Differentiable Transportation Pruning

Yunqiang Li[1] Jan C. van Gemert[2] Torsten Hoefler[3]
Bert Moons[1] Evangelos Eleftheriou[1] Bram-Ernst Verhoef[1]

[1]Axelera AI [2]TU Delft [3]ETH Zurich

A. Supplementary Material

A.1. Optimal coupling to regularized problem

Here we prove the optimal coupling to the regularized problem in Eq. (7) is:

$$\mathbf{P}_\varepsilon = e^{\mathbf{f}/\varepsilon} \odot e^{-\mathbf{C}/\varepsilon} \odot e^{\mathbf{g}/\varepsilon}. \tag{16}$$

Proof. Introducing two dual variables $\mathbf{f} \in \mathbb{R}^n$ and $\mathbf{g} \in \mathbb{R}^2$ for marginal constraints $\mathbf{P}\mathbb{1}_2 = \mathbf{a}$ and $\mathbf{P}^\mathsf{T}\mathbb{1}_n = \mathbf{b}$, given the discrete entropy of a coupling matrix $\mathcal{H}(\mathbf{P}) = -\sum_{ij} \mathbf{P}_{ij}(\log(\mathbf{P}_{ij}) - 1)$, the Lagrangian of Eq. (7) is optimized as follows:

$$\xi(\mathbf{P},\mathbf{f},\mathbf{g}) = \langle \mathbf{C},\mathbf{P}\rangle - \varepsilon\mathcal{H}(\mathbf{P}) - \langle \mathbf{f}, \mathbf{P}\mathbb{1}_2 - \mathbf{a}\rangle - \langle \mathbf{g}, \mathbf{P}^\mathsf{T}\mathbb{1}_n - \mathbf{b}\rangle . \tag{17}$$

First order conditions then yield:

$$\frac{\partial \xi(\mathbf{P},\mathbf{f},\mathbf{g})}{\partial \mathbf{P}_{ij}} = \mathbf{C} + \varepsilon \log(\mathbf{P}_{ij}) - \mathbf{f}_i - \mathbf{g}_j = 0, \tag{18}$$

which result for an optimal coupling to the regularized problem, in the matrix expression shown in Eq. (16).

A.2. Dual problem computation

Here we prove that minimizing the regularized optimal transport distance in Eq. (7) is equivalent to maximizing its dual problem:

$$\max_{\mathbf{f}\in\mathbb{R}^n, \mathbf{g}\in\mathbb{R}^2} \langle \mathbf{f},\mathbf{a}\rangle + \langle \mathbf{g},\mathbf{b}\rangle - \varepsilon \left\langle e^{\mathbf{f}/\varepsilon}, e^{-\mathbf{C}(\mathbf{s})/\varepsilon} \cdot e^{\mathbf{g}/\varepsilon} \right\rangle \tag{19}$$

Proof. We start from the result in Eq. (16), and substitute it in the Lagrangian $\xi(\mathbf{P},\mathbf{f},\mathbf{g})$ of Eq. (17), where the optimal $\mathbf{P}$ is a function of $\mathbf{f}$ and $\mathbf{g}$, we obtain that the Lagrange dual function equals:

$$\begin{aligned} \mathbf{f},\mathbf{g} \mapsto & \left\langle e^{\mathbf{f}/\varepsilon}, \left(e^{-\mathbf{C}(\mathbf{s})/\varepsilon} \odot \mathbf{C}(\mathbf{s})\right) e^{\mathbf{g}/\varepsilon} \right\rangle \\ & - \varepsilon \mathcal{H}\left(e^{\mathbf{f}/\varepsilon} \odot e^{-\mathbf{C}(\mathbf{s})/\varepsilon} \odot e^{\mathbf{g}/\varepsilon}\right) \end{aligned} \tag{20}$$

The negative entropy of $\mathbf{P}$ scaled by ε, namely $\varepsilon \langle \mathbf{P},\ \log\mathbf{P} - \mathbb{1}_{n\times 2}\rangle$, can be stated explicitly as a function of $\mathbf{f},\mathbf{g},\mathbf{C}$:

$$\begin{aligned} & \varepsilon \langle \mathbf{P},\ \log\mathbf{P} - \mathbb{1}_{n\times 2}\rangle \\ & = \left\langle e^{\mathbf{f}/\varepsilon} \odot e^{-\mathbf{C}(\mathbf{s})/\varepsilon} \odot e^{\mathbf{g}/\varepsilon}, \mathbf{f}\mathbb{1}_2^\mathsf{T} + \mathbb{1}_n\mathbf{g}^\mathsf{T} - \mathbf{C}(\mathbf{s}) - \varepsilon\mathbb{1}_{n\times 2} \right\rangle \\ & = - \left\langle e^{\mathbf{f}/\varepsilon}, \left(e^{-\mathbf{C}(\mathbf{s})/\varepsilon} \odot \mathbf{C}(\mathbf{s})\right) e^{\mathbf{g}/\varepsilon} \right\rangle + \langle \mathbf{f},\mathbf{a}\rangle + \langle \mathbf{g},\mathbf{b}\rangle - \\ & \quad \varepsilon \left\langle e^{\mathbf{f}/\varepsilon}, e^{-\mathbf{C}(\mathbf{s})/\varepsilon} \cdot e^{\mathbf{g}/\varepsilon} \right\rangle \end{aligned}$$

therefore, the first term in Eq. (20) cancels out with the first term in the entropy above. The remaining terms are those appearing in Eq. (19).

A.3. Expensive to train with Sinkhorn's algorithm

The bi-level optimization problem in Eq. (10) and Eq. (11) is expensive to train with Sinkhorn's algorithm: During forward pass, per mini-batch the inner optimization in Eq. (11) needs to perform Sinkhorn's iterative algorithm for hundreds of iterations to converge, which is computationally inefficient. During the back-propagation pass, the gradient of $\mathbf{P}_\varepsilon^*(\mathbf{s})$ with respect to the importance scores $\mathbf{s}$ is computed by differentiating [48] through all Sinkhorn iterations, which is expensive. We display the forward and backward of Sinkhorn algorithm in Fig. 5.

A.4. Bregman divergence based optimization

In this paper we use similar Bregman Divergence D_h as [66] based on entropy function $\mathcal{H}(\mathbf{x}) = -\sum_i x_i \left(\log(x_i) - 1\right)$ as:

$$D_\mathcal{H}(\mathbf{x},\mathbf{y}) = -\sum_i x_i \log\frac{x_i}{y_i} + \sum_i x_i. \tag{21}$$

Based on the defined Bregman Divergence, a single proximal point iteration for problem (4) can be written as:

$$\begin{aligned} \mathbf{P}^{(\ell+1)} &= \min_{\mathbf{P}\in\mathcal{U}(\mathbf{a},\mathbf{b})} \langle \mathbf{C},\mathbf{P}\rangle - \varepsilon D_\mathcal{H}(\mathbf{P},\mathbf{P}^{(\ell)}) \\ &= \min_{\mathbf{P}\in\mathcal{U}(\mathbf{a},\mathbf{b})} \left\langle \mathbf{C} - \varepsilon \log \mathbf{P}^{(\ell)}, \mathbf{P} \right\rangle - \varepsilon\mathcal{H}(\mathbf{P}). \end{aligned} \tag{22}$$

Denote $\mathbf{C}' = \mathbf{C} - \varepsilon \log \mathbf{P}^{(\ell)}$. Note that for optimization problem 22, $\mathbf{P}^{(\ell)}$ is a fixed value that is not relevant to optimization variable $\mathbf{P}$. Comparing to Eq. (7), the problem

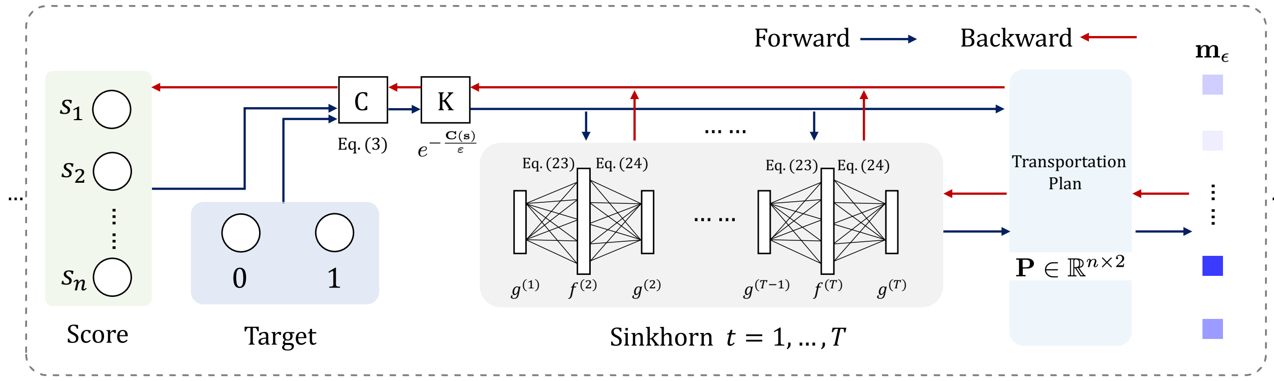

Figure 5. The display of training with Sinkhorn's algorithm. We show the forward and backward pass over one single layer: given importance score $\mathbf{s}$, the cost matrix is computed by Eq. (3), we thus get $\mathbf{K}$; in forward pass we input $\mathbf{K}$ to iteratively compute $\mathbf{f}^{(t+1)}$ with Eq. (27) and $\mathbf{g}^{(t+1)}$ with Eq. (28) using Sinkhorn algorithm for T iterations; during back-propagation pass, the gradient of soft mask with respect to the importance scores is computed by differentiating through all Sinkhorn iterations. A large T makes it expensive to train.

Table 5. Training setting: For the SGD optimizer, in the parentheses are the momentum and weight decay. For ImageNet, batch size is 64 per GPU. We learn the soft mask using cosine learning rate schedule.

Settings	CIFAR	ImageNet
Optimizer	SGD (0.9, 5e-4)	SGD (0.9, 1e-4)
LR schedule (soft mask)	Cosine LR schedule (0.1)	
LR schedule (finetune)	Multi-step (0:1e-2, 60:1e-3, 90:1e-4)	Multi-step (0:1e-2, 30:1e-3, 60:1e-4, 75:1e-5)
Training Epoch	120 + 120	90 + 90
Batch size	256	256

in Eq. (22) can be solved by Sinkhorn iteration by replacing Gibbs kernel $\mathbf{K}$ by $\mathbf{K}' = e^{-\frac{\mathbf{C}'}{\varepsilon}} = e^{-\frac{\mathbf{C}}{\varepsilon}} \odot \mathbf{P}^{(\ell)}$. With this reorganization, we can solve it with Sinkhorn algorithm. [66] have shown both theoretically and empirically that a *single Sinkhorn inner iteration* is sufficient to converge under a large range of fixed ε, we therefore compute $\mathbf{P}^{(\ell+1)}$ as:

$$\mathbf{P}^{(\ell+1)} = e^{\mathbf{f}^{(\ell+1)}/\varepsilon} \odot \left(e^{-\frac{\mathbf{C}}{\varepsilon}} \odot \mathbf{P}^{(\ell)}\right) \odot e^{\mathbf{g}^{(\ell+1)}/\varepsilon}, \tag{23}$$

which is the expression of Eq. (13).

A.5. Proximal point iteration as iterative Sinkhorn

We refer to the proof in Chapter 4.2 of the book [51]. The general setting for proximal point iteration is to define Gibbs kernel as $\mathbf{K} = e^{-\frac{\mathbf{C}}{\varepsilon}} \odot \mathbf{P}^{(\ell)}$, the proximal point iterations thus have the form:

$$\begin{aligned}\mathbf{P}_\varepsilon^{(\ell+1)}(\mathbf{s}) =& e^{\mathbf{f}^{(\ell+1)}/\varepsilon} \odot \left(e^{-\frac{\mathbf{C}}{\varepsilon}} \odot \mathbf{P}^{(\ell)}\right) \odot e^{\mathbf{g}^{(\ell+1)}/\varepsilon} \\ =& \left(e^{\mathbf{f}^{(\ell+1)}/\varepsilon} \odot \cdots \odot e^{\mathbf{f}^{(1)}/\varepsilon}\right) \odot \left(e^{-\frac{(\ell+1)\mathbf{C}}{\varepsilon}}\right. \\ & \left.\odot \mathbf{P}^{(\ell)}\right) \odot \left(e^{\mathbf{g}^{(\ell+1)}/\varepsilon} \odot \cdots \odot e^{\mathbf{g}^{(1)}/\varepsilon}\right).\end{aligned} \tag{24}$$

The proximal point iteration iteratively applies Sinkhorn's algorithm with a $e^{-\frac{\mathbf{C}}{\varepsilon/\ell}}$ kernel, *i.e.* with a decaying regularization parameter ε/ℓ, therefore it can perform an automatic decaying schedule on the regularization as $\ell \to \infty$ to gradually approach the optimal discrete plan.

A.6. Sinkhorn iterations on dual problem

A simple approach to solving the unconstrained maximization problem in Eq. (19) is to use an exact block coordinate ascent strategy, namely to update alternatively $\mathbf{f}$ and $\mathbf{g}$ to cancel the respective gradients in these variables of the objective of (19). Indeed, one can notice after a few elementary computations that, writing $\mathcal{L}_{\text{dual}}(\mathbf{f}, \mathbf{g}, \mathbf{s})$ for the objective of (19), we have the gradients, w.r.t., $\mathbf{f}$ and $\mathbf{g}$ as follows:

$$\nabla_{\mathbf{f}} \mathcal{L}_{\text{dual}}(\mathbf{f}, \mathbf{g}, \mathbf{s}) = \mathbf{a} - e^{\mathbf{f}/\varepsilon} \odot (\mathbf{K} e^{\mathbf{g}/\varepsilon}), \tag{25}$$

$$\nabla_{\mathbf{g}} \mathcal{L}_{\text{dual}}(\mathbf{f}, \mathbf{g}, \mathbf{s}) = \mathbf{b} - e^{\mathbf{g}/\varepsilon} \odot (\mathbf{K} e^{\mathbf{f}/\varepsilon}), \tag{26}$$

where $\mathbf{K} = e^{-\mathbf{C}(\mathbf{s})/\varepsilon}$ is defined as the Gibbs kernel in Sinkhorn's algorithm. Block coordinate ascent can therefore be implemented in a closed form by applying successively the following updates, starting from any arbitrary $\mathbf{g}^{(1)}$, for $t \geq 1$:

$$\mathbf{f}^{(t+1)} = \varepsilon \log \mathbf{a} - \varepsilon \log \left(\mathbf{K} e^{\mathbf{g}^{(t)}/\varepsilon}\right); \tag{27}$$

$$\mathbf{g}^{(t+1)} = \varepsilon \log \mathbf{b} - \varepsilon \log \left(\mathbf{K}^\top e^{\mathbf{f}^{(t+1)}/\varepsilon}\right). \tag{28}$$

A.7. Experimental setting details

How is the ε selected. The ε controls a trade-off between exploration and exploitation. A large ε leads to a greater

Table 6. Training setting: For the SGD optimizer, in the parentheses are the momentum and weight decay. For ImageNet, batch size is 64 per GPU. We learn the soft mask using cosine learning rate schedule.

Datasets	Backbone	Speedup	Pruning ratio
CIFAR-10	ResNet-56	$--$	[0, p, p, p], $p \in \{0.5, 0.7, 0.9, 0.925, 0.95\}$
CIFAR-100	VGG-19	$--$	[0:0, 1-15:p], $p \in \{0.5, 0.6, 0.7, 0.8, 0.9\}$
ImageNet	ResNet-34	1.32×	[0, 0.50, 0.60, 0.40, 0]
ImageNet	ResNet-50	2.31×	[0, 0.60, 0.60, 0.60, 0.21]
ImageNet	ResNet-50	2.56×	[0, 0.74, 0.74, 0.60, 0.21]
ImageNet	ResNet-50	3.06×	[0, 0.68, 0.68, 0.68, 0.50]

exploration by a softer mask. We verify that soft masks converge to hard masks by training for few epochs and measuring the average of the squared differences between them for different ε candidates. As is common to hyperparameter tuning, this requires some effort. Therefore for similar architectures we used the same ε, *e.g.*, $\varepsilon = 1$ for lightweight ResNet-56 and MobileNetV2.

Training setting. In Table 5 we summarize the detailed training settings of this work. For easier comparison, we use same training settings as [63] in finetuning. In our soft mask learning phase, we use a cosine learning rate schedule for updating the importance scores and network weights with SGD.

Pruning ratio. We use same pruning ratio as in [63] for fair comparison. In Table 6 we give the specific pruning ratio used for our experiments in the paper. We here briefly explain how we set the pruning ratio. We use two different architectures: single-branch (*i.e.* VGG-19) and multi-branch (*i.e.* ResNet). *(i)* For VGG19, we use the following pruning ratio setting. As an example, "[0:0, 1-9:0.3, 10-15:0.5]" means "for the first layer (index starting from 0), the pruning ratio is 0; for layer 1 to 9, the pruning ratio is 0.3; for layer 10 to 15, the pruning ratio is 0.5". *(ii)* For a ResNet, if it has N stages, we will use a list of N floats to represent its pruning ratios for the N stages. For example, ResNet-56 has 4 stages in conv layers, then "[0, 0.7, 0.7, 0.7]" means "for the first stage (the first conv layer), the pruning ratio is set as 0; the other three stages have pruning ratio of 0.7". Besides, since we do not prune the last conv layer in a residual block, which means for a two-layer residual block (for ResNet-56), we only prune the first layer; for a three-layer bottleneck block (for ResNet-34 and ResNet-50), we only prune the first and second layers.

Unstructured pruning. For unstructured pruning, each individual weight needs to couple an importance score which may cause higher memory cost, therefore we follow previous works that use the magnitude of the weight parameter as an importance score.

A.8. Training recipe variants

Our primary goal was to compare pruning methods as fairly as possible. The reported results for ResNet-50 used the *baseline training recipe* from CReg for fair comparisons. Here, we performed additional analyses, pruning ResNet-50 on ImageNet with Label Smoothing, TrivialAugment, Random Erasing, Mixup, and Cutmix to form a *stronger training recipe*. We did not apply other effective techniques such as Long Training (*e.g.* 600 epochs), LR optimizations, EMA, Weight Decay tuning, and Inference Resize tuning, which may further improve by ∼3% accuracy as shown in PyTorch. We trained on a large batch size of 4,096 on 8 NVIDIA A100 GPUs, scaling up the learning rate by 5×. As expected, Table 7 shows increased pruning accuracy, showing that our method can be augmented with different training recipes.

Table 7. Training recipes for ResNet50 under structured pruning on ImageNet-1K. Our method is flexible for different training recipes.

Recipe	Speed Up	Acc.
Baseline	2.31×	75.54
Stonger		77.16

Table 8. FLOPs budgets for WideResNet-26 on CIFAR-100

Method	Budget (%)	Acc.
Unpruned	100	80.21
ChipNet [60]	40	76.88
Ours		78.63
ChipNet [60]	20	77.15
Ours		78.17

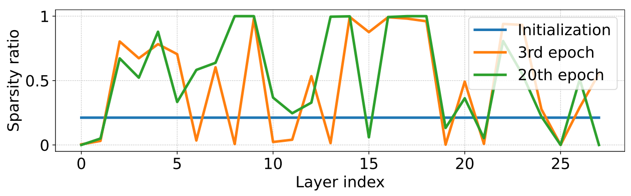

Figure 6. Learned sparsity ratios over layers under FLOPs budget.

A.9. Extension to FLOPs and latency budgets

The FLOPs or latency budget pruning can be viewed as a learnable differentiable sparsity allocation problem. We formulate it in the following.

Formulation. Given budget $B_{\mathcal{F}}$, the network weights as $\mathbf{w}$, the optimization problem of kept ratios $\mathcal{A} = \{\alpha^{(l)}\}_{l=1,\ldots,L}$ (*i.e.*, $1-$sparsity ratio) of L layers can be written as:

$$\begin{aligned} &\arg\min_{\mathbf{m},\mathbf{w},\mathcal{A}} \ \mathcal{L}_{\text{train}}\big(f(\mathbf{x};\mathbf{w},\mathbf{m},\mathcal{A})\big), \\ &\text{s.t.} \ \ \frac{1}{n}\sum\nolimits_{i=1}^{n}\mathbf{m}_i^{(l)} = \alpha^{(l)}, \ \ \mathbf{m}^{(l)} \in \{0,1\}^n, \\ &\qquad \mathcal{F}(\mathcal{A}) \leq B_{\mathcal{F}}, \ \ 0 \leq \mathcal{A} \leq 1. \end{aligned} \tag{29}$$

where $\mathcal{L}_{\text{train}}$ is training loss. $\mathcal{F}(\mathcal{A})$ is the consumed resource corresponding to the kept ratios. If $\mathcal{A}$ is predefined layer-wise or global kept ratio, the formulation degrades to our original formulation in Eq. (1). For FLOPs or latency budgets, we learn the kept ratios. Note $\mathcal{F}(\mathcal{A})$ can be denoted as a function of kept ratio, see [60] and [54]. By setting $\mathcal{A} = \text{Sigmoid}(\Theta)$ where Θ is a group of learnable parameters where $0 \leq \mathcal{A} \leq 1$ is satisfied naturally, we optimize Θ to learn kept ratio $\mathcal{A}$ by minimizng $\mathcal{L}_{\text{train}}$ and a budgets penalty loss $||\mathcal{F}(\mathcal{A}) - B_{\mathcal{F}}||^2$. Our optimal transport method dynamically aligns to learned ratios $\mathcal{A}$.

We add the experimental comparisons in Table 8 with FLOPs budget for WideResNet-26 for CIFAR-100.

FLOPs budget. We use the same way as [60] to calculate the FLOPs budget that assumes a sliding window is used to achieve convolution and the nonlinear computational overhead is ignored. We define FLOPs budget as:

$$\mathcal{F}(\mathcal{A}) = \frac{\sum_{j=1}^{\mathcal{N}}(K_j \cdot n_{j-1} \cdot \alpha^{(j-1)} + 1) \cdot n_j \cdot \alpha^{(j)} \cdot A_j}{\sum_{j=1}^{\mathcal{N}}(K_j \cdot n_{j-1} + 1) \cdot n_j \cdot A_j}, \tag{30}$$

where $\mathcal{N}$ denotes the number of convolutional layers in the network, K_j denotes area of the kernel, and A_j and n_j denote area of the feature maps and the channel count, respectively, in the j^{th} layer.

We note that the FLOPs budget is formulated as a function of kept ratios $\alpha^{(j)}$. Also the the latency cost of each layer can be approximated by the kept ratios of previous layer and current layer as shown in [24].