Differentiable Top-k Operator with Optimal Transport

Yujia Xie, Hanjun Dai, Minshuo Chen, Bo Dai, Tuo Zhao, Hongyuan Zha, Wei Wei, Tomas Pfister *

Abstract

The top-k operation, i.e., finding the k largest or smallest elements from a collection of scores, is an important model component, which is widely used in information retrieval, machine learning, and data mining. However, if the top-k operation is implemented in an algorithmic way, e.g., using bubble algorithm, the resulting model cannot be trained in an end-to-end way using prevalent gradient descent algorithms. This is because these implementations typically involve swapping indices, whose gradient cannot be computed. Moreover, the corresponding mapping from the input scores to the indicator vector of whether this element belongs to the top-k set is essentially discontinuous. To address the issue, we propose a smoothed approximation, namely the SOFT (Scalable Optimal transport-based diFferenTiable) top-k operator. Specifically, our SOFT top-k operator approximates the output of the top-k operation as the solution of an Entropic Optimal Transport (EOT) problem. The gradient of the SOFT operator can then be efficiently approximated based on the optimality conditions of EOT problem. We apply the proposed operator to the k-nearest neighbors and beam search algorithms, and demonstrate improved performance.

1 Introduction

The top-k operation, i.e., finding the k largest or smallest elements from a set, is widely used for predictive modeling in information retrieval, machine learning, and data mining. For example, in image retrieval (Babenko et al., 2014; Radenović et al., 2016; Gordo et al., 2016), one needs to query the k nearest neighbors of an input image under certain metrics; in the beam search (Reddy et al., 1977; Wiseman & Rush, 2016) algorithm for neural machine translation, one needs to find the k sequences of largest likelihoods in each decoding step.

Although the ubiquity of top-k operation continues to grow, the operation itself is difficult to be integrated into the training procedure of a predictive model. For example, we consider a neural network-based k-nearest neighbor classifier. Given an input, we use the neural network to extract features from the input. Next, the extracted features are fed into the top-k operation for identifying

*Work done when Yujia was an intern in Google. Yujia Xie, Minshuo Chen, Tuo Zhao are affiliated with Georgia Institute of Technology. Hanjun Dai, Bo Dai, Wei Wei, Tomas Pfister are affiliated with Google Inc. Yujia Xie and Wei Wei are the corresponding authors. Emails: `Xie.Yujia000@gmail.com, wewei@google.com`.

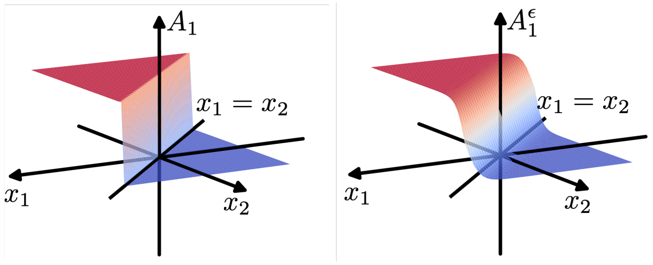


Figure 1: Indicator vector with respect to input scores. Left: original top-k operator; right: SOFT top-k operator.

the k nearest neighbors under some distance metric. We then obtain a prediction based on the k nearest neighbors of the input. In order to train such a model, we choose a proper loss function, and minimize the average loss across training samples using (stochastic) first-order methods. This naturally requires the loss function being differentiable with respect to the input at each update step. Nonetheless, the top-k operation does not exhibit an explicit mathematical formulation: most implementations of the top-k operation, e.g., bubble algorithm and QUICKSELECT (Hoare, 1961), involve operations on indices such as indices swapping. Consequently, the training objective is difficult to formulate explicitly.

Alternative perspective — taking the top-k operation as an operator — still cannot resolve the differentibility issue. Specifically, the top-k operator[1] maps a set of inputs $x_1, \ldots, x_n$ to an index vector $\{0,1\}^n$. Whereas the Jacobian matrix of such a mapping is not well defined. As a simple example, consider two scalars x_1, x_2. The top-1 operation as in Figure 1 returns a vector $[A_1, A_2]^\top$, with each entry denoting whether the scalar is the larger one (1 for true, 0 for false). Denote $A_1 = f(x_1, x_2)$. For a fixed x_2, A_1 jumps from 0 to 1 at $x_1 = x_2$. It is clear that f is not differentiable at $x_1 = x_2$, and the derivative is identically zero otherwise.

Due to the aforementioned difficulty, existing works resort to two-stage training for models with the top-k operation. We consider the neural network-based k-nearest neighbor classifier again. As proposed in Papernot & McDaniel (2018), one first trains the neural network using some surrogate loss on the extracted features, e.g., using softmax activation in the output layer and the cross-entropy loss. Next, one uses the k-nearest neighbor for prediction based on the features extracted by the well-trained neural network. This training procedure, although circumventing the top-k operation, makes the training and prediction misaligned; and the actual performance suffers.

In this work, we propose the SOFT (Scalable Optimal transport-based diFferenTiable) top-k operation as a differentiable approximation of the standard top-k operation in Section. 2. Specifically, motivated by the implicit differentiation (Duchi et al., 2008; Griewank & Walther, 2008; Amos & Kolter, 2017; Luise et al., 2018) techniques, we first parameterize the top-k operation in terms of the optimal solution of an Optimal Transport (OT) problem. Such a re-parameterization is still not differentiable with respect to the input. To rule out the discontinuity, we impose entropy regularization to the optimal transport problem, and show that the optimal solution to the Entropic OT (EOT) problem yields a differentiable approximation to the top-k operation. Moreover, we prove that under mild assumptions, the approximation error can be properly controlled.

We then develop an efficient implementation of the SOFT top-k operation in Section. 3. Specif-

[1]Throughout the rest of the paper, we refer to the top-k operator as the top-k operation.

ically, we solve the EOT problem via the Sinkhorn algorithm (Cuturi, 2013). Given the optimal solution, we can explicitly formulate the gradient of SOFT top-k operation using the KKT (Karush-Kuhn-Tucker) condition. As a result, the gradient at each update step can be efficiently computed with complexity $\mathcal{O}(n)$, where n is the number of elements in the input set to the top-k operation.

Our proposed SOFT top-k operation allows end-to-end training, and we apply SOFT top-k operation to kNN for classification in Section 4, beam search in Section 5 and learning sparse attention for neural machine translation in Section 6. The experimental results demonstrate significant performance gain over competing methods, as an end-to-end training procedure resolves the misalignment between training and prediction.

Notations. We denote $\|\cdot\|_2$ as the ℓ_2 norm of vectors, $\|\cdot\|_{\mathrm{F}}$ as the Frobenius norm of matrices. Given two matrices $B, D \in \mathbb{R}^{n\times m}$, we denote $\langle B, D\rangle$ as the inner product, i.e., $\langle B, D\rangle = \sum_{i=1,j=1}^{n,m} B_{ij}D_{ij}$. We denote $B \odot D$ as the element-wise multiplication of B and D. We denote $\mathbb{1}(\cdot)$ as the indicator function, i.e., the output of $\mathbb{1}(\cdot)$ is 1 if the input condition is satisfied, and is 0 otherwise. For matrix $B \in \mathbb{R}^{n\times m}$, we denote $B_{i,:}$ as the i-th row of the matrix. The softmax function for matrix B is defined as $\mathrm{softmax}_i(B_{ij}) = e^{B_{ij}}/\sum_{\ell=1}^{n} e^{B_{\ell j}}$. For a vector $b \in \mathbb{R}^n$, we denote $\mathrm{diag}(b)$ as the matrix where the i-th diagonal entries is b_i.

2 SOFT Top-k Operator

In this section we derive the proposed SOFT (Scalable Optimal transport-based diFferenTialble) top-k operator.

2.1 Problem Statement

Given a set of scalars $\mathcal{X} = \{x_i\}_{i=1}^n$, the standard top-$k$ operator returns a vector $A = [A_1, \ldots, A_n]^\top$, such that

$$A_i = \begin{cases} 1, & \text{if } x_i \text{ is a top-}k \text{ element in } \mathcal{X}, \\ 0, & \text{otherwise.} \end{cases}$$

In this work, our goal is to design a smooth relaxation of the standard top-k operator, whose Jacobian matrix exists and is smooth. Without loss of generality, we refer to top-k elements as the *smallest* k elements.

2.2 Parameterizing Top-k Operator as OT Problem

We first show that the standard top-k operator can be parameterized in terms of the solution of an Optimal Transport (OT) problem (Monge, 1781; Kantorovich, 1960). We briefly introduce OT problems for self-containedness. An OT problem finds a transport plan between two distributions, while the expected cost of the transportation is minimized. We consider two discrete distributions defined on supports $\mathcal{A} = \{a_i\}_{i=1}^n$ and $\mathcal{B} = \{b_j\}_{j=1}^m$, respectively. Denote $\mathbb{P}(\{a_i\}) = \mu_i$ and $\mathbb{P}(\{b_j\}) = \nu_j$, and let $\mu = [\mu_1, \ldots, \mu_n]^\top$ and $\nu = [\nu_1, \ldots, \nu_m]^\top$. We further denote $C \in \mathbb{R}^{n\times m}$ as the cost matrix with

C_{ij} being the cost of transporting mass from a_i to b_j. An OT problem can be formulated as

$$\Gamma^* = \underset{\Gamma \geq 0}{\operatorname{argmin}} \langle C, \Gamma \rangle, \quad \text{s.t.,} \;\; \Gamma \mathbf{1}_m = \mu, \; \Gamma^\top \mathbf{1}_n = \nu, \tag{1}$$

where $\mathbf{1}$ denotes a vector of ones. The optimal solution Γ^* is referred to as the *optimal transport plan*.

In order to parameterize the top-k operator using the optimal transport plan Γ^*, we set the support $\mathcal{A} = \mathcal{X}$ and $\mathcal{B} = \{0, 1\}$ in (1), with μ, ν defined as

$$\mu = \mathbf{1}_n / n, \quad \nu = [k/n, (n-k)/n]^\top.$$

We take the cost to be the squared Euclidean distance, i.e., $C_{i1} = x_i^2$ and $C_{i2} = (x_i - 1)^2$ for $i = 1, \ldots, n$. We then establish the relationship between the output A of the top-k operator and Γ^*.

Proposition 1. Consider the setup in the previous paragraph. Without loss of generality, we assume $\mathcal{X}$ has no duplicates. Then the optimal transport plan Γ^* of (1) is

$$\Gamma^*_{\sigma_i,1} = \begin{cases} 1/n, & \text{if } i \leq k, \\ 0, & \text{if } k+1 \leq i \leq n. \end{cases} \tag{2}$$

$$\Gamma^*_{\sigma_i,2} = \begin{cases} 0, & \text{if } i \leq k, \\ 1/n, & \text{if } k+1 \leq i \leq n, \end{cases} \tag{3}$$

with σ being the sorting permutation, i.e., $x_{\sigma_1} < x_{\sigma_2} < \cdots < x_{\sigma_n}$. Moreover, we have

$$A = n\Gamma^* \cdot [1, 0]^\top. \tag{4}$$

Proof. We expand the objective function of (1) as

$$\begin{aligned} \langle C, \Gamma \rangle &= \sum_{i=1}^n \Big((x_i - 0)^2 \Gamma_{i,1} + (x_i - 1)^2 \Gamma_{i,2} \Big) \\ &= \sum_{i=1}^n \Big(x_i^2 (\Gamma_{i,1} + \Gamma_{i,2}) + \Gamma_{i,2} - 2x_i \Gamma_{i,2} \Big) \\ &= \frac{1}{n} \sum_{i=1}^n x_i^2 + \frac{n-k}{n} - 2 \sum_{i=1}^n x_i \Gamma_{i,2}. \end{aligned}$$

Therefore, to minimize $\langle C, \Gamma \rangle$, it suffices to maximize $\sum_{i=1}^n x_i \Gamma_{i,2}$. It is straightforward to check

$$\sum_{i=1}^n \Gamma_{i,2} = \frac{n-k}{n} \quad \text{and} \quad \Gamma_{i,2} \leq \frac{1}{n}$$

for any $i = 1, \ldots, n$. Hence, maximizing $\sum_{i=1}^n x_i \Gamma_{i,2}$ is essentially selecting the largest $n - K$ elements from $\mathcal{X}$, and the maximum is attained at

$$\Gamma^*_{\sigma_i,2} = \begin{cases} 0, & \text{if } i \leq k, \\ 1/n, & \text{if } k+1 \leq i \leq n. \end{cases}$$

The constraint $\Gamma \mathbf{1}_m = \mu$ further implies that $\Gamma^*_{i,1}$ satisfies (2). Thus, A can be parameterized as $A = n\Gamma^* \cdot [1, 0]^\top$. $\square$

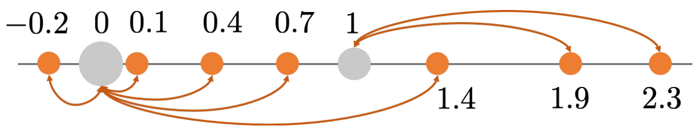
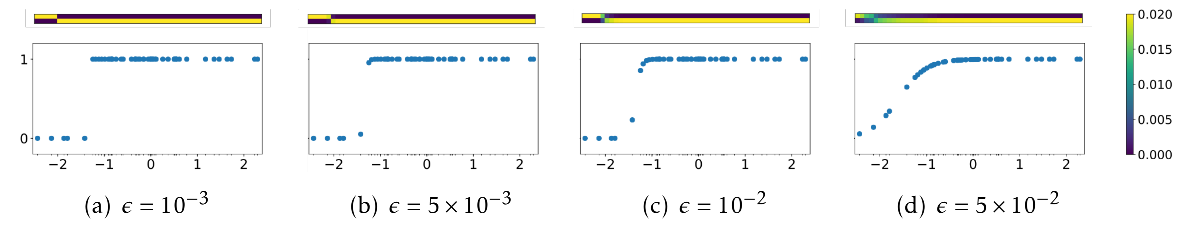


Figure 2: Illustration of the optimal transport plan with input $\mathcal{X} = [0.4, 0.7, 2.3, 1.9, -0.2, 1.4, 0.1]^\top$ and $k = 5$. Here, we set $\nu = [\frac{5}{7}, \frac{2}{7}]^\top$. In this way, 5 of the 7 scores, i.e., $\{0.4, 0.7, -0.2, 1.4, 0.1\}$, would align with 0, while $\{2.3, 1.9\}$ align with 1.



(a) $\epsilon = 10^{-3}$ (b) $\epsilon = 5 \times 10^{-3}$ (c) $\epsilon = 10^{-2}$ (d) $\epsilon = 5 \times 10^{-2}$

Figure 3: Color maps of Γ^ϵ (upper) and the corresponding scatter plots of values in A^ϵ (lower), where $\mathcal{X}$ contains 50 standard Gaussian samples, and $K = 5$. The scatter plots show the correspondence of the input $\mathcal{X}$ and output A^ϵ.

Figure 2 illustrates the corresponding optimal transport plan for parameterizing the top-5 operator applied to a set of 7 elements. As can be seen, the mass from the 5 closest points is transported to 0, and meanwhile the mass from the 2 remaining points is transported to 1. Therefore, the optimal transport plan exactly indicates the top-5 elements.

2.3 Smoothing by Entropy Regularization

We next rule out the discontinuity of (1) to obtain a smoothed approximation to the standard top-k operator.

Specifically, we employ entropy regularization to the OT problem (1):

$$\Gamma^{*,\epsilon} = \operatorname*{argmin}_{\Gamma \geq 0} \langle C, \Gamma \rangle + \epsilon H(\Gamma), \quad \text{s.t.,} \quad \Gamma \mathbf{1}_m = \mu, \Gamma^\top \mathbf{1}_n = \nu, \tag{5}$$

where $h(\Gamma) = \sum_{i,j} \Gamma_{ij} \log \Gamma_{ij}$ is the entropy regularizer. We define $A^\epsilon = n\Gamma^{*,\epsilon} \cdot [0,1]^\top$ as a smoothed counterpart of output A in the standard top-k operator. Accordingly, SOFT top-k operator is defined as the mapping from $\mathcal{X}$ to A^ϵ. We show that the Jacobian matrix of SOFT top-k operator exists and is nonzero in the following theorem.

Theorem 1. For any $\epsilon > 0$, SOFT top-k operator: $\mathcal{X} \mapsto A^\epsilon$ is differentiable, as long as the cost C_{ij} is differentiable with respect to x_i for any i, j. Moreover, the Jacobian matrix of SOFT top-k operator always has a nonzero entry for any $\mathcal{X} \in \mathbb{R}^n$.

The proof can be found in Appendix A. We remark that the entropic OT (5) is computationally more friendly, since it allows the usage of first-order algorithms (Cuturi, 2013).

The Entropic OT introduces bias to the SOFT top-k operator. The following theorem shows that such a bias can be effectively controlled.

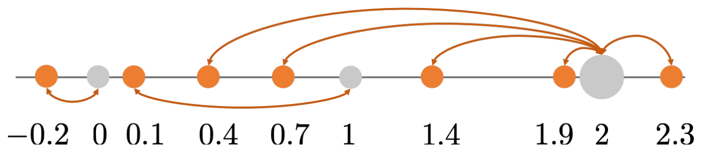


Figure 4: Illustration of the optimal transport plan for sorted top-k with input $\mathcal{X} = [0.4, 0.7, 2.3, 1.9, -0.2, 1.4, 0.1]^\top$ and $K = 2$. Here, we set $\nu = [\frac{1}{7}, \frac{1}{7}, \frac{5}{7}]^\top$ and $\mathcal{B} = [0, 1, 2]^\top$. In this way, the smallest score -0.2 aligns with 0, the second smallest score 0.1 aligns with 1, and the rest of the scores align with 2.

Theorem 2. Given a distinct sequence $\mathcal{X}$ and its sorting permutation σ, with Euclidean square cost function, for the proposed top-k solver we have

$$\|\Gamma^{*,\epsilon} - \Gamma^*\|_F \le \frac{\epsilon(\ln n + \ln 2)}{n(x_{\sigma_{k+1}} - x_{\sigma_k})}.$$

Therefore, with a small enough ϵ, the output vector A^ϵ can well approximate A, especially when there is a large gap between x_{σ_k} and $x_{\sigma_{k+1}}$. Besides, Theorem 2 suggests a trade-off between the bias and regularization of SOFT top-k operator. See Section 8 for a detailed discussion.

2.4 Sorted SOFT Top-k Operator

In some applications like beam search, we not only need to distinguish the top-k elements, but also sort the top-k elements. For example, in image retrieval (Gordo et al., 2016), the retrieved k images are expected to be sorted. We show that our proposed SOFT top-k operator can be extended to the sorted SOFT top-k operator.

Analogous to the derivation of the SOFT top-k operator, we first parameterize the sorted top-k operator in terms of an OT problem. Specifically, we keep $\mathcal{A} = \mathcal{X}$ and $\mu = \mathbf{1}_n/n$ and set

$$\mathcal{B} = [0, 1, 2, \cdots, k]^\top, \quad \text{and} \quad \nu = [1/n, \cdots, 1/n, (n-k)/n]^\top.$$

One can check that the optimal transport plan of the above OT problem transports the smallest element in $\mathcal{A}$ to 0 in $\mathcal{B}$, the second smallest element to 1, and so on so forth. This in turn yields the sorted top-k elements. Figure 4 illustrates the sorted top-2 operator and its corresponding optimal transport plan.

The sorted SOFT top-k operator is obtained similarly to SOFT top-k operator by solving the entropy regularized OT problem. We can show that the sorted SOFT top-k operator is differentiable and the bias can be properly controlled.

3 Efficient Implementation

We now present our implementation of SOFT top-k operator, which consists of 1) computing A^ϵ from $\mathcal{X}$ and 2) computing the Jacobian matrix of A^ϵ with respect to $\mathcal{X}$. We refer to 1) as the forward pass and 2) as the backward pass.

Forward Pass. The forward pass from $\mathcal{X}$ to A^ϵ can be efficiently computed using Sinkhorn algorithm. Specifically, we run iterative Bregman projections (Benamou et al., 2015), where at the ℓ-th iteration, we update

$$p^{(\ell+1)} = \frac{\mu}{Gq^{(\ell)}}, \quad q^{(\ell+1)} = \frac{\nu}{G^\top p^{(\ell+1)}}.$$

Here the division is entrywise, $q^{(0)} = \mathbf{1}_2/2$, and $G \in \mathbb{R}^{n\times m}$ with $G_{ij} = e^{-\frac{C_{ij}}{\epsilon}}$. Denote p^* and q^* as the stationary point of the Bregman projections. The optimal transport plan $\Gamma^{*,\epsilon}$ can be otained by $\Gamma^{*,\epsilon}_{ij} = p^*_i G_{ij} q^*_j$. The algorithm is summarized in Algorithm 1.

Algorithm 1 SOFT Top-k

```
Require: $\mathcal{X} = [x_i]_{i=1}^n, k, \epsilon, L$
  $\mathcal{Y} = [y_1, y_2]^\top = [0,1]^\top$
  $\mu = \mathbf{1}_n/n, \nu = [k/n, (n-K)/n]^\top$
  $C_{ij} = |x_i - y_j|^2, G_{ij} = e^{-\frac{C_{ij}}{\epsilon}}, q = \mathbf{1}_2/2$
  for $l = 1, \cdots, L$ do
    $p = \mu/(Gq), q = \nu/(G^\top p)$
  end for
  $\Gamma = \mathrm{diag}(p) \odot G \odot \mathrm{diag}(q)$
  $A^\epsilon = n\Gamma \cdot [0,1]^\top$
```

Backward Pass. Given A^ϵ, we compute the Jacobian matrix $\frac{dA^\epsilon}{d\mathcal{X}}$ using implicit differentiation and differentiable programming techinques. Specifically, the Lagrangian function of Problem (5) is

$$\mathcal{L} = \langle C, \Gamma\rangle - \xi^\top(\Gamma\mathbf{1}_m - \mu) - \zeta^\top(\Gamma^\top\mathbf{1}_n - \nu) + \epsilon H(\Gamma),$$

where ξ and ζ are dual variables. The KKT condition implies that the optimal solution $\Gamma^{*,\epsilon}$ can be formulated using the optimal dual variables ξ^* and ζ^* as (SinkhornâĂŹs scaling theorem, Sinkhorn & Knopp (1967)),

$$\Gamma^{*,\epsilon} = \mathrm{diag}(e^{\frac{\xi^*}{\epsilon}})e^{-\frac{C}{\epsilon}}\mathrm{diag}(e^{\frac{\zeta^*}{\epsilon}}). \tag{6}$$

Substituting (6) into the Lagrangian function, we obtain

$$\mathcal{L}(\xi^*, \zeta^*; C) = (\xi^*)^\top\mu + (\zeta^*)^\top\nu - \epsilon\sum_{i,j=1}^{n,m} e^{-\frac{C_{ij}-\xi^*_i-\zeta^*_j}{\epsilon}}.$$

We now compute the gradient of ξ^* and ζ^* with respect to C, such that we can obtain $d\Gamma^{*,\epsilon}/dC$ by the chain rule applied to (6). Denote $\omega^* = [(\xi^*)^\top, (\zeta^*)^\top]^\top$, and $\phi(\omega^*; C) = \partial\mathcal{L}(\omega^*; C)/\partial\omega^*$. At the optimal dual variable ω^*, the KKT condition immediately yields

$$\phi(\omega^*; C) \equiv 0.$$

By the chain rule, we have

$$\frac{d\phi(\omega^*; C)}{dC} = \frac{\partial\phi(\omega^*; C)}{\partial C} + \frac{\partial\phi(\omega^*; C)}{\partial\omega^*}\frac{d\omega^*}{dC} = 0.$$

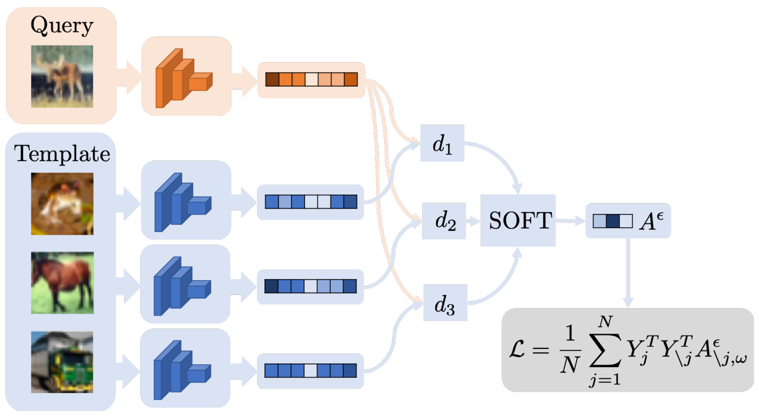

Query
Template
d_1
d_2
d_3
SOFT
A^ϵ
$\mathcal{L} = \frac{1}{N}\sum_{j=1}^{N} Y_j^T Y_{\setminus j}^T A^\epsilon_{\setminus j,\omega}$

Figure 5: Illustration of the entire forward pass of kNN.

Rerranging terms, we obtain

$$\frac{d\omega^*}{dC} = -\left(\frac{\partial\phi(\omega^*;C)}{\partial\omega^*}\right)^{-1}\frac{\partial\phi(\omega^*;C)}{\partial C}. \tag{7}$$

Combining (6), (7), $C_{ij} = (x_i - y_j)^2$, and $A^\epsilon = n\Gamma^{*,\epsilon}\cdot[1,0]^\top$, the Jacobian matrix $dA^\epsilon/d\mathcal{X}$ can then be derived using the chain rule again.

The detailed derivation and the corresponding algorithm for computing the Jacobian matrix can be found in Appendix B. The time and space complexity of the derived algorithm is $\mathcal{O}(n)$ and $\mathcal{O}(kn)$ for top-k and sorted top-k operators, respectively. We also include a Pytorch (Paszke et al., 2017) implementation of the forward and backward pass in Appendix B by extending the `autograd` automatic differentiation package.

4 k-NN for Image Classification

The proposed SOFT top-k operator enables us to train an end-to-end neural network-based kNN classifier. Specifically, we receive training samples $\{Z_i, y_i\}_{i=1}^N$ with Z_i being the input data and $y_i \in \{1,\ldots,M\}$ the label from M classes. During the training, for an input data Z_j (also known as the query sample), we associate a loss as follows. Denote $Z_{\setminus j}$ as all the input data excluding Z_j (also known as the template samples). We use a neural network to extract features from all the input data, and measure the pairwise Euclidean distances between the extracted features of $Z_{\setminus j}$ and that of Z_j. Denote $\mathcal{X}_{\setminus j,\theta}$ as the collection of these pairwise distances, i.e.,

$$\mathcal{X}_{\setminus j,\theta} = \{\|f_\theta(Z_1)-f_\theta(Z_j)\|_2,\ldots,\|f_\theta(Z_{j-1})-f_\theta(Z_j)\|_2,\|f_\theta(Z_{j+1})-f_\theta(Z_j)\|_2,\ldots,\|f_\theta(Z_N)-f_\theta(Z_j)\|_2\},$$

where f_θ is the neural network parameterized by θ, and the subscript of $\mathcal{X}$ emphasizes its dependence on θ.

Next, we apply SOFT top-k operator to $\mathcal{X}_{\setminus j,\omega}$, and the returned vector is denoted by $A^\epsilon_{\setminus j,\theta}$. Let $Y_{\setminus j} \in \mathbb{R}^{M\times(N-1)}$ be the matrix by concatenating the one-hot encoding of labels y_i for $i \neq j$ as columns, and $Y_j \in \mathbb{R}^M$ the one-hot encoding of the label y_j. The loss of Z_j is defined as

$$\ell(Z_j, y_j) = Y_j^\top Y_{\setminus j}^\top A^\epsilon_{\setminus j,\theta}.$$

Consequently, the training loss is

$$\mathcal{L}(\{Z_j, y_j\}_{j=1}^N) = \frac{1}{N}\sum_{j=1}^N \ell(Z_j, y_j) = \frac{1}{N}\sum_{j=1}^N Y_j^\top Y_{\backslash j}^\top A_{\backslash j,\theta}^\epsilon.$$

Recall that the Jacobian matrix of $A_{\backslash j,\theta}^\epsilon$ exists and has no zero entries. This allows us to utilize stochastic gradient descent algorithms to update θ in the neural network. Moreover, since N is often large, to ease the computation, we randomly sample a batch of samples to compute the stochastic gradient at each iteration.

In the prediction stage, we use all the training samples to obtain a predicted label of a query sample. Specifically, we feed the query sample into the neural network to extract its features, and compute pairwise Euclidean distances to all the training samples. We then run the standard kNN algorithm (Hastie et al., 2009) to obtain the predicted label.

4.1 Experiment

We evaluate the performance of the proposed neural network-based kNN classifier on two benchmark datasets: MNIST dataset of handwritten digits (LeCun et al., 1998) and the CIFAR-10 dataset of natural images (Krizhevsky et al., 2009) with the canonical splits for training and testing without data augmentation. We adopt the coefficient of entropy regularizer $\epsilon = 10^{-3}$ for MNIST dataset and $\epsilon = 10^{-5}$ for CIFAR-10 dataset. Detailed settings of the model and training procedure are deferred to Appendix C.

Baselines. We consider several baselines:

1. Standard kNN method.
2. Two-stage training methods: we first extract the features of the images, and then perform kNN on the features. The feature is extracted using Principle Component Analysis (PCA, top-50 principle components is adopted), autoencoder (AE), or a pretrained Convolutional Neural Network (CNN) using the Cross-Entropy (CE) loss.
3. Differentiable *ranking* + kNN: This includes NeuralSort (Grover et al., 2019) and Cuturi et al. (2019). Cuturi et al. (2019) is not directly applicable, which requires some adaptation (see Appendix C).
4. Stochastic kNN with Gumbel top-k relaxation (Xie & Ermon, 2019): The model is referred as RelaxSubSample.
5. Softmax Augmentation for smoothed top-k operation: A combination of k softmax operation is used to replace the top-k operator. Specifically, we recursively perform softmax on $\mathcal{X}$ for k times (Similar idea appears in Plötz & Roth (2018) and Goyal et al. (2018)). At the k-th iteration, we mask the top-$(k-1)$ entries with negative infinity.
6. CNNs trained with CE without any top-k component[2].

For the pretrained CNN and CNN trained with CE, we adopt identical neural networks as our method.

Results. We report the classification accuracies on the standard test sets in Table 1. On both datasets, the SOFT kNN classifier achieves comparable or better accuracies.

[2]Our implementation is based on github.com/pytorch/vision.git

Table 1: Classification accuracy of kNN.

Algorithm	MNIST	CIFAR10
*k*NN	97.2%	35.4%
*k*NN+PCA	97.6%	40.9%
*k*NN+AE	97.6%	44.2%
*k*NN+pretrained CNN	98.4%	91.1%
RelaxSubSample	99.3%	90.1%
*k*NN+NeuralSort	**99.5**%	90.7%
*k*NN+Cuturi et al. (2019)	99.0%	84.8%
*k*NN+Softmax *k* times	99.3%	92.2%
CE+CNN (He et al., 2016)	99.0%	91.3%
*k*NN+**SOFT Top-*k***	99.4%	**92.6**%

5 Beam Search for Machine Translation

Beam search is a popular method for the *inference* of Neural Language Generation (NLG) models, e.g., machine translation models. Here, we propose to incorporate beam search into the *training* procedure based on SOFT top-k operator.

5.1 Misalignment between Training and Inference

Denote the predicted sequence as $y = [y^{(1)}, \cdots, y^{(T)}]$, and the vocabularies as $\{z_1, \cdots, z_V\}$. Consider a recurrent network based NLG model. The output of the model at the t-th decoding step is a probability simplex $[\mathbb{P}(y^{(t)} = z_i | h^{(t)}]_{i=1}^{V}$, where $h^{(t)}$ is the hidden state associated with the sequence $y^{(1:t)} = [y^{(1)}, ..., y^{(t)}]$.

Beam search recursively keeps the sequences with the k largest likelihoods, and discards the rest. Specifically, at the $(t+1)$-th decoding step, we have k sequences $\widetilde{y}^{(1:t),i}$'s obtained at the t-th step, where $i = 1, ..., k$ indexes the sequences. The likelihood of $\widetilde{y}^{(1:t),i}$ is denoted by $\mathcal{L}_{\mathrm{s}}(\widetilde{y}^{(1:t),i})$. We then select the next k sequences by varying $i = 1, \dots, k$ and $j = 1, \dots, V$:

$$\{\widetilde{y}^{(1:t+1),\ell}\}_{\ell=1}^{k} = \arg\text{top-k}_{[\widetilde{y}^{(1:t),i}, z_j]} \mathcal{L}_{\mathrm{s}}([\widetilde{y}^{(1:t),i}, z_j]).$$

where $\mathcal{L}_{\mathrm{s}}([\widetilde{y}^{(1:t),i}, z_j])$ is the likelihood of the sequence appending z_j to $\widetilde{y}^{(1:t),i}$ defined as

$$\mathcal{L}_{\mathrm{s}}([\widetilde{y}^{(1:t),i}, z_j]) = \mathbb{P}(y^{(t+1)} = z_j | h^{(t+1),i}) \mathcal{L}_{\mathrm{s}}(\widetilde{y}^{(1:t),i}), \tag{8}$$

and $h^{(t+1),i}$ is the hidden state generated from $\widetilde{y}^{(1:t),i}$. Note that z_j's and $\widetilde{y}^{(1:t),i}$'s together yield Vk choices. Here we abuse the notation: $\widetilde{y}^{(1:t+1),\ell}$ denotes the ℓ-th selected sequence at the $(t+1)$-th decoding step, and is not necessarily related to $\widetilde{y}^{(1:t),i}$ at the t-th decoding step, even if $i = \ell$.

For $t = 1$, we set $\widetilde{y}^{(1)} = z_{\mathrm{s}}$ as the start token, $\mathcal{L}_{\mathrm{s}}(y^{(1)}) = 1$, and $h^{(1)} = h_{\mathrm{e}}$ as the output of the encoder. We repeat the above procedure, until the end token is selected or the pre-specified max length is reached. At last, we select the sequence $y^{(1:T),*}$ with the largest likelihood as the predicted sequence.

Moreover, the most popular training procedure for NLG models directly uses the so-called *"teacher forcing"* framework. As the ground truth of the target sequence (i.e., gold sequence) $\bar{y} = [\bar{y}^{(1)}, \cdots, \bar{y}^{(T)}]$ is provided at the training stage, we can directly maximize the likelihood

$$\mathcal{L}_{\mathrm{tf}} = \prod_{t=1}^{T} \mathbb{P}(y^{(t)} = \bar{y}^{(t)} | h^{(t)}(\bar{y}^{(1:t\text{-}1)})). \tag{9}$$

As can be seen, such a training framework only involve the gold sequence, and cannot take the uncertainty of the recursive exploration of the beam search into consideration. Therefore, it yields a misalignment between model training and inference (Bengio et al., 2015), which is also referred as *exposure bias* (Wiseman & Rush, 2016).

5.2 Differential Beam Search with Sorted SOFT Top-k

To mitigate the aforementioned misalignment, we propose to integrate beam search into the training procedure, where the top-k operator in the beam search algorithm is replaced with our proposed sorted SOFT top-k operator proposed in Section 2.4.

Specifically, at the $(t+1)$-th decoding step, we have k sequences denoted by $E^{(1:t),i}$, where $i = 1,...,k$ indexes the sequences. Here $E^{(1:t),i}$ consists of a sequence of D-dimensional vectors, where D is the embedding dimension. We are not using the tokens, and the reason behind will be explained later. Let $\widetilde{h}^{(t),i}$ denote the hidden state generated from $E^{(1:t),i}$. We then consider

$$\mathcal{X}^{(t)} = \{-\mathcal{L}_{\mathrm{s}}([E^{(1:t),i}, w_j]), j = 1,...,V,\ i = 1,...,k\},$$

where $\mathcal{L}_{\mathrm{s}}(\cdot)$ is defined analogously to (8), and $w_j \in \mathbb{R}^D$ is the embedding of token z_j.

Recall that ϵ is the smoothing parameter. We then apply the sorted SOFT top-k operator to $\mathcal{X}^{(t)}$ to obtain $\{E^{(1:t+1),\ell}\}_{\ell=1}^{k}$, which are k sequences with the largest likelihoods. More precisely, the sorted SOFT top-k operator yields an output tensor $A^{(t),\epsilon} \in \mathbb{R}^{V\times k\times k}$, where $A_{ji,\ell}^{(t),\epsilon}$ denotes the smoothed indicator of whether $[E^{(1:t),i}, w_j]$ has a rank ℓ. We then obtain

$$E^{(1:t+1),\ell} = \Big[E^{(1:t),r}, \sum_{j=1}^{V} \sum_{i=1}^{k} A_{ji,\ell}^{(t),\epsilon} w_j\Big], \tag{10}$$

where r denotes the index i (for $E^{(1:t),i}$'s) associated with the index ℓ (for $E^{(1:t+1),\ell}$'s). This is why we use vector representations instead of tokens: this allows us to compute $E^{(t+1),\ell}$ as a weighted sum of all the word embeddings $[w_j]_{j=1}^{V}$, instead of discarding the un-selected words.

Accordingly, we generate the k hidden states for the $(t+1)$-th decoding step:

$$\widetilde{h}^{(t),\ell} = \sum_{j=1}^{V} \sum_{i=1}^{k} A_{ji,\ell}^{(t),\epsilon} h^{(t),i}, \tag{11}$$

where $h^{(t),i}$ is the intermediate hidden state generated by the decoder based on $E^{(1:t),i}$.

After decoding, we select the sequence with largest likelihood $E^{(1:T),*}$, and maximize the likelihood as follows,

$$\mathcal{L}_{\text{SOFT}} = \prod_{t=1}^{T} \mathbb{P}(y^{(t)} = \bar{y}^{(t)} | \widetilde{h}^{(t-1),*}(E^{(1:t-1),*})).$$

We provide the sketch of training procedure in Algorithm 2, where we denote $\text{logit}^{(t),i}$ as $[\log \mathbb{P}(y^{(t+1)} = \omega_j | \widetilde{h}^{(t),i}(E^{(1:t),i}))]_{j=1}^{V}$, which is part of the output of the decoder. More technical details (e.g., backtracking algorithm for finding the index r in (10)) are provided in Appendix C.

Algorithm 2 Beam search training with SOFT Top-k

```
Require: Input sequence $s$, target sequence $\bar{y}$; embedding matrix $W \in \mathbb{R}^{V \times D}$; max length $T$; $k$;
  regularization coefficient $\epsilon$; number of Sinkhorn iteration $L$
  $\widetilde{h}_i^{(1)} = h_e = \text{Encoder}(s)$, $E^{(1),i} = w_s$
  for $t = 1, \cdots, T-1$ do
    for $i = 1, \cdots, k$ do
      $\text{logit}^{(t),i}, h^{(t),i} = \text{Decoder}(E^{(t),i}, \widetilde{h}^{(t),i})$
      $\log \mathcal{L}_s([E^{(1:t),i}, w_j]) = \log \mathcal{L}_s(E^{(1:t),i}) + \text{logit}_j^{(t),i}$
      $\mathcal{X}^{(t)} = \{-\log \mathcal{L}_s([E^{(1:t),i}, w_j]) \mid j = 1, \cdots, V\}$
    end for
    $A^{(t),\epsilon} = \text{Sorted-SOFT-Top-}k(\mathcal{X}^{(t)}, k, \epsilon, L)$
    Compute $E^{(t+1),\ell}$, $\widetilde{h}^{(t+1),\ell}$ as in (10) and (11)
  end for
  Compute $\nabla \mathcal{L}_{\text{SOFT}}$ and update the model
```

Note that integrating the beam search into training essentially yields a very large search space for the model, which is not necessarily affordable sometimes. To alleviate this issue, we further propose a hybrid approach by combining the teacher forcing training with beam search-type training. Specifically, we maximize the weighted likelihood defined as follows,

$$\mathcal{L}_{\text{final}} = \rho \mathcal{L}_{\text{tf}} + (1-\rho)\mathcal{L}_{\text{SOFT}},$$

where $\rho \in (0,1)$ is referred to as the "teaching forcing ratio". The teaching forcing loss $\mathcal{L}_{\text{tf}}$ can help reduce the search space and improve the overall performance.

5.3 Experiment

We evaluate our proposed beam search + sorted SOFT top-k training procedure using WMT2014 English-French dataset.

Settings. We adopt beam size 5, teacher forcing ratio $\rho = 0.8$, and $\epsilon = 10^{-1}$. For detailed settings of the training procedure, please refer to Appendix C.

We reproduce the experiment in Bahdanau et al. (2014), and run our proposed training procedure with the identical data pre-processing procedure and the LSTM-based sequence-to-sequence model. Different from Bahdanau et al. (2014), here we also preprocess the data with *byte pair encoding* (Sennrich et al., 2015).

Results. As shown in Table 2, the proposed SOFT beam search training procedure achieves an improvement in BLEU score of approximately 0.9. We also include other LSTM-based models for baseline comparison.

Table 2: BLEU scores on WMT'14 with single LSTM model.

Algorithm	BLEU
Luong et al. (2014)	33.10
Durrani et al. (2014)	30.82
Cho et al. (2014)	34.54
Sutskever et al. (2014)	30.59
Bahdanau et al. (2014)	28.45
Jean et al. (2014)	34.60
Bahdanau et al. (2014) (Our implementation)	35.38
Beam Search + Sorted SOFT Top-k	**36.27**

6 Top-k Attention for Machine Translation

We apply SOFT top-k operator to yield sparse attention scores. Attention module is an integral part of various natural language processing tasks, allowing modeling of long-term and local dependencies. Specifically, given the vector representations of a source sequence $s = [s_1, \cdots, s_N]^\top$ and target sequence $y = [y_1, \cdots, y_M]^\top$, we compute the alignment score between s_i and y_j by a compatibility function $f(s_i, y_j)$, e.g., $f(s_i, y_j) = s_i^\top y_j$, which measures the dependency between s_i and y_j. A softmax function then transforms the scores $[f(s_i, y_j)]_{i=1}^N$ to a sum-to-one weight vector w_j for each y_j. The output o_j of this attention module is a weighted sum of s_i's, i.e., $o_j = w_j^\top s$.

The attention module described above is called the soft attention, i.e., the attention scores w_j of y_j is not sparse. This may lead to redundancy of the attention (Zhu et al., 2018; Schlemper et al., 2019). Empirical results show that hard attention, i.e., enforcing sparsity structures in the score w_j's, yields more appealing performance (Shankar et al., 2018). Therefore, we propose to replace the softmax operation on $[f(s_i, y_j)]_{i=1}^N$ by the standard top-k operator to select the top-k elements. In order for an end-to-end training, we further deploy SOFT top-k operator to substitute the standard top-k operator. Given $[f(s_i, y_j)]_{i=1}^N$, the output of SOFT top-k operator is denoted by A_j^ϵ, and the weight vector w_j is now computed as

$$w_j = \mathrm{softmax}([f(s_1, y_j), \ldots, f(s_N, y_j)]^\top + \log A_j^\epsilon).$$

Here log is the entrywise logarithm. The output o_j of the attention module is computed the same $o_j = w_j^\top s$. Such a SOFT top-k attention will promote the top-k elements in $[f(s_i, y_j)]_{i=1}^N$ to be even larger than the non-top-k elements, and eventually promote the attention of y_j to focus on k tokens in s.

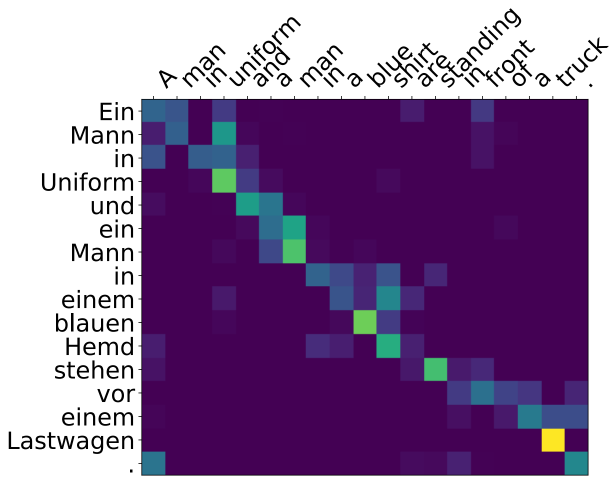


Figure 6: Visualization of the top-K attention.

6.1 Experiment

We evaluate the proposed top-k attention on WMT2016 English-German dataset. Our implementation and settings are based on Klein et al. (2017)[3]. For a fair comparison, we implement a standard soft attention using the same settings as the baseline. The details are provided in Appendix C.
Results. As shown in Table 3, the proposed SOFT top-k attention training procedure achieves an improvement in BLEU score of approximately 0.8. We visualize the top-k attention in Figure 6. The attention matrix is sparse, and has a clear semantic meaning – "truck" corresponds to "Lastwagen", "blue" corresponds to "blauen", "standing" corresponds to "stehen", etc.

Table 3: BLEU scores on WMT'16.

Algorithm	BLEU
Proposed Top-k Attention	**37.30**
Soft Attention	36.54

7 Related Work

We parameterize the top-k operator as an optimal transport problem, which shares the same spirit as Cuturi et al. (2019). Specifically, Cuturi et al. (2019) formulate the ranking and sorting problems as optimal transport problems. Ranking is more complicated than identifying the top-k elements, since one needs to align different ranks to corresponding elements. Therefore, the algorithm complexity per iteration for ranking whole n elements is $\mathcal{O}(n^2)$. Cuturi et al. (2019) also propose an optimal transport problem for finding the τ-quantile in a set of n elements and the algorithm complexity reduces to $\mathcal{O}(n)$. Top-k operator essentially finds all the elements more extreme than the $(n-k)/n$-quantile, and our proposed algorithm achieves the same complexity $\mathcal{O}(n)$ per iteration.

[3]Settings on data pre-processing, model, and training procedure is identical to https://opennmt.net/OpenNMT-py/extended.html.

The difference is that top-k operator returns the top-k elements in a given input set, while finding a quantile only yields a certain threshold.

Gumbel-Softmax trick (Jang et al., 2016) can also be utilized to derive a continuous relaxation of the top-k operator. Specifically, Kool et al. (2019) adapted such a trick to sample k elements from n choices, and Xie & Ermon (2019) further applied the trick to stochastic kNN, where neural networks are used to approximating the sorting operator. However, as shown in our experiments (see Table 1), the performance of stochastic kNN is not as good as deterministic kNN.

Our SOFT beam search training procedure is inspired by several works that incorporate some of the characteristics of beam search into the training procedure (Wiseman & Rush, 2016; Goyal et al., 2018; Bengio et al., 2015). Specifically, Wiseman & Rush (2016) and Goyal et al. (2018) both address the exposure bias issue in beam search. Wiseman & Rush (2016) propose a new loss function in terms of the error made during beam search. This mitigates the misalignment of training and testing in beam search. Later, Goyal et al. (2018) approximates the top-k operator using k softmax operations (This method is described and compared to our proposed method in 4). Such an approximation allows an end-to-end training of beam search. In addition, our proposed training loss $\mathcal{L}_{\text{final}}$ is inspired by Bengio et al. (2015), which combines the teacher forcing training procedure and greedy decoding, i.e., beam search with beam size 1.

8 Discussion

Relation to automatic differentiation. We compute the Jacobian matrix of SOFT top-k operator with respect to its input using the optimal transport plan of the entropic OT problem (5) in the backward pass. The optimal transport plan can be obtained by the Sinkhorn algorithm (Algorithm 1), which is iterative and each iteration only involves multiplication and addition. Therefore, we can also apply automatic differentiation (auto-diff) to compute the Jacobian matrix. Specifically, we denote Γ_ℓ as the transport plan at the t-th iteration of Sinkhorn algorithm. The update of Γ_ℓ can be written as $\Gamma_{\ell+1} = \mathcal{T}(\Gamma_\ell)$, where $\mathcal{T}$ denotes the update of the Sinkhorn algorithm. In order to apply auto-diff, we need to store all the intermediate states, e.g., p, q, G in each iteration, as defined in Algorithm 1 at each iteration. This requires a huge memory size proportional to the total number of iterations of the algorithm. In contrast, our backward pass allows us to save memory.
Bias and regularization trade-off. Theorem 2 suggests a trade-off between the regularization and bias of SOFT top-k operator. Specifically, a large ϵ has a strong smoothing effect on the entropic OT problem, and the corresponding entries of the Jacobian matrix are neither too large nor too small. This eases the end-to-end training process. However, the bias of SOFT top-k operator is large, which can deteriorate the model performance. On the contrary, a smaller ϵ ensures a smaller bias. Yet the SOFT top-k operator is less smooth, which in turn makes the end-to-end training less efficient.

On the other hand, the bias of SOFT top-k operator also depends on the gap between $x_{\sigma_{k+1}}$ and x_{σ_k}. In fact, such a gap can be viewed as the signal strength of the problem. A large gap implies that the top-k elements are clearly distinguished from the rest of the elements. Therefore, the bias is expected to be small since the problem is relatively easy. Moreover, in real applications

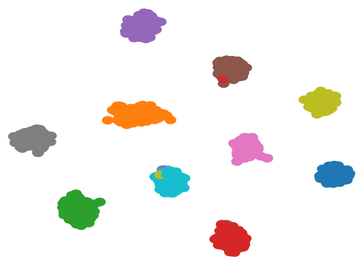


Figure 7: Visualization of the MNIST data based on features extracted by the neural network-based k-NN classifier trained by our proposed method in Section 4.

such as neural network-based kNN classification, the end-to-end training process promotes neural networks to extract features that exhibit a large gap (as illustrated in Figure 7). Hence, the bias of SOFT top-k operator can be well controlled in practice.

References

Amos, B. and Kolter, J. Z. Optnet: Differentiable optimization as a layer in neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 136–145. JMLR. org, 2017.

Babenko, A., Slesarev, A., Chigorin, A., and Lempitsky, V. Neural codes for image retrieval. In *European conference on computer vision*, pp. 584–599. Springer, 2014.

Bahdanau, D., Cho, K., and Bengio, Y. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Benamou, J.-D., Carlier, G., Cuturi, M., Nenna, L., and Peyré, G. Iterative bregman projections for regularized transportation problems. *SIAM Journal on Scientific Computing*, 37(2):A1111–A1138, 2015.

Bengio, S., Vinyals, O., Jaitly, N., and Shazeer, N. Scheduled sampling for sequence prediction with recurrent neural networks. In *Advances in Neural Information Processing Systems*, pp. 1171–1179, 2015.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Cuturi, M. Sinkhorn distances: Lightspeed computation of optimal transport. In *Advances in neural information processing systems*, pp. 2292–2300, 2013.

Cuturi, M., Teboul, O., and Vert, J.-P. Differentiable ranking and sorting using optimal transport. In *Advances in Neural Information Processing Systems*, pp. 6858–6868, 2019.

Duchi, J., Shalev-Shwartz, S., Singer, Y., and Chandra, T. Efficient projections onto the l 1-ball for learning in high dimensions. In *Proceedings of the 25th international conference on Machine learning*, pp. 272–279, 2008.

Durrani, N., Haddow, B., Koehn, P., and Heafield, K. EdinburghâĂŹs phrase-based machine translation systems for wmt-14. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*, pp. 97–104, 2014.

Gordo, A., Almazán, J., Revaud, J., and Larlus, D. Deep image retrieval: Learning global representations for image search. In *European conference on computer vision*, pp. 241–257. Springer, 2016.

Goyal, K., Neubig, G., Dyer, C., and Berg-Kirkpatrick, T. A continuous relaxation of beam search for end-to-end training of neural sequence models. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Griewank, A. and Walther, A. *Evaluating derivatives: principles and techniques of algorithmic differentiation*, volume 105. Siam, 2008.

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

Hastie, T., Tibshirani, R., and Friedman, J. *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media, 2009.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Hoare, C. A. Algorithm 65: Find. *Commun. ACM*, 4(7):321–322, July 1961. ISSN 0001-0782. doi: 10.1145/366622.366647.

Jang, E., Gu, S., and Poole, B. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

Jean, S., Cho, K., Memisevic, R., and Bengio, Y. On using very large target vocabulary for neural machine translation. *arXiv preprint arXiv:1412.2007*, 2014.

Kantorovich, L. V. Mathematical methods of organizing and planning production. *Management science*, 6(4):366–422, 1960.

Klein, G., Kim, Y., Deng, Y., Senellart, J., and Rush, A. M. OpenNMT: Open-source toolkit for neural machine translation. In *Proc. ACL*, 2017. doi: 10.18653/v1/P17-4012. URL https://doi.org/10.18653/v1/P17-4012.

Kool, W., Van Hoof, H., and Welling, M. Stochastic beams and where to find them: The gumbel-top-k trick for sampling sequences without replacement. *arXiv preprint arXiv:1903.06059*, 2019.

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

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Luise, G., Rudi, A., Pontil, M., and Ciliberto, C. Differential properties of sinkhorn approximation for learning with wasserstein distance. In *Advances in Neural Information Processing Systems*, pp. 5859–5870, 2018.

Luong, M.-T., Sutskever, I., Le, Q. V., Vinyals, O., and Zaremba, W. Addressing the rare word problem in neural machine translation. *arXiv preprint arXiv:1410.8206*, 2014.

Monge, G. Mémoire sur la théorie des déblais et des remblais. *Histoire de l'Académie Royale des Sciences de Paris*, 1781.

Papernot, N. and McDaniel, P. Deep k-nearest neighbors: Towards confident, interpretable and robust deep learning. *arXiv preprint arXiv:1803.04765*, 2018.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. Automatic differentiation in pytorch. 2017.

Plötz, T. and Roth, S. Neural nearest neighbors networks. In *Advances in Neural Information Processing Systems*, pp. 1087–1098, 2018.

Radenović, F., Tolias, G., and Chum, O. Cnn image retrieval learns from bow: Unsupervised fine-tuning with hard examples. In *European conference on computer vision*, pp. 3–20. Springer, 2016.

Reddy, D. R. et al. Speech understanding systems: A summary of results of the five-year research effort. department of computer science, 1977.

Schlemper, J., Oktay, O., Schaap, M., Heinrich, M., Kainz, B., Glocker, B., and Rueckert, D. Attention gated networks: Learning to leverage salient regions in medical images. *Medical image analysis*, 53:197–207, 2019.

Sennrich, R., Haddow, B., and Birch, A. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*, 2015.

Shankar, S., Garg, S., and Sarawagi, S. Surprisingly easy hard-attention for sequence to sequence learning. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 640–645, 2018.

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

Sutskever, I., Vinyals, O., and Le, Q. V. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pp. 3104–3112, 2014.

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

Xie, S. M. and Ermon, S. Reparameterizable subset sampling via continuous relaxations. In *International Joint Conference on Artificial Intelligence*, 2019.

Zhu, C., Tan, X., Zhou, F., Liu, X., Yue, K., Ding, E., and Ma, Y. Fine-grained video categorization with redundancy reduction attention. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 136–152, 2018.

A Theoretical Guarantees

First, we show that after adding entropy regularization the problem is differentiable.

Theorem 1. For any $\epsilon > 0$, SOFT top-k operator: $\mathcal{X} \mapsto A^\epsilon$ is differentiable, as long as the cost C_{ij} is differentiable with respect to x_i for any i, j. Moreover, the Jacobian matrix of SOFT top-k operator always has a nonzero entry for any $\mathcal{X} \in \mathbb{R}^n$.

Proof. We first prove the differentiability. This part of proof mirrors the proof in Luise et al. (2018). By Sinkhorn's scaling theorem,

$$\Gamma^{*,\epsilon} = \mathrm{diag}(e^{\frac{\xi^*}{\epsilon}})e^{-\frac{C}{\epsilon}}\mathrm{diag}(e^{\frac{\zeta^*}{\epsilon}}).$$

Therefore, since C_{ij} is differentiable, $\Gamma^{*,\epsilon}$ is differentiable if (ξ^*, ζ^*) is differentiable as a function of input scores X.

Let us set

$$\mathcal{L}(\xi, \zeta; \mu, \nu, C) = \xi^T \mu + \zeta^T \nu - \epsilon \sum_{i,j=1}^{n,m} e^{-\frac{C_{ij}-\xi_i-\zeta_j}{\epsilon}}.$$

and recall that $(\xi^*, \zeta^*) = \mathrm{argmax}_{\xi,\zeta} L(\xi, \zeta; \mu, \nu, C)$. The differentiability of (ξ^*, ζ^*) is proved using the Implicit Function theorem and follows from the differentiability and strong convexity in (ξ^*, ζ^*) of the function $\mathcal{L}$.

Now we prove that dA^ϵ/dx_ℓ always has a nonzero entry for $l = 1, \cdots, n$. First, we prove that for any $\ell \in \{1, \cdots, n\}$, $d\Gamma^{*,\epsilon}/dx_\ell$ always has a nonzero entry. We will prove it by contradiction. Specifically, the KKT conditions for the stationarity are as follows

$$\xi_i^* + \zeta_j^* = (x_i - y_j)^2 - \epsilon \log \Gamma_{ij}^{*,\epsilon}, \quad \forall i = 1, \cdots, n, j = 1, \cdots, m.$$

If we view the above formula as a linear equation set of the dual variables, it has nm equations and $m+n$ variables. Therefore, there are $nm - m - n$ redundant equations. Suppose one of the scores x_ℓ, has an infinitesimal change δx_ℓ. Assuming $\Gamma^{*,\epsilon}$ does not change, we have a new set of linear equations,

$$\begin{aligned} \xi_i^* + \zeta_j^* &= (x_i - y_j)^2 - \epsilon \log \Gamma_{ij}^{*,\epsilon}, \quad \forall i \neq \ell, \\ \xi_\ell^* + \zeta_j^* &= (x_\ell + \delta x_\ell - y_j)^2 + \delta C_{\ell j} - \epsilon \log \Gamma_{\ell j}^{*,\epsilon}. \end{aligned}$$

Easy to verify that this set of linear equations has no solution. Therefore, there must be at least one entry in $\Gamma^{*,\epsilon}$ has changed. As a result, $d\Gamma^{*,\epsilon}/dx_\ell$ always has a nonzero entry. We denote this entry as $\Gamma_{i'j'}^{*,\epsilon}$. Since $\Gamma_{i'j'}^{*,\epsilon} + \Gamma_{i',3-j'}^{*,\epsilon} = \mu_{i'}$, we have

$$\frac{d\Gamma_{i',3-j'}^{*,\epsilon}}{dx_\ell} = -\frac{d\Gamma_{i'j'}^{*,\epsilon}}{dx_\ell} \neq 0.$$

Therefore, there must be a nonzero entry in the first column of $d\Gamma^{*,\epsilon}/dx_\ell$. Recall A^ϵ is the first column of $\Gamma^{*,\epsilon}$. As a result, there must be a nonzero entry in dA^ϵ/dx_ℓ for any $\ell \in \{1, \cdots, n\}$.

□

Second, we would like to know after smoothness relaxation, how much bias is introduced to A^ϵ.

Lemma 1. Denote the feasible set of optimal transport problem as $\Delta = \{\Gamma : \Gamma \in [0,1]^{n\times m}, \Gamma \mathbf{1}_m = \mu, \Gamma \mathbf{1}_n = \nu\}$. Assume the optimal transport plan is unique. Denote Γ^* as the optimal transport plan,

$$\Gamma^* = \operatorname*{argmin}_{\Gamma\in\Delta} f(\Gamma) = \operatorname*{argmin}_{\Gamma\in\Delta} \langle C, \Gamma\rangle,$$

and $\Gamma^{*,\epsilon}$ as the entropy regularized transport plan,

$$\Gamma^{*,\epsilon} = \operatorname*{argmin}_{\Gamma\in\Delta} f^\epsilon(\Gamma) = \operatorname*{argmin}_{\Gamma\in\Delta} f(\Gamma) - \epsilon H(\Gamma) = \operatorname*{argmin}_{\Gamma\in\Delta} \langle C, \Gamma\rangle + \epsilon \sum_{i,j} \Gamma_{ij} \ln \Gamma_{ij}.$$

We can bound the difference between Γ^* and $\Gamma^{*,\epsilon}$ to be

$$\|\Gamma^* - \Gamma^{*,\epsilon}\|_F \le \epsilon \frac{(\ln n + \ln m)}{B},$$

where $\|\cdot\|_F$ is the Frobenius norm, and B is a positive constant irrelevant to ϵ.

Proof. Note that $H(\Gamma)$ is the entropy function. Since $0 \le \Gamma_{ij} \le 1$ and $\sum_{ij} \Gamma_{ij} = 1$ for any $\Gamma \in \Delta$, we can view Δ as the subset of a simplex. Therefore,

1. $H(\Gamma)$ is non-negative.
2. The maximum of $H(\Gamma)$ in the simplex can be obtained at $\Gamma_{ij} \equiv \frac{1}{nm}$. Therefore the maximum value is $(\ln n + \ln m)$.

Therefore, $0 \le H(\Gamma) \le (\ln n + \ln m)$ for any $\Gamma \in \Delta$.

Since $H(\Gamma) \ge 0$, we have $f^\epsilon(\Gamma) \le f(\Gamma)$ for any $\Gamma \in \Delta$. As a result, we have $f^\epsilon(\Gamma^{*,\epsilon}) \le f(\Gamma^*)$. In other words, we have

$$\langle C, \Gamma^{*,\epsilon}\rangle - \epsilon H(\Gamma^{*,\epsilon}) - \langle C, \Gamma^*\rangle \le 0.$$

Therefore,

$$\langle C, \Gamma^{*,\epsilon} - \Gamma^*\rangle = \langle C, \Gamma^{*,\epsilon}\rangle - \langle C, \Gamma^*\rangle \le \epsilon H(\Gamma^{*,\epsilon}) \le \epsilon(\ln n + \ln m).$$

Since the optimal transport problem is a linear optimization problem, Γ^* is one of the vertices of Δ. Denote $e_0, e_1, \cdots, e_J$ as the vertices of Δ, and without loss of generality we assume $e_0 = \Gamma^*$. Since $\Gamma^{*,\epsilon} \in \Delta$, we can denote $\Gamma^{*,\epsilon} = \sum_{j=0}^J \lambda_j e_j$, where $\lambda_j \ge 0$, and $\sum_j \lambda_j = 1$. Since Γ^* is unique, we have

$$\langle C, e_j - e_0\rangle > 0, \quad \forall j = 1, \cdots, J.$$

Denote $B_j = \langle C, e_j - e_0\rangle$. Since the space we are considering is Euclidean space (if we reshape the matrices into vectors), we can write the inner product as

$$B_j = \langle C, e_j - e_0\rangle = \|C\|_F \|e_j - e_0\|_F \cos\theta_{(C, e_j - e_0)} > 0.$$

So we have $\cos\theta_{(C,e_j-e_0)} > 0$. In other words, the angle between C and $e_j - e_0$ is always smaller than $\frac{\pi}{2}$. Therefore, the angle between C and the affine combination of $e_j - e_0$, namely $\sum_{j=0}^{J}\lambda_j(e_j - e_0)$, is also smaller than $\frac{\pi}{2}$. More specifically, we have

$$\cos\theta_{(C,\Gamma^{*,\epsilon}-\Gamma^*)} = \cos\theta_{(C,\sum_{j=0}^{J}\lambda_j(e_j-e_0))} \geq \min_j \cos\theta_{(C,e_j-e_0)} = \min_j \frac{B_j}{\|C\|_F\|e_j - e_0\|_F}.$$

Therefore, we have

$$\|\Gamma^{*,\epsilon} - \Gamma^*\|_F = \frac{\langle C, \Gamma^{*,\epsilon} - \Gamma^*\rangle}{\|C\|_F \cos\theta_{(C,\Gamma^{*,\epsilon}-\Gamma^*)}} \leq \frac{\epsilon(\ln n + \ln m)}{\|C\|_F \min_j \frac{B_j}{\|C\|_F\|e_j-e_0\|_F}} = \frac{\epsilon(\ln n + \ln m)}{\min_j \frac{B_j}{\|e_j-e_0\|_F}}.$$

Denote $B = \min_j \frac{B_j}{\|e_j-e_0\|_F}$, and we have the conclusion. □

Remark 1. In Theorem 1 we restricted the optimal solution to be unique, only for clarity purpose. If it is not unique, similar conclusion holds, except that the proof is more tedious – instead of divide the vertices into e_0 and others, we need to divide it into the vertices that are optimal solutions and the others.

Lemma 2. At each of the vertices of Δ, the entries of Γ are either 0 or $1/n$ for $\Gamma \in \Delta$.

Proof. The key idea is to prove by contradiction: If there exist i, j such that $\Gamma_{ij} \in (0, 1/n)$, then Γ cannot be a vertex.

To ease the discussion, we denote $Z = n\Gamma$. We will first prove that the entries of Z are either 0 or 1 at the vertices.

Notice that

$$\begin{aligned} &Z_{i,1} + Z_{i,2} = 1, \quad \forall i = 1, \cdots, n, \\ &\sum_i Z_{i,1} = k, \\ &\sum_i Z_{i,2} = n - k. \end{aligned}$$

If there exists an entry $Z_{i',j'} \in (0,1)$, then

1. $Z_{i',3-j'} \in (0,1)$.

2. there must exist $i'' \neq i'$, such that $Z_{i'',j'} \in (0,1)$. This is because $\sum_{i=1}^{n} Z_{i,j}$ is an integer, and $Z_{i',j'}$ is not.

3. As a result, $Z_{i'',3-j'} \in (0,1)$.

Therefore, consider $\delta \in (-\min\{1 - Z_{i'.j'}, Z_{i',j'}\}, \min\{1 - Z_{i'.j'}, Z_{i',j'}\})$ and denote

$$\widetilde{Z}^{(1)}_{ij} = \begin{cases} Z_{i',j'} + \delta, \text{ if } i = i', j = j', \\ Z_{i',3-j'} - \delta, \text{ if } i = i', j = 3 - j', \\ Z_{i'',j'} - \delta, \text{ if } i = i'', j = j', \\ Z_{i'',3-j'} + \delta, \text{ if } i = i'', j = 3 - j', \\ Z_{i,j}, \text{ otherwise.} \end{cases}$$

$$\widetilde{Z}^{(2)}_{ij} = \begin{cases} Z_{i',j'} - \delta, \text{ if } i = i', j = j', \\ Z_{i',3-j'} + \delta, \text{ if } i = i', j = 3 - j', \\ Z_{i'',j'} + \delta, \text{ if } i = i'', j = j', \\ Z_{i'',3-j'} - \delta, \text{ if } i = i'', j = 3 - j', \\ Z_{i,j}, \text{ otherwise.} \end{cases}$$

We can easily verify that $\widetilde{Z}^{(1)}/n, \widetilde{Z}^{(2)}/n \in \Delta$, and also $Z = (\widetilde{Z}^{(1)} + \widetilde{Z}^{(2)})/2$. Therefore, Z cannot be a vertex.

□

Lemma 3. Given a set of scalar $\{x_1, \cdots, x_n\}$, we sort it to be $\{x_{\sigma_1}, \cdots, x_{\sigma_n}\}$. If Euclidean square cost is adopted, Γ^* has the following form,

$$\Gamma^*_{ij} = \begin{cases} 1/n, \text{ if } i = \sigma_\ell, j = 1, \ell \le k \\ 0, \text{ if } i = \sigma_\ell, j = 1, k < \ell \le n \\ 1/n, \text{ if } i = \sigma_\ell, j = 2, k < \ell \le n \\ 0, \text{ if } i = \sigma_\ell, j = 2, \ell \le k \end{cases}$$

And $\min_j \frac{B_j}{\|e_j - e_0\|_F}$ is attained at at a vertex Γ^{**}, where $\Gamma^{**}_{ij} = \Gamma^*_{ij}$ except that the σ_k-th row and the σ_{k+1}-th row are swapped. As a result, we have

$$\min_j \frac{B_j}{\|e_j - e_0\|_F} = n(x_{\sigma_{k+1}} - x_{\sigma_k}).$$

Proof. From Lemma 2, in each vertex the entries of Γ is either 0 or $1/n$. Also, $\Gamma^* \in \Delta = \{\Gamma : \Gamma \in [0,1]^{n\times m}, \Gamma \mathbf{1}_m = \mathbf{1}_n/n, \Gamma \mathbf{1}_n = [k/n, (n-k)/n]^\top\}$. Therefore, for the j-th vertex, there are k entries with value $1/n$ in the first row of Γ. Denote the row indices of these k entries as $\mathcal{I}_j$, and $\Omega = \{1, \cdots, n\}$. Then for each vertex we have

$$\begin{aligned} &\Gamma_{i,1} = 1/n, \quad \forall i \in \mathcal{I}_j \\ &\Gamma_{i,1} = 0, \quad \forall i \in \Omega \backslash \mathcal{I}_j \\ &\Gamma_{i,2} = 1/n, \quad \forall i \in \Omega \backslash \mathcal{I}_j \\ &\Gamma_{i,2} = 0, \quad \forall i \in \mathcal{I}_j. \end{aligned}$$

Denote $\mathcal{I}^* = \{\sigma_1, \cdots, \sigma_k\}$. We now prove that $\mathcal{I}^*$ corresponds to the optimal solution Γ^*. This is because for any $j \in \{1, \cdots, J\}$

$$\begin{aligned}
\Gamma(\mathcal{I}_j) - \Gamma(\mathcal{I}^*) &= \left(\sum_{i \in \mathcal{I}_j} x_i^2 + \sum_{i \in \Omega \backslash \mathcal{I}_j} (x_i - 1)^2 \right) - \left(\sum_{i \in \mathcal{I}^*} x_i^2 + \sum_{i \in \Omega \backslash \mathcal{I}^*} (x_i - 1)^2 \right) \\
&= \left(\sum_{i \in \Omega} x_i^2 - \sum_{i \in \Omega \backslash \mathcal{I}_j} 2x_i + (n-k) \right) - \left(\sum_{i \in \Omega} x_i^2 - \sum_{i \in \Omega \backslash \mathcal{I}^*} 2x_i + (n-k) \right) \\
&= 2 \left(\sum_{i \in \Omega \backslash \mathcal{I}^*} x_i - \sum_{i \in \Omega \backslash \mathcal{I}_j} x_i \right) \geq 0,
\end{aligned}$$

where the last step is because the elements with indices $\Omega \backslash \mathcal{I}_j$ is the largest $n-k$ elements. Therefore we have $\Gamma(\mathcal{I}^*) = \Gamma^*$.

Now let's compute $\min_{j \neq 0} B_j / \|e_j - e_0\|$. Denote set subtraction $\mathcal{A} - \mathcal{B}$ as the set if elements that belongs to $\mathcal{A}$ but do not belong to $\mathcal{B}$, and $|\mathcal{A}|$ as the number of elements in $\mathcal{A}$.

$$\begin{aligned}
\frac{B_j}{\|e_j - e_0\|} &= \frac{B_j}{\|\Gamma(\mathcal{I}_j) - \Gamma(\mathcal{I}^*)\|} \\
&= 2 \frac{\sum_{i \in \Omega \backslash \mathcal{I}^*} x_i - \sum_{i \in \Omega \backslash \mathcal{I}_j} x_i}{2\sqrt{|\mathcal{I}^* - \mathcal{I}_j|/n}} \\
&= n \frac{\sum_{i \in (\mathcal{I}_j - \mathcal{I}^*)} x_i - \sum_{i \in (\mathcal{I}^* - \mathcal{I}_j)} x_i}{\sqrt{|\mathcal{I}^* - \mathcal{I}_j|}},
\end{aligned}$$

where the second line can be obtained by substituting the definition of B_j. Notice that $\mathcal{I}_j - \mathcal{I}^* \in \Omega \backslash \mathcal{I}^*$ and $\mathcal{I}^* - \mathcal{I}_j \in \mathcal{I}^*$. Any element with index in $\Omega \backslash \mathcal{I}^*$ is larger than any element in $\mathcal{I}^*$ by at least $x_{\sigma_{K+1}} - x_{\sigma_K}$. Then we have

$$\begin{aligned}
\frac{B_j}{\|e_j - e_0\|} &= N \frac{\sum_{i \in (\mathcal{I}_j - \mathcal{I}^*)} x_i - \sum_{i \in (\mathcal{I}^* - \mathcal{I}_j)} x_i}{\sqrt{|\mathcal{I}^* - \mathcal{I}_j|}} \\
&\geq N \frac{|\mathcal{I}^* - \mathcal{I}_j| (x_{\sigma_{K+1}} - x_{\sigma_K})}{\sqrt{|\mathcal{I}^* - \mathcal{I}_j|}} \\
&\geq N (x_{\sigma_{K+1}} - x_{\sigma_K}),
\end{aligned}$$

where the last step is because for $j \neq 0$, $|\mathcal{I}^* - \mathcal{I}_j|$ is at least 1.

Also notice that the value $n(x_{\sigma_{k+1}} - x_{\sigma_k})$ can be attained at $\mathcal{I}_{j^*} = \{\sigma_1, \cdots, \sigma_{k-1}, \sigma_{k+1}\}$. Therefore we have

$$\min_j \frac{B_j}{\|e_j - e_0\|} = n(x_{\sigma_{k+1}} - x_{\sigma_k}).$$

□

Theorem 2. Given a distinct sequence $\mathcal{X}$ and its sorting permutation σ, with Euclidean square cost function, for the proposed top-k solver we have

$$\|\Gamma^{*,\epsilon} - \Gamma^*\| \leq \frac{\epsilon(\ln n + \ln 2)}{n(x_{\sigma_{k+1}} - x_{\sigma_k})}.$$

Proof. This is a direct conclusion with Lemma 1 and Lemma 3. □

B The Expression of the Gradient of A^ϵ

In this section we will derive the expression of dA^ϵ/dx_i. We first list a few reminders that will be used later:

- $\{x_i\}_{i=1}^n$ is a scalar set to be solved for top-k. $\{y_j\}_{j=1}^m$ is taken to be $\{0,1\}$.
- $C \in \mathbb{R}^{n\times m}$ is the cost matrix, usually defined as $C_{ij} = (x_i - y_j)^2$.
- The loss function of entropic optimal transport is

$$\Gamma^{*,\epsilon} = \underset{\Gamma\in\Delta}{\operatorname{argmin}} f^\epsilon(\Gamma) = \underset{\Gamma\in\Delta}{\operatorname{argmin}}\langle C, \Gamma\rangle + \epsilon \sum_{i,j} \Gamma_{ij} \ln \Gamma_{ij},$$

  where $\Delta = \{\Gamma : \Gamma \in [0,1]^{n\times m}, \Gamma\mathbf{1}_m = \mu, \Gamma\mathbf{1}_n = \nu\}$.
- The dual problem of the above optimization problem is

$$\xi^*, \zeta^* = \underset{\xi,\zeta}{\operatorname{argmax}} \mathcal{L}(\xi, \zeta; C),$$

  where

$$\mathcal{L}(\xi,\zeta;C) = \xi^\top \mu + \zeta^\top \nu - \epsilon \sum_{i,j=1}^{n,m} e^{-\frac{C_{ij}-\xi_i-\zeta_j}{\epsilon}}.$$

  And it is connected to the prime form by

$$\Gamma^{*,\epsilon} = \operatorname{diag}(e^{\frac{\xi^*}{\epsilon}}) e^{-\frac{C}{\epsilon}} \operatorname{diag}(e^{\frac{\zeta^*}{\epsilon}}).$$

  The converged p, q in Algorithm 1 is actually $e^{\frac{\xi^*}{\epsilon}}$ and $e^{\frac{\zeta^*}{\epsilon}}$.

If we obtain the expression for $\frac{d\xi^*}{dC}$ and $\frac{d\zeta^*}{dC}$, we can obtain the expression for $\frac{dA^\epsilon}{dx_i}$.

In this section only, we denote $\Gamma = \Gamma^{*,\epsilon}$, to shorten the notation. The multiplication of 3rd-order tensors mirrors the multiplication of matrices: we always use the last dimension of the first input to multiplies the first dimension of the second input. We denote $\bar{b} = b_{:-1}$ as b removing the last entry, $\bar{\nu} = \nu_{:-1}$ as ν removing the last entry, $\bar{\Gamma} = \Gamma_{:,:-1}$ as Γ removing the last column.

Theorem 3. $\frac{d\xi^*}{dC}$ and $\frac{d\zeta^*}{dC}$ have the following expression,

$$\begin{bmatrix} \frac{d\xi^*}{dC} \\ \frac{d\zeta^*}{dC} \end{bmatrix} = \begin{bmatrix} -H^{-1}D \\ \mathbf{0} \end{bmatrix}$$

where $-H^{-1}D \in \mathbb{R}^{(n+m-1)\times n\times m}$, $\mathbf{0} \in \mathbb{R}^{1\times n\times m}$, and

$$
\begin{aligned}
D_{\ell ij} &= \frac{1}{\epsilon}\begin{cases}\delta_{\ell i}\Gamma_{ij}, \ell = 1, \cdots, n \\ \delta_{\ell j}\Gamma_{ij}, \ell = n+1, \cdots, n+m-1\end{cases} \\
H^{-1} &= -\epsilon\begin{bmatrix}(\mathrm{diag}(\mu))^{-1} + (\mathrm{diag}(\mu))^{-1}\bar{\Gamma}\mathcal{K}^{-1}\bar{\Gamma}^T(\mathrm{diag}(\mu))^{-1} & -(\mathrm{diag}(\mu))^{-1}\bar{\Gamma}\mathcal{K}^{-1} \\ -\mathcal{K}^{-1}\bar{\Gamma}^T(\mathrm{diag}(\mu))^{-1} & \mathcal{K}^{-1}\end{bmatrix} \\
\mathcal{K} &= \mathrm{diag}(\bar{\nu}) - \bar{\Gamma}^T(\mathrm{diag}(\mu))^{-1}\bar{\Gamma}.
\end{aligned}
$$

Proof. Notice that there is one redundant dual variable, since $\mu\mathbf{1}_N = \nu\mathbf{1}_M = 1$. Therefore, we can rewrite $\mathcal{L}(\xi, \zeta; C)$ as

$$
\mathcal{L}(\xi, \bar{\zeta}; C) = \xi^T\mu + \bar{\zeta}^T\bar{\nu} - \epsilon\sum_{i,j=1}^{n,m-1} e^{\frac{-C_{ij}+\xi_i+\zeta_j}{\epsilon}} - \epsilon\sum_{i=1}^{n} e^{\frac{-C_{im}+\xi_i}{\epsilon}}.
$$

Denote

$$
\phi(\xi, \bar{\zeta}, C) = \frac{d\mathcal{L}(\xi, \bar{\zeta}; C)}{d\xi} = \mu - F\mathbf{1}_m, \tag{12}
$$

$$
\psi(\xi, \bar{\zeta}, C) = \frac{d\mathcal{L}(\xi, \bar{\zeta}; C)}{d\bar{\zeta}} = \bar{\nu} - \bar{F}^\top\mathbf{1}_n, \tag{13}
$$

where

$$
\begin{aligned}
F_{ij} &= e^{\frac{-C_{ij}+\xi_i+\zeta_j}{\epsilon}}, \quad \forall i = 1, \cdots, n, \quad j = 1, \cdots, m-1 \\
F_{im} &= e^{\frac{-C_{im}+\xi_i}{\epsilon}}, \quad \forall i = 1, \cdots, n, \\
\bar{F} &= F_{:,:-1}.
\end{aligned}
$$

Since $(\xi^*, \bar{\zeta}^*)$ is a maximum of $\mathcal{L}(\xi, \bar{\zeta}; C)$, we have

$$
\begin{aligned}
\phi(\xi^*, \bar{\zeta}^*, C) &= 0, \\
\psi(\xi^*, \bar{\zeta}^*, C) &= 0.
\end{aligned}
$$

Therefore,

$$
\begin{aligned}
\frac{d\phi(\xi^*, \bar{\zeta}^*, C)}{dC} &= \frac{\partial\phi(\xi^*, \bar{\zeta}^*, C)}{\partial C} + \frac{\partial\phi(\xi^*, \bar{\zeta}^*, C)}{\partial\xi^*}\frac{d\xi^*}{dC} + \frac{\partial\phi(\xi^*, \bar{\zeta}^*, \mu, \nu, C)}{\partial\bar{\zeta}^*}\frac{d\bar{\zeta}^*}{dC} = 0, \\
\frac{d\psi(\xi^*, \bar{\zeta}^*, C)}{dC} &= \frac{\partial\psi(\xi^*, \bar{\zeta}^*, C)}{\partial C} + \frac{\partial\psi(\xi^*, \bar{\zeta}^*, C)}{\partial\xi^*}\frac{d\xi^*}{dC} + \frac{\partial\psi(\xi^*, \bar{\zeta}^*, C)}{\partial\bar{\zeta}^*}\frac{d\bar{\zeta}^*}{dC} = 0.
\end{aligned}
$$

Therefore,

$$
\begin{aligned}
\begin{bmatrix}\frac{d\xi^*}{dC} \\ \frac{d\bar{\zeta}^*}{dC}\end{bmatrix} &= -\begin{bmatrix}\frac{\partial\phi(\xi^*, \bar{\zeta}^*, C)}{\partial\xi^*} & \frac{\partial\phi(\xi^*, \bar{\zeta}^*, C)}{\partial\bar{\zeta}^*} \\ \frac{\partial\psi(\xi^*, \bar{\zeta}^*, C)}{\partial\xi^*} & \frac{\partial\psi(\xi^*, \bar{\zeta}^*, C)}{\partial\bar{\zeta}^*}\end{bmatrix}^{-1}\begin{bmatrix}\frac{\partial\phi(\xi^*, \bar{\zeta}^*, C)}{\partial C} \\ \frac{\partial\psi(\xi^*, \bar{\zeta}^*, C)}{\partial C}\end{bmatrix} \\
&\triangleq -H^{-1}\begin{bmatrix}D^{(1)} \\ D^{(2)}\end{bmatrix} \\
&\triangleq -H^{-1}D.
\end{aligned}
$$

Now let's compute each of the terms.

$$\begin{aligned}
\frac{\partial \phi(\xi^*,\bar{\zeta}^*,C)_h}{\partial C_{ij}} &= -\frac{\partial [F\mathbf{1}_m]_h}{\partial C_{ij}} = -\frac{\partial}{\partial C_{ij}}\left(\sum_{\ell=1}^{m-1} e^{\frac{-C_{h\ell}+a_h+b_\ell}{\epsilon}} + e^{\frac{-C_{hm}+a_h}{\epsilon}}\right) \\
&= \frac{1}{\epsilon}\delta_{hi}F_{ij} = \frac{1}{\epsilon}\delta_{hi}\Gamma_{ij} \\
&\forall h = 1,\cdots,n, \quad i = 1,\cdots,n, \quad j = 1,\cdots,m \\
\frac{\partial \psi(\xi^*,\bar{\zeta}^*,C)_\ell}{\partial C_{ij}} &= -\frac{\partial [\bar{F}^\top \mathbf{1}_n]_\ell}{\partial C_{ij}} = -\frac{\partial}{\partial C_{ij}}\sum_{h=1}^{n} e^{\frac{-C_{h\ell}+a_h+b_\ell}{\epsilon}} \\
&= \frac{1}{\epsilon}\delta_{\ell j}F_{ij} = \frac{1}{\epsilon}\delta_{\ell j}\Gamma_{ij} \\
&\forall \ell = 1,\cdots,m-1, \quad i = 1,\cdots,n, \quad j = 1,\cdots,m \\
\frac{\partial \phi(\xi^*,\bar{\zeta}^*,C)_h}{\partial \xi_i^*} &= -\frac{\partial [F\mathbf{1}_m]_h}{\partial \xi_i^*} = -\frac{\partial}{\partial \xi_i^*}\left(\sum_{\ell=1}^{m-1} e^{\frac{-C_{h\ell}+a_h+b_\ell}{\epsilon}} + e^{\frac{-C_{hm}+a_h}{\epsilon}}\right) \\
&= -\frac{1}{\epsilon}\delta_{hi}\sum_{\ell=1}^{m} F_{h\ell} = -\frac{1}{\epsilon}\delta_{hi}\mu_h \\
&\forall h = 1,\cdots,n, \quad i = 1,\cdots,n \\
\frac{\partial \phi(\xi^*,\bar{\zeta}^*,C)_h}{\partial \bar{\zeta}_j^*} &= -\frac{\partial [F\mathbf{1}_m]_h}{\partial \bar{\zeta}_j^*} = -\frac{\partial}{\partial \bar{\zeta}_j^*}\left(\sum_{\ell=1}^{m-1} e^{\frac{-C_{h\ell}+a_h+b_\ell}{\epsilon}} + e^{\frac{-C_{hm}+a_h}{\epsilon}}\right) \\
&= -\frac{1}{\epsilon}\sum_{\ell=1}^{m-1}\delta_{\ell j}F_{h\ell} = -\frac{1}{\epsilon}F_{hj} = -\frac{1}{\epsilon}\Gamma_{hj} \\
&\forall h = 1,\cdots,n, \quad j = 1,\cdots,m-1 \\
\frac{\partial \psi(\xi^*,\bar{\zeta}^*,C)_\ell}{\partial \xi_i^*} &= -\frac{\partial [\bar{F}^\top \mathbf{1}_n]_\ell}{\partial \xi_i^*} = -\frac{\partial}{\partial \xi_i^*}\sum_{h=1}^{n} e^{\frac{-C_{h\ell}+a_h+b_\ell}{\epsilon}} \\
&= -\frac{1}{\epsilon}\sum_{h=1}^{n}\delta_{hi}F_{h\ell} = -\frac{1}{\epsilon}F_{i\ell} = -\frac{1}{\epsilon}\Gamma_{i\ell} \\
&\forall \ell = 1,\cdots,m-1, \quad i = 1,\cdots,n \\
\frac{\partial \psi(\xi^*,\bar{\zeta}^*,C)_\ell}{\partial \bar{\zeta}_j^*} &= -\frac{\partial [\bar{F}^\top \mathbf{1}_n]_\ell}{\partial \bar{\zeta}_j^*} = -\frac{\partial}{\partial \bar{\zeta}_j^*}\sum_{h=1}^{n} e^{\frac{-C_{h\ell}+a_h+b_\ell}{\epsilon}} \\
&= -\frac{1}{\epsilon}\sum_{h=1}^{n}\delta_{\ell j}F_{h\ell} = -\frac{1}{\epsilon}\delta_{\ell j}\nu_\ell \\
&\forall \ell = 1,\cdots,m-1, \quad j = 1,\cdots,m-1.
\end{aligned}$$

To sum up, we have

$$H = -\frac{1}{\epsilon}\begin{bmatrix} \mathrm{diag}(\mu) & \bar{\Gamma} \\ \bar{\Gamma}^T & \mathrm{diag}(\bar{\nu}) \end{bmatrix}.$$

Following the formula for inverse of block matrices,

$$\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix}^{-1} = \begin{bmatrix} \mathbf{A}^{-1} + \mathbf{A}^{-1}\mathbf{B}(\mathbf{D} - \mathbf{C}\mathbf{A}^{-1}\mathbf{B})^{-1}\mathbf{C}\mathbf{A}^{-1} & -\mathbf{A}^{-1}\mathbf{B}(\mathbf{D} - \mathbf{C}\mathbf{A}^{-1}\mathbf{B})^{-1} \\ -(\mathbf{D} - \mathbf{C}\mathbf{A}^{-1}\mathbf{B})^{-1}\mathbf{C}\mathbf{A}^{-1} & (\mathbf{D} - \mathbf{C}\mathbf{A}^{-1}\mathbf{B})^{-1} \end{bmatrix},$$

denote

$$\mathcal{K} = \operatorname{diag}(\bar{\nu}) - \bar{\Gamma}^T (\operatorname{diag}(\mu))^{-1} \bar{\Gamma}.$$

Note that $\mathcal{K}$ is just a scalar for SOFT top-k operator, and is a $(k-1) \times (k-1)$ matrix for sorted SOFT top-k operator. Therefore computing its inverse is not expensive. Finally we have

$$H^{-1} = -\epsilon \begin{bmatrix} (\operatorname{diag}(\mu))^{-1} + (\operatorname{diag}(\mu))^{-1}\bar{\Gamma}\mathcal{K}^{-1}\bar{\Gamma}^T(\operatorname{diag}(\mu))^{-1} & -(\operatorname{diag}(\mu))^{-1}\bar{\Gamma}\mathcal{K}^{-1} \\ -\mathcal{K}^{-1}\bar{\Gamma}^T(\operatorname{diag}(\mu))^{-1} & \mathcal{K}^{-1} \end{bmatrix}.$$

And also

$$D^{(1)}_{hij} = \frac{1}{\epsilon}\delta_{hi}\Gamma_{ij}$$
$$D^{(2)}_{\ell ij} = \frac{1}{\epsilon}\delta_{\ell j}\Gamma_{ij}.$$

The above derivation can actually be viewed as we explicitly force $b_m = 0$, i.e., no matter how C changes, b_m does not change. Therefore, we can treat $\frac{db_m}{dC} = \mathbf{0}_{n\times m}$, and we get the equation in the theorem. □

After we obtain $\frac{d\xi^*}{dC}$ and $\frac{d\zeta^*}{dC}$, we can now compute $\frac{d\Gamma}{dC}$.

$$\frac{d\Gamma_{h\ell}}{dC_{ij}} = \frac{d}{dC_{ij}} e^{\frac{-C_{h\ell}+a_h+b_\ell}{\epsilon}} = \frac{1}{\epsilon}\left(-\Gamma_{h\ell}\delta_{ih}\delta_{j\ell} + \Gamma_{h\ell}\frac{d\xi^*_h}{dC_{ij}} + \Gamma_{h\ell}\frac{db^*_\ell}{dC_{ij}}\right).$$

Finally, in the back-propagation step, we can compute the gradient of the loss L w.r.t. C,

$$\begin{aligned} \frac{dL}{dC_{ij}} &= \sum_{h,\ell=1}^{n,m} \frac{dL}{d\Gamma_{h\ell}} \frac{d\Gamma_{h\ell}}{dC_{ij}} \\ &= \frac{1}{\epsilon}\left(-\sum_{h,\ell=1}^{n,m} \frac{dL}{d\Gamma_{h\ell}}\Gamma_{h\ell}\delta_{in}\delta_{j\ell} + \sum_{h,\ell=1}^{n,m} \frac{dL}{d\Gamma_{h\ell}}\Gamma_{h\ell}\frac{d\xi^*_h}{dC_{ij}} + \sum_{h,\ell=1}^{n,m} \frac{dL}{d\Gamma_{h\ell}}\Gamma_{h\ell}\frac{db^*_\ell}{dC_{ij}}\right) \\ &= \frac{1}{\epsilon}\left(-\frac{dL}{d\Gamma_{ij}}\Gamma_{ij} + \sum_{h,\ell=1}^{n,m} \frac{dL}{d\Gamma_{h\ell}}\Gamma_{h\ell}\frac{d\xi^*_h}{dC_{ij}} + \sum_{h,\ell=1}^{n,m} \frac{dL}{d\Gamma_{h\ell}}\Gamma_{h\ell}\frac{db^*_\ell}{dC_{ij}}\right). \end{aligned}$$

We summarize the above procedure for computing the gradient for sorted SOFT top-k operator in Algorithm 3. This naive implementation takes $\mathcal{O}(n^2k)$ complexity, which is not efficient. Therefore, we modify the algorithm using the associative law of matrix multiplications, so that the complexity is lowered to $\mathcal{O}(nk)$. We summarize the modified algorithm in Algorithm 4.

We also include the `PyTorch` implementation of the forward pass and backward pass as shown below. The code is executed by creating an instance of `TopK_custom`, and the forward pass and the backward pass is run similar to any other `PyTorch` model.

Algorithm 3 Gradient for Sorted Top-K

```
Require: $C \in \mathbb{R}^{n\times(k+1)}, \mu \in \mathbb{R}^n, \nu \in \mathbb{R}^{k+1}, \frac{d\mathcal{L}}{d\Gamma} \in \mathbb{R}^{n\times(k+1)}, \epsilon$
  Run forward pass to get $\Gamma$
  $\bar{\nu} = \nu[:-1], \bar{\Gamma} = \Gamma[:,:-1]$
  $\mathcal{K} \leftarrow \text{diag}(\bar{\nu}) - \bar{\Gamma}^T(\text{diag}(\mu))^{-1}\bar{\Gamma}$    # $\mathcal{K} \in \mathbb{R}^{k\times k}$
  $H1 \leftarrow (\text{diag}(\mu))^{-1} + (\text{diag}(\mu))^{-1}\bar{\Gamma}\mathcal{K}^{-1}\bar{\Gamma}^T(\text{diag}(\mu))^{-1}$    # $H1 \in \mathbb{R}^{n\times n}$
  $H2 \leftarrow -(\text{diag}(\mu))^{-1}\bar{\Gamma}\mathcal{K}^{-1}$    # $H2 \in \mathbb{R}^{n\times k}$
  $H3 \leftarrow (H2)^T$    # $H3 \in \mathbb{R}^{k\times n}$
  $H4 \leftarrow \mathcal{K}^{-1}$    # $H4 \in \mathbb{R}^{k\times k}$
  Pad $H2$ to be $[n, k+1]$ in the last column with value 0
  Pad $H4$ to be $[k, k+1]$ in the last column with value 0
  $[\frac{d\xi^*}{dC}]_{hij} \leftarrow [H1]_{hi}\Gamma_{ij} + [H2]_{hj}\Gamma_{ij}$    # $\frac{d\xi^*}{dC} \in \mathbb{R}^{n\times n\times(k+1)}$
  $[\frac{db^*}{dC}]_{\ell ij} \leftarrow [H3]_{\ell i}\Gamma_{ij} + [H4]_{\ell j}\Gamma_{ij}$    # $\frac{db^*}{dC} \in \mathbb{R}^{k\times n\times(k+1)}$
  Pad $\frac{db^*}{dC}$ to be $[k+1, n, k+1]$ with value 0
  $[\frac{d\mathcal{L}}{dC}]_{ij} \leftarrow \frac{1}{\epsilon}(-[\frac{d\mathcal{L}}{d\Gamma}]_{ij}\Gamma_{ij} + \sum_{h,\ell}[\frac{d\mathcal{L}}{d\Gamma}]_{h\ell}\Gamma_{h\ell}[\frac{d\xi^*}{dC}]_{hij} + \sum_{h,\ell}[\frac{d\mathcal{L}}{d\Gamma}]_{h\ell}\Gamma_{h\ell}[\frac{db^*}{dC}]_{\ell ij})$
```

Algorithm 4 Gradient for Sorted Top-k, with reduced memory

```
Require: $C \in \mathbb{R}^{N\times(K+1)}, \mu \in \mathbb{R}^N, \nu \in \mathbb{R}^{K+1}, \frac{d\mathcal{L}}{d\Gamma} \in \mathbb{R}^{N\times(K+1)}, \epsilon$
  Run forward pass to get $\Gamma$
  $\bar{\nu} = \nu[:-1], \bar{\Gamma} = \Gamma[:,:-1]$
  $\mathcal{K} \leftarrow \text{diag}(\bar{\nu}) - \bar{\Gamma}^T(\text{diag}(\mu))^{-1}\bar{\Gamma}$    # $\mathcal{K} \in \mathbb{R}^{K\times K}$
  $\mu'_i = \mu_i^{-1}$
  $L \leftarrow (\text{diag}(\mu))^{-1}\bar{\Gamma}\mathcal{K}^{-1}$    # $L \in \mathbb{R}^{N\times K}$
  $G1 \leftarrow \frac{d\mathcal{L}}{d\Gamma} \odot \Gamma$    # $G1 \in \mathbb{R}^{N\times K}$
  $g1 \leftarrow [G1]\mathbf{1}_K, g2 \leftarrow [G1]^T\mathbf{1}_N$    # $g1 \in \mathbb{R}^N, g2 \in \mathbb{R}^K$
  $G21 \leftarrow (g1 \odot \mu').expand\_dims(1) \odot \Gamma$    # $G21 \in \mathbb{R}^{N\times(K+1)}$
  $G22 \leftarrow ((g1)^T L\bar{\Gamma}^T \odot \mu').expand\_dims(1) \odot \Gamma$    # $G22 \in \mathbb{R}^{N\times(K+1)}$
  $G23 \leftarrow -((g1)^T L).pad\_last\_entry(0).expand\_dims(0) \odot \Gamma$    # $G23 \in \mathbb{R}^{N\times(K+1)}$
  $G2 = G21 + G22 + G23$    # $G2 \in \mathbb{R}^{N\times(K+1)}$
  $g2 \leftarrow g2[:-1]$
  $G31 \leftarrow -(L(g2)).expand\_dims(1) \odot \Gamma$    # $G31 \in \mathbb{R}^{N\times(K+1)}$
  $G32 \leftarrow (\mathcal{K}^{-1}(g2)).pad\_last\_entry(0).expand\_dims(0) \odot \Gamma$    # $G32 \in \mathbb{R}^{N\times(K+1)}$
  $G3 = G31 + G32$    # $G3 \in \mathbb{R}^{N\times(K+1)}$
  $\frac{d\mathcal{L}}{dC} \leftarrow \frac{1}{\epsilon}(-G1 + G2 + G3)$
```

```
def sinkhorn_forward(C, mu, nu, epsilon, max_iter):
    bs, n, k_ = C.size()

    v = torch.ones([bs, 1, k_])/(k_)
    G = torch.exp(-C/epsilon)
    if torch.cuda.is_available():
        v = v.cuda()

    for i in range(max_iter):
        u = mu/(G*v).sum(-1, keepdim=True)
        v = nu/(G*u).sum(-2, keepdim=True)

    Gamma = u*G*v
    return Gamma

def sinkhorn_forward_stablized(C, mu, nu, epsilon, max_iter):
    bs, n, k_ = C.size()
    k = k_-1

    f = torch.zeros([bs, n, 1])
    g = torch.zeros([bs, 1, k+1])
    if torch.cuda.is_available():
        f = f.cuda()
        g = g.cuda()

    epsilon_log_mu = epsilon*torch.log(mu)
    epsilon_log_nu = epsilon*torch.log(nu)

    def min_epsilon_row(Z, epsilon):
        return -epsilon*torch.logsumexp((-Z)/epsilon, -1, keepdim=True)

    def min_epsilon_col(Z, epsilon):
        return -epsilon*torch.logsumexp((-Z)/epsilon, -2, keepdim=True)

    for i in range(max_iter):
        f = min_epsilon_row(C-g, epsilon)+epsilon_log_mu
        g = min_epsilon_col(C-f, epsilon)+epsilon_log_nu

    Gamma = torch.exp((-C+f+g)/epsilon)
    return Gamma

def sinkhorn_backward(grad_output_Gamma, Gamma, mu, nu, epsilon):

    nu_ = nu[:,:,:-1]
    Gamma_ = Gamma[:,:,:-1]

    bs, n, k_ = Gamma.size()

    inv_mu = 1./(mu.view([1,-1]))   #[1, n]
    Kappa = torch.diag_embed(nu_.squeeze(-2)) \
            -torch.matmul(Gamma_.transpose(-1, -2) * inv_mu.unsqueeze(-2), Gamma_)    #[bs, k, k]

    inv_Kappa = torch.inverse(Kappa) #[bs, k, k]

    Gamma_mu = inv_mu.unsqueeze(-1)*Gamma_
    L = Gamma_mu.matmul(inv_Kappa) #[bs, n, k]
    G1 = grad_output_Gamma * Gamma #[bs, n, k+1]

    g1 = G1.sum(-1)
    G21 = (g1*inv_mu).unsqueeze(-1)*Gamma   #[bs, n, k+1]
    g1_L = g1.unsqueeze(-2).matmul(L)   #[bs, 1, k]
    G22 = g1_L.matmul(Gamma_mu.transpose(-1,-2)).transpose(-1,-2)*Gamma   #[bs, n, k+1]
    G23 = - F.pad(g1_L, pad=(0, 1), mode='constant', value=0)*Gamma   #[bs, n, k+1]
    G2 = G21 + G22 + G23   #[bs, n, k+1]

    del g1, G21, G22, G23, Gamma_mu

    g2 = G1.sum(-2).unsqueeze(-1) #[bs, k+1, 1]
    g2 = g2[:,:-1,:]   #[bs, k, 1]
    G31 = - L.matmul(g2)*Gamma   #[bs, n, k+1]
    G32 = F.pad(inv_Kappa.matmul(g2).transpose(-1,-2), pad=(0, 1), mode='constant', value=0)*Gamma   #[bs, n, k+1]
    G3 = G31 + G32   #[bs, n, k+1]

    grad_C = (-G1+G2+G3)/epsilon   #[bs, n, k+1]
    return grad_C

class TopKFunc(Function):
    @staticmethod
    def forward(ctx, C, mu, nu, epsilon, max_iter):
        with torch.no_grad():
            if epsilon>1e-2:
                Gamma = sinkhorn_forward(C, mu, nu, epsilon, max_iter)
                if bool(torch.any(Gamma!=Gamma)):
                    print('Nan appeared in Gamma, re-computing...')
                    Gamma = sinkhorn_forward_stablized(C, mu, nu, epsilon, max_iter)
            else:
                Gamma = sinkhorn_forward_stablized(C, mu, nu, epsilon, max_iter)
            ctx.save_for_backward(mu, nu, Gamma)
            ctx.epsilon = epsilon
        return Gamma

    @staticmethod
    def backward(ctx, grad_output_Gamma):

        epsilon = ctx.epsilon
        mu, nu, Gamma = ctx.saved_tensors
        # mu [1, n, 1]
        # nu [1, 1, k+1]
        #Gamma [bs, n, k+1]
        with torch.no_grad():
            grad_C = sinkhorn_backward(grad_output_Gamma, Gamma, mu, nu, epsilon)
        return grad_C, None, None, None, None


class TopK_custom(torch.nn.Module):
    def __init__(self, k, epsilon=0.1, max_iter = 200):
        super(TopK_custom1, self).__init__()
        self.k = k
        self.epsilon = epsilon
        self.anchors = torch.FloatTensor([k-i for i in range(k+1)]).view([1,1, k+1])
        self.max_iter = max_iter

        if torch.cuda.is_available():
            self.anchors = self.anchors.cuda()

    def forward(self, scores):
        bs, n = scores.size()
        scores = scores.view([bs, n, 1])

        #find the -inf value and replace it with the minimum value except -inf
        scores_ = scores.clone().detach()
        max_scores = torch.max(scores_).detach()
        scores_[scores_==float('-inf')] = float('inf')
        min_scores = torch.min(scores_).detach()
        filled_value = min_scores - (max_scores-min_scores)
        mask = scores==float('-inf')
        scores = scores.masked_fill(mask, filled_value)

        C = (scores-self.anchors)**2
        C = C / (C.max().detach())

        mu = torch.ones([1, n, 1], requires_grad=False)/n
        nu = [1./n for _ in range(self.k)]
        nu.append((n-self.k)/n)
        nu = torch.FloatTensor(nu).view([1, 1, self.k+1])

        if torch.cuda.is_available():
            mu = mu.cuda()
            nu = nu.cuda()

        Gamma = TopKFunc.apply(C, mu, nu, self.epsilon, self.max_iter)

        A = Gamma[:,:,:self.k]*n

        return A, None
```

C Experiment Settings

C.1 *k*NN

The settings of the neural networks, the training procedure, and the number of neighbors k, and the tuning procedures are similar to Grover et al. (2019). The tuning o ϵ ranging from 10^{-6} to 10^{-2}.

Other settings are shown in Table 4.

Table 4: Parameter settings for kNN experiments.

Dataset	MNIST	CIFAR-10
k	9	9
ϵ	10^{-3}	10^{-5}
Batch size of query samples	100	100
Batch size of template samples	100	100
Optimizer	SGD	SGD
Learning rate	10^{-3}	10^{-3}
Momentum	0.9	0.9
Weight decay	5×10^{-4}	5×10^{-4}
Model	2-layer convolutional network	ResNet18

Note that f_θ is a feature extraction neural network, so that model specified in the last row of Table 4 does not contain the final activation layer and the linear layer.

Baselines. In the baselines, the results of kNN, kNN+PCA, kNN+AE, kNN+NeuralSort is copied from Grover et al. (2019). The result of RelaxSubSample is copied from Xie & Ermon (2019).

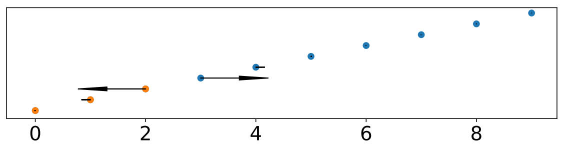
The implementation of kNN+Cuturi et al. (2019) is based on Grover et al. (2019). Specifically, the outputs of the models in Cuturi et al. (2019) and Grover et al. (2019) are both doubly stochastic matrices. So in the implementation of kNN+Cuturi et al. (2019), we adopt the algorithm in Grover et al. (2019), except that we replace the module of computing the doubly stochastic matrix to be the one in Cuturi et al. (2019). We extensively tuned k, ϵ and the learning rate, but cannot achieve a better score for this experiment.

The baselines kNN+Softmax k times, kNN+pretrained CNN, and CE+CNN adopts the identical neural networks as our model. We remark that the scores reported in Grover et al. (2019) for CNN+CE are 99.4% for MNIST and 95.1% for CIFAR-10. However, our experiments *using their code* cannot reproduce the reported scores: and the scores are 99.0% and 90.9%, respectively. Therefore, the reported score for MNIST is implemented by us, and the score for CIFAR-10 is copied from He et al. (2016).

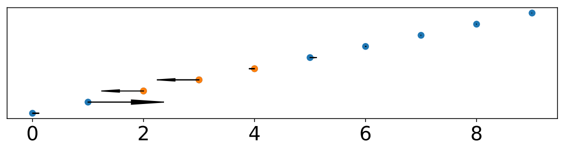
C.2 Beam Search

Algorithm. We now elaborate how to backtrack the predecessors $E^{(1:t),r}$ for an embedding $E^{(t+1),\ell}$, and how to compute the likelihood $\mathcal{L}_s(E^{(1:t+1),\ell})$, which we have omitted in Algorithm 2. Specifically, in standard beam search algorithm, each selected token $\widetilde{y}^{(t+1),\ell}$ is generated from a specific predecessor, and thus the backtracking is straightforward. In beam search with sorted SOFT top-k operator, however, each computed embedding $E^{(1:t),r}$ is a weighted sum of the output from all predecessors, so that it is not corresponding to one specific predecessor. To address this difficulty, we select the predecessor for $E^{(t+1),\ell}$ with the largest weight, i.e.,

$$(o,r) = \operatorname*{argmax}_{(j,i)} A^{(t),\epsilon}_{ji,\ell}.$$



(a) $\mathcal{I} = \{0, 1, 2\}$.



(b) $\mathcal{I} = \{2, 3, 4\}$.

Figure 8: Illustration of the gradient of the SOFT top-k operators. The arrows represent the direction and magnitude of the gradient. The orange dots corresponds to the ground truth elements.

This is a good approximation because $A^{(t),\epsilon}$ is a smoothed 0-1 tensor, i.e., for each ℓ, there is only one entry that is approximately 1 in $A^{(t),\epsilon}_{:,:,\ell}$, while the others are approximately 0. The likelihood is then computed as follows

$$\mathcal{L}_s(E^{(1:t+1),\ell}) = \mathcal{L}_s(E^{(1:t),r})\mathbb{P}(y^{t+1} = \omega_o | \widetilde{h}^{(t),r}(E^{(1:t),r})).$$

Implementation. The implemented model is identical to Bahdanau et al. (2014). Different from Bahdanau et al. (2014), here we also preprocess the data with *byte pair encoding* (Sennrich et al., 2015).

We adopt beam size 5, teacher forcing ratio $\rho = 0.8$, and $\epsilon = 10^{-1}$. The training procedure is as follows: We first pretrain the model with teacher forcing training procedure. The pretraining procedure has initial learning rate 1, learning rate decay 0.1 starting from iteration 5×10^5 for every 10^5 iterations. We pretrain it for 10^6 iterations in total. We then train the model using the combined training procedure for 10^5 iterations with learning rate 0.05.

C.3 Top-k Attention

The settings of the baseline model on data pre-processing, model, and the training procedure, evaluation procedure is identical to https://opennmt.net/OpenNMT-py/extended.html. The settings of the proposed model only differs in that we adopt SOFT top-k attention instead of the standard soft attention.

D Visualization of the Gradients

In this section we visualize the computed gradient using a toy example mimicking the settings of kNN classification. Specifically, we input 10 scores computed from 10 images, i.e., $\mathcal{X} = \{0, 1, 2, \cdots, 9\}$, into the SOFT top-$k$ operator, and select the top-3 elements. Denote the indices of the images with the same labels as the query sample as $\mathcal{I}$. Similar to kNN classification, we want to maximize $\sum_{i\in\mathcal{I}} A_i^\epsilon$.

We visualize the gradient on $\mathcal{X}$ with respect to this objective function in Figure 8. In Figure 8(a), $\mathcal{I}$ is the same as the indices of top-3 scores. In this case, the gradient will push the gap between the top-3 scores and the rest scores even further. In Figure 8(b), $\mathcal{I}$ is different from the indices of top-3 scores. In this case, the scores corresponding to $\mathcal{I}$ are pushed to be smaller, while the others are pushed to be larger.