Fast, Differentiable and Sparse Top-k: a Convex Analysis Perspective

Michael E. Sander [* 1] **Joan Puigcerver** [2] **Josip Djolonga** [2] **Gabriel Peyré** [1 3] **Mathieu Blondel** [2]

Abstract

The top-k operator returns a sparse vector, where the non-zero values correspond to the k largest values of the input. Unfortunately, because it is a discontinuous function, it is difficult to incorporate in neural networks trained end-to-end with backpropagation. Recent works have considered differentiable relaxations, based either on regularization or perturbation techniques. However, to date, no approach is fully differentiable *and* sparse. In this paper, we propose new differentiable *and* sparse top-k operators. We view the top-k operator as a linear program over the permutahedron, the convex hull of permutations. We then introduce a p-norm regularization term to smooth out the operator, and show that its computation can be reduced to isotonic optimization. Our framework is significantly more general than the existing one and allows for example to express top-k operators that select values *in magnitude*. On the algorithmic side, in addition to pool adjacent violator (PAV) algorithms, we propose a new GPU/TPU-friendly Dykstra algorithm to solve isotonic optimization problems. We successfully use our operators to prune weights in neural networks, to fine-tune vision transformers, and as a router in sparse mixture of experts.

1. Introduction

Finding the top-k values and their corresponding indices in a vector is a widely used building block in modern neural networks. For instance, in sparse mixture of experts (MoEs) (Shazeer et al., 2017; Fedus et al., 2022), a top-k router maps

each token to a selection of k experts (or each expert to a selection of k tokens). In beam search for sequence decoding (Wiseman & Rush, 2016), a beam of k possible output sequences is maintained and updated at each decoding step. For pruning neural networks, the top-k operator can be used to sparsify a neural network, by removing weights with the smallest magnitude (Han et al., 2015; Frankle & Carbin, 2018). Finally, top-k accuracy (e.g., top-3 or top-5) is frequently used to evaluate the performance of neural networks at inference time.
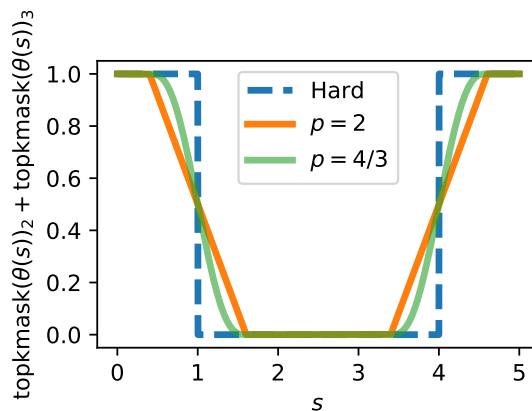

Figure 1. **Illustration of our differentiable and sparse top-k mask.** For $k = 2$, we consider $\boldsymbol{\theta}(s) = (3, 1, -1 + s, s) \in \mathbb{R}^4$ and plot $\mathrm{topkmask}(\boldsymbol{\theta}(s))_2 + \mathrm{topkmask}(\boldsymbol{\theta}(s))_3$ as a function of s. We compare the hard version (no regularization) with our proposed operator using p-norm regularization: $p = 2$ leads to differentiable a.e. operator; $p = 4/3$, leads to a differentiable operator. Both operators are **sparse**: they are exactly 0 for some values of s.

However, the top-k operator is a discontinuous piecewise affine function with derivatives either undefined or constant (the related top-k *mask* operator, which returns a binary encoding of the indices corresponding to the top-k values, has null derivatives). This makes it hard to use in a neural network trained with gradient backpropagation. Recent works have considered differentiable relaxations, based either on regularization or perturbation techniques (see §2 for a review). However, to date, no approach is differentiable everywhere *and* sparse. Sparsity is crucial in neural networks that require *conditional* computation. This is for instance the case in sparse mixture of experts, where the top-k operator is used to "route" tokens to selected experts.

---

[*]Work done during an internship at Google Research, Brain team. [1]Ecole Normale Supérieure [2]Google Research, Brain team [3]CNRS. Correspondence to: Michael E. Sander <michael.sander@ens.fr>, Joan Puigcerver <jpuigcerver@google.com>, Josip Djolonga <josipd@google.com>, Gabriel Peyré <gabriel.peyre@ens.fr>, Mathieu Blondel <mblondel@google.com>.

Without sparsity, all experts would need to process all tokens, leading to high computational cost. This is also the case when selecting weights with highest magnitude in neural networks: the non-selected weights should be exactly 0 in order to optimize the computational and memory costs.

In this work, we propose novel differentiable everywhere *and* sparse top-k operators (see Figure 1). We build upon the framework of Blondel et al. (2020b), which casts sorting and ranking as linear programs over the permutahedron, and uses a reduction to isotonic optimization. We significantly generalize that framework in several ways. Specifically, we make the following contributions:

- After reviewing related work in §2 and background in §3, we introduce our generalized framework in §4. We introduce a new nonlinearity φ, allowing us to express new operators (such as the top-k in magnitude). In doing so, we also establish new connections between so-called k-support norms and the permutahedron.

- We introduce a regularization term to obtain a relaxed top-k operator. In particular, using p-norm regularization, we obtain the first differentiable everywhere *and* sparse top-k operator (Figure 3).

- In §5, we derive pool adjacent violator (PAV) algorithms for solving isotonic optimization when using φ and/or when using p-norm regularization. We show that the Jacobian of our operator can be computed in closed form.

- As a GPU/TPU friendly alternative to PAV, we propose a Dykstra algorithm to solve isotonic optimization, which is easy to vectorize in the case $p = 2$.

- In §6, we chose to focus on three applications of our operators. First, we use them to prune weights in a multilayer perceptron during training and show that they lead to better accuracy than with a hard top-k. Second, we define top-k losses to fine-tune vision transformers (ViTs) and obtain better top-k accuracy than with the cross-entropy loss. Finally, we use our operators as a router in vision mixture of experts, and show that they outperform the hard top-k router.

2. Related work

Differentiable loss functions.   Several works have proposed a differentiable loss function as a surrogate for a discrete, discontinuous metric. For example, loss functions have been proposed for top-k accuracy (Lapin et al., 2015; 2016; Berrada et al., 2018; Petersen et al., 2022) and various ranking metrics (Chapelle & Wu, 2010; Adams & Zemel, 2011; Rolínek et al., 2020).

Differentiable operators.   In a different line of research, which is the main focus of this work, a differentiable operator is proposed, which can be used either as an intermediate

layer in a neural network or as final output, fed into an arbitrary loss function. For example, Niepert et al. (2021) proposed a framework for computing gradients of discrete probability distributions or optimization problems. Amos et al. (2019) and Qian et al. (2022) proposed a smooth (but not sparse) top-k operator based on binary entropy regularization. Related projections on the capped simplex were proposed (Martins & Kreutzer, 2017; Malaviya et al., 2018; Blondel, 2019) in different contexts. These works use ad-hoc algorithms, while we use a reduction to isotonic optimization. Cuturi et al. (2019) proposed a relaxation of the sorting and ranking operators based on entropy-regularized optimal transport and used it to obtain a differentiable (but again not sparse) top-k operator. Its computation relies on Sinkhorn's algorithm (Sinkhorn, 1967; Cuturi, 2013), which in addition makes it potentially slow to compute and differentiate. A similar approach was proposed by Xie et al. (2020). Petersen et al. (2021) propose smooth differentiable sorting networks by combining differentiable sorting functions with sorting networks. Other relaxation of the sort and operators have been proposed by Grover et al. (2019) and Prillo & Eisenschlos (2020).

The closest work to ours is that of Blondel et al. (2020b), in which sorting and ranking are cast as linear programs over the permutahedron. To make these operators differentiable, regularization is introduced in the formulation and it is shown that the resulting operators can be computed via isotonic optimization in $O(n \log n)$ time. Unfortunately, the proposed operators still include kinks: they are not differentiable everywhere. The question of how to construct a differentiable everywhere *and* sparse relaxation with $O(n \log n)$ time complexity is therefore still open. In this work, we manage to do so by using p-norm regularization. Furthermore, by introducing a new nonlinearity φ, we significantly generalize the framework of Blondel et al. (2020b), allowing us for instance to express a new top-k operator *in magnitude*. We introduce a new GPU/TPU friendly Dykstra algorithm as an alternative to PAV.

Instead of introducing regularization, another technique relies on perturbation (Berthet et al., 2020). This technique has been used to obtain a differentiable (but still *not sparse*) top-k for image patch selection (Cordonnier et al., 2021).

Pruning weights with small magnitude.   Many recent works focus on neural network pruning, where parameters are removed to significantly reduce the size of a model. See Blalock et al. (2020) for a recent survey. A simple yet popular method for pruning neural networks is by global magnitude pruning (Collins & Kohli, 2014; Han et al., 2015): weights with lowest absolute value are set to 0. While most of the pruning techniques are performed after the model is fully trained (Blalock et al., 2020), some works prune periodically during training (Gale et al., 2019). However,

to the best of our knowledge, pruning by magnitude is not done in a differentiable fashion. In this work, we empirically show that pruning weights with a differentiable (or differentiable almost everywhere) top-k operator in magnitude during training leads to faster convergence and better accuracy than with a "hard" one.

Top-k operator for mixture of experts. Sparse mixture of experts models (MoEs) (Shazeer et al., 2017) are a class of deep learning models where only a small proportion of the model, known as experts, is activated, depending on its input. Therefore, sparse MoEs are able to increase the number of parameters without increasing the time complexity of the model. Sparse MoEs have achieved great empirical successes in computer vision (Riquelme et al., 2021; Zhou et al., 2022) as well as natural language processing (Shazeer et al., 2017; Lewis et al., 2021; Fedus et al., 2021). At the heart of the sparse MoE model is its routing mechanism, which determines which inputs (or tokens) are assigned to which experts. In the sparse mixture of experts literature, some works have recently proposed new top-k operators in the routing module. Hazimeh et al. (2021) proposed a binary encoding formulation to select non-zero weights. However, their formulation does not approximate the true top-k operator and sparsity is only supported at inference time, not during training. Liu et al. (2022) proposed an optimal transport formulation supporting k-sparsity constraints and used it for sparse mixture of experts. In this work, we propose to replace the hard top-k router, which is a discontinuous function, by our smooth relaxation.

3. Background

Notation. We denote a permutation of $[n]$ by $\sigma = (\sigma_1, \ldots, \sigma_n)$ and its inverse by σ^{-1}. When seen as a vector, we denote it $\boldsymbol{\sigma}$. We denote the set of all $n!$ permutations by Σ. Given a vector $\boldsymbol{x} \in \mathbb{R}^n$, we denote the version of $\boldsymbol{x}$ permuted according to σ by $\boldsymbol{x}_\sigma := (x_{\sigma_1}, \ldots, x_{\sigma_n})$. We denote the i-th largest value of $\boldsymbol{x} \in \mathbb{R}^n$ by $x_{[i]}$. Without loss of generality, we always sort values in descending order. The conjugate of $f(\boldsymbol{x})$ is denoted by $f^*(\boldsymbol{y}) := \sup_{\boldsymbol{x} \in \mathbb{R}^n} \langle \boldsymbol{x}, \boldsymbol{y} \rangle - f(\boldsymbol{x})$.

Review of operators. We denote the **argsort** operator as the permutation $\boldsymbol{\sigma}$ sorting $\boldsymbol{x} \in \mathbb{R}^n$, i.e.,

$$\operatorname{argsort}(\boldsymbol{x}) := \boldsymbol{\sigma}, \quad \text{where} \quad x_{\sigma_1} \geq \cdots \geq x_{\sigma_n}.$$

We denote the **sort** operator as the values of $\boldsymbol{x} \in \mathbb{R}^n$ in sorted order, i.e.,

$$\operatorname{sort}(\boldsymbol{x}) := \boldsymbol{x}_\sigma, \quad \text{where} \quad \boldsymbol{\sigma} = \operatorname{argsort}(\boldsymbol{x}).$$

The value $[\operatorname{sort}(\boldsymbol{x})]_i$ is also known as the i-th order statistic. We denote the **rank** operator as the function returning the

positions of the vector $\boldsymbol{x} \in \mathbb{R}^n$ in the sorted vector. It is formally equal to the argsort's inverse permutation:

$$\operatorname{rank}(\boldsymbol{x}) := \boldsymbol{\sigma}^{-1}, \quad \text{where} \quad \boldsymbol{\sigma} = \operatorname{argsort}(\boldsymbol{x}).$$

Smaller rank $[\operatorname{rank}(\boldsymbol{x})]_i$ means that x_i has higher value. The **top-k mask** operator returns a bit-vector encoding whether each value x_i is within the top-k values or not:

$$[\operatorname{topkmask}(\boldsymbol{x})]_i := \begin{cases} 1, & \text{if } [\operatorname{rank}(\boldsymbol{x})]_i \leq k \\ 0, & \text{otherwise.} \end{cases}.$$

The **top-k** operator returns the values themselves if they are within the top-k values or 0 otherwise, i.e.,

$$\operatorname{topk}(\boldsymbol{x}) := \boldsymbol{x} \circ \operatorname{topkmask}(\boldsymbol{x}),$$

where $\circ$ denotes element-wise multiplication. The **top-k in magnitude** operator is defined similarly as

$$\operatorname{topkmag}(\boldsymbol{x}) := \boldsymbol{x} \circ \operatorname{topkmask}(|\boldsymbol{x}|).$$

To illustrate, if $x_3 \geq x_1 \geq x_2$ and $|x_2| \geq |x_3| \geq |x_1|$, then

- $\operatorname{argsort}(\boldsymbol{x}) = (3, 1, 2)$
- $\operatorname{sort}(\boldsymbol{x}) = (x_3, x_1, x_2)$
- $\operatorname{rank}(\boldsymbol{x}) = (2, 3, 1)$
- $\operatorname{topkmask}(\boldsymbol{x}) = (1, 0, 1)$
- $\operatorname{topk}(\boldsymbol{x}) = (x_1, 0, x_3)$,
- $\operatorname{topkmag}(\boldsymbol{x}) = (0, x_2, x_3)$,

where in the last three, we used $k = 2$.

Permutahedron. The permutahedron associated with a vector $\boldsymbol{w} \in \mathbb{R}^n$, a well-known object in combinatorics (Bowman, 1972; Ziegler, 2012), is the convex hull of the permutations of $\boldsymbol{w}$, i.e.,

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

We define the linear maximization oracles (LMO) associated with $P(\boldsymbol{w})$ by

$$\begin{aligned} f(\boldsymbol{x}, \boldsymbol{w}) &:= \max_{\boldsymbol{y} \in P(\boldsymbol{w})} \langle \boldsymbol{x}, \boldsymbol{y} \rangle \\ \boldsymbol{y}(\boldsymbol{x}, \boldsymbol{w}) &:= \operatorname{argmax}_{\boldsymbol{y} \in P(\boldsymbol{w})} \langle \boldsymbol{x}, \boldsymbol{y} \rangle = \nabla_1 f(\boldsymbol{x}, \boldsymbol{w}), \end{aligned} \quad (1)$$

where $\nabla_1 f(\boldsymbol{x}, \boldsymbol{w})$ is technically a subgradient of f w.r.t. $\boldsymbol{x}$. The LMO can be computed in $O(n \log n)$ time. Indeed, the calculation of the LMO reduces to a sorting operation, as shown in the following known proposition. A proof is included for completeness in Appendix A.1.

> **Proposition 1.** *(Linear maximization oracles)*
>
> *If $w_1 \geq \cdots \geq w_n$ (if not, sort $\boldsymbol{w}$), then*
>
> $$f(\boldsymbol{x}, \boldsymbol{w}) = \sum_{i=1}^n w_i x_{[i]} \quad and \quad \boldsymbol{y}(\boldsymbol{x}, \boldsymbol{w}) = \boldsymbol{w}_{\mathrm{rank}(\boldsymbol{x})}.$$

LP formulations. Let us denote the reversing permutation by $\boldsymbol{\rho} := (n, n-1, \ldots, 1)$. Blondel et al. (2020b) showed that the sort and rank operators can be formulated as linear programs (LP) over the permutahedron:

$$\mathrm{sort}(\boldsymbol{x}) = \boldsymbol{y}(\boldsymbol{\rho}, \boldsymbol{x}) = \underset{\boldsymbol{y} \in P(\boldsymbol{x})}{\mathrm{argmax}} \langle \boldsymbol{\rho}, \boldsymbol{y} \rangle$$

$$\mathrm{rank}(\boldsymbol{x}) = \boldsymbol{y}(-\boldsymbol{x}, \boldsymbol{\rho}) = \underset{\boldsymbol{y} \in P(\boldsymbol{\rho})}{\mathrm{argmax}} \langle -\boldsymbol{x}, \boldsymbol{y} \rangle.$$

In the latter expression, the minus sign is due to the fact that we use the convention that smaller rank indicates higher value (i.e., the maximum value has rank 1).

Although not mentioned by Blondel et al. (2020b), it is also easy to express the top-k mask operator as an LP

$$\mathrm{topkmask}(\boldsymbol{x}) = \boldsymbol{y}(\boldsymbol{x}, \boldsymbol{1}_k) = \underset{\boldsymbol{y} \in P(\boldsymbol{1}_k)}{\mathrm{argmax}} \langle \boldsymbol{x}, \boldsymbol{y} \rangle, \quad (2)$$

where $\boldsymbol{1}_k := (\underbrace{1, ..., 1}_{k}, \underbrace{0, \ldots, 0}_{n-k})$. For this choice of $\boldsymbol{w}$, the permutahedron enjoys a particularly simple expression

$$P(\boldsymbol{1}_k) = \{\boldsymbol{y} \in \mathbb{R}^n \colon \langle \boldsymbol{y}, \boldsymbol{1} \rangle = k, \boldsymbol{y} \in [0, 1]^n\}$$

and $P(\boldsymbol{1}_k/k)$ is known as the capped simplex (Warmuth & Kuzmin, 2008; Blondel et al., 2020a). This is illustrated in Figure 2. To obtain relaxed operators, Blondel et al. (2020b) proposed to introduce regularization in (1) (see "recovering the previous framework" in the next section) and used a reduction to isotonic optimization.

4. Proposed generalized framework

In this section, we generalize the framework of Blondel et al. (2020b) by adding an optional nonlinearity $\varphi(\boldsymbol{x})$. In addition to the operators covered by the previous framework, this allows us to directly express the top-k in magnitude operator, which was not possible before. We also support p-norm regularization, which allows to express differentiable *and* sparse operators when $1 < p < 2$.

Introducing a mapping φ. Consider a mapping $\varphi(\boldsymbol{x}) := (\phi(x_1), \ldots, \phi(x_n))$. Given $\boldsymbol{x} \in \mathbb{R}^n$ and $\boldsymbol{w} \in \mathbb{R}^n$, we define

$$f_\varphi(\boldsymbol{x}, \boldsymbol{w}) := f(\varphi(\boldsymbol{x}), \boldsymbol{w})$$
$$\boldsymbol{y}_\varphi(\boldsymbol{x}, \boldsymbol{w}) := \nabla_1 f_\varphi(\boldsymbol{x}, \boldsymbol{w}).$$

When $\varphi(\boldsymbol{x}) = \boldsymbol{x}$ (identity mapping), we clearly recover the existing framework, i.e., $f_\varphi(\boldsymbol{x}, \boldsymbol{w}) = f(\boldsymbol{x}, \boldsymbol{w})$ and $\boldsymbol{y}_\varphi(\boldsymbol{x}, \boldsymbol{w}) = \boldsymbol{y}(\boldsymbol{x}, \boldsymbol{w})$.

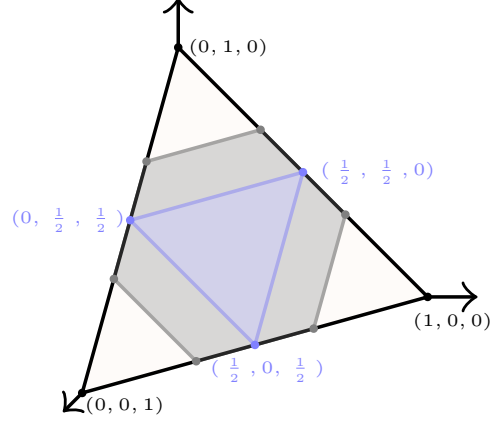*Figure 2.* The permutahedron $P(\boldsymbol{w})$ is a polytope whose vertices are permutations of $\boldsymbol{w}$. Depending, on the choice of $\boldsymbol{w}$, it can express several known polytopes. When $\boldsymbol{w} = (1, 0, 0)$, $P(\boldsymbol{w})$ is the probability simplex (light beige), which corresponds to the top-1 setting. When $\boldsymbol{w} = (\frac{1}{2}, \frac{1}{2}, 0)$, $P(\boldsymbol{w})$ is the capped probability simplex (blue), which corresponds to the top-k setting (here, with $k = 2$). When $\boldsymbol{w} = (\frac{2}{3}, \frac{1}{3}, 0)$, $P(\boldsymbol{w})$ is an hexagon, which corresponds to the partial ranking setting (gray).

When $\varphi(\boldsymbol{x}) \neq \boldsymbol{x}$, our framework starts to differ from the previous one, since φ affects differentiation. By the chain rule and Danskin's theorem (1966), we get

$$\begin{aligned}
\boldsymbol{y}_\varphi(\boldsymbol{x}, \boldsymbol{w}) &:= \nabla_1 f_\varphi(\boldsymbol{x}, \boldsymbol{w}) \\
&= \partial\varphi(\boldsymbol{x})^\top \boldsymbol{y}(\varphi(\boldsymbol{x}), \boldsymbol{w}) \\
&= (\phi'(x_1), \ldots, \phi'(x_n)) \circ \boldsymbol{y}(\varphi(\boldsymbol{x}), \boldsymbol{w}),
\end{aligned} \quad (3)$$

where $\partial\varphi(\boldsymbol{x}) \in \mathbb{R}^{n \times n}$ denotes the Jacobian of $\varphi(\boldsymbol{x})$ and $\boldsymbol{y}(\boldsymbol{x}, \boldsymbol{w})$ is given by Proposition 1.

Top-k in magnitude. As we emphasized, one advantage of our proposed generalization is that we can express the top-k in magnitude operator. Indeed, with $\phi(x) = \frac{1}{2}x^2$, we can see from (2) and (3) that we have for all $\boldsymbol{x} \in \mathbb{R}^n$

$$\mathrm{topkmag}(\boldsymbol{x}) = \boldsymbol{y}_\varphi(\boldsymbol{x}, \boldsymbol{1}_k) = \nabla_1 f_\varphi(\boldsymbol{x}, \boldsymbol{1}_k). \quad (4)$$

Obviously, for all $\boldsymbol{x} \in \mathbb{R}_+^n$, we also have $\mathrm{topkmag}(\boldsymbol{x}) = \mathrm{topk}(\boldsymbol{x})$. Top-k in magnitude is useful for pruning weights with small magnitude in a neural network, as we demonstrate in our experiments in §6.

Introducing regularization. We now explain how to make our generalized operator differentiable. We introduce convex regularization $R : \mathbb{R}^n \to \mathbb{R}$ in the dual space:

$$f_{\varphi,R}^*(\boldsymbol{y}, \boldsymbol{w}) := f_\varphi^*(\boldsymbol{y}, \boldsymbol{w}) + R(\boldsymbol{y}),$$

where f_φ^* is the conjugate of f_φ in the first argument. Going back to the primal space, we obtain a new relaxed operator. A proof is given in Appendix A.2.

Proposition 2. *(Relaxed operator)*

Let $R\colon \mathbb{R}^n \to \mathbb{R}$ be a convex regularizer. Then

$$f_{\varphi,R}(\boldsymbol{x}, \boldsymbol{w}) := \max_{\boldsymbol{y} \in \mathbb{R}^n} \langle \boldsymbol{y}, \boldsymbol{x} \rangle - f_\varphi^*(\boldsymbol{y}, \boldsymbol{w}) - R(\boldsymbol{y})$$

$$= \min_{\boldsymbol{u} \in \mathbb{R}^n} R^*(\boldsymbol{x} - \boldsymbol{u}) + f_\varphi(\boldsymbol{u}, \boldsymbol{w})$$

$$\boldsymbol{y}_{\varphi,R}(\boldsymbol{x}, \boldsymbol{w}) := \boldsymbol{y}^\star = \nabla R^*(\boldsymbol{x} - \boldsymbol{u}^\star) = \nabla_1 f_{\varphi,R}(\boldsymbol{x}, \boldsymbol{w}).$$

The mapping φ also affects conjugacy. We have the following proposition.

Proposition 3. *(Conjugate of f_φ in the first argument)*

If ϕ is convex and $\boldsymbol{w} \in \mathbb{R}_+^n$, then

$$f_\varphi^*(\boldsymbol{y}, \boldsymbol{w}) = \min_{\boldsymbol{z} \in P(\boldsymbol{w})} D_{\phi^*}(\boldsymbol{y}, \boldsymbol{z}),$$

where $D_f(\boldsymbol{y}, \boldsymbol{z}) := \sum_{i=1}^n z_i f(y_i/z_i)$.

A proof is given in Appendix A.3. The function $(y_i, z_i) \mapsto y_i' \phi^*(y_i/z_i)$ is known as the perspective of ϕ^* and is jointly convex when $z_i > 0$. The function $D_f(\boldsymbol{y}, \boldsymbol{z})$ is known as the f-divergence between $\boldsymbol{y}$ and $\boldsymbol{z}$. Therefore, $f_\varphi^*(\boldsymbol{y}, \boldsymbol{w})$ can be seen as the minimum "distance" between $\boldsymbol{y}$ and $P(\boldsymbol{w})$ in the ϕ^*-divergence sense.

Recovering the previous framework. If $\phi(x) = x$, then

$$\phi^*(y_i/z_i) = \begin{cases} 0, & \text{if } y_i = z_i \\ \infty, & \text{otherwise} \end{cases}.$$

This implies that $f_\varphi^*(\boldsymbol{y}, \boldsymbol{w}) = f^*(\boldsymbol{y}, \boldsymbol{w}) = \delta_{P(\boldsymbol{w})}(\boldsymbol{y})$, the indicator function of $P(\boldsymbol{w})$, which is 0 if $\boldsymbol{y} \in P(\boldsymbol{w})$ and ∞ otherwise. In this case, we therefore obtain

$$f_{\varphi,R}(\boldsymbol{x}, \boldsymbol{w}) = \max_{\boldsymbol{y} \in P(\boldsymbol{w})} \langle \boldsymbol{y}, \boldsymbol{x} \rangle - R(\boldsymbol{y}),$$

which is exactly the relaxation of Blondel et al. (2020b).

Differentiable and sparse top-k operators. To obtain a relaxed top-k operator with our framework, we simply replace f_φ with $f_{\varphi,R}$ and $\boldsymbol{y}_\varphi$ with $\boldsymbol{y}_{\varphi,R}$ in (4) to define

$$\mathrm{topkmag}_R(\boldsymbol{x}) := \boldsymbol{y}_{\varphi,R}(\boldsymbol{x}, \boldsymbol{1}_k) = \nabla_1 f_{\varphi,R}(\boldsymbol{x}, \boldsymbol{1}_k).$$

A relaxed top-k mask can be defined in a similar way, but using $\phi(x) = x$ instead of $\phi(x) = \frac{1}{2}x^2$. For the regularization R, we propose to use p-norms to the power p:

$$R(\boldsymbol{y}) = \frac{1}{p}\|\boldsymbol{y}\|_p^p := \frac{1}{p}\sum_{i=1}^n |y_i|^p.$$

The choice $p = 2$ used in previous works leads to sparse outputs but is not differentiable everywhere. Any p between

1 (excluded) and 2 (excluded) leads to differentiable *and* sparse outputs. We propose to use $p = 4/3$ to obtain a differentiable everywhere operator, and $p = 2$ to obtain a differentiable a.e. operator, which is more convenient numerically. This is illustrated in Figure 3.

Connection with k-support and OWL norms. When $\phi(x) = \frac{1}{2}x^2$ and $\boldsymbol{w} = \boldsymbol{1}_k$, we obtain

$$f_\varphi^*(\boldsymbol{y}, \boldsymbol{w}) = \frac{1}{2}\min_{\boldsymbol{z} \in [0,1]^n} \sum_{i=1}^n \frac{y_i^2}{z_i} \quad \text{s.t.} \quad \langle \boldsymbol{z}, \boldsymbol{1} \rangle = k,$$

which is known as the squared k-support norm (Argyriou et al., 2012; McDonald et al., 2014; Eriksson et al., 2015). Our formulation is a generalization of the squared k-support norm, as it supports other choices of $\boldsymbol{w}$ and ϕ. For instance, we use it to define a new notion of k-support negentropy in Appendix B.1. When $\varphi(\boldsymbol{x}) = |\boldsymbol{x}|$, we recover the ordered weighted lasso (OWL) norm (Zeng & Figueiredo, 2014) as

$$f_\varphi(\boldsymbol{x}, \boldsymbol{w}) = \sum_{i=1}^n w_i |x|_{[i]}.$$

With that choice of φ, it is easy to see from (3) that (4) becomes a *signed* top-k mask. Note that, interestingly, k-support and OWL norms are not defined in the same space.

Biconjugate interpretation. Let us define the set of k-sparse vectors, which is nonconvex, as $S_k := \{\boldsymbol{x} \in \mathbb{R}^n \colon \|\boldsymbol{x}\|_0 \le k\}$, where $\|\boldsymbol{x}\|_0$ is the number of non zero elements in $\boldsymbol{x}$. We saw that if $\phi(x) = \frac{1}{2}x^2$ then $f_\varphi^*(\boldsymbol{y}, \boldsymbol{1}_k)$ is the squared k-support norm. It is known to be the biconjugate (i.e., the tightest convex relaxation) of the squared L_2 norm restricted to S_k (Eriksson et al., 2015; Liu et al., 2022). We now prove a more general result: $f_\varphi^*(\boldsymbol{y}, \boldsymbol{1}_k)$ is the biconjugate of $\sum_{i=1}^n \phi^*(y_i)$ restricted to S_k.

Proposition 4. *(Biconjugate interpretation)*

Let $\Phi_k(\boldsymbol{y}) := \sum_{i=1}^n \phi^*(y_i) + \delta_{S_k}(\boldsymbol{y})$. Suppose that ϕ is convex. Then the biconjugate of Φ_k is given by

$$\Phi_k^{**}(\boldsymbol{y}) = f_\varphi^*(\boldsymbol{y}, \boldsymbol{1}_k)$$

See Appendix A.4 for a proof.

5. Algorithms

In this section, we propose efficient algorithms for computing our operators. We first show that the calculation of our relaxed operator reduces to isotonic optimization.

Reduction to isotonic optimization. We now show how to compute $\boldsymbol{u}^\star$ in Proposition 2 by reduction to isotonic optimization, from which $\boldsymbol{y}^\star$ can then be recovered by $\boldsymbol{y}^\star =$
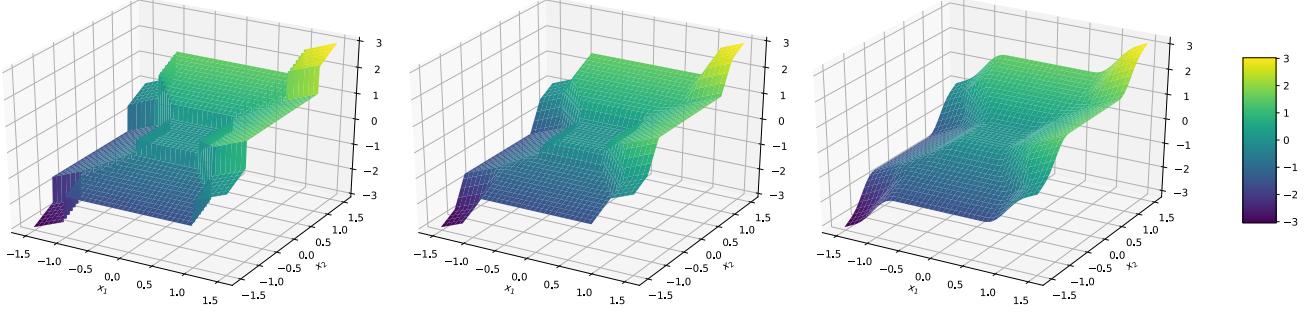
Figure 3. **Example of our relaxed top-k operators.** We take $\phi(x) = \frac{1}{2}x^2$ and $k = 2$. For an input $\boldsymbol{x} = (x_1, x_2, \frac{1}{2}, 1)$, we plot $y_{\varphi,R}(\boldsymbol{x}, 1_k)_1 + y_{\varphi,R}(\boldsymbol{x}, 1_k)_2$ for $R = 0$ (left), $R = \frac{\lambda}{2}\|\boldsymbol{x}\|_2^2$ (center) and $R = \frac{\lambda}{p}\|\boldsymbol{x}\|_p^p$ with $p = \frac{4}{3}$ (right). We take $\lambda = 0.3$. While no regularization leads to a discontinuous mapping, the 2-norm regularization leads to continuity and a.e. differentiability, and the $\frac{4}{3}$-norm regularization provides a continuously differentiable mapping. We emphasize that, although in the left plot the graph looks connected, it is actually a discontinuous function. Note that our relaxed operators are **sparse** as they are exactly 0 in the center.

$\nabla R^*(\boldsymbol{x} - \boldsymbol{u}^\star)$. We first recall the case $\varphi(\boldsymbol{x}) = \boldsymbol{x}$, which was already proved in existing works (Lim & Wright, 2016; Blondel et al., 2020b).

Proposition 5. *(Reduction, $\varphi(\boldsymbol{x}) = \boldsymbol{x}$ case)*

Suppose that $R(\boldsymbol{y}) = \sum_{i=1}^n r(y_i)$. Let σ be the permutation sorting $\boldsymbol{x}$, $\boldsymbol{s} := \boldsymbol{x}_\sigma$ and

$$\boldsymbol{v}^\star = \operatorname*{argmin}_{v_1 \geq \cdots \geq v_n} R^*(\boldsymbol{s} - \boldsymbol{v}) + f(\boldsymbol{v}, \boldsymbol{w}).$$

Then $\boldsymbol{u}^\star$ from Proposition 2 is given by $\boldsymbol{u}^\star = \boldsymbol{v}^\star_{\sigma^{-1}}$.

The set $\{\boldsymbol{v} \in \mathbb{R}^n : v_1 \geq \cdots \geq v_n\}$ is called the monotone cone. Next, we show that a similar result is possible when $\phi(\boldsymbol{x})$ and r^* are both even functions (sign-invariant) and increasing on $\mathbb{R}_+$.

Proposition 6. *(Reduction, $\varphi(\boldsymbol{x}) = \varphi(-\boldsymbol{x})$ case)*

Suppose that $R(\boldsymbol{y}) = \sum_{i=1}^n r(y_i)$ and $\varphi(\boldsymbol{x}) = (\phi(x_1), \ldots, \phi(x_n))$. Assume ϕ and r^* are both even functions (sign-invariant) and increasing on $\mathbb{R}_+$. Let σ be the permutation sorting $|\boldsymbol{x}|$, $\boldsymbol{s} := |\boldsymbol{x}|_\sigma$ and

$$\boldsymbol{v}^\star = \operatorname*{argmin}_{v_1 \geq \cdots \geq v_n \geq 0} R^*(\boldsymbol{s} - \boldsymbol{v}) + f_\varphi(\boldsymbol{v}, \boldsymbol{w}).$$

Then, $\boldsymbol{u}^\star$ (Proposition 2) is equal to $\operatorname{sign}(\boldsymbol{x}) \circ \boldsymbol{v}^\star_{\sigma^{-1}}$.

See Appendix A.6 for a proof. Less general results are proved in (Zeng & Figueiredo, 2014; Eriksson et al., 2015) for specific cases of ϕ and R. The set $\{v \in \mathbb{R}^n : v_1 \geq \cdots \geq v_n \geq 0\}$ is called the non-negative monotone cone. In practice, the additional non-negativity constraint is easy to handle: we can solve the isotonic optimization problem without it and truncate the solution if it is not non-negative (Németh & Németh, 2012).

Pool adjacent violator (PAV) algorithms. Under the conditions of Proposition 6, assuming $\boldsymbol{v}$ and $\boldsymbol{w}$ are both sorted, we have from Proposition 1 that

$$f(\boldsymbol{v}, \boldsymbol{w}) = \sum_{i=1}^n w_i v_i, \quad f_\varphi(\boldsymbol{v}, \boldsymbol{w}) = \sum_{i=1}^n w_i \phi(v_i).$$

We then get that the problems in Proposition 5 and 6 are coordinate-wise separable:

$$\boldsymbol{v}^\star = \operatorname*{argmin}_{v_1 \geq \cdots \geq v_n} \sum_{i=1}^n h_i(v_i), \tag{5}$$

for $h_i(v_i) = r^*(s_i - v_i) + w_i \phi(v_i)$. Such problems can be solved in $O(n)$ time using the pool adjacent violator (PAV) algorithm (Best et al., 2000). This algorithm works by partitioning the set $[n]$ into disjoint sets $(B_1, \cdots B_m)$, starting from $m = n$ and $B_i = \{i\}$, and by merging these sets until the isotonic condition is met. A pseudo-code is available for completness in Appendix B.2. At its core, PAV simply needs a routine to solve the "pooling" subproblem

$$\gamma_B^\star = \operatorname*{argmin}_{\gamma \in \mathbb{R}} \sum_{i \in B} h_i(\gamma) \tag{6}$$

for any $B \subseteq [n]$. Once the optimal partition $(B_1, \cdots B_m)$ is identified, we have that

$$\boldsymbol{v}^\star = (\underbrace{\gamma_{B_1}^\star, \ldots, \gamma_{B_1}^\star}_{|B_1|}, \cdots, \underbrace{\gamma_{B_m}^\star, \ldots, \gamma_{B_m}^\star}_{|B_m|}) \in \mathbb{R}^n.$$

Because Proposition 5 and 6 require to obtain the sorting permutation σ beforehand, the total time complexity for our operators is $O(n \log n)$.

Example. Suppose $\phi(x) = \frac{1}{2}x^2$ and $R = \frac{\lambda}{2}\|.\|_2^2$, where $\lambda > 0$. Since $R^* = \frac{1}{2\lambda}\|.\|^2$, this gives $h_i(v_i) = \frac{1}{2}(w_i v_i^2 +$

$\frac{1}{\lambda}(s_i - v_i)^2$. The solution of the sub-problem is then

$$\gamma_B^\star = \frac{\sum_{i \in B} s_i}{\sum_{i \in B}(\lambda w_i + 1)}.$$

Using this formula, when λ is small enough, we can upper-bound the error between the hard and relaxed operators: $\|\text{topkmag}_R(\boldsymbol{x}) - \text{topkmag}(\boldsymbol{x})\|_\infty \leq \lambda \|\boldsymbol{x}\|_\infty$. See Appendix A.6 for details and for the case $p = \frac{4}{3}$.

Dykstra's alternating projection algorithm. The PAV algorithm returns an exact solution of (5) in $O(n)$ time. Unfortunately, it relies on element-wise dynamical array assignments, which makes it potentially slow on GPUs and TPUs. We propose an alternative to obtain faster computations. Our key insight is that by defining

$$C_1 := \{\boldsymbol{v} \in \mathbb{R}^n : v_1 \geq v_2, v_3 \geq v_4, \dots\}$$
$$C_2 := \{\boldsymbol{v} \in \mathbb{R}^n : v_2 \geq v_3, v_4 \geq v_5, \dots\},$$

one has $\{\boldsymbol{v} \in \mathbb{R}^n : v_1 \geq \cdots \geq v_n\} = C_1 \cap C_2$. We can therefore rewrite (5) as

$$\boldsymbol{v}^\star = \underset{\boldsymbol{v} \in C_1 \cap C_2}{\operatorname{argmin}} \sum_{i=1}^n h_i(v_i).$$

In the case $R = \frac{1}{2}\|.\|^2$ and $\phi(x) = x$ or $\frac{1}{2}x^2$, this reduces to a projection onto $C_1 \cap C_2$, for which we can use Dykstra's celebrated projection algorithm (Boyle & Dykstra, 1986; Combettes & Pesquet, 2011). Jegelka et al. (2013) have used this method for computing the projection onto the intersection of submodular polytopes, whereas we project onto a single polyhedral face. When $\phi(x) = x$, Dykstra's projection algorithm takes the simple form in Algorithm 1.

Algorithm 1. *(Dykstra's projection algorithm)*

Starting from $\boldsymbol{v}^0 = \boldsymbol{s}$, $\boldsymbol{p}^0 = \boldsymbol{q}^0 = \boldsymbol{0}$, Dykstra's algorithm iterates

$$\boldsymbol{y}^k = \underset{\boldsymbol{y} \in C_1}{\operatorname{argmin}} \frac{1}{2}\|\boldsymbol{v}^k + \boldsymbol{p}^k - \boldsymbol{y}\|_2^2 + \langle \boldsymbol{y}, \boldsymbol{w} \rangle$$
$$\boldsymbol{p}^{k+1} = \boldsymbol{v}^k + \boldsymbol{p}^k - \boldsymbol{y}^k$$
$$\boldsymbol{v}^{k+1} = \underset{\boldsymbol{v} \in C_2}{\operatorname{argmin}} \frac{1}{2}\|\boldsymbol{y}^k + \boldsymbol{q}^k - \boldsymbol{v}\|_2^2 + \langle \boldsymbol{v}, \boldsymbol{w} \rangle$$
$$\boldsymbol{q}^{k+1} = \boldsymbol{y}^k + \boldsymbol{q}^k - \boldsymbol{v}^{k+1}.$$

Each argmin calculation corresponds to a Euclidean projection onto C_1 or C_2, which can be computed in closed form. Therefore, $\boldsymbol{v}^k$ provably converges to $\boldsymbol{v}^\star$. Each iteration of Dykstra's algorithm is learning-rate free, has linear time complexity and can be efficiently written as a matrix-vector product using a mask, which makes it particularly appealing for GPUs and TPUs. Interestingly, we find out that when $\boldsymbol{w} = \boldsymbol{1}_k$, then Dykstra's algorithm converges surprisingly

fast to the exact solution. We validate that Dykstra leads to a faster runtime than PAV in Figure 4. We use 100 iterations of Dykstra, and verify that we obtain the same output as PAV.

For the general non-Euclidean R^* case, i.e., $p \neq 2$, we can use block coordinate ascent in the dual of (5). We can divide the dual variables into two blocks, corresponding to C_1 and C_2; see Appendix A.8. In fact, it is known that in the Euclidean case, Dykstra's algorithm in the primal and block coordinate ascent in the dual are equivalent (Tibshirani, 2017). Therefore, although a vectorized implementation could be challenging for $p \neq 2$, block coordinate ascent can be seen as an elegant way to generalize Dykstra's algorithm. Convergence is guaranteed as long as each h_i is strictly convex.

Differentiation. The Jacobian of the solution of the isotonic optimization problems can be expressed in closed form for any p-norm regularization.
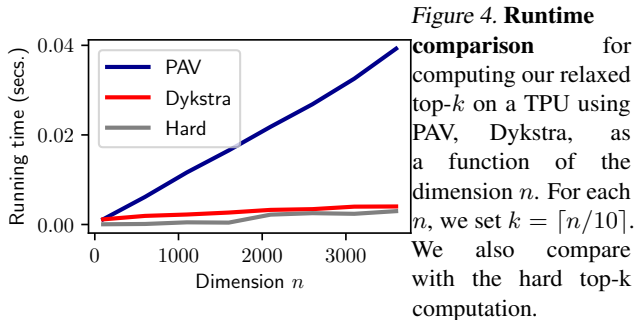

Proposition 7. *(Differentiation)*

Let $\boldsymbol{v}^\star = (\gamma_{B_1}^\star, \dots, \gamma_{B_1}^\star \cdots, \gamma_{B_m}^\star, \dots, \gamma_{B_m}^\star)$ be the optimal solution of the isotonic optimization problem with $R = \frac{1}{p}\|.\|_p^p$ and $p > 0$. Then one has that $\boldsymbol{v}^\star$ is differentiable with respect to $\boldsymbol{s} = (s_1, \dots, s_n)$. Furthermore, for any $r \in \{1, \dots, m\}$ and $i \in B_r$,

$$\frac{\partial \gamma_{B_r}^\star}{\partial s_i} = \begin{cases} \dfrac{|\gamma_{B_r}^\star - s_i|^{q-2}}{\sum_{j \in B_r} |\gamma_{B_r}^\star - s_j|^{q-2}} & \text{if } \phi(x) = x \\[2ex] \dfrac{(q-1)|\gamma_{B_r}^\star - s_i|^{q-2}}{\sum_{j \in B_r}(q-1)|\gamma_{B_r}^\star - s_j|^{q-2} + w_j} & \text{if } \phi(x) = \frac{1}{2}x^2 \end{cases}$$

where q is such that $\frac{1}{p} + \frac{1}{q} = 1$. When $i \notin B_r$, one simply has $\frac{\partial \gamma_{B_r}^\star}{\partial s_i} = 0$.

One then has $\partial v_j^\star / \partial s_i = \partial \gamma_{B_{r_j}}^\star / \partial s_i$ where r_j is such that $v_j^\star = \gamma_{B_{r_j}}^\star$. See Appendix A.7 for a proof. Thanks to Proposition 7, we do not need to solve a linear system to compute the Jacobian of the solution $\boldsymbol{v}^\star$, in contrast to implicit differentiation of general optimization problems (Blondel et al., 2021). In practice, this also means that we do not need to perform backpropagation through the unrolled iterations of PAV or Dykstra's projection algorithm to obtain the gradient of a scalar loss function, in which our operator is incorporated. In particular, we do not need to store the intermediate iterates of these algorithms in memory. Along with PAV and Dykstra's projection algorithm, we implement the corresponding Jacobian vector product routines in JAX, using Proposition 7.

Figure 4. **Runtime comparison** for computing our relaxed top-k on a TPU using PAV, Dykstra, as a function of the dimension n. For each n, we set $k = \lceil n/10 \rceil$. We also compare with the hard top-k computation.

6. Experiments

We now demonstrate the applicability of our top-k operators through experiments. Our JAX (Bradbury et al., 2018) implementation is available at the following URL. See Appendix C for additional experimental details.
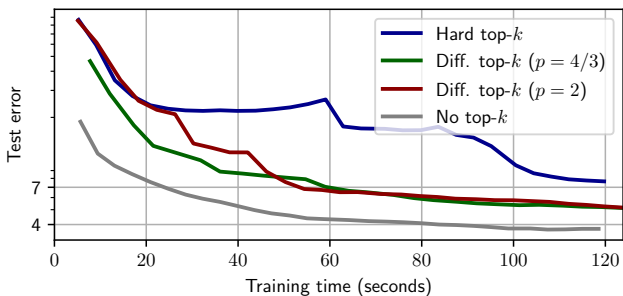
Figure 5. **Test error** with respect to training time when training an MLP on MNIST. We compare the baseline (grey) with the case where 90% of the weights are set to 0 by magnitude pruning, using a differentiable a.e. (red), fully differentiable (green) or a hard top-k (blue).

Weight pruning in neural networks. We experimentally validate the advantage of using a smoothed top-k for weight pruning in neural networks. We use a multilayer perceptron (MLP) with 2 hidden layers and with ReLU activation. The width of the layers are respectively 784, 32, 32 followed by a linear classification head of width 10. More precisely, our model takes as input an image $\boldsymbol{a} \in \mathbb{R}^{784}$ and outputs

$$\boldsymbol{x} = W_3\sigma(W_2\sigma(W_1\boldsymbol{a} + \boldsymbol{b}_1) + \boldsymbol{b}_2) + \boldsymbol{b}_3 \quad \text{(logits)},$$

where $W_1 \in \mathbb{R}^{32\times784}$, $\boldsymbol{b}_1 \in \mathbb{R}^{32}$, $W_2 \in \mathbb{R}^{32\times32}$, $\boldsymbol{b}_2 \in \mathbb{R}^{32}$, $W_1 \in \mathbb{R}^{10\times32}$, $\boldsymbol{b}_3 \in \mathbb{R}^{10}$ and σ is a ReLU. In order to perform weight pruning we parametrize each W_i as $W_i = \text{topkmag}_R(W_i')$ and learn W_i' instead of learning W_i directly. The output is then fed into a cross-entropy loss. We compare the performance of the model when applying a hard vs differentiable top-k operator to keep only 10% of the coefficients. For the differentiable top-k, we use a regularization $R(\boldsymbol{y}) = \frac{\lambda}{p}\|\boldsymbol{y}\|^p$ with $p \in \{\frac{4}{3}, 2\}$ and $\lambda = 10^{-4}$. We find out that the model trained with the differentiable top-k trains significantly faster than the one trained with the hard top-k. We also verify that our relaxed

top-k maintains the 10% rate of non-zero weights. Results on MNIST are displayed in Figure 5. We also compare with an entropy-regularized approximation of the top-k operator using the framework proposed in Cuturi et al. (2019) and adapted in Petersen et al. (2022). To guarantee the sparsity of the weights, we use the "straight-through" trick: the hard top-k is run on the forward pass but we use the gradient of the relaxed top-k in the backward pass. This method leads to a test error of 5.9%, which is comparable to the results obtained with our differentiable operators.

Smooth top-k loss. To train a neural network on a classification task, one typically minimizes the cross-entropy loss, whereas the performance of the network is evaluated using a top-k test accuracy. There is therefore a mismatch between the loss used at train time and the metric used at evaluation time. Cuturi et al. (2019) proposed to replace the cross-entropy loss with a differentiable top-k loss. In the same spirit, we propose to finetune a ViT-B/16 (Dosovitskiy et al., 2020) pretrained on the ImageNet21k dataset on CIFAR 100 (Krizhevsky et al., 2009) using a smooth and sparse top-k loss instead of the cross-entropy loss. We use a Fenchel-Young loss (Blondel et al., 2020a). It takes as input the vector $\boldsymbol{a}$ and parameters θ of a neural network g_θ:

$$\boldsymbol{x} = g_\theta(\boldsymbol{a}) \quad \text{(logits)}$$
$$\ell(\boldsymbol{x}, \boldsymbol{t}) = f_{\varphi,R}(\boldsymbol{x}, \mathbf{1}_k) - \langle \boldsymbol{x}, \boldsymbol{t} \rangle,$$

where $f_{\varphi,R}(\boldsymbol{x}, \mathbf{1}_k)$ is given by Proposition 2 and $\boldsymbol{t}$ is a one-hot encoding of the class of $\boldsymbol{a}$. We set $\phi(x) = x$ as we want a top-k mask. We consider p norm regularizations for R, where $p = 2$ or $p = 4/3$. We take $k = 3$. We use the exact same training procedure as described in Dosovitskiy et al. (2020) and use the corresponding pretrained ViT model B/16, and train our model for 100 steps. Results are reported in Figure 6. We find that the ViT finetuned with the smooth top-3 loss outperforms the one finetuned with the cross-entropy loss in terms of top-k error, for various k.
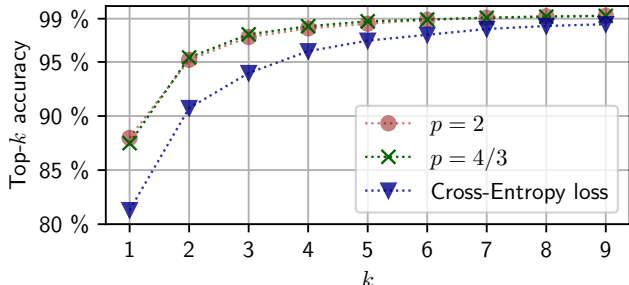
Figure 6. **Validation top-k accuracy** when fine-tuning a ViT-B/16 on CIFAR 100 using either the cross-entropy loss or our smooth top-3 loss for training. We have the following running times obtained with a TPUv3-8. Baseline: 9.5 sec/step, $p = 2$: 9.7 sec/step and $p = 4/3$: 10 sec/step.

Sparse MoEs. Finally, we demonstrate the applicability of our proposed smooth top-k operators on a large-scale classification task using vision sparse mixture of experts (V-MoE) (Riquelme et al., 2021). Vision transformers (ViTs) are made of a succession of self-attention layers and MLP layers. The idea of V-MoEs is to replace MLPs in ViTs by a sparsely-gated mixture of MLPs called experts. This way, only some of the experts are activated by a given patch token. At the heart of the token-expert assignment lies a routing mechanism which performs a top-k operation on gate values. We focus on the MoE with expert choice routing framework (Zhou et al., 2022), where each expert is assigned to k tokens. We train a S/32 variant of the V-MoE model, with 32×32 patches on the JFT-300M dataset (Sun et al., 2017), a dataset with more than 305 million images. Our model has 32 experts, each assigned to $k = 28$ tokens selected among $n = 400$ at each MoE layer. We compare the validation accuracy when using the baseline (hard top-k) with our relaxed operator. We use $p = 2$ and Dykstra's projection algorithm, as we found it was the fastest method on TPU. We used the training procedure proposed by Zhou et al. (2022) to obtain a fair comparison with the baseline. Due to the large size of the JFT-300M dataset (305 million images), we performed one run, as in Liu et al. (2022). We find that our approach improves validation performance. Results are displayed in Figure 7.
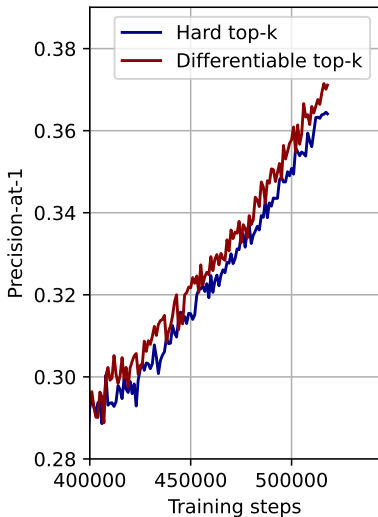

Figure 7. **Precision-at-1** on the JFT-300M dataset when using a hard top-k (baseline, in blue) or a differentiable a.e. one (in red) in a sparse MoE with a ViT-S/32. We zoom in on the last training steps, where our proposed method outperforms the baseline. The runtime is 10 hours for the baseline and 15 for the differentiable a.e. top-k (gradient calculation is the bottleneck here).

7. Discussion

Advantage of the non-linearity. As an alternative to performing a relaxed top-k operator in magnitude of x, one can perform a differentiable top-k mask on $|x|$, and then multiply the output by x. This alternative would also lead to a differentiable top-k operator in magnitude. However, our operator has more principled behavior at the limit cases. For instance, as $\lambda \to \infty$, it is easy to see that the relaxed

top-k mask converges to the vector $(k/n) \times 1_n$. Therefore, a rescaling by n/k is needed to obtain the identity as $\lambda \to \infty$, in contrast to our top-k in magnitude. From a theoretical point of view, the introduction of a non-linearity allows us to draw connections with the k-support norm. It also has a bi-conjugate interpretation, which we believe has an interest by itself.

Sensitivity to the choice of p. The subproblem needed within PAV enjoys a closed form only for specific choices of p. This is why we focused on $p = 2$ and $p = \frac{4}{3}$ in our experiments. However, we stress out that the proposed methods work for any choice of p. As an example, we provide the same illustration as for Figure 1 in Figure 8.
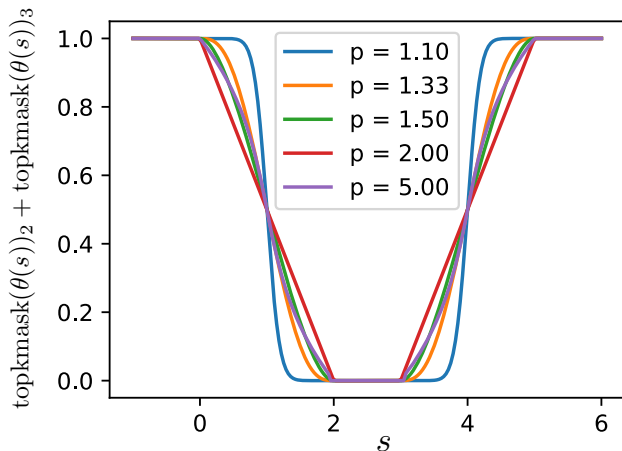

Figure 8. **Illustration of our differentiable and sparse top-k mask.** Same setup as for Figure 1, with more values for p.

8. Conclusion

In this work, we proposed a generalized framework to obtain fast, differentiable (or differentiable a.e.) and sparse top-k and top-k masks operators, including operators that select values in magnitude. Thanks to a reduction to isotonic optimization, we showed that these operators can be computed using either the Pool Adjacent Violators (PAV) algorithm or Dykstra's projection algorithm, the latter being faster on TPU hardware. We successfully demonstrated the usefulness of our operators for weight pruning, top-k losses and as routers in vision sparse mixture of experts.

References

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

Amos, B., Koltun, V., and Kolter, J. Z. The limited multi-label projection layer. *arXiv preprint arXiv:1906.08707*, 2019.

Argyriou, A., Foygel, R., and Srebro, N. Sparse prediction with the k-support norm. *Advances in Neural Information Processing Systems*, 25, 2012.

Beck, A. *First-order methods in optimization*. SIAM, 2017.

Berrada, L., Zisserman, A., and Kumar, M. P. Smooth loss functions for deep top-k classification. *International Conference on Learning Representations*, 2018.

Berthet, Q., Blondel, M., Teboul, O., Cuturi, M., Vert, J.-P., and Bach, F. Learning with differentiable pertubed optimizers. *Advances in neural information processing systems*, 33:9508–9519, 2020.

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.

Blalock, D., Gonzalez Ortiz, J. J., Frankle, J., and Guttag, J. What is the state of neural network pruning? *Proceedings of machine learning and systems*, 2:129–146, 2020.

Blondel, M. Structured prediction with projection oracles. *Advances in neural information processing systems*, 32, 2019.

Blondel, M., Martins, A. F., and Niculae, V. Learning with fenchel-young losses. *J. Mach. Learn. Res.*, 21(35):1–69, 2020a.

Blondel, M., Teboul, O., Berthet, Q., and Djolonga, J. Fast differentiable sorting and ranking. In *International Conference on Machine Learning*, pp. 950–959. PMLR, 2020b.

Blondel, M., Berthet, Q., Cuturi, M., Frostig, R., Hoyer, S., Llinares-López, F., Pedregosa, F., and Vert, J.-P. Efficient and modular implicit differentiation. *arXiv preprint arXiv:2105.15183*, 2021.

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

Boyle, J. P. and Dykstra, R. L. A method for finding projections onto the intersection of convex sets in hilbert spaces. In *Advances in order restricted statistical inference*, pp. 28–47. Springer, 1986.

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., and Zhang, Q. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

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

Collins, M. D. and Kohli, P. Memory bounded deep convolutional networks. *arXiv preprint arXiv:1412.1442*, 2014.

Combettes, P. L. and Pesquet, J.-C. Proximal splitting methods in signal processing. In *Fixed-point algorithms for inverse problems in science and engineering*, pp. 185–212. Springer, 2011.

Cordonnier, J.-B., Mahendran, A., Dosovitskiy, A., Weissenborn, D., Uszkoreit, J., and Unterthiner, T. Differentiable patch selection for image recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2351–2360, 2021.

Cuturi, M. Sinkhorn distances: Lightspeed computation of optimal transport. *Advances in neural information processing systems*, 26, 2013.

Cuturi, M., Teboul, O., and Vert, J.-P. Differentiable ranking and sorting using optimal transport. *Advances in neural information processing systems*, 32, 2019.

Danskin, J. M. The theory of max-min, with applications. *SIAM Journal on Applied Mathematics*, 14(4):641–664, 1966.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

Eriksson, A., Thanh Pham, T., Chin, T.-J., and Reid, I. The k-support norm and convex envelopes of cardinality and rank. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3349–3357, 2015.

Fedus, W., Zoph, B., and Shazeer, N. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity, 2021.

Fedus, W., Dean, J., and Zoph, B. A review of sparse expert models in deep learning. *arXiv preprint arXiv:2209.01667*, 2022.

Frankle, J. and Carbin, M. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

Gale, T., Elsen, E., and Hooker, S. The state of sparsity in deep neural networks. *arXiv preprint arXiv:1902.09574*, 2019.

Grover, A., Wang, E., Zweig, A., and Ermon, S. Stochastic optimization of sorting networks via continuous relaxations. In *International Conference on Learning Representations*, 2019.

Han, S., Pool, J., Tran, J., and Dally, W. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28, 2015.

Hazimeh, H., Zhao, Z., Chowdhery, A., Sathiamoorthy, M., Chen, Y., Mazumder, R., Hong, L., and Chi, E. Dselect-k: Differentiable selection in the mixture of experts with applications to multi-task learning. *Advances in Neural Information Processing Systems*, 34:29335–29347, 2021.

Järvelin, K. and Kekäläinen, J. Ir evaluation methods for retrieving highly relevant documents. In *ACM SIGIR Forum*, volume 51, pp. 243–250. ACM New York, NY, USA, 2017.

Jegelka, S., Bach, F., and Sra, S. Reflection methods for user-friendly submodular optimization. *Advances in Neural Information Processing Systems*, 26, 2013.

Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. 2009.

Kyrillidis, A., Becker, S., Cevher, V., and Koch, C. Sparse projections onto the simplex. In *International Conference on Machine Learning*, pp. 235–243. PMLR, 2013.

Lapin, M., Hein, M., and Schiele, B. Top-k multiclass svm. *Advances in Neural Information Processing Systems*, 28, 2015.

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

Lewis, M., Bhosale, S., Dettmers, T., Goyal, N., and Zettlemoyer, L. Base layers: Simplifying training of large, sparse models. In *International Conference on Machine Learning*, pp. 6265–6274. PMLR, 2021.

Lim, C. H. and Wright, S. J. Efficient bregman projections onto the permutahedron and related polytopes. In *Artificial Intelligence and Statistics*, pp. 1205–1213. PMLR, 2016.

Liu, T., Puigcerver, J., and Blondel, M. Sparsity-constrained optimal transport. *arXiv preprint arXiv:2209.15466*, 2022.

Malaviya, C., Ferreira, P., and Martins, A. F. Sparse and constrained attention for neural machine translation. *arXiv preprint arXiv:1805.08241*, 2018.

Martins, A. F. and Kreutzer, J. Learning what's easy: Fully differentiable neural easy-first taggers. In *Proceedings of the 2017 conference on empirical methods in natural language processing*, pp. 349–362, 2017.

McDonald, A. M., Pontil, M., and Stamos, D. Spectral k-support norm regularization. *Advances in neural information processing systems*, 27, 2014.

Németh, A. and Németh, S. How to project onto the monotone nonnegative cone using pool adjacent violators type algorithms. *arXiv preprint arXiv:1201.2343*, 2012.

Niepert, M., Minervini, P., and Franceschi, L. Implicit mle: backpropagating through discrete exponential family distributions. *Advances in Neural Information Processing Systems*, 34:14567–14579, 2021.

Petersen, F., Borgelt, C., Kuehne, H., and Deussen, O. Differentiable sorting networks for scalable sorting and ranking supervision. In *International Conference on Machine Learning*, pp. 8546–8555. PMLR, 2021.

Petersen, F., Kuehne, H., Borgelt, C., and Deussen, O. Differentiable top-k classification learning. In *International Conference on Machine Learning*, pp. 17656–17668. PMLR, 2022.

Prillo, S. and Eisenschlos, J. Softsort: A continuous relaxation for the argsort operator. In *International Conference on Machine Learning*, pp. 7793–7802. PMLR, 2020.

Qian, Y., Lee, J., Duddu, S. M. K., Dai, Z., Brahma, S., Naim, I., Lei, T., and Zhao, V. Y. Multi-vector retrieval as sparse alignment. *arXiv preprint arXiv:2211.01267*, 2022.

Riquelme, C., Puigcerver, J., Mustafa, B., Neumann, M., Jenatton, R., Susano Pinto, A., Keysers, D., and Houlsby, N. Scaling vision with sparse mixture of experts. *Advances in Neural Information Processing Systems*, 34: 8583–8595, 2021.

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.

Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q., Hinton, G., and Dean, J. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

Sinkhorn, R. Diagonal equivalence to matrices with prescribed row and column sums. *The American Mathematical Monthly*, 74(4):402–405, 1967.

Sun, C., Shrivastava, A., Singh, S., and Gupta, A. Revisiting unreasonable effectiveness of data in deep learning era. In *Proceedings of the IEEE international conference on computer vision*, pp. 843–852, 2017.

Tibshirani, R. J. Dykstra's algorithm, admm, and coordinate descent: Connections, insights, and extensions. *Advances in Neural Information Processing Systems*, 30, 2017.

Warmuth, M. K. and Kuzmin, D. Randomized online pca algorithms with regret bounds that are logarithmic in the dimension. *Journal of Machine Learning Research*, 9 (Oct):2287–2320, 2008.

Wiseman, S. and Rush, A. M. Sequence-to-sequence learning as beam-search optimization. *arXiv preprint arXiv:1606.02960*, 2016.

Xie, Y., Dai, H., Chen, M., Dai, B., Zhao, T., Zha, H., Wei, W., and Pfister, T. Differentiable top-k with optimal transport. *Advances in Neural Information Processing Systems*, 33:20520–20531, 2020.

Zeng, X. and Figueiredo, M. A. The ordered weighted l_1 norm: Atomic formulation, projections, and algorithms. *arXiv preprint arXiv:1409.4271*, 2014.

Zhou, Y., Lei, T., Liu, H., Du, N., Huang, Y., Zhao, V., Dai, A., Chen, Z., Le, Q., and Laudon, J. Mixture-of-experts with expert choice routing. *arXiv preprint arXiv:2202.09368*, 2022.

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

A. Proofs

A.1. Linear Maximization Oracle - Proof of Proposition 1

Assuming $\boldsymbol{w}$ is sorted in descending order, we have for any $\boldsymbol{x} \in \mathbb{R}^n$

$$
\sum_{i=1}^{n} w_i x_{[i]} = \max_{\sigma \in \Sigma} \langle \boldsymbol{x}_\sigma, \boldsymbol{w} \rangle
$$
$$
= \max_{\pi \in \Sigma} \langle \boldsymbol{x}, \boldsymbol{w}_\pi \rangle
$$
$$
= \max_{\boldsymbol{y} \in P(\boldsymbol{w})} \langle \boldsymbol{x}, \boldsymbol{y} \rangle.
$$

In the first line, we used that the inner product is maximized by finding the permutation σ sorting $\boldsymbol{x}$ in descending order. In the second line, we used that $\langle \boldsymbol{x}_\sigma, \boldsymbol{w} \rangle = \langle \boldsymbol{x}, \boldsymbol{w}_\pi \rangle$, if π is the inverse permutation of σ. In the third line, we used the fundamental theorem of linear programming, which guarantees that the solution happens at one of the vertices of the polytope. To summarize, if σ is the permutation sorting $\boldsymbol{x}$ in descending order, then $\boldsymbol{y}^\star = \boldsymbol{w}_{\sigma^{-1}}$.

A.2. Relaxed operator - Proof of Proposition 2

Recall that $f_{\varphi,R}^*(\boldsymbol{y}, \boldsymbol{w}) := f_\varphi^*(\boldsymbol{y}, \boldsymbol{w}) + R(\boldsymbol{y})$. We then have

$$
f_{\varphi,R}(\boldsymbol{x}, \boldsymbol{w}) = \max_{\boldsymbol{y} \in \mathbb{R}^n} \langle \boldsymbol{y}, \boldsymbol{x} \rangle - f_\varphi^*(\boldsymbol{y}, \boldsymbol{w}) - R(\boldsymbol{y}).
$$

It is well-known that if h_1 and h_2 are two convex functions, then $(h_1 + h_2)^*$ is equal to the infimal convolution of h_1^* with h_2^* (Beck, 2017, Theorem 4.17):

$$
(h_1 + h_2)^*(\boldsymbol{x}) = (h_1^* \square h_2^*)(\boldsymbol{x}) := \min_{\boldsymbol{u} \in \mathbb{R}^n} h_1^*(\boldsymbol{u}) + h_2^*(\boldsymbol{x} - \boldsymbol{u}).
$$

With $h_1 = f_\varphi^*$ and $h_2 = R$, we therefore get

$$
f_{\varphi,R}(\boldsymbol{x}, \boldsymbol{w}) = \min_{\boldsymbol{u} \in \mathbb{R}^n} R^*(\boldsymbol{x} - \boldsymbol{u}) + f_\varphi(\boldsymbol{u}, \boldsymbol{w}).
$$

Finally, the expression of $\boldsymbol{y}^\star$ follows from Danskin's theorem applied.

A.3. Conjugate - Proof of Proposition 3

We have

$$
f_\varphi^*(\boldsymbol{y}, \boldsymbol{w}) = \max_{\boldsymbol{x} \in \mathbb{R}^n} \langle \boldsymbol{x}, \boldsymbol{y} \rangle - f_\varphi(\boldsymbol{x}, \boldsymbol{w})
$$
$$
= \max_{\boldsymbol{x} \in \mathbb{R}^n} \langle \boldsymbol{x}, \boldsymbol{y} \rangle - \max_{\boldsymbol{y}' \in P(\boldsymbol{w})} \langle \varphi(\boldsymbol{x}), \boldsymbol{y}' \rangle
$$
$$
= \max_{\boldsymbol{x} \in \mathbb{R}^n} \min_{\boldsymbol{y}' \in P(\boldsymbol{w})} \langle \boldsymbol{x}, \boldsymbol{y} \rangle - \langle \varphi(\boldsymbol{x}), \boldsymbol{y}' \rangle.
$$

If $\boldsymbol{w} \in \mathbb{R}_+^n$, then $\boldsymbol{y}' \in \mathbb{R}_+^n$ for all $\boldsymbol{y}' \in P(\boldsymbol{w})$. Then the function $(\boldsymbol{x}, \boldsymbol{y}') \mapsto \langle \boldsymbol{x}, \boldsymbol{y} \rangle - \langle \varphi(\boldsymbol{x}), \boldsymbol{y}' \rangle$ is concave-convex and we can switch the min and the max to obtain

$$
f_\varphi^*(\boldsymbol{y}, \boldsymbol{w}) = \min_{\boldsymbol{y}' \in P(\boldsymbol{w})} \max_{\boldsymbol{x} \in \mathbb{R}^n} \langle \boldsymbol{x}, \boldsymbol{y} \rangle - \langle \varphi(\boldsymbol{x}), \boldsymbol{y}' \rangle
$$
$$
= \min_{\boldsymbol{y}' \in P(\boldsymbol{w})} \sum_{i=1}^{n} y_i' \phi_i^*(y_i / y_i').
$$

A.4. Biconjugate interpretation - Proof of Proposition 4

One has

$$
\Phi_k^*(\boldsymbol{x}) = \max_{\boldsymbol{y} \in S_k} \langle \boldsymbol{x}, \boldsymbol{y} \rangle - \sum_{i=1}^{n} \phi^*(y_i).
$$

As in (Kyrillidis et al., 2013), let Σ_k be the set of subsets of $[n]$ with cardinality smaller than k. Then

$$\Phi_k^*(\boldsymbol{x}) = \max_{I \subset \Sigma_k} \max_{\boldsymbol{y} \in \mathbb{R}^n} \sum_{i \in I} x_i y_i - \phi^*(y_i).$$

This gives

$$\Phi_k^*(\boldsymbol{x}) = \max_{I \subset \Sigma_k} \sum_{i \in I} \phi(x_i) = \sum_{i=1}^k \varphi(\boldsymbol{x})_{[i]}.$$

Taking the conjugate gives the desired result.

A.5. Reduction to isotonic optimization

We focus on the case when φ is sign-invariant, i.e., $\varphi(\boldsymbol{x}) = \varphi(-\boldsymbol{x})$, since the case $\varphi(\boldsymbol{x}) = \boldsymbol{x}$ is already tackled in (Lim & Wright, 2016; Blondel et al., 2020b).

We first show that $\boldsymbol{u}^\star$ preserves the sign of $\boldsymbol{x}$. We do so by showing that for any $\boldsymbol{u} \in \mathbb{R}^n$, $\boldsymbol{u}' := \text{sign}(\boldsymbol{x}) \circ |\boldsymbol{u}|$ achieves smaller objective value than $\boldsymbol{u}$. Recall that

$$f_{\varphi,R}(\boldsymbol{x}, \boldsymbol{w}) = \min_{\boldsymbol{u} \in \mathbb{R}^n} R^*(\boldsymbol{x} - \boldsymbol{u}) + f_\varphi(\boldsymbol{u}, \boldsymbol{w}) = \min_{\boldsymbol{u} \in \mathbb{R}^n} R^*(\boldsymbol{x} - \boldsymbol{u}) + f(\varphi(\boldsymbol{u}), \boldsymbol{w}).$$

Clearly, we have $f(\varphi(\boldsymbol{u}'), \boldsymbol{w}) = f(\varphi(\boldsymbol{u}), \boldsymbol{w})$. Moreover, if $R(\boldsymbol{y}) = \sum_{i=1}^n r(y_i)$, then $R^*(\boldsymbol{x} - \boldsymbol{u}) = \sum_{i=1}^n r^*(x_i - u_i)$. If r^* is sign-invariant and increasing on $\mathbb{R}_+$, we then have

$$\begin{aligned}
r^*(x_i - u_i') &= r^*(\text{sign}(x_i)(|x_i| - |u_i|)) \\
&= r^*(|x_i| - |u_i|) \\
&\leq r^*(x_i - u_i),
\end{aligned}$$

where we used the reverse triangle inequality $||x_i| - |u_i|| \leq |x_i - u_i|$. We conclude that $\boldsymbol{u}^\star$ has the same sign as $\boldsymbol{x}$. From now on, we can therefore assume that $\boldsymbol{x} \in \mathbb{R}_+^n$, which implies that $\boldsymbol{u} \in \mathbb{R}_+^n$.

Since $\boldsymbol{u} \in \mathbb{R}_+^n$ and ϕ is increasing on $\mathbb{R}_+$, we have $u_{\sigma_1} \geq \cdots \geq u_{\sigma_n} \Rightarrow \phi(u_{\sigma_1}) \geq \cdots \geq \phi(u_{\sigma_n})$. We know that for all $\boldsymbol{u} \in \mathbb{R}^n$ and $\boldsymbol{w} \in \mathbb{R}^n$, we have $f(\varphi(u), \boldsymbol{w}) = \langle \varphi(\boldsymbol{u})_\sigma, \boldsymbol{w} \rangle = \langle \varphi(\boldsymbol{u}_\sigma), \boldsymbol{w} \rangle$, where σ is the permutation sorting $\boldsymbol{u}$ in descending order. From now on, let us fix σ to the permutation sorting $\boldsymbol{u}^\star$. We will show in the sequel that this is the same permutation as the one sorting $\boldsymbol{x}$. We then have

$$f_{\varphi,R}(\boldsymbol{x}, \boldsymbol{w}) = \min_{\boldsymbol{u} \in \mathbb{R}^n} R^*(\boldsymbol{x} - \boldsymbol{u}) + \langle \varphi(\boldsymbol{u}_\sigma), \boldsymbol{w} \rangle.$$

Using the change of variable $\boldsymbol{v} = \boldsymbol{u}_\sigma \Leftrightarrow \boldsymbol{v}_{\sigma^{-1}} = \boldsymbol{u}$, we obtain

$$f_{\varphi,R}(\boldsymbol{x}, \boldsymbol{w}) = \max_{v_1 \geq \cdots \geq v_n} R^*(\boldsymbol{x}_\sigma - \boldsymbol{v}) + \langle \varphi(\boldsymbol{v}), \boldsymbol{w} \rangle$$

where we used that if $R(\boldsymbol{y}) = \sum_{i=1}^n r(y_i)$, then

$$R^*(\boldsymbol{x} - \boldsymbol{u}) = R^*(\boldsymbol{x} - \boldsymbol{v}_{\sigma^{-1}}) = R^*(\boldsymbol{x}_\sigma - \boldsymbol{v}).$$

Let $\boldsymbol{s} := \boldsymbol{x}_\sigma$. It remains to show that $s_1 \geq \cdots \geq s_n$, i.e., that $\boldsymbol{s}$ and $\boldsymbol{v}^\star$ are both in descending order. Suppose $s_j > s_i$ for some $i < j$. Let $\boldsymbol{s}'$ be a copy of $\boldsymbol{s}$ with s_i and s_j swapped. Since R^* is convex, by (Blondel et al., 2020b, Lemma 4),

$$R^*(\boldsymbol{s} - \boldsymbol{v}^\star) - R^*(\boldsymbol{s}' - \boldsymbol{v}^\star) = r^*(s_i - v_i^\star) + r^*(s_j - v_j^\star) - r^*(s_j - v_i^\star) - r^*(s_i - v_j^\star) \geq 0,$$

which contradicts the assumption that $\boldsymbol{v}^\star$ and the corresponding σ are optimal.

A.6. Subproblem derivation

Case $\phi(x) = \frac{1}{2}x^2$ and $R(\boldsymbol{x}) = \frac{\lambda}{2}\|\boldsymbol{x}\|^2$. One has $h_i(\gamma) = \frac{1}{2}(w_i\gamma^2 + \frac{1}{\lambda}(s_i - \gamma)^2)$ so that $\frac{dh_i}{d\gamma} = (w_i + \frac{1}{\lambda})\gamma - \frac{1}{\lambda}s_i$. Therefore,

$$\frac{d\sum_{i \in B} h_i(\gamma)}{d\gamma} = \gamma \sum_{i \in B}(w_i + \frac{1}{\lambda}) - \frac{1}{\lambda}\sum_{i \in B} s_i.$$

Since in addition $\sum_{i \in B} h_i$ is convex we obtain that its minimum is given by canceling the derivative, hence

$$\gamma_B^\star = \frac{\sum_{i \in B} s_i}{\sum_{i \in B} (\lambda w_i + 1)}.$$

Remark. This result shows that when λ is small enough, one can control the approximation error induced by our proposed operator in comparison to the hard operator. For simplicity, let us focus on the case where there are no ties: $\forall i \neq j$, $x_i \neq x_j$. This implies that $s_1 > s_2 > \cdots > s_n$. In this case, for λ small enough, we get

$$\gamma_{\{1\}}^\star > \gamma_{\{2\}}^\star > \cdots > \gamma_{\{n\}}^\star$$

so that the optimal partition in PAV's algorithm is given by taking $B_i = \{i\}$. Therefore, $v_i^\star = \frac{s_i}{\lambda w_i + 1}$. One then has

$$\boldsymbol{u}^\star = \mathrm{sign}(\boldsymbol{x}) \circ \boldsymbol{v}_{\sigma^{-1}}^\star = \frac{\boldsymbol{x}}{\lambda \boldsymbol{w}_{\sigma^{-1}} + 1}.$$

Plugging it into $\boldsymbol{y}^\star$ given by Proposition 2 gives

$$\boldsymbol{y}^\star = \frac{\boldsymbol{w}_{\sigma^{-1}} \circ \boldsymbol{x}}{\lambda \boldsymbol{w}_{\sigma^{-1}} + 1}.$$

Since the hard operator is given by $\boldsymbol{w}_{\sigma^{-1}} \circ \boldsymbol{x}$, the approximation error $e_\lambda(\boldsymbol{x}, \boldsymbol{w})$ in infinite norm is then simply bounded by

$$e_\lambda(\boldsymbol{x}, \boldsymbol{w}) := \|\boldsymbol{w}_{\sigma^{-1}} \circ \boldsymbol{x}(\frac{1}{\lambda \boldsymbol{w}_{\sigma^{-1}} + 1} - 1)\|_\infty \leq \|\boldsymbol{w}\|_\infty \|\boldsymbol{x}\|_\infty \|\frac{\lambda \boldsymbol{w}}{\lambda \boldsymbol{w} + 1}\|_\infty.$$

In the $\mathrm{topkmag}$ case, $w = \mathbf{1}_k$, so that $e_\lambda(\boldsymbol{x}, \boldsymbol{w}) \leq \lambda \|\boldsymbol{x}\|_\infty$.

Case $\phi(x) = x$ and $R(\boldsymbol{x}) = \frac{\lambda}{p}\|\boldsymbol{x}\|^p$ with $p = \frac{4}{3}$. One has $h_i(\gamma) = w_i \gamma + \frac{1}{4\lambda^3}(s_i - \gamma)^4$ so that $\frac{dh_i}{d\gamma} = w_i + \frac{1}{\lambda^3}(\gamma - s_i)^3$. Therefore,

$$\frac{d\sum_{i \in B} h_i(\gamma)}{d\gamma} = \frac{1}{\lambda^3} \sum_{i \in B} (\gamma - s_i)^3 + \sum_{i \in B} w_i.$$

Since in addition $\sum_{i \in B} h_i$ is convex we obtain that its minimum is given by canceling the derivative, and hence by solving the third-order polynomial equation

$$\frac{1}{\lambda^3} \sum_{i \in B} (\gamma - s_i)^3 + \sum_{i \in B} w_i = 0.$$

In practice, we solve this equation using the root solver from the *numpy* library. Note that taking $p = \frac{4}{3}$ leads to an easier subproblem than $p = \frac{3}{2}$, hence our choice for p.

Case $\phi(x) = \frac{1}{2}x^2$ and $R(\boldsymbol{x}) = \frac{\lambda}{p}\|\boldsymbol{x}\|^p$ with $p = \frac{4}{3}$. The derivation is very similar to the previous case. Indeed, one has $h_i(\gamma) = \frac{1}{2}w_i\gamma^2 + \frac{1}{4\lambda^3}(s_i - \gamma)^4$ so that $\frac{dh_i}{d\gamma} = w_i\gamma + \frac{1}{\lambda^3}(\gamma - s_i)^3$. Therefore,

$$\frac{d\sum_{i \in B} h_i(\gamma)}{d\gamma} = \frac{1}{\lambda^3} \sum_{i \in B} (\gamma - s_i)^3 + \gamma \sum_{i \in B} w_i.$$

A.7. Differentiation - Proof of Proposition 7

Case $\phi(x) = x$. One has $h_i(\gamma) = \frac{1}{q}|s_i - \gamma|^q + w_i v_i$. For any optimal B in PAV one has the optimality condition

$$\sum_{i \in B} (\mathrm{sign}(\gamma_B^\star - s_i)|\gamma_B^\star - s_i|^{q-1} + w_i) = 0.$$

Using the implicit function theorem gives that $\gamma_B^\star$ is differentiable, and differentiating with respect to any s_i for $i \in B$ leads to

$$\frac{\partial \gamma_B^\star}{\partial s_i} = \frac{|\gamma_B^\star - s_i|^{q-2}}{\sum_{j \in B} |\gamma_B^\star - s_j|^{q-2}}.$$

Case $\phi(x) = \frac{1}{2}x^2$. Similar calculations lead to

$$\frac{\partial \gamma_B^\star}{\partial s_i} = \frac{(q-1)|\gamma_B^\star - s_i|^{q-2}}{\sum_{j \in B}(q-1)|\gamma_B^\star - s_j|^{q-2} + w_j}.$$

A.8. Dual of isotonic optimization

$$\min_{v_1 \geq \cdots \geq v_n} \sum_{i=1}^n h_i(v_i) = \min_{\boldsymbol{v} \in \mathbb{R}^n} \max_{\boldsymbol{\alpha} \in \mathbb{R}_+^{n-1}} \sum_{i=1}^n h_i(v_i) - \alpha_i(v_i - v_{i+1})$$

$$= \min_{\boldsymbol{v} \in \mathbb{R}^n} \max_{\boldsymbol{\alpha} \in \mathbb{R}_+^{n-1}} \sum_{i=1}^n h_i(v_i) - v_i(\alpha_i - \alpha_{i-1})$$

$$= \max_{\boldsymbol{\alpha} \in \mathbb{R}_+^{n-1}} - \left[\sum_{i=1}^n h_i^*(\alpha_i - \alpha_{i-1}) \right]$$

where $\alpha_0 \coloneqq 0$ and $\alpha_n \coloneqq 0$ are constants (i.e., not optimized). An optimal solution $\boldsymbol{v}^\star$ is recovered from $\boldsymbol{\alpha}^\star$ by $v_i^\star = (h_i^*)'(\alpha_i^\star - \alpha_{i-1}^\star)$. Since $\boldsymbol{\alpha}$ is only constrained to be non-negative, we can solve the dual by coordinate ascent. The subproblem associated with α_i, for $i \in \{1, \dots, n-1\}$, is

$$\max_{\alpha_i \in \mathbb{R}_+} -h_i^*(\alpha_i - \alpha_{i-1}) - h_{i+1}^*(\alpha_{i+1} - \alpha_i).$$

The subproblem is a simple univariate problem with non-negative constraint. Let us define α_i' as the solution of $(h_i^*)'(\alpha_i' - \alpha_{i-1}) - (h_{i+1}^*)'(\alpha_{i+1} - \alpha_i') = 0$. The solution is then $\alpha_i^\star = [\alpha_i']_+$.

In practice, we can alternate between updating $\alpha_1, \alpha_3, \dots$ in parallel and $\alpha_2, \alpha_4, \dots$ in parallel. The dual variables with odd coordinates correspond to the set $C_1 = \{\boldsymbol{v} \in \mathbb{R}^n : v_1 \geq v_2, v_3 \geq v_4, \dots\}$ and the dual variables with even coordinates correspond to the set $C_2 = \{\boldsymbol{v} \in \mathbb{R}^n : v_2 \geq v_3, v_4 \geq v_5, \dots\}$. Coordinate ascent converges to an optimal dual solution, assuming each h_i^* is differentiable, which is equivalent to each h_i being strictly convex.

Note that the subproblem can be rewritten in primal space as

$$\max_{\alpha_i \in \mathbb{R}_+} -h_i^*(\alpha_i - \alpha_{i-1}) - h_{i+1}^*(\alpha_{i+1} - \alpha_i)$$

$$= \max_{\alpha_i \in \mathbb{R}_+} - \left[\max_{v_i}(\alpha_i - \alpha_{i-1})v_i - h_i(v_i) \right] - \left[\max_{v_{i+1}}(\alpha_{i+1} - \alpha_i)v_{i+1} - h_{i+1}(v_{i+1}) \right]$$

$$= \min_{v_i, v_{i+1}} \alpha_{i-1}v_i + h_i(v_i) - \alpha_{i+1}v_{i+1} + h_{i+1}(v_{i+1}) + \max_{\alpha_i \in \mathbb{R}_+} \alpha_i(v_{i+1} - v_i)$$

$$= \min_{v_i \geq v_{i+1}} h_i(v_i) + h_{i+1}(v_{i+1}) + \alpha_{i-1}v_i - \alpha_{i+1}v_{i+1}.$$

In fact, in the Euclidean case, it is known that Dykstra's algorithm in the primal and block coordinate ascent in the dual are equivalent (Tibshirani, 2017). Therefore, block coordinate ascent can be seen as an elegant way to generalize Dykstra's algorithm to the non-Euclidean case.

B. Additional material

B.1. k-support negentropies

When $\phi(x) = e^{x-1}$ and $\boldsymbol{w} = \mathbf{1}_k$, we obtain

$$f_\varphi^*(\boldsymbol{y}, \boldsymbol{w}) = \min_{\boldsymbol{z} \in [0,1]^n} \sum_{i=1}^n y_i \log(\frac{y_i}{z_i}) \quad \text{s.t.} \quad \langle \boldsymbol{z}, \mathbf{1} \rangle = k.$$
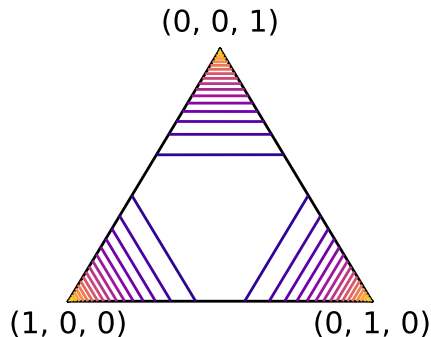
We call it a k-support negative entropy.

Figure 9. **Contours of the k-support entropy** on the simplex for $n = 3$ and $k = 2$. Lighter colors indicate lower values.

B.2. PAV algorithm

We present the pseudo code for PAV, adapted from Lim & Wright (2016). Recall that we define

$$\gamma_B^\star = \underset{\gamma \in \mathbb{R}}{\operatorname{argmin}} \sum_{i \in B} h_i(\gamma).$$

Algorithm 1 Pool Adjacent Violators (PAV)

Input: Convex functions $\{h_i : \mathbb{R} \to \mathbb{R}\}_{i \in [n]}$
Initalize partitions $P \leftarrow \{\{i\}|i \in [n]\}$
Initialize $v_i \leftarrow \gamma_{\{i\}}^\star$ for all $i \in [n]$
while there exists i such that $v_i < v_{i+1}$ **do**
    Find B_{r_i} and $B_{r_{i+1}}$ in P such that $i \in B_{r_i}$ and $i + 1 \in B_{r_{i+1}}$
    Remove B_{r_i} and $B_{r_{i+1}}$ from P
    Add $B_{r_i} \cup B_{r_{i+1}}$ to P
    Compute $\gamma_{B_{r_i} \cup B_{r_{i+1}}}^\star$ by solving (6)
    Assign $v_r \leftarrow \gamma_{B_{r_i} \cup B_{r_{i+1}}}^\star$ for all $r \in B_{r_i} \cup B_{r_{i+1}}$
end while
return v

C. Experimental details

C.1. Weight pruning in neural networks

For our experiment on the MNIST dataset, we train the MLP using SGD with a batch size of 128 and a constant learning rate of 10^{-2}. We trained the model for 30 epochs. In terms of hardware, we use a single GPU.

C.2. Smooth top-k loss

For our experiment on the CIFAR-100 dataset, we train the ViT-B/16 using SGD with a momentum of 0.9 and with a batch size of 512.

For the cross-entropy loss, we follow the training procedure of Dosovitskiy et al. (2020): warmup phase until the learning rate reaches 3×10^{-3}. The model is trained for 100 steps using a cosine learning rate scheduler. This choice of learning rate gave the best performance for this number of training steps.

For our top-3 losses: warmup phase until the learning rate reaches 5×10^{-3}. The model is trained for 100 steps using a cosine learning rate scheduler.

In terms of hardware, we use 8 TPUs.

C.3. Sparse MoEs

We train the V-MoE S/32 model (Riquelme et al., 2021) on the JFT-300M dataset (Sun et al., 2017). JFT is a multilabel dataset, and thus accuracy is not an appropriate metric since each image may have multiple labels. Therefore, we measure the quality of the models using the commonly-used *precision-at-1* metric (Järvelin & Kekäläinen, 2017). The training procedure is analogous to the one described in Riquelme et al. (2021), except that we replace the routing algorithm. In particular, we use the Expert Choice Routing algorithm described in Zhou et al. (2022) as our baseline, and replace the non-differentiable top-k operation used there with our differentiable approach (we perform 10 iterations of Dykstra's algorithm).

We use exactly the same hyperparameters as described in Riquelme et al. (2021), except for the fact that Expert Choice Routing does not require any auxiliary loss. Specifically, we train for 7 epochs using a batch size of 4 096. We use the Adam optimizer ($\beta_1 = 0.9$, $\beta_2 = 0.999$), with a peak learning rate of 10^{-3}, warmed up for 10 000 steps and followed by linear decay. We use mild data augmentations (random cropping and horizontal flipping) and weight decay of 10^{-1} in all parameters as means of regularization. We trained both models on TPUv2-128 devices.