boldsymbol{z}_{\pi^{-1}})_\pi$, which allows us to reuse the $O(n)$ multiplication with the Jacobian of isotonic optimization.

**Differentiating $s_{\varepsilon\Psi}$ and $r_{\varepsilon\Psi}$.** With the Jacobian of $P_\Psi(\boldsymbol{z}, \boldsymbol{w})$ w.r.t. $\boldsymbol{z}$ and $\boldsymbol{w}$ at hand, differentiating $s_{\varepsilon\Psi}$ and $r_{\varepsilon\Psi}$ boils down to a mere application of the chain rule to (5) and (6). To summarize, we can multiply with the Jacobians of our soft operators in $O(n)$ time and space.

## 6. Experiments

We present in this section our empirical findings. NumPy, JAX, PyTorch and Tensorflow versions of our sorting and ranking operators are available at `https://github.com/google-research/fast-soft-sort/`.

### 6.1. Top-k classification loss function

**Experimental setup.** To demonstrate the effectiveness of our proposed soft rank operators as a drop-in replacement for exisiting ones, we reproduce the top-$k$ classification experiment of Cuturi et al. (2019). The authors propose a loss for top-$k$ classification between a ground truth class $y \in [n]$ and a vector of soft ranks $\boldsymbol{r} \in \mathbb{R}^n$, which is higher if the predicted soft ranks correctly place $y$ in the top-$k$ elements. We compare the following soft operators

- OT (Cuturi et al., 2019): optimal transport formulation.
- All-pairs (Qin et al., 2010): noting that $[r(\boldsymbol{\theta})]_i$ is equivalent to $\sum_{j \neq i} \mathbf{1}[\theta_i < \theta_j] + 1$, one can obtain soft ranks in $O(n^2)$ by replacing the indicator function with a sigmoid.
- Proposed: our $O(n \log n)$ soft ranks $r_Q$ and $r_E$. Although not used in this experiment, for top-$k$ ranking, the complexity can be reduced to $O(n \log k)$ by computing $P_\Psi$ using the algorithm of Lim & Wright (2016).

We use the CIFAR-10 and CIFAR-100 datasets, with $n = 10$ and $n = 100$ classes, respectively. Following Cuturi et al. (2019), we use a vanilla CNN (4 Conv2D with 2 max-pooling layers, ReLU activation, 2 fully connected layers with batch norm on each), the ADAM optimizer (Kingma & Ba, 2014) with a constant step size of $10^{-4}$, and set $k = 1$. Similarly to Cuturi et al. (2019), we found that squashing the scores $\boldsymbol{\theta}$ to $[0, 1]^n$ with a logistic map was beneficial.

**Results.** Our empirical results, averaged over 12 runs, are shown in Figure 4 (left, center). On both CIFAR-10 and CIFAR-100, our soft rank formulations achieve comparable accuracy to the OT formulation, though significantly faster, as we elaborate below. Confirming the results of Cuturi

et al. (2019), we found that the soft top-$k$ loss slightly outperforms the classical cross-entropy (logistic) loss for these two datasets. However, we did not find that the All-pairs formulation could outperform the cross-entropy loss.

The training times for 600 epochs on CIFAR-100 were 29 hours (OT), 21 hours ($r_Q$), 23 hours ($r_E$) and 16 hours (All-pairs). Training times on CIFAR-10 were similar. While our soft operators are several hours faster than OT, they are slower than All-pairs, despite its $O(n^2)$ complexity. This is due the fact that, with $n = 100$, All-pairs is very efficient on GPUs, while our PAV implementation runs on CPU.

### 6.2. Runtime comparison: effect of input dimension

To measure the impact of the dimensionality $n$ on the runtime of each method, we designed the following experiment.

**Experimental setup.** We generate score vectors $\boldsymbol{\theta} \in \mathbb{R}^n$ randomly according to $\mathcal{N}(0, 1)$, for $n$ ranging from 100 up to 5000. For fair comparison with GPU implementations (OT, All-pairs, Cross-entropy), we create a batch of 128 such vectors and we compare the time to compute soft ranking operators on this batch. We run this experiment on top of TensorFlow (Abadi et al., 2016) on a six core Intel Xeon W-2135 with 64 GBs of RAM and a GeForce GTX 1080 Ti.

**Results.** Run times for one batch computation with backpropagation disabled are shown in Figure 4 (Right). While their runtime is reasonable in small dimension, OT and All-pairs scale quadratically with respect to the dimensionality $n$ (note the log scale on the $y$-axis). Although slower than a softmax, our formulations scale well, with the dimensionality $n$ having negligible impact on the runtime. OT and All-pairs go out-of-memory starting from $n = 2000$ and $n = 3000$, respectively. With backpropagation enabled, they go out-of-memory at $n = 1000$ and $n = 2500$, due to the need for recording the computational graph. This shows that the lack of memory available on GPUs is problematic for these methods. In contrast, our approaches only require $O(n)$ memory and comes with the theoretical Jacobian (they do not rely on differentiating through iterates). They therefore suffer from no such issues.

### 6.3. Label ranking via soft Spearman's rank correlation coefficient

We now consider the label ranking setting where supervision is given as full rankings (e.g., $2 \succ 1 \succ 3 \succ 4$) rather than as label relevance scores. The goal is therefore to learn to predict permutations, i.e., a function $f_{\boldsymbol{w}} \colon \mathcal{X} \to \Sigma$. A classical metric between ranks is Spearman's rank correlation coefficient, defined as the Pearson correlation coefficient between the ranks. Maximizing this coefficient is equivalent to minimizing the squared loss between ranks. A naive idea would be therefore to use as loss $\frac{1}{2}\|\boldsymbol{r} - r(\boldsymbol{\theta})\|^2$, where

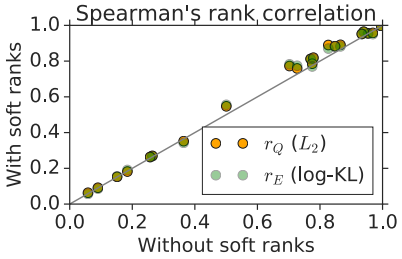


*Figure 5.* **Label ranking accuracy** with and without soft rank layer. Each point above the line represents a dataset where our soft rank layer improves Spearman's rank correlation coefficient.

$\boldsymbol{\theta} = g_{\boldsymbol{w}}(\boldsymbol{x})$. This is unfortunately a discontinuous function of $\boldsymbol{\theta}$. We therefore propose to rather use $\frac{1}{2}\|\boldsymbol{r} - r_\Psi(\boldsymbol{\theta})\|^2$, hence the name differentiable Spearman's rank correlation coefficient. At test time, we replace $r_\Psi$ with $r$, which is justified by the order-preservation property (Proposition 2).

**Experimental setup.** We consider the 21 datasets from (Hüllermeier et al., 2008; Cheng et al., 2009), which has both semi-synthetic data obtained from classification problems, and real biological measurements. Following (Korba et al., 2018), we average over two 10-fold validation runs, in each of which we train on 90% and evaluate on 10% of the data. Within each repetition, we run an internal 5-fold cross-validation to grid-search for the best parameters. We consider linear models of the form $g_{W,\boldsymbol{b}}(\boldsymbol{x}) = W\boldsymbol{x} + \boldsymbol{b}$, and for ablation study we drop the soft ranking layer $r_\Psi$.

**Results.** Due to the large number of datasets, we choose to present a summary of the results in Figure 5. We postpone detailed results to the appendix (Table 1). Out of 21 datasets, introducing a soft rank layer with $\Psi = Q$ works better on 15 datasets, similarly on 4 and worse on 2 datasets. We can thus conclude that even for such simple model, introducing our layer is beneficial, and even achieving state of the art results on some of the datasets (full details in the appendix).

### 6.4. Robust regression via soft least trimmed squares

We explore in this section the application of our soft sorting operator $s_{\varepsilon\Psi}$ to robust regression. Let $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in \mathcal{X} \subseteq \mathbb{R}^d$ and $y_1, \ldots, y_n \in \mathcal{Y} \subseteq \mathbb{R}$ be a training set of input-output pairs. Our goal is to learn a model $g_{\boldsymbol{w}} \colon \mathbb{R}^d \to \mathbb{R}$ that predicts outputs from inputs, where $\boldsymbol{w}$ are model parameters. We focus on $g_{\boldsymbol{w}}(\boldsymbol{x}) \coloneqq \langle \boldsymbol{w}, \boldsymbol{x} \rangle$ for simplicity. We further assume that a certain proportion of examples are outliers including some label noise, which makes the task of robustly estimating $g_{\boldsymbol{w}}$ particularly challenging.

The classical ridge regression can be cast as

$$\min_{\boldsymbol{w}} \frac{1}{n} \sum_{i=1}^{n} \ell_i(\boldsymbol{w}) + \frac{1}{2\varepsilon}\|\boldsymbol{w}\|^2, \tag{9}$$

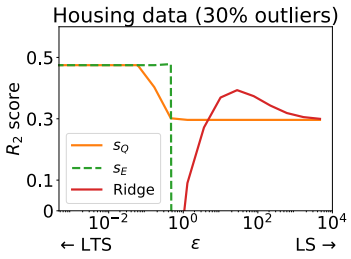


*Figure 6.* Empirical validation of interpolation between LTS and LS.

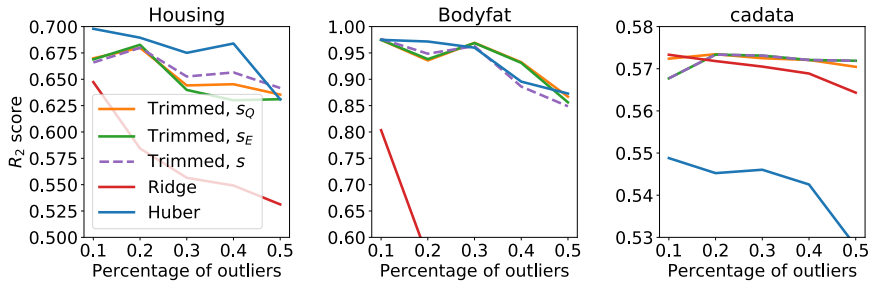


*Figure 7.* $R_2$ score (higher is better) averaged over 10 train-time splits for increasing percentage of outliers. Hyper-parameters are tuned by 5-fold cross-validation.

where $\ell_i(\boldsymbol{w}) := \frac{1}{2}(y_i - g_{\boldsymbol{w}}(\boldsymbol{x}_i))^2$. In order to be robust to label noise, we propose instead to sort the losses (from larger to smaller) and to ignore the first $k$ ones. Introducing our soft sorting operator, this can be formulated as

$$\min_{\boldsymbol{w}} \frac{1}{n-k} \sum_{i=k+1}^{n} \ell_i^{\varepsilon}(\boldsymbol{w}) \tag{10}$$

where $\ell_i^{\varepsilon}(\boldsymbol{w}) := [s_{\varepsilon\Psi}(\boldsymbol{\ell}(\boldsymbol{w}))]_i$ is the $i^{\text{th}}$ loss in the soft sort, and $\boldsymbol{\ell}(\boldsymbol{w}) \in \mathbb{R}^n$ is the loss vector that gathers $\ell_i(\boldsymbol{w})$ for each $i \in [n]$. Solving (10) with existing $O(n^2)$ soft sorting operators could be particularly computationally prohibitive, since here $n$ is the number of training samples.

When $\varepsilon \to 0$, we have $s_{\varepsilon\Psi}(\boldsymbol{\ell}(\boldsymbol{w})) \to s(\boldsymbol{\ell}(\boldsymbol{w}))$ and (10) is known as least trimmed squares (LTS) (Rousseeuw, 1984; Rousseeuw & Leroy, 2005). When $\varepsilon \to \infty$, we have, from Proposition 2, $s_{\varepsilon\Psi}(\boldsymbol{\ell}(\boldsymbol{w})) \to \text{mean}(\boldsymbol{\ell}(\boldsymbol{w}))\mathbf{1}$ and therefore both (9) and (10) converge to the least squares (LS) objective, $\min_{\boldsymbol{w}} \frac{1}{n}\sum_{i=1}^{n} \ell_i(\boldsymbol{w})$. To summarize, our proposed objective (10), dubbed soft least trimmed squares, **interpolates** between least trimmed squares ($\epsilon \to 0$) and least squares ($\epsilon \to \infty$), as also confirmed empirically in Figure 6.

**Experimental setup.** To empirically validate our proposal, we compare cross-validated results for increasing percentage of outliers of the following methods:

- Least trimmed squares, with truncation parameter $k$,
- Soft least trimmed squares (10), with truncation parameter $k$ and regularization parameter $\varepsilon$,
- Ridge regression (9), with regularization parameter $\varepsilon$,
- Huber loss (Huber, 1964) with regularization parameter $\varepsilon$ and threshold parameter $\tau$, as implemented in scikit-learn (Pedregosa et al., 2011).

We consider datasets from the LIBSVM archive (Fan & Lin, 2011). We hold out 20% of the data as test set and use the rest as training set. We artifically create outliers, by adding noise to a certain percentage of the training labels, using $y_i \leftarrow y_i + e$, where $e \sim \mathcal{N}(0, 5 \times \text{std}(\boldsymbol{y}))$. We do not add noise to the test set. For all methods, we use L-BFGS (Liu & Nocedal, 1989), with a maximum of 300 iterations. For hyper-parameter optimization, we use 5-fold cross-validation. We choose $k$ from $\{\lceil 0.1n \rceil, \lceil 0.2n \rceil, \ldots, \lceil 0.5n \rceil\}$, $\varepsilon$ from 10 log-spaced values between $10^{-3}$ and $10^4$, and $\tau$ from 5 linearly spaced values between 1.3 and 2. We repeat this procedure 10 times with a different train-test split, and report the averaged $R_2$ scores (a.k.a. coefficient of determination).

**Results.** The averaged $R_2$ scores (higher is better) are shown in Figure 7. On all datasets, the accuracy of ridge regression deteriorated significantly with increasing number of outliers. Least trimmed squares (hard or soft) performed slightly worse than the Huber loss on housing, comparably on bodyfat and much better on cadata. We found that hard least trimmed squares (i.e., $\varepsilon = 0$) worked well on all datasets, showing that regularization is less important for sorting operators (which are piecewise linear) than for ranking operators (which are piecewise constant). Nevertheless, regularization appeared useful in some cases. For instance, on cadata, the cross-validation procedure picked $\varepsilon > 1000$ when the percentage of outliers is less than 20%, and $\varepsilon < 10^{-3}$ when the percentage of outliers is larger than 20%. This is confirmed visually on Figure 7, where the soft sort with $\Psi = Q$ works slightly better than the hard sort with few outliers, then performs comparably with more outliers. The interpolation effect enabled by $\varepsilon$ therefore allows some adaptivity to the (unknown) percentage of outliers.

## 7. Conclusion

Building upon projections onto permutahedra, we constructed differentiable sorting and ranking operators. We derived **exact** $O(n \log n)$ computation and $O(n)$ differentiation of these operators, a key technical contribution of this paper. We demonstrated that our operators can be used as a drop-in replacement for existing $O(n^2)$ ones, with an order-of-magnitude speed-up. We also showcased two applications enabled by our soft operators: label ranking with differentiable Spearman's rank correlation coefficient and robust regression via soft least trimmed squares.

## Acknowledgements

We are grateful to Marco Cuturi and Jean-Philippe Vert for useful discussions, and to Carlos Riquelme for comments on a draft of this paper.

## References

Abadi, M. et al. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation*, pp. 265–283, 2016.

Adams, R. P. and Zemel, R. S. Ranking via sinkhorn propagation. *arXiv e-prints*, 2011.

Ailon, N., Hatano, K., and Takimoto, E. Bandit online optimization over the permutahedron. *Theoretical Computer Science*, 650:92–108, 2016.

Bach, F. Learning with submodular functions: A convex optimization perspective. *Foundations and Trends® in Machine Learning*, 6(2-3):145–373, 2013.

Best, M. J., Chakravarti, N., and Ubhaya, V. A. Minimizing separable convex functions subject to simple chain constraints. *SIAM Journal on Optimization*, 10(3):658–672, 2000.

Blondel, M. Structured prediction with projection oracles. In *Proc. of NeurIPS*, 2019.

Blondel, M., Martins, A. F., and Niculae, V. Learning classifiers with Fenchel-Young losses: Generalized entropies, margins, and algorithms. In *Proc. of AISTATS*, 2019.

Blondel, M., Martins, A. F., and Niculae, V. Learning with Fenchel-Young losses. *arXiv preprint arXiv:1901.02324*, 2019.

Bowman, V. Permutation polyhedra. *SIAM Journal on Applied Mathematics*, 22(4):580–589, 1972.

Chapelle, O. and Wu, M. Gradient descent optimization of smoothed information retrieval metrics. *Information retrieval*, 13(3):216–235, 2010.

Cheng, W., Hühn, J., and Hüllermeier, E. Decision tree and instance-based learning for label ranking. In *International Conference on Machine Learning (ICML-09)*, 2009.

Cuturi, M. Sinkhorn distances: Lightspeed computation of optimal transport. In *Proc. of NeurIPS*, 2013.

Cuturi, M., Teboul, O., and Vert, J.-P. Differentiable ranking and sorting using optimal transport. In *Proc. of NeurIPS*, 2019.

Dantzig, G. B., Orden, A., and Wolfe, P. The generalized simplex method for minimizing a linear form under linear inequality restraints. *Pacific Journal of Mathematics*, 5 (2):183–195, 1955.

David, H. A. and Nagaraja, H. N. Order statistics. *Encyclopedia of Statistical Sciences*, 2004.

Djolonga, J. and Krause, A. Differentiable learning of submodular models. In *Proc. of NeurIPS*, pp. 1013–1023, 2017.

Edmonds, J. Submodular functions, matroids, and certain polyhedra. In *Combinatorial structures and their applications*, pp. 69–87, 1970.

Fan, R.-E. and Lin, C.-J. LIBSVM datasets, 2011.

Grover, A., Wang, E., Zweig, A., and Ermon, S. Stochastic optimization of sorting networks via continuous relaxations. *arXiv preprint arXiv:1903.08850*, 2019.

Huber, P. J. Robust estimation of a location parameter. *The Annals of Mathematical Statistics*, 35(1):73–101, 1964.

Hüllermeier, E., Fürnkranz, J., Cheng, W., and Brinker, K. Label ranking by learning pairwise preferences. *Artificial Intelligence*, 172, 2008.

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

Korba, A., Garcia, A., and d'Alché Buc, F. A structured prediction approach for label ranking. In *Proc. of NeurIPS*, 2018.

Lapin, M., Hein, M., and Schiele, B. Loss functions for top-k error: Analysis and insights. In *Proc. of CVPR*, 2016.

LeCun, Y. Deep Learning est mort. Vive Differentiable Programming!, 2018.

Lim, C. H. and Wright, S. J. Efficient bregman projections onto the permutahedron and related polytopes. In *Proc. of AISTATS*, pp. 1205–1213, 2016.

Liu, D. C. and Nocedal, J. On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, 45(1):503–528, 1989.

Martins, A. F. and Astudillo, R. F. From softmax to sparsemax: A sparse model of attention and multi-label classification. In *Proc. of ICML*, 2016.

Negrinho, R. and Martins, A. Orbit regularization. In *Proc. of NeurIPS*, 2014.

Niculae, V., Martins, A. F., Blondel, M., and Cardie, C. SparseMAP: Differentiable sparse structured inference. In *Proc. of ICML*, 2018.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Peyré, G. and Cuturi, M. *Computational Optimal Transport*. Foundations and Trends in Machine Learning, 2017.

Qin, T., Liu, T.-Y., and Li, H. A general approximation framework for direct optimization of information retrieval measures. *Information retrieval*, 13(4):375–397, 2010.

Rolínek, M., Musil, V., Paulus, A., Vlastelica, M., Michaelis, C., and Martius, G. Optimizing rank-based metrics with blackbox differentiation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7620–7630, 2020.

Rousseeuw, P. J. Least median of squares regression. *Journal of the American statistical association*, 79(388):871–880, 1984.

Rousseeuw, P. J. and Leroy, A. M. *Robust regression and outlier detection*, volume 589. John wiley & sons, 2005.

Sinkhorn, R. and Knopp, P. Concerning nonnegative matrices and doubly stochastic matrices. *Pacific Journal of Mathematics*, 21(2):343–348, 1967.

Spearman, C. The proof and measurement of association between two things. *American Journal of Psychology*, 1904.

Suehiro, D., Hatano, K., Kijima, S., Takimoto, E., and Nagano, K. Online prediction under submodular constraints. In *International Conference on Algorithmic Learning Theory*, pp. 260–274, 2012.

Taylor, M., Guiver, J., Robertson, S., and Minka, T. Softrank: optimizing non-smooth rank metrics. In *Proceedings of the 2008 International Conference on Web Search and Data Mining*, pp. 77–86, 2008.

Yasutake, S., Hatano, K., Kijima, S., Takimoto, E., and Takeda, M. Online linear optimization over permutations. In *International Symposium on Algorithms and Computation*, pp. 534–543. Springer, 2011.

Zeng, X. and Figueiredo, M. A. The ordered weighted $\ell_1$ norm: Atomic formulation and conditional gradient algorithm. In *Proc. of SPARS*, 2015.

Ziegler, G. M. *Lectures on polytopes*, volume 152. Springer Science & Business Media, 2012.