Fast differentiable sorting and ranking

M. Blondel  O. Teboul  Q. Berthet  J. Djolonga

March 12th, 2020
Background

Proposed method

Experimental results
DL as Differentiable Programming
Deep learning increasingly synonymous with differentiable programming

"People are now building a **new kind of software** by assembling networks of parameterized **functional blocks** (including loops and conditionals) and by **training** them from examples using some form of gradient-based optimization."

Yann LeCun, 2018
Deep learning increasingly synonymous with differentiable programming

“People are now building a new kind of software by assembling networks of parameterized functional blocks (including loops and conditionals) and by training them from examples using some form of gradient-based optimization.”

Yann LeCun, 2018

Many computer programming operations remain poorly differentiable

In this work, we focus on sorting and ranking.
Sorting as subroutine in ML

**k-NN**
1. select neighbours
2. majority vote

**Trimmed regression**
ignore large errors

**Classifiers**
select top-$k$ activations

**Ranking / Sorting**
$O(n \log n)$

**Learning to rank**
NDCG loss and others

**Rank-based statistics**
data viewed as ranks

**Descriptive statistics**
Empirical distribution function
quantile normalization

**MoM estimators**

**Trimmed regression**
ignore large errors

**Slide credit:** Marco Cuturi
Sorting

\[ \sigma(\theta) = (2, 4, 3, 1) \]

Argsort (descending)
**Sorting**

\[ \theta_1, \theta_3, \theta_4, \theta_2 \]

- **Argsort (descending)**: \( \sigma(\theta) = (2, 4, 3, 1) \)
- **Sort (descending)**: \( s(\theta) \triangleq \theta_{\sigma(\theta)} \)
\[
\sigma(\theta) = (2, 4, 3, 1)
\]

So \(\theta_1, \theta_3, \theta_4, \theta_2\) are sorted in descending order:

\[
\sigma(\theta) = (2, 4, 3, 1) \implies \theta_2, \theta_4, \theta_3, \theta_1
\]
sorting

\[ \sigma(\theta) = (2, 4, 3, 1) \]

Argsort (descending)

Sort (descending)

\[ s(\theta) \triangleq \theta_{\sigma(\theta)} = (\theta_2, \theta_4, \theta_3, \theta_1) \]

piecewise linear
induces
non-convexity
\[ r(\theta) \triangleq \sigma^{-1}(\theta) \]
Ranks

\[ r(\theta) \triangleq \sigma^{-1}(\theta) = (4,1,3,2) \]
Ranks

\[ r(\theta) \triangleq \sigma^{-1}(\theta) = (4,1,3,2) \]
Related work on soft ranks

Soft ranks: differentiable proxies to “hard” ranks
Related work on soft ranks

Soft ranks: differentiable proxies to "hard" ranks

- Random perturbation technique to compute expected ranks in $O(n^3)$ time [Taylor et al., 2008]
Related work on soft ranks

Soft ranks: differentiable proxies to “hard” ranks

- Random perturbation technique to compute expected ranks in $O(n^3)$ time [Taylor et al., 2008]

- Using pairwise comparisons in $O(n^2)$ time [Qin et al., 2010]

$$r_i(\theta) \triangleq 1 + \sum_{i \neq j} 1[\theta_i < \theta_j]$$
Related work on soft ranks

Soft ranks: differentiable proxies to "hard" ranks

- Random perturbation technique to compute expected ranks in $O(n^3)$ time [Taylor et al., 2008]

- Using pairwise comparisons in $O(n^2)$ time [Qin et al., 2010]

$$r_i(\theta) \triangleq 1 + \sum_{i \neq j} 1[\theta_i < \theta_j]$$

- Regularized optimal transport approach and Sinkhorn in $O(T n^2)$ time [Cuturi et al., 2019]
Related work on soft ranks

**Soft ranks**: differentiable proxies to “hard” ranks

- Random perturbation technique to compute expected ranks in $O(n^3)$ time [Taylor et al., 2008]

- Using pairwise comparisons in $O(n^2)$ time [Qin et al., 2010]

$$r_i(\theta) \triangleq 1 + \sum_{i \neq j} 1[\theta_i < \theta_j]$$

- Regularized optimal transport approach and Sinkhorn in $O(T n^2)$ time [Cuturi et al., 2019]

None of these works achieves $O(n \log n)$ complexity
Background

Proposed method

Experimental results
Our proposal
• Differentiable (soft) relaxations of $s(\theta)$ and $r(\theta)$
Our proposal

- Differentiable (soft) relaxations of $s(\theta)$ and $r(\theta)$
- Two formulations: $L^2$ and Entropy regularised
Our proposal

- Differentiable (soft) relaxations of $s(\theta)$ and $r(\theta)$
- Two formulations: $L^2$ and Entropy regularised
- "Convexification" effect
Our proposal

- Differentiable (soft) relaxations of $s(\theta)$ and $r(\theta)$
- Two formulations: $L2$ and Entropy regularised
- "Convexification" effect
- Exact computation in $O(n \log n)$ time (forward pass)
Our proposal

- Differentiable (soft) relaxations of $s(\theta)$ and $r(\theta)$
- Two formulations: $L_2$ and Entropy regularised
- "Convexification" effect
- Exact computation in $O(n \log n)$ time (forward pass)
- Exact multiplication with the Jacobian in $O(n)$ time without unrolling (backward pass)
1. Express $s(\theta)$ and $r(\theta)$ as **linear programs** (LP) over convex polytopes
1. Express $s(\theta)$ and $r(\theta)$ as linear programs (LP) over convex polytopes

→ Turn algorithmic function into an optimization problem
Strategy outline

1. Express $s(\theta)$ and $r(\theta)$ as **linear programs** (LP) over convex polytopes
   
   $\rightarrow$ Turn algorithmic function into an optimization problem

2. Introduce **regularization** in the LP
1. Express $s(\theta)$ and $r(\theta)$ as **linear programs** (LP) over convex polytopes

   → Turn algorithmic function into an optimization problem

2. Introduce **regularization** in the LP

   → Turn LP into a projection onto convex polytopes
Strategy outline

1. Express $s(\theta)$ and $r(\theta)$ as **linear programs** (LP) over convex polytopes
   - Turn algorithmic function into an optimization problem

2. Introduce **regularization** in the LP
   - Turn LP into a projection onto convex polytopes

3. Derive algorithm for **computing** the projection
1. Express $s(\theta)$ and $r(\theta)$ as **linear programs** (LP) over convex polytopes
   $\rightarrow$ Turn algorithmic function into an optimization problem

2. Introduce **regularization** in the LP
   $\rightarrow$ Turn LP into a projection onto convex polytopes

3. Derive algorithm for **computing** the projection
   $\rightarrow$ Ideally, the projection should be computable in the same cost as the original function…
1. Express $s(\theta)$ and $r(\theta)$ as **linear programs** (LP) over convex polytopes
   → Turn algorithmic function into an optimization problem

2. Introduce **regularization** in the LP
   → Turn LP into a projection onto convex polytopes

3. Derive algorithm for **computing** the projection
   → Ideally, the projection should be computable in the same cost as the original function…

4. Derive algorithm for **differentiating** the projection
1. Express $s(\theta)$ and $r(\theta)$ as **linear programs** (LP) over convex polytopes
   - Turn algorithmic function into an optimization problem

2. Introduce **regularization** in the LP
   - Turn LP into a projection onto convex polytopes

3. Derive algorithm for **computing** the projection
   - Ideally, the projection should be computable in the same cost as the original function…

4. Derive algorithm for **differentiating** the projection
   - Could be challenging (argmin differentiation problem)
<table>
<thead>
<tr>
<th>Strategy outline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cuturi et al. [2019]</td>
</tr>
</tbody>
</table>
Strategy outline

1. LP

Cuturi et al. [2019]

Birkhoff polytope

\[ \mathcal{B} \subset \mathbb{R}^{n \times n} \]

- \( \varphi((1, 3, 2)) \)
- \( \varphi((2, 3, 1)) \)
- \( \varphi((1, 2, 3)) \)
- \( \varphi((2, 1, 3)) \)
- \( \varphi((3, 1, 2)) \)
- \( \varphi((3, 2, 1)) \)

This work

Permutahedron

\[ \mathcal{P} \subset \mathbb{R}^n \]

- \( (1, 3, 2) \)
- \( (2, 3, 1) \)
- \( (3, 2, 1) \)
- \( (1, 2, 3) \)
- \( (3, 1, 2) \)
- \( (2, 1, 3) \)
We assume computed exactly using the Hungarian algorithm. Otherwise, return the projection of $B$. Otherwise, return the projection of $1$. Otherwise, return the projection of $l$. Otherwise, return the projection of $h$. Otherwise, return the projection of $y$. Otherwise, return the projection of $k$. Otherwise, return the projection of $k$. Otherwise, return the projection of $u$. Otherwise, return the projection of $k$. Otherwise, return the projection of $k$. Otherwise, return the projection of $u$.

The next proposition, proved in §C.2, shows how to project efficiently.
1. LP

Cuturi et al. [2019]

This work

2. Regularization

Birkhoff polytope

Permutahedron

3. Computation

Entropy

Sinkhorn

L2 or Entropy

Pool Adjacent Violators (PAV)
where we assume Dykstra’s algorithm computed using the Sinkhorn algorithm.

Noticeably, marginal inference is known to be $\#P$-complete.

•

1 $k$

The total cost is $O$ $l$ $i^q$ $u$

where we assume the set of row-stochastic matrices, a strict superset of the Birkhoff polytope.

Otherwise, return the projection of $L^2$ $i^q$ $u$

otherwise.

Proposition 3 $h$
in the Euclidean case and $O$ $m$

else.

We now set $P$ $2$
in the Euclidean case and $O$

We view ranking as a structured prediction problem and let $P$

(a) Probability simplex

(b) Unit cube

(c) Permutahedron

(d) Birkhoff polytope

Figure 2: Examples of polytopes

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Birkhoff polytope</td>
<td>Entropy</td>
<td>Sinkhorn</td>
<td>Backprop through Sinkhorn iterates</td>
</tr>
<tr>
<td>$\mathcal{B} \subset \mathbb{R}^{n \times n}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\varphi((1, 2, 3))$</td>
<td>$\varphi((2, 1, 3))$</td>
<td>$\varphi((3, 1, 2))$</td>
<td>$\varphi((3, 2, 1))$</td>
</tr>
<tr>
<td>$\varphi((1, 3, 2))$</td>
<td>$\varphi((2, 3, 1))$</td>
<td>$\varphi((3, 2, 1))$</td>
<td>$\varphi((3, 1, 2))$</td>
</tr>
</tbody>
</table>

This work

Permutahedron

$\mathcal{P} \subset \mathbb{R}^n$

$P$ $2$
or Entropy

L2 or Entropy

Pool Adjacent Violators (PAV)

Differentiate PAV solution
\( \mathcal{P}(w) \triangleq \text{conv}(\{w_\sigma : \sigma \in \Sigma\}) \subset \mathbb{R}^n \)

\( \mathcal{P}(\rho) \subset \mathbb{R}^n \)

\( \rho \triangleq (n, n-1, \ldots, 1) \)
Step 1: linear programming formulations
Proposition

\[ s(\theta) = \arg \max_{y \in \mathcal{P}(\theta)} \langle y, \rho \rangle \]

\[ \rho \triangleq (n, n - 1, \ldots, 1) \]
Step 1: linear programming formulations

Proposition

\[ s(\theta) = \arg \max_{y \in \mathcal{P}(\theta)} \langle y, \rho \rangle \]

\[ r(\theta) = \arg \max_{y \in \mathcal{P}(\rho)} \langle y, -\theta \rangle \]

\[ \rho \triangleq (n, n - 1, \ldots, 1) \]
Proof of the first claim

\[ \rho_n > \rho_{n-1} > \ldots > 1 \Rightarrow \sigma(\theta) = \underset{\sigma \in \Sigma}{\arg\max} \langle \theta_\sigma, \rho \rangle \]
Proof of the first claim

\[ \rho_n > \rho_{n-1} > \ldots > 1 \Rightarrow \sigma(\theta) = \arg \max_{\sigma \in \Sigma} \langle \theta_\sigma, \rho \rangle \]

\[ s(\theta) \triangleq \theta_{\sigma(\theta)} \]
Proof of the first claim

\[ \rho_n > \rho_{n-1} > \ldots > 1 \Rightarrow \sigma(\theta) = \arg \max_{\sigma \in \Sigma} \langle \theta_\sigma, \rho \rangle \]

\[ s(\theta) \triangleq \theta_{\sigma(\theta)} \]

\[ = \arg \max_{\theta_\sigma : \sigma \in \Sigma} \langle \theta_\sigma, \rho \rangle \]
Proof of the first claim

\[ \rho_n > \rho_{n-1} > \ldots > 1 \Rightarrow \sigma(\theta) = \arg \max_{\sigma \in \Sigma} \langle \theta_\sigma, \rho \rangle \]

\[ s(\theta) \triangleq \theta_{\sigma(\theta)} \]

\[ = \arg \max_{\theta_\sigma : \sigma \in \Sigma} \langle \theta_\sigma, \rho \rangle \]

\[ = \arg \max_{y \in \Sigma(\theta)} \langle y, \rho \rangle \]
Proof of the first claim

\[ \rho_n > \rho_{n-1} > \ldots > 1 \Rightarrow \sigma(\theta) = \arg \max_{\sigma \in \Sigma} \langle \theta_\sigma, \rho \rangle \]

\[ s(\theta) \triangleq \theta_{\sigma(\theta)} \]

\[ = \arg \max_{\theta_\sigma: \sigma \in \Sigma} \langle \theta_\sigma, \rho \rangle \]

\[ = \arg \max_{y \in \Sigma(\theta)} \langle y, \rho \rangle \]

\[ = \arg \max_{y \in \mathcal{P}(\theta)} \langle y, \rho \rangle \]
Step 2: introducing regularization
Step 2: introducing regularization

Quadratic regularization \( Q(y) \triangleq \frac{1}{2} \|y\|^2 \)

\[
P_Q(z, w) \triangleq \arg \max_{y \in \mathcal{P}(w)} \langle y, z \rangle - Q(y)
\]
Step 2: introducing regularization

Quadratic regularization $Q(y) \triangleq \frac{1}{2} \|y\|^2$

$P_Q(z, w) \triangleq \arg \max_{y \in \mathcal{P}(w)} \langle y, z \rangle - Q(y) = \arg \min_{y \in \mathcal{P}(w)} \|y - z\|^2$
Step 2: introducing regularization

Quadratic regularization \( Q(y) \triangleq \frac{1}{2} \|y\|^2 \)

\( P_Q(z, w) \triangleq \arg \max_{y \in \mathcal{P}(w)} \langle y, z \rangle - Q(y) = \arg \min_{y \in \mathcal{P}(w)} \|y - z\|^2 \)

Definition

\( s_{\varepsilon Q}(\theta) \triangleq P_{\varepsilon Q}(\rho, \theta) = P_Q(\rho/\varepsilon, \theta) \)
Step 2: introducing regularization

Quadratic regularization \( Q(y) \triangleq \frac{1}{2} \|y\|^2 \)

\[
P_Q(z, w) \triangleq \arg \max_{y \in \mathcal{P}(w)} \langle y, z \rangle - Q(y) = \arg \min_{y \in \mathcal{P}(w)} \|y - z\|^2
\]

Definition

\[
s_{\varepsilon Q}(\theta) \triangleq P_{\varepsilon Q}(\rho, \theta) = P_Q(\rho/\varepsilon, \theta)
\]

\[
r_{\varepsilon Q}(\theta) \triangleq P_{\varepsilon Q}(-\theta, \rho) = P_Q(-\theta/\varepsilon, \rho)
\]
We illustrate the behavior of both of these soft operations by showing that our operators can be computed in closed form. Specifically, for soft ranking, we choose to use a soft sort (Q) that tends to be even smoother compared to the hard counterpart. For soft ranking with top-

This diagram illustrates the soft sort (Q) for different values of \( \epsilon \):

- Blue line: \( \epsilon = 0 \)
- Orange line: \( \epsilon = 1 \)
- Green line: \( \epsilon = 10 \)

As \( \epsilon \) increases, the soft sort (Q) becomes smoother, resembling a straight line. This is consistent with the observation that the function eventually converges to a mean (Proposition B.3).

We reduce the number of kinks, and the function eventually converges to a mean (Proposition B.3). Together with the proof of Proposition B.3, we establish a slightly stronger result. Namely, we derive a differentiable operator, from which we can rewrite the change of variable as linear assignment over the Birkhoff polytope.

The parameter \( \epsilon \) controls the smoothness of the function, making the objective function increasingly easy to optimize as \( \epsilon \) increases.

On tuning \( \theta_2 \), depicted by a straight line.

Fast Differentiable Sorting and Ranking

Adams & Zemel 2019

Cuturi et al. 2011

Relation to linear assignment formulation.
Figure 3. We illustrate the behavior of both of these soft operations with an explicit value of \( r \). With \( \varepsilon \), we reduce the number of kinks, and the function eventually converges to a mean (Proposition 2).

We can further characterize these approximations. Namely, as we now formalize, they are differentiable (at the cost of departing from "hard" sorting or ranking). This behavior is also visible in Figure 3.

For all \( \varepsilon \), the function again tends to be smoother though it may contain kinks. For soft sorting with \( \varepsilon \), we define the function, from \( \cdot \), as linear assignment over the Birkhoff polytope. The last property describes the behavior as we include in the model. As for the hard versions, this behavior is also visible in Figure 3.

Effect of the regularization parameter \( \varepsilon \) shows that if \( \varepsilon < 0 \), the function again tends to be smoother though it may contain kinks. For soft ranking with \( \varepsilon \), we confirm the empirical finding of Cuturi et al. (2011).
Continuity and differentiability

**Soft sort (Q)**

- \( s_{\theta_2}(\theta) \)
- \( \varepsilon = 0 \) (blue)
- \( \varepsilon = 1 \) (orange)
- \( \varepsilon = 10 \) (green)

**Soft rank (Q)**

- \( r_{\theta_1}(\theta) \)
- \( \varepsilon = 0 \) (blue)
- \( \varepsilon = 1 \) (orange)
- \( \varepsilon = 2 \) (green)

**Properties**

\( s_Q \) and \( r_Q \) are 1-Lipchitz continuous and differentiable almost everywhere.
Effect of regularization strength $\varepsilon$

**Soft sorting**

**Soft ranking**

![Graph showing the effect of regularization strength $\varepsilon$ on soft sorting and ranking]

- **Soft sorting**
  - Colors represent different values of $\theta$.
  - The x-axis represents $\varepsilon$.

- **Soft ranking**
  - Colors represent different values of $\theta$.
  - The x-axis represents $\varepsilon$. 

---

**Text Content**

- Clearly, we can express regularizations in our linear programming formulations. This is done by introducing strongly convex regularizations.

- **Differentiability a.e. of sorting.**

- **Lack of useful Jacobian of ranking.**

- **Fast Differentiable Sorting and Ranking**

- For sorting, we choose $\theta = 1$, which is a convex combination of the permutations of $r$.

- When varying the regularization strength, we introduce a parameter $\varepsilon$.

- We now build upon these projections to consider entropic regularization.

- More generally, we can use any strongly convex regularization.

- We also define soft sorting and ranking operators. To control the $\theta$ parameter, we use entropic regularization.

- When $\varepsilon = 0$, we get the hard counterpart. As we increase $\varepsilon$, we move towards the soft, smooth solutions.

- We state all propositions for these two cases and postpone a more detailed analysis.
Effect of regularization strength $\epsilon$

**Soft sorting**

- Properties
  - Converge to hard version when $\epsilon \leq \epsilon_{\text{min}}$

**Soft ranking**
Effect of regularization strength $\varepsilon$

**Soft sorting**

- Converge to hard version when $\varepsilon \leq \varepsilon_{\text{min}}$
- Collapse to a mean when $\varepsilon \to \infty$

**Soft ranking**
Effect of regularization strength \( \varepsilon \)

**Properties**

Converge to hard version when \( \varepsilon \leq \varepsilon_{\text{min}} \)

Collapse to a mean when \( \varepsilon \rightarrow \infty \)

Order preserving (paths don’t cross)
Figure 1. Illustration of the permutahedron. The gray line indicates the regularization path of the permutahedron. The permutahedron is a polytope in which the vertices are permutations of \((1, 2, 3)\), \((1, 3, 2)\), \((2, 1, 3)\), \((2, 3, 1)\), \((3, 1, 2)\), and \((3, 2, 1)\). The soft sorting and ranking operators, \(r_Q(\theta)\), \(r_{2Q}(\theta)\), \(r_{3Q}(\theta)\), and \(r_{100Q}(\theta)\), are shown at different points on the permutahedron.

Collapsing to a mean \(\mu\) when \(\epsilon \to \infty\)
Step 3: Computation
Step 3: Computation

Reduction to isotonic regression

**Proposition**

\[
P_Q(z, w) = z - v_Q(z_{\sigma(z)}, w)_{\sigma^{-1}(z)}
\]

\[
v_Q(s, w) \triangleq \arg \min_{v_1 \geq \ldots \geq v_n} \|v - (s - w)\|^2
\]

Total time cost: $O(n \log n)$

e.g. [Negrignho & Martins, 2014; Lim & Wright 2016]
Step 3: Computation

Reduction to isotonic regression

Proposition

\[ P_Q(z, w) = z - v_Q(z_{\sigma(z)}, w)_{\sigma^{-1}(z)} \]

\[ v_Q(s, w) \triangleq \arg \min_{v_1 \geq \ldots \geq v_n} \| v - (s - w) \|^2 \]

dual solution

Total time cost: \( O(n \log n) \)

e.g. [Negrignho & Martins, 2014; Lim & Wright 2016]
Step 3: Computation

Reduction to isotonic regression

**Proposition**

\[ P_Q(z, w) = z - v_Q(z_{\sigma(z)}, w)_{\sigma^{-1}(z)} \]

\[ v_Q(s, w) \triangleq \arg \min_{v_1 \geq \ldots \geq v_n} \| v - (s - w) \|^2 \]

Total time cost: \( O(n \log n) \)

e.g. [Negrignho & Martins, 2014; Lim & Wright 2016]
Step 3: Computation

Boils down to solving  \[ v^* = \arg \min_{v_1 \geq \ldots \geq v_n} \|v - u\|^2 \] \quad u = s - w

[Best, 2000]
Step 3: Computation

Boils down to solving  \( v^* = \arg \min_{v_1 \geq \ldots \geq v_n} \|v - u\|^2 \)  

\( u = s - w \)

Pool Adjacent Violators (PAV): Finds a partition \( \mathcal{B}_1, \ldots, \mathcal{B}_m \) by repeatedly splitting coordinates. The worst-case cost is \( O(n) \).

[Best, 2000]
Step 3: Computation

Boils down to solving \( v^* = \arg \min_{v_1 \geq \ldots \geq v_n} \| v - u \|^2 \) \( u = s - w \)

Pool Adjacent Violators (PAV): Finds a partition \( \mathcal{B}_1, \ldots, \mathcal{B}_m \) by repeatedly splitting coordinates. The worst-case cost is \( O(n) \).

Ex: \( n = 6 \)

\[ \begin{align*}
\mathcal{B}_1 &= \{1, 2\} & v_1^* = v_2^* &= \text{mean}(u_1, u_2) \\
\mathcal{B}_2 &= \{3\} & v_3^* &= \text{mean}(u_3) = u_3 \\
\mathcal{B}_3 &= \{4, 5, 6\} & v_4^* = v_5^* = v_6^* &= \text{mean}(u_4, u_5, u_6)
\end{align*} \]

[Best, 2000]
Step 4: Differentiation

See also [Djolonga & Krause, 2017]
Step 4: Differentiation

Differentiate \( v_Q(s, w) = \arg \min_{v_1 \geq \ldots \geq v_n} \| v - (s - w) \|^2 \) w.r.t. \( s \) and \( w \)

See also [Djolonga & Krause, 2017]
Differentiate $v_Q(s, w) = \arg \min_{v_1 \geq \ldots \geq v_n} \|v - (s - w)\|^2$ w.r.t. $s$ and $w$

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

$\mathbf{B}_j \triangleq \frac{1}{|\mathcal{B}_j|} \in \mathbb{R}^{|\mathcal{B}_j| \times |\mathcal{B}_j|}$, $j \in [m]$

See also [Djolonga & Krause, 2017]
Step 4: Differentiation

Differentiate $P_Q(z, w)$ w.r.t. $z$ and $w$
Step 4: Differentiation

Differentiate $P_Q(z, w)$ w.r.t. $z$ and $w$

\[ \frac{\partial P_Q(z, w)}{\partial z} = J_Q(z_{\sigma(z)}, w)_{\sigma^{-1}(z)} \]

Proposition

\[ J_Q(s, w) \triangleq I - \frac{\partial v_Q(s, w)}{\partial s} \]

Multiplication with the Jacobian in $O(n)$ time and space (see paper)
Robust regression
Robust regression

Least squares (LS)

$$\min_w \frac{1}{n} \sum_{i=1}^{n} \ell_i(w) \quad \ell_i(w) \triangleq \frac{1}{2} (y_i - g_w(x_i))^2$$
Robust regression

**Least squares (LS)**

\[
\min_{w} \frac{1}{n} \sum_{i=1}^{n} \ell_i(w) \quad \ell_i(w) \equiv \frac{1}{2} (y_i - g_w(x_i))^2
\]

**Soft Least trimmed squares (SLTS)**

\[
\min_{w} \frac{1}{n - k} \sum_{i=k+1}^{n} \ell_i^\varepsilon(w) \quad \ell_i^\varepsilon(w) \equiv [s_{\varepsilon Q}(\ell(w))]_i
\]
Robust regression

**Least squares (LS)**

\[
\min_w \frac{1}{n} \sum_{i=1}^{n} \ell_i(w) \quad \ell_i(w) \triangleq \frac{1}{2}(y_i - g_w(x_i))^2
\]

**Soft Least trimmed squares (SLTS)**

\[
\min_w \frac{1}{n-k} \sum_{i=k+1}^{n} \ell_i^\varepsilon(w) \quad \ell_i^\varepsilon(w) \triangleq [s_{\varepsilon Q(\ell(w))}]_i
\]

\[\varepsilon \to 0 \quad SLTS \to LTS\]
Robust regression

**Least squares (LS)**

$$\min_w \frac{1}{n} \sum_{i=1}^{n} \ell_i(w) \quad \ell_i(w) \triangleq \frac{1}{2} (y_i - g_w(x_i))^2$$

**Soft Least trimmed squares (SLTS)**

$$\min_w \frac{1}{n-k} \sum_{i=k+1}^{n} \ell_i^\varepsilon(w) \quad \ell_i^\varepsilon(w) \triangleq [s_{\varepsilon Q}(\ell(w))]_i$$

$$\varepsilon \rightarrow 0 \quad SLTS \rightarrow LTS \quad \varepsilon \rightarrow \infty \quad SLTS \rightarrow LS$$
Robust regression

Evaluation: 10-fold CV

Hyper-parameter selection: 5-fold CV
Top-k classification

\[ \ell : [n] \times \mathbb{R}^n \rightarrow \mathbb{R}_+ \]

Ground truth  Predicted soft ranks

Cuturi et al. [2019]
Top-k classification

\[ \ell : [n] \times \mathbb{R}^n \rightarrow \mathbb{R}_+ \]

Cuturi et al. [2019]

Ground truth  Predicted soft ranks

![Graph showing test accuracy over epochs for different methods on CIFAR-10.](image)

- OT
- \( r_Q (L_2) \)
- \( r_E (\log\text{-KL}) \)
- Cross-entropy
- All-pairs
Top-k classification

\[ \ell : [n] \times \mathbb{R}^n \rightarrow \mathbb{R}_+ \]

Ground truth | Predicted soft ranks

Cuturi et al. [2019]

\[
\begin{align*}
\text{CIFAR-10} & \quad \text{CIFAR-100} \\
\text{Test accuracy} & \quad \text{Test accuracy}
\end{align*}
\]

- OT
- \( r_Q (L_2) \)
- \( r_E \) (log-KL)
- Cross-entropy
- All-pairs

\[ \text{Epochs} \]

\[ \text{Epochs} \]
Again, the Jacobian w.r.t. the Jacobian of isotonic optimization. Differentiating linear time by using the simple identity rows and columns. We can nonetheless multiply with it in the Jacobian of isotonic optimization, the Jacobian of the regularization. For quadratic regularization, the Jacobian computation with backpropagation disabled. OT and All-pairs go out-of-memory starting from "where Proposition 4. \( (z_1, \ldots, z_n) \) and \( (s_1, \ldots, s_n) \) are multiplied with the Jacobians \( J_{z}, w \) and \( J_{s}, w \). To summarize, we can multiply with the Jacobians and the blocks are constant.

Proposed: our All-pairs (\( r_{Q} (L_2) \)) and \( r_{E} (\text{log-KL}) \) is equivalent to \( \prod_{i = 1}^{n} \frac{1}{y_i} \) and \( \log \left( \prod_{i = 1}^{n} y_i^{x_i} \right) \) respectively. Following Qin et al.\( (2010) \) \& Lim & Wright\( (2019) \), one can obtain soft ranks in \( n \times n \) elements. We compare the following soft operators with batch norm on each), the ADAM optimizer (\( P = 100 \)). Our empirical results, averaged over \( \frac{100}{n} \) runs, are the ADAM optimizer (\( P = 100 \)).

6. Experiments

OT: The optimal transport projection is not block diagonal, as we need to permute its incoming gradients, one uniformly and the other weighted. The blocks have constant value. For entropic regularization, only depends on the partition \( \tau \). With the Jacobian of the projection is at hand, differentiating rows and columns. We now combine Proposition 2 and 4.

\[ \begin{align*}
\text{Speed benchmark} \\
\text{Runtime comparison for one iteration (batch size: 128)} \\
\end{align*} \]
We now consider the label ranking setting where supervision \( W = \{W_1, \ldots, W_N\} \) with 64 GBs of RAM and a GeForce GTX 1080 Ti. All-pairs go out-of-memory starting from \((\text{OT, All-pairs, Cross-entropy})\), we create a batch of TensorFlow (\( \text{Abadi et al. 2016} \)).

To measure the impact of the dimensionality \( n \) vectors and we compare the time to compute soft ranking between the ranks. Maximizing this coefficient is equivalent to the classical ridge regression can be cast as the operator \( f(x) \approx \mathbb{E}_{h^*} \mathbb{E}_{w^*} \left( \ell_i \right) \), which we therefore propose to rather use the notation \( f(x) \approx \mathbb{E}_{h^*} \mathbb{E}_{w^*} \left( \ell_i \right) \) rather than \( f(x) \approx \mathbb{E}_{h^*} \mathbb{E}_{w^*} \left( \ell_i \right) \).

**Results.**

For fair comparison with GPU implementations of Softmax, our formulations scale well, with the dimensionality \( n \) and the runtime is reasonable in small dimension, OT and All-pairs scale quadratically with respect to the dimensionality \( n \).

**Discussion.**

Our soft operators are several hours faster than OT, they are particularly challenging. In contrast, our approaches only show that the lack of memory available on GPUs is problematic for these methods. In contrast, our approaches only require the need for recording the computational graph. This is unfortunately a discontinuous function \( f(x) \approx \mathbb{E}_{h^*} \mathbb{E}_{w^*} \left( \ell_i \right) \).

**Comparison.**

For the label ranking experiment, we designed the following experiment. Run times for one batch computation with backpropagation enabled, \( \text{Cuturi 2009} \), due to the large number of datasets, we choose \( \text{Korba 2008} \), \( \text{Cheng et al. 2008} \), \( \text{Hullermeier et al. 2009} \) and for ablation study we drop the soft ranking layer.

**Figure.**

The classical ridge regression can be cast as estimating the output pairs. Our goal is to learn a model \( R \) that predicts outputs from inputs, where \( R \) is an ordered set of \( (y_1, \ldots, y_n) \) and \( (x_1, \ldots, x_n) \) function. This is unfortunately a discontinuous function \( f(x) \approx \mathbb{E}_{h^*} \mathbb{E}_{w^*} \left( \ell_i \right) \).

**Label ranking experiment**

\[
\ell_i \triangleq \frac{1}{2} \| y_i - f(x_i) \|^2 \quad y_i \in \Sigma
\]

**Label ranking experiment**

\[
f(x) \approx \mathbb{E}_{h^*} \mathbb{E}_{w^*} \left( \ell_i \right)
\]

**Label ranking experiment**

\[
r_Q(g(x))
\]

**Label ranking experiment**

\[
r_E(\log-KL)
\]

**Label ranking experiment**

Comparison on 21 datasets, 5-fold CV
Summary

- We proposed sorting and ranking relaxations with $O(n \log n)$ computation and $O(n)$ differentiation.
Summary

• We proposed sorting and ranking relaxations with $O(n \log n)$ computation and $O(n)$ differentiation

• Key techniques: projections onto the permutahedron and reduction to isotonic optimization
We proposed sorting and ranking relaxations with \( O(n \log n) \) computation and \( O(n) \) differentiation.

Key techniques: projections onto the permutahedron and reduction to isotonic optimization.

Applications to least trimmed squares, top-k classification and label ranking.
• We proposed sorting and ranking relaxations with $O(n \log n)$ computation and $O(n)$ differentiation

• Key techniques: projections onto the permutahedron and reduction to isotonic optimization

• Applications to least trimmed squares, top-k classification and label ranking


Code: coming soon!