Spartan: Differentiable Sparsity via Regularized Transportation

Kai Sheng Tai
Meta AI

Taipeng Tian
Meta AI

Ser-Nam Lim
Meta AI

Abstract

We present Spartan, a method for training sparse neural network models with a predetermined level of sparsity. Spartan is based on a combination of two techniques: (1) soft top-k masking of low-magnitude parameters via a regularized optimal transportation problem and (2) dual averaging-based parameter updates with hard sparsification in the forward pass. This scheme realizes an exploration-exploitation tradeoff: early in training, the learner is able to explore various sparsity patterns, and as the soft top-k approximation is gradually sharpened over the course of training, the balance shifts towards parameter optimization with respect to a fixed sparsity mask. Spartan is sufficiently flexible to accommodate a variety of sparsity allocation policies, including both unstructured and block structured sparsity, as well as general cost-sensitive sparsity allocation mediated by linear models of per-parameter costs. On ImageNet-1K classification, Spartan yields 95% sparse ResNet-50 models and 90% block sparse ViT-B/16 models while incurring absolute top-1 accuracy losses of less than 1% compared to fully dense training.

1 Introduction

Sparse learning algorithms search for model parameters that minimize training loss while retaining only a small fraction of non-zero entries. Parameter sparsity yields benefits along several axes: reduced model storage costs, greater computational and energy efficiency during training and inference, and potentially improved model generalization [33]. However, sparse training is a challenging optimization problem: for the general problem of learning parameters $\theta \in \mathbb{R}^d$ subject to the constraint that θ is k-sparse, the feasible set is the union of $\binom{d}{k}$ axis-aligned hyperplanes intersecting at the origin, each of dimension k. This complex, nonconvex geometry of the feasible set compounds the existing difficulty of optimizing deep neural network models.

This work focuses on improving the generalization error of sparse neural network models. To this end, we introduce *Spartan*, or *Sparsity via Regularized Transportation*—a sparse learning algorithm that leverages an optimal transportation-based top-k masking operation to induce parameter sparsity during training. Spartan belongs to the family of "parameter dense" training algorithms that maintains a dense parameter vector $\theta \in \mathbb{R}^d$ throughout training [48; 24], in contrast to "parameter sparse" training algorithms that adhere to a $\tilde{\mathcal{O}}(k)$ memory budget for representing the parameters of a k-sparse model [3; 31; 32; 13; 16]. While computational cost and memory usage at training time are important considerations, Spartan primarily optimizes for performance at inference time.

Intuitively, Spartan aims to achieve a controlled transition between the *exploration* and *exploitation* of various sparsity patterns during training. In the exploration regime, the goal is for the learner to easily transition between differing sparsity patterns in order to escape those that correspond to poor minima of the loss function. On the other hand, the goal of exploitation is for the learner to optimize model parameters given a fixed sparsity pattern, or a small set of similar patterns. This latter setting

Correspondence to: Kai Sheng Tai (kst@meta.com)

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

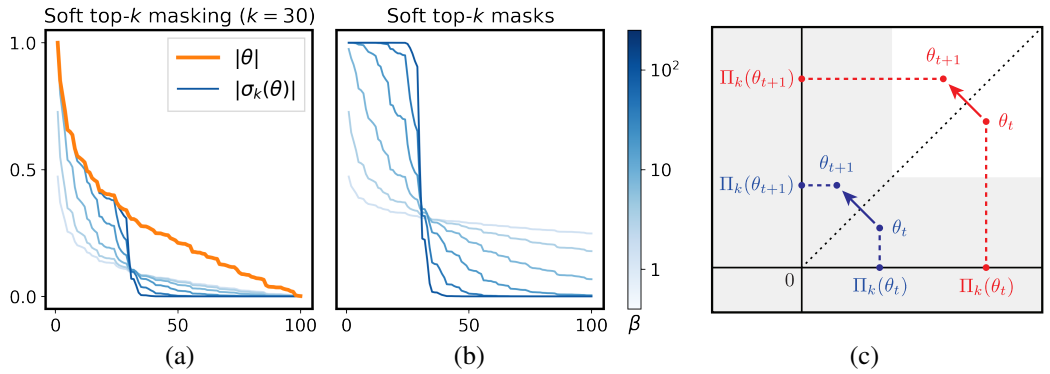

Figure 1: **Soft top-k masking.** (a, b) The soft top-k masking operator σ_k^β computes approximately k-sparse outputs, with the sharpness of the mask controlled by the parameter β. (c) Small updates to iterates far from the k-sparse feasible set ($\theta_t \to \theta_{t+1}$) can correspond to a large perturbation in parameter space after projection by Π_k. Updates to iterates θ_t in the approximately sparse region (shaded in grey) correspond to smaller post-projection perturbations.

avoids frequent oscillations between disparate sparsity patterns, which may be detrimental to the optimization process.

We operationalize this intuition through the use of a differentiable *soft top-k masking* operation $\sigma_k^\beta : \mathbb{R}^d \to \mathbb{R}^d$. This function maps parameters θ to an approximately sparse output that suppresses low-magnitude entries in θ. We parameterize the soft top-k mask with a *sharpness parameter* β: at $\beta = 0$, σ_k^β simply scales the input by a constant, and as $\beta \to \infty$, the mapping reduces to hard top-k magnitude-based selection (Figure 1 (a, b)). Soft top-k masking therefore constrains the iterates θ_t to be close to the set of exactly k-sparse vectors, with the strength of this constraint mediated by the parameter β. Figures 1 (c) and 2 give some geometeric intuition for the effect of this mapping. We implement soft top-k masking using a regularized optimal transportation formulation [11; 42] and demonstrate that this technique scales to networks on the order of 10^8 parameters in size.

We evaluate Spartan on ResNet-50 [21] and ViT [15] models trained on the ImageNet-1K dataset. On ResNet-50, we find that sparse models trained with Spartan achieve higher generalization accuracies than those trained with existing methods at sparsity levels of 90% and above. In particular, we train ResNet-50 models to 76.5% top-1 validation accuracy at 95% sparsity and to 74.2% accuracy at 97.5% sparsity, improving on the previous state-of-the-art by 0.6% and 1.6% respectively. Our sparse ViT-B/16 models reduce model storage size by $10\times$ and inference FLOPs by $7.4\times$ at the cost of a 0.6% accuracy reduction relative to DeiT-B [38]. We further demonstrate that Spartan is effective for block structured pruning, a form of structured sparsity that is more amenable to acceleration on current GPU hardware than unstructured sparsity [18]. On a ViT-B/16 model with 16×16 block structured pruning, Spartan achieves 79.1% top-1 accuracy at 90% sparsity, compared to the baseline of 74.1% at the same sparsity level.

To summarize, we make the following contributions in this paper:

- We present Spartan, a sparsity-inducing training algorithm based on a soft top-k masking operation. We show that Spartan interpolates between two existing sparse learning algorithms: iterative magnitude pruning [48] and Top-K Always Sparse Training [24].
- We empirically evaluate Spartan using ResNet-50 and ViT models trained on the ImageNet-1K dataset, demonstrating consistent improvements over the prior state-of-the-art.
- We study the effect of Spartan's hyperparameters on an exploration-exploitation tradeoff during training and on the final accuracy of the trained models.

We provide an open source implementation of Spartan at `https://github.com/facebookresearch/spartan`.

Notation. We use $\mathbb{R}_+$ and $\mathbb{R}_{++}$ to denote the nonnegative real numbers and the strictly positive real numbers respectively. $\mathbb{1}_d$ is the all-ones vector of dimension d. Given a vector $x \in \mathbb{R}^d$, $\|x\|_p$ denotes

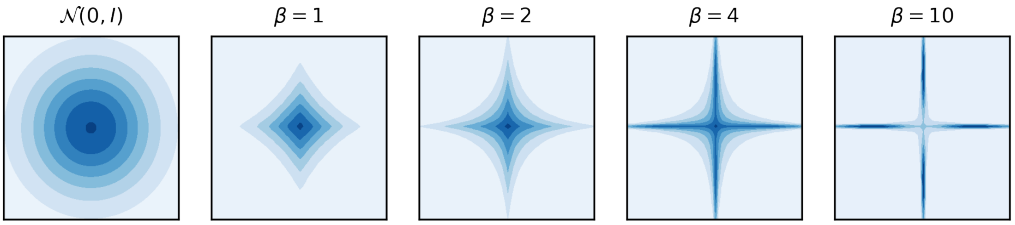

Figure 2: **A 2D example of soft masking.** We plot probability densities corresponding to the action of the soft top-1 operator σ_1^β on the 2D standard Gaussian distribution (darker colors indicate higher density). Specifically, we visualize the densities of $\sigma_1^\beta(x) \mid x \sim \mathcal{N}(0, I_2)$ for a range of sharpness parameters β. Higher values of β constrain iterates to be closer to the k-sparse feasible set.

the p-norm of x, $\|x\|_0$ is the number of nonzero entries in x, and $|x|$ is the vector of elementwise absolute values. When x and y are vectors, $x \odot y$ denotes elementwise multiplication and x/y denotes elementwise division. The operator $\Pi_k(x) := \arg\min_{y:\|y\|_0 \leq k} \|x - y\|_2$ denotes Euclidean projection onto the set of k-sparse vectors.

2 Related Work

Neural network pruning. Our use of a magnitude-based criterion for sparse learning draws on early work in the area of neural network pruning [23]. Magnitude-based pruning is computationally cheap relative to alternative criteria that rely on first- or second-order information [33; 27; 20], and is a perhaps surprisingly performant option despite its simplicity [17]. More generally, Spartan builds on a substantial body of previous work that aims to jointly optimize sparsity patterns and model parameters for deep neural networks [43; 8; 19; 29; 28; 48, *inter alia*; see, *e.g.*, 22 for a survey].

Spartan as a generalization of existing methods. We highlight two particularly relevant sparse training methods in the literature: the *iterative magnitude pruning* (IMP) method (Algorithm 1, [48]), and *Top-K Always Sparse Training*, or Top-KAST (Algorithm 2, [24]). At the extreme points of the sharpness parameter β, Spartan's parameter update procedure reduces to those of IMP and Top-KAST. We can thus view Spartan as a method that generalizes and smoothly interpolates between this pair of algorithms. As $\beta \to \infty$, Spartan is equivalent to IMP: this approach sparsifies parameters using a top-k magnitude-based binary mask, and entries that were masked in the forward pass receive zero gradient in the backward pass. At $\beta = 0$, Spartan reduces to Top-KAST: this method again sparsifies parameters by magnitude with a binary mask, but unlike IMP, all entries are updated in the backward pass using the gradient of the loss with respect to the *masked* parameters. The Top-KAST update is thus an application of the *straight-through gradient method* [5; 9], otherwise known as *lazy projection* or *dual averaging* (DA) in optimization [34; 41; 2; 14]. See Sec. 5 for further discussion of this connection.

Algorithm 1 Iterative magnitude pruning update

Input: parameters θ_t, loss function $L(\theta)$, sparsity budget k, step size η_t
Output: parameters θ_{t+1}
1: $\tilde{\theta}_t = \Pi_k(\theta_t)$
2: $\theta_{t+1} = \theta_t - \eta_t \nabla \tilde{\theta}_t \nabla L(\tilde{\theta}_t)$
3: **return** θ_{t+1}

Algorithm 2 Dual averaging / Top-KAST update

Input: parameters θ_t, loss function $L(\theta)$, sparsity budget k, step size η_t
Output: parameters θ_{t+1}
1: $\tilde{\theta}_t = \Pi_k(\theta_t)$
2: $\theta_{t+1} = \theta_t - \eta_t \nabla L(\tilde{\theta}_t)$
3: **return** θ_{t+1}

Note that both Algorithms 1 and 2 compute the forward pass through the network using the sparsified parameters $\tilde{\theta}$, obtained by setting all but the top-k entries of θ by magnitude to zero.

Smooth approximations. Our use of the soft top-k operation is related to prior methods that use the logistic sigmoid function as a differentiable approximation to the step function for sparse training [30; 36; 1]. These approaches similarly regulate the sharpness of the approximation with a temperature parameter that scales the input logits to the sigmoid function. A distinctive characteristic

Algorithm 3 Spartan parameter update

Input: parameters θ_t, loss function $L(\theta)$, sparsity budget k, sharpness parameter β, step size η_t
Output: parameters θ_{t+1}

1: $\sigma_k^\beta(\theta_t) = \theta_t \odot \mathrm{softtopk}(|\theta_t|, k, \beta)$ ▷ *apply soft masking*
2: $\tilde{\theta}_t = \Pi_k\left(\sigma_k^\beta(\theta_t)\right)$ ▷ *project onto k-sparse set*
3: $\theta_{t+1} = \theta_t - \eta_t \nabla \sigma_k^\beta(\theta_t) \nabla L(\tilde{\theta}_t)$ ▷ *compute dual averaging update*
4: **return** θ_{t+1}

of Spartan is that the parameter k directly controls the degree of sparsity of the mask; this differs from previous approaches involving the logistic sigmoid approximation that only indirectly control the degree of sparsity using an L_1 penalty term.

Transformer-specific methods. Several approaches are specialized to transformer-based architectures. These include structured pruning of pre-trained transformers for NLP tasks [39; 26; 40] and sparse training methods specifically applied to vision transformers [7; 6]. In contrast, Spartan is a general-purpose sparse training algorithm that is designed to be agnostic to the model architecture.

3 Sparsity via Regularized Transportation

We begin by motivating our proposed approach. A key advantage of the dual averaging method (Algorithm 2) over IMP (Algorithm 1) is that it mitigates the issue of gradient sparsity. In the IMP update, only those parameters that were not masked in the forward pass receive a nonzero gradient, resulting in slow progress at high levels of sparsity. In contrast, dual averaging applies a dense update to the parameter vector θ in each iteration—intuitively, these dense updates can help "revive" parameters that are below the top-k magnitude threshold and are consequently masked to zero. Dual averaging iterates are thus able to more quickly explore the space of possible sparsity patterns.

On the other hand, large variations in the sparsity patterns realized post-projection can lead to instability in optimization, thus hampering the final performance of the model. For instance, Top-KAST empirically benefits from the application of additional sparsity-inducing regularization on the parameters [24 (Sec. 2.3), 37]. This issue motivates our use of a soft top-k approximation as a mechanism for controlling the stability of our training iterates.

Each iteration of Spartan consists of the following two steps (Algorithm 3): (1) an approximately k-sparse masking of the model parameters using the soft top-k operator, and (2) dual averaging-based parameter updates with the set of exactly k-sparse vectors as the feasible set. This procedure aims to combine the advantages of Algorithms 1 and 2 within a single parameter update scheme. Note that the Spartan parameter update is essentially identical to Algorithm 2, save for the inclusion of the intermediate soft top-k masking step.

In the following, we begin by describing the soft top-k masking operation and address the issues of scalability and of applying soft top-k masking to structured sparsity. We subsequently discuss how the update procedure outlined in Algorithm 3 can be incorporated within a complete training loop.

3.1 Soft Top-k Masking

Our soft top-k masking scheme is based on the soft top-k operator $\mathrm{softtopk}(v, k, \beta)$ described by Xie et al. [42]. $\mathrm{softtopk}$ takes as input a vector $v \in \mathbb{R}^d$, a budget parameter $k > 0$ and a sharpness parameter $\beta \geq 0$, and outputs $m \in [0, 1]^d$, a smoothed version of the top-k indicator vector. By using the magnitudes $|\theta|$ of the parameter vector as a measure of the "value" of each entry, we obtain a soft top-k magnitude pruning operator $\sigma_k^\beta(\cdot)$ by masking the entries of the parameter vector with the output of the soft top-k operator:

$$\sigma_k^\beta(\theta) \coloneqq \theta \odot \mathrm{softtopk}(|\theta|, k, \beta).$$

We introduce a mild generalization of the soft top-k operator from Xie et al. [42] by incorporating a strictly positive *cost vector* c. In particular, we require that the output mask $m \in [0, 1]^d$ satisfies the budget constraint $c^T m = k$. This abstraction is useful for modeling several notions of heterogeneous

Algorithm 4 Soft top-k forward pass

Input: values $v \in \mathbb{R}^d$, costs $c \in \mathbb{R}^d_{++}$, budget $k \in (0, \mathbb{1}_d^T c]$, sharpness parameter $\beta \geq 0$, max. iterations T, tolerance ϵ, initial dual variable μ_0
Output: mask $m \in [0,1]^d$, dual variables $\mu \in \mathbb{R}$, $\nu \in \mathbb{R}^d$

$z = \beta v / c, \quad m_0 = \mathbb{1}_d$ ▷ *normalize values v by costs c*
for $t = 1, \ldots, T$ **do**
  $\nu_t = \log c - \log(\mathbb{1}_d + \exp(z + \mu_{t-1}))$ ▷ *normalize mask entries*
  $\mu_t = \log k - \log \sum_i \exp(z_i + \nu_{t,i})$ ▷ *normalize sum to be equal to k*
  $m_t = \exp(z + \mu_t + \nu_t - \log c)$ ▷ *compute mask*
  if $|v^T(m_t - m_{t-1})| < \epsilon |v^T m_{t-1}|$ **then** ▷ *check stopping criterion*
    return m_t, μ_t, ν_t
return m_T, μ_T, ν_T

Algorithm 5 Soft top-k backward pass

Input: gradient w.r.t. outputs $g \in \mathbb{R}^d$, mask $m \in [0,1]^d$, costs c, budget k, sharpness parameter β
Output: gradient w.r.t. inputs

$a_1 = \sum_i g_i m_i (1 - m_i), \quad a_2 = \sum_i c_i m_i^2$
return $\beta m \odot (\mathbb{1}_d - m) \odot (g/c - a_1/(k - a_2))$

parameter costs within a model; for instance, parameters that are repeatedly invoked within a network have a higher associated computational cost. As a concrete example, Evci et al. [16] observed that the FLOP count of a ResNet-50 model at a fixed global sparsity can vary by a factor of over $2\times$ due to differences in the individual sparsities of convolutional layers with differing output dimensions.

To derive this cost-sensitive soft top-k operator, we begin by expressing the mask m as the solution to the following linear program (LP):

$$\underset{m \in \mathbb{R}^d}{\text{maximize}} \quad v^T m \quad \text{subject to} \quad 0 \preceq m \preceq \mathbb{1}_d, \;\; c^T m = k. \tag{1}$$

Deferring the derivation to the Appendix, we can rewrite this LP as the following optimal transportation (OT) problem:

$$\underset{Y \in \mathbb{R}_+^{d \times 2}}{\text{minimize}} \quad \sum_{ij} C_{ij} Y_{ij} \quad \text{subject to} \quad Y \mathbb{1}_2 = c, \; \mathbb{1}_d Y = [k, \, \mathbb{1}_d^T c - k], \tag{2}$$

with the cost matrix $C \coloneqq [-v/c, \, 0]$. We can recover the mask from Y as $m_i = Y_{i1}/c_i$. Now following [42; 11], we introduce entropic regularization to obtain a smoothed version of the top-k operator:

$$\begin{aligned} \underset{Y \in \mathbb{R}_+^{d \times 2}}{\text{minimize}} \quad & \sum_{ij} C_{ij} Y_{ij} - \frac{1}{\beta} H(Y) \\ \text{subject to} \quad & Y \mathbb{1}_2 = c, \; \mathbb{1}_d Y = [k, \, \mathbb{1}_d^T c - k], \end{aligned} \tag{3}$$

where $H(Y) \coloneqq -\sum_{ij} Y_{ij}(\log Y_{ij} - 1)$. In this standard form, we can efficiently solve this regularized OT problem using the Sinkhorn-Knopp algorithm [11; 4; 12].

Algorithms 4 and 5 describe the forward and backward passes of the resulting cost-sensitive soft top-k operator (see Appendices B and C for their derivations). The expression for the gradient in Algorithm 5 follows from Theorem 3 in Xie et al. [42] with some algebraic simplification. We remark that this closed-form backward pass is approximate in the sense that it assumes that we obtain the *optimal* dual variables μ, ν in the forward pass; in practice, we do not encounter issues when using estimates of the dual variables obtained with tolerance $\epsilon = 0.01$ and a maximum iteration count of $T = 100$ in Algorithm 4.

Scalability. Since we apply soft top-k masking to high-dimensional parameters in each iteration of training, it is important to minimize the computational overhead of Sinkhorn iteration in the forward pass. We observe that in order to compute the hard top-k projection step in dual averaging (as in Algorithm 3), it is necessary to find the index i_k of the kth largest entry of $|\theta|$. By using the

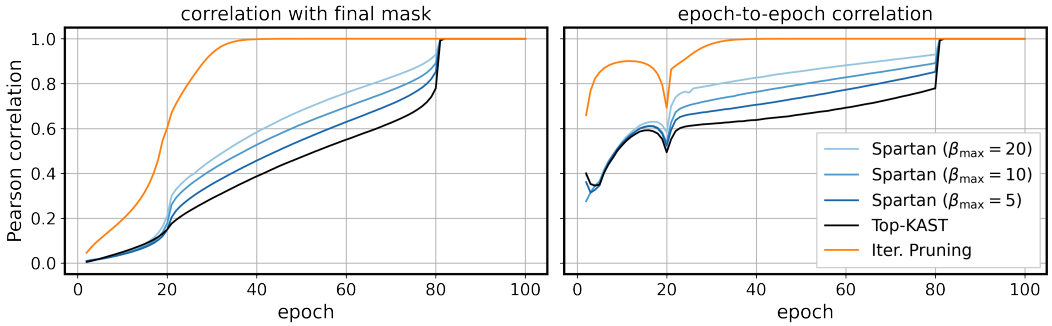

Figure 3: **Spartan parameterizes an exploration-exploitation tradeoff.** For each run of ResNet-50 training on ImageNet-1K, we plot Pearson correlation coefficients between the sparsity mask at the end of each epoch and the mask obtained at the end of training (left), and between sparsity masks at the end of subsequent epochs (right). Kinks in the correlation curves at epochs 20 and 80 are respectively due to the end of sparsity annealing and the start of fine-tuning with fixed masks (see Section 3.2 for details on the training schedule).

heuristic initialization $\mu_0 = -\beta|\theta_{i_k}|/c_{i_k}$ for the dual variable μ, we can accelerate the convergence of Sinkhorn iteration. This heuristic is based on the fact that $|\theta_{i_k}|/c_{i_k}$ is the threshold value of the normalized value vector $|\theta|/c$ in *hard* top-k masking. Concretely, we demonstrate in Sec. 4.3 that our implementation of Spartan incurs a per-iteration runtime overhead of approximately 5% over standard dense ResNet-50 training.

Structured sparsity. To implement block structured sparsity, we instantiate one mask variable per block and mask all parameters within that block with the same mask variable. To compute the mask, we use the sum of the magnitudes of the entries in each block as the corresponding value v_i. In a standard pruning setting with uniform costs across all parameters, the corresponding cost c_i is simply the total number of entries in the block.

3.2 Training with Spartan

Training schedule. When training with Spartan, we divide the training process into a warmup phase, an intermediate phase, and a fine-tuning phase. In our experiments, the warmup phase consists of the first 20% of epochs, the intermediate phase the next 60%, and the fine-tuning phase the final 20%. In the warmup phase, we linearly anneal the global sparsity of the model from a value of 1 at the start of training to the target level of sparsity. Throughout the intermediate phase, we maintain the target sparsity level, and during the fine-tuning phase, we fix the sparsity mask to that used at the start of fine-tuning. This training schedule is similar to those used in prior work [37].

Exploration vs. exploitation as a function of β. From the start of training up to the start of the fine-tuning phase, we linearly ramp the sharpness parameter β from an initial value of 1 to the final value $\beta_{\max}$. We interpret the spectrum of updates parameterized by β as being characteristic of an exploration-exploitation tradeoff. In Figure 3, we illustrate this phenomenon by plotting Pearson correlation coefficients between sparsity masks at different stages of training. We observe that iterative magnitude pruning converges on the final mask relatively early in training, which is indicative of insufficient exploration of different sparsity patterns—consequently, this model achieves lower validation accuracy than the remaining models. The three models trained with Spartan each ramp β from 1 at the start of training to $\beta_{\max}$ at epoch 80. The value of $\beta_{\max}$ correlates well both with the Pearson correlation with the final mask, and with the Pearson correlation between masks at the end of subsequent epochs. Spartan thus interpolates between the high-exploration regime of standard dual averaging and the low-exploration regime of iterative magnitude pruning. In this example, the intermediate setting $\beta_{\max} = 10$ achieves the highest top-1 validation accuracy of 74.6%, with Top-KAST at 73.5% and IMP at 68.2%.

Table 1: Top-1 accuracies on ImageNet-1K validation set with fully dense training.

Method	Epochs	Accuracy (%)
ResNet-50	100	77.08 ±0.11
	200	**77.46** ±0.09
	400	77.17 ±0.04

Method	Epochs	Accuracy (%)
ViT-B/16	100	77.78 ±0.22
	300	**80.06** ±0.11

Table 2: ResNet-50 top-1 accuracies on ImageNet-1K validation set at varying levels of sparsity and epochs of training. When available, we report means and standard deviations over 3 trials.

			Sparsity				
	Method	Epochs	80%	90%	95%	97.5%	99%
[16]	RigL (ERK)	100	75.1 ±0.05	73.0 ±0.04	69.7 ±0.17	-	-
		500	77.1 ±0.06	76.4 ±0.05	74.5 ±0.09	-	-
[25]	STR†	100	76.19	74.31	70.40	62.84	51.82
[47]	ProbMask	100	-	74.68	71.50	66.83‡	61.07
[46]	OptG	100	-	74.28	72.38	-	62.10
[37]	IMP	100	75.3 ±0.07	73.7 ±0.14	70.6 ±0.05	-	-
	Top-KAST	100	76.08 ±0.02	75.13 ±0.03	73.19 ±0.02	-	-
	(with PP)	100	76.24 ±0.07	75.23 ±0.02	73.25 ±0.02	-	-
	(with ERK)	100	76.42 ±0.03	75.51 ±0.05	-	-	-
	(with PP & ERK)	100	76.76 ±0.08	75.74 ±0.08	-	-	-
		200	77.51 ±0.03	76.94 ±0.10	-	-	-
		300	77.64 ±0.05	77.16 ±0.19	-	-	-
	Top-KAST	100	76.66 ±0.04	75.48 ±0.15	73.51 ±0.16	70.23 ±0.05	-
		200	77.55 ±0.08	76.84 ±0.11	75.20 ±0.11	71.72 ±0.06	-
		400	**77.75** ±0.09	77.37 ±0.07	75.90 ±0.04	72.58 ±0.13	-
	Spartan	100	**76.89** ±0.09	**76.17** ±0.10	**74.68** ±0.24	**71.95** ±0.11	**63.87** ±0.12
		200	**77.56** ±0.19	**77.06** ±0.13	**75.92** ±0.01	**73.41** ±0.15	65.77 ±0.03
		400	77.57 ±0.08	**77.40** ±0.06	**76.48** ±0.20	**74.15** ±0.10	66.80 ±0.03

†Reported accuracies for models closest to the listed sparsity level. ‡Model trained at 98% sparsity.

4 Empirical Evaluation

In this section, we report the results of our sparse training experiments on the ImageNet-1K dataset with two standard architectures: ResNet-50 [21] and ViT-B/16 [15], consisting of 25.6M and 86.4M parameters respectively. On ResNet-50, we evaluate only unstructured pruning, whereas on ViT-B/16, we evaluate both unstructured and block structured pruning. We subsequently present empirical studies of the sensitivity of Spartan to the value of β, the effect of running Spartan without the dual averaging step, and the computational overhead of soft top-k masking.

4.1 ImageNet-1K Classification

In all our experiments, we run Spartan with the training schedule described in Section 3.2. We train and evaluate our models on the ImageNet-1K dataset with the standard training-validation split and report means and standard deviations over 3 independent trials. We provide additional details on our experimental setup in the Appendix.

ResNet-50 experimental setup. For consistency with our baselines, we use standard data augmentation with horizontal flips and random crops at 224×224 resolution. For all Spartan runs, we use $\beta_{\max} = 10$, which we selected based on models trained at 95% accuracy. We sparsify the parameters of all linear and convolutional layers with a global sparsity budget, excluding bias parameters and the parameters of batch normalization layers. Our baselines are iterative magnitude pruning [48], RigL with the Erdos-Renyi-Kernel (ERK) sparsity distribution [16], Soft Threshold Weight Reparameterization (STR) [25], probabilistic masking (ProbMask) [47], OptG [46], and Top-KAST with Powerpropagation and ERK [24; 37]. We additionally re-run the most performant baseline method,

Table 3: ViT-B/16 top-1 accuracies on ImageNet-1K validation set at 90% sparsity with unstructured sparsity and block structured pruning of attention layers.

		Sparsity Structure		
Method	**Epochs**	**Unstructured**	**16×16 blocks**	**32×32 blocks**
Top-KAST	100	78.05 ± 0.07	67.69 ± 0.23	64.79 ± 0.14
	300	80.86 ± 0.03	74.11 ± 0.20	70.67 ± 0.96
Spartan	100	**79.88** ± 0.16	**75.62** ± 0.07	**74.50** ± 0.23
	300	**81.18** ± 0.04	**79.12** ± 0.16	**78.43** ± 0.10

Top-KAST, using a reimplementation in our own codebase. For Top-KAST, we exclude the first convolutional layer from pruning (following [37; 16]) and we use fully dense backward passes (*i.e.*, with a backward sparsity of 0), since this setting achieved the highest accuracy in prior work [24]. We use mixed precision training with a batch size of 4096 on 8 NVIDIA A100 GPUs.

ViT experimental setup. We use the ViT-B architecture with 16×16 patches at 224×224 input resolution. We augment the training data using RandAugment [10], MixUp [45] and CutMix [44]. Our ViT models are trained from random initialization, without any pretraining. We set $\beta_{\max} = 20$ for Spartan with unstructured sparsity, and $\beta_{\max} = 320$ and $\beta_{\max} = 640$ for Spartan with 16×16 and 32×32 blocks respectively. These latter settings are the values of $\beta_{\max}$ for the unstructured case scaled up by factors of 16 and 32—since each block averages the magnitudes of B^2 entries, we expect the variance of the block values to correspondingly decrease by a factor of B^2, and we thus compensate by scaling up β by B. In the block structured case, we exclude the input convolutional layer and the output classification head from pruning since their parameter dimensions are not divisible by the block size. We use mixed precision training with a batch size of 4096 on 16 NVIDIA A100 GPUs across 2 nodes.

Results. Table 1 lists the top-1 validation accuracies achieved by fully dense ResNet-50 and ViT-B/16 models. Table 2 reports validation accuracies for ResNet-50 models at 80%, 90%, 95%, 97.5% and 99% sparsity, and Table 3 reports validation accuracies for ViT at 90% sparsity. In the Appendix, we report additional measurements of inference FLOP costs for our sparse models and the results of experiments with FLOP-sensitive pruning.

For ResNet-50 models, we find that Spartan outperforms all our baselines across all training durations at sparsity levels of 90% and above. In particular, Spartan achieves a mean top-1 accuracy within 1% of fully dense training at 95% sparsity. We observe that additional epochs of sparse training consistently improves the final generalization accuracy; in contrast, validation accuracy peaks at 200 epochs for dense training. This trend persists at 800 training epochs, where Spartan achieves 67.18 ± 0.13 top-1 accuracy at 99% sparsity. For the Top-KAST baseline, we omit results at 99% sparsity due to training instability. We note that the accuracy improvements in the Spartan-trained ResNet-50 models relative to Top-KAST are not a result of increased FLOP counts at a given level of sparsity, as can be seen from the FLOP measurements reported in Appendix E.

For ViT-B/16 models, Spartan outperforms Top-KAST for both unstructured and block structured pruning. We observe a particularly large improvement over Top-KAST in the block structured case, where Spartan improves absolute validation accuracy by 7.8% for 32×32 block structured pruning. For unstructured pruning, Spartan achieves comparable accuracy to SViTE [7] (81.2% for Spartan vs. 81.3% for SViTE), but with 30% higher sparsity (90% for Spartan vs. 60% for SViTE). In exchange for a 0.6% reduction in accuracy relative to DeiT-B [38], Spartan reduces model storage cost by $10\times$ and the FLOP cost of inference by $7.4\times$.

4.2 Sensitivity and Ablation Analysis

Figure 4 (left) shows the effect of ablating the dual averaging step in Spartan—*i.e.*, omitting hard top-k projection in the forward pass—over a range of settings of $\beta_{\max}$ for 95% sparse ResNet-50 models trained for 100 epochs. The dashed line shows top-1 accuracy with Top-KAST. For Spartan training without dual averaging, we compute a hard top-k mask at the end of epoch 80, and as with standard Spartan training, we fix this mask until the end of training. In the absence of top-k projection, we find that accuracy increases with increasing $\beta_{\max}$ up to $\beta_{\max} = 80$; at lower settings of $\beta_{\max}$,

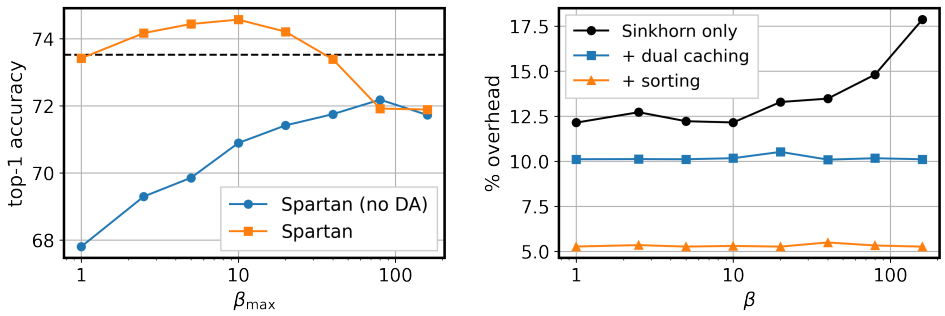

Figure 4: **Effect of β and dual averaging (left).** Top-1 ImageNet-1K validation accuracies for Spartan with and without dual averaging (DA) and with varying $\beta_{\max}$ at 95% sparsity. **Computational overhead (right).** Percentage increase in per-iteration wall clock runtime over dense training for Spartan with standard Sinkhorn iteration, Sinkhorn with dual caching, and Sinkhorn with sorting.

the higher approximation error of soft masking is detrimental to the final accuracy of the model. In contrast, the use of top-k projection in the forward pass mitigates this mismatch between training and inference and improves the final accuracy of the sparse model.

4.3 Computational Overhead

We evaluate the computational overhead of Spartan over standard dense training by measuring wall clock training times for a ResNet-50 model (Figure 4, right). Our benchmark measures runtime on a single NVIDIA A100 GPU over 50 iterations of training with a batch size of 256. We use random inputs and labels to avoid incurring data movement costs. We compare three approaches: standard Sinkhorn iteration, Sinkhorn iteration with the dual variable μ initialized using its final value from the previous iteration ("dual caching"), and Sinkhorn iteration with μ initialized using the value $-\beta|\theta_{i_k}|/c_{i_k}$, where we compute the index i_k of the kth largest entry of the normalized value vector $|\theta|/c$ by sorting its entries in each iteration ("sorting").

Standard Sinkhorn iteration incurs higher overheads as β increases—this is due to the additional iterations required to reach convergence as the regularized OT problem more closely approximates the original LP. We find that dual caching and sorting both prevent this growth in runtime over the range of β values that we tested. In our remaining experiments, we use the simple sorting-based approach since it corresponds to the lowest relative overhead over standard dense training (approximately 5%). We note that this relative overhead decreases as the batch size increases since the cost of computing the mask is independent of the batch size.

5 Discussion

Extensions. As an alternative to magnitude-based pruning, we may also apply Spartan in conjunction with *learned value parameters*, as in methods such as Movement Pruning [35]. In this approach, we would compute masks using a set of auxiliary parameters $\phi \in \mathbb{R}^d$ instead of the magnitudes $|\theta|$: $\sigma_k^\beta(\theta; \phi) = \theta \odot \mathrm{softtopk}(\phi, k, \beta)$, and similarly for hard projection. We remark that while this variant requires additional memory during training to store the value parameters, there is no additional cost during inference since the sparsity mask is fixed at the end of training.

Limitations. Since Spartan retains a dense parameter vector and computes dense backward passes during training, it incurs higher memory and computational costs in each iteration than sparse-to-sparse methods like RigL [16]. Nevertheless, we note that in terms of *total* computational or energy cost over a full training run, Spartan may remain a competitive option as it requires fewer iterations of training to reach a given accuracy threshold relative to sparse-to-sparse methods. However, we do not compare total training FLOP costs in our empirical evaluation.

A further limitation is that cost-sensitive pruning with Spartan is only compatible with relatively crude linear cost models. This restriction arises due to the requirements of the regularized OT formulation used to compute the soft top-k mask. In particular, this limitation precludes the use of cost models involving interaction terms such as those arising from spatially coherent sparsity patterns.

For example, the cost models that we consider here cannot encode the notion that coherently pruning an entire row or column of a parameter matrix will yield more cost savings than pruning random entries of the matrix.

Optimization with Dual Averaging. As discussed in Section 3, Spartan and Top-KAST are both applications of the dual averaging method [34; 41]. The empirically observed effectiveness of dual averaging in sparse training is reminiscent of its success in a related domain where continuous optimization is complicated by discrete constraints: neural network quantization. In this area, the BinaryConnect algorithm [9], which trains binary neural networks with parameters in $\{\pm 1\}^d$, is considered to be a foundational method. As observed by Bai et al. [2], BinaryConnect is itself a special case of dual averaging, and the theoretical underpinnings of this connection to dual averaging were further analyzed by Dockhorn et al. [14]. These parallels with neural network quantization suggest that similar ideas may well apply towards deepening our understanding of sparse learning.

Societal Impacts. Inference with deep neural network models is a computationally intensive process. At present, the total energy footprint associated with serving these models in production is expected to continue growing in tandem with the rising prevalence of large transformer-based architectures in vision and NLP applications. Research towards improving the energy efficiency of deep neural networks is therefore an important counterbalancing force against increasing resource usage by these models. The development of sparse learning algorithms is particularly relevant to these efforts, and we expect that the impact of these approaches will further increase as sparsity-aware hardware acceleration becomes more widely available.

6 Conclusions & Future Work

In this work, we describe a sparse learning algorithm that interpolates between two parameter update schemes: standard stochastic gradient updates with hard masking and the dual averaging method. We show that there exists an intermediate regime between these two methods that yields improved generalization accuracy for sparse convolutional and transformer models, particularly at higher levels of sparsity. While we have demonstrated promising empirical results with our proposed method, the learning dynamics of stochastic optimization for deep networks under sparsity constraints remains relatively poorly understood from a theoretical standpoint. There thus exists ample opportunity for further work towards better understanding sparse learning algorithms, which may in turn inspire future algorithmic advances in this area.

Acknowledgements and Disclosure of Funding

We are grateful to Trevor Gale for his feedback on this work. We also thank our anonymous reviewers for their valuable suggestions on improving the manuscript. This work was funded by Meta.

References

[1] Kambiz Azarian, Yash Bhalgat, Jinwon Lee, and Tijmen Blankevoort. Learned threshold pruning. *arXiv preprint arXiv:2003.00075*, 2020.

[2] Yu Bai, Yu-Xiang Wang, and Edo Liberty. ProxQuant: Quantized Neural Networks via Proximal Operators. In *International Conference on Learning Representations*, 2018.

[3] Guillaume Bellec, David Kappel, Wolfgang Maass, and Robert Legenstein. Deep Rewiring: Training very sparse deep networks. In *International Conference on Learning Representations*, 2018.

[4] Jean-David Benamou, Guillaume Carlier, Marco Cuturi, Luca Nenna, and Gabriel Peyré. Iterative Bregman projections for regularized transportation problems. *SIAM Journal on Scientific Computing*, 37(2), 2015.

[5] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

[6] Arnav Chavan, Zhiqiang Shen, Zhuang Liu, Zechun Liu, Kwang-Ting Cheng, and Eric Xing. Vision transformer slimming: Multi-dimension searching in continuous optimization space. *arXiv preprint arXiv:2201.00814*, 2022.

[7] Tianlong Chen, Yu Cheng, Zhe Gan, Lu Yuan, Lei Zhang, and Zhangyang Wang. Chasing sparsity in vision transformers: An end-to-end exploration. In *Advances in Neural Information Processing Systems*, 2021.

[8] Maxwell D Collins and Pushmeet Kohli. Memory bounded deep convolutional networks. *arXiv preprint arXiv:1412.1442*, 2014.

[9] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. BinaryConnect: Training deep neural networks with binary weights during propagations. In *Advances in Neural Information Processing Systems*, 2015.

[10] Ekin D Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V Le. Randaugment: Practical automated data augmentation with a reduced search space. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2020.

[11] Marco Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. In *Advances in Neural Information Processing Systems*, 2013.

[12] Marco Cuturi, Olivier Teboul, and Jean-Philippe Vert. Differentiable ranking and sorting using optimal transport. In *Advances in Neural Information Processing Systems*, 2019.

[13] Tim Dettmers and Luke Zettlemoyer. Sparse networks from scratch: Faster training without losing performance. *arXiv preprint arXiv:1907.04840*, 2019.

[14] Tim Dockhorn, Yaoliang Yu, Eyyüb Sari, Mahdi Zolnouri, and Vahid Partovi Nia. Demystifying and generalizing binaryconnect. 2021.

[15] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.

[16] Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. Rigging the lottery: Making all tickets winners. In *International Conference on Machine Learning*, 2020.

[17] Trevor Gale, Erich Elsen, and Sara Hooker. The state of sparsity in deep neural networks. *arXiv e-prints*, arXiv:1902.09574, 2019.

[18] Scott Gray, Alec Radford, and Diederik P Kingma. GPU kernels for block-sparse weights. *arXiv preprint arXiv:1711.09224*, 2017.

[19] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. *Advances in Neural Information Processing Systems*, 2015.

[20] Babak Hassibi and David Stork. Second order derivatives for network pruning: Optimal brain surgeon. *Advances in Neural Information Processing Systems*, 1992.

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[22] Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *Journal of Machine Learning Research*, 2021.

[23] Steven A Janowsky. Pruning versus clipping in neural networks. *Physical Review A*, 1989.

[24] Siddhant Jayakumar, Razvan Pascanu, Jack Rae, Simon Osindero, and Erich Elsen. Top-KAST: Top-K Always Sparse Training. *Advances in Neural Information Processing Systems*, 2020.

[25] Aditya Kusupati, Vivek Ramanujan, Raghav Somani, Mitchell Wortsman, Prateek Jain, Sham Kakade, and Ali Farhadi. Soft threshold weight reparameterization for learnable sparsity. In *International Conference on Machine Learning*, 2020.

[26] François Lagunas, Ella Charlaix, Victor Sanh, and Alexander M Rush. Block pruning for faster transformers. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, 2021.

[27] Yann LeCun, John Denker, and Sara Solla. Optimal brain damage. *Advances in Neural Information Processing Systems*, 2, 1989.

[28] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip Torr. SNIP: Single-shot Network Pruning based on Connection Sensitivity. In *International Conference on Learning Representations*, 2018.

[29] C Louizos, M Welling, and DP Kingma. Learning sparse neural networks through L_0 regularization. In *International Conference on Learning Representations*, 2018.

[30] Jian-Hao Luo and Jianxin Wu. Autopruner: An end-to-end trainable filter pruning method for efficient deep model inference. *Pattern Recognition*, 2020.

[31] Decebal Constantin Mocanu, Elena Mocanu, Peter Stone, Phuong H Nguyen, Madeleine Gibescu, and Antonio Liotta. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature Communications*, 2018.

[32] Hesham Mostafa and Xin Wang. Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization. In *International Conference on Machine Learning*, 2019.

[33] Michael C Mozer and Paul Smolensky. Skeletonization: A technique for trimming the fat from a network via relevance assessment. In *Advances in Neural Information Processing Systems*, 1988.

[34] Yurii Nesterov. Primal-dual subgradient methods for convex problems. *Mathematical Programming*, 2009.

[35] Victor Sanh, Thomas Wolf, and Alexander Rush. Movement pruning: Adaptive sparsity by fine-tuning. In *Advances in Neural Information Processing Systems*, 2020.

[36] Pedro Savarese, Hugo Silva, and Michael Maire. Winning the lottery with continuous sparsification. *Advances in Neural Information Processing Systems*, 2020.

[37] Jonathan Schwarz, Siddhant Jayakumar, Razvan Pascanu, Peter Latham, and Yee Teh. Powerpropagation: A sparsity inducing weight reparameterisation. In *Advances in Neural Information Processing Systems*, 2021.

[38] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *International Conference on Machine Learning*, 2021.

[39] Ziheng Wang, Jeremy Wohlwend, and Tao Lei. Structured pruning of large language models. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2020.

[40] Mengzhou Xia, Zexuan Zhong, and Danqi Chen. Structured pruning learns compact and accurate models. *arXiv preprint arXiv:2204.00408*, 2022.

[41] Lin Xiao. Dual averaging method for regularized stochastic learning and online optimization. *Advances in Neural Information Processing Systems*, 2009.

[42] Yujia Xie, Hanjun Dai, Minshuo Chen, Bo Dai, Tuo Zhao, Hongyuan Zha, Wei Wei, and Tomas Pfister. Differentiable top-k with optimal transport. In *Advances in Neural Information Processing Systems*, 2020.

[43] Dong Yu, Frank Seide, Gang Li, and Li Deng. Exploiting sparseness in deep neural networks for large vocabulary speech recognition. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2012.

[44] Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *IEEE/CVF International Conference on Computer Vision*, 2019.

[45] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *International Conference on Learning Representations*, 2018.

[46] Yuxin Zhang, Mingbao Lin, Mengzhao Chen, Zihan Xu, Fei Chao, Yunhan Shen, Ke Li, Yongjian Wu, and Rongrong Ji. Optimizing gradient-driven criteria in network sparsity: Gradient is all you need. *arXiv preprint arXiv:2201.12826*, 2022.

[47] Xiao Zhou, Weizhong Zhang, Hang Xu, and Tong Zhang. Effective sparsification of neural networks with global sparsity constraint. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021.

[48] Michael Zhu and Suyog Gupta. To Prune, or Not to Prune: Exploring the Efficacy of Pruning for Model Compression. In *International Conference on Learning Representations*, 2018.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Section 5.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] See Section 5.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Section 4 and the Appendix.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [Yes]
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Derivation of the Optimal Transport LP (Problem 2)

Here, we show how the original top-k LP with costs c can be straightforwardly rewritten in the form of an optimal transportation problem. For a given value vector $v \in \mathbb{R}^d$, cost vector $c \in \mathbb{R}^d_{++}$, and budget k, the top-k LP is:

$$\begin{aligned} \underset{m \in \mathbb{R}^d}{\text{maximize}} \quad & v^T m \\ \text{subject to} \quad & 0 \preceq m \preceq \mathbb{1}_d, \\ & c^T m = k. \end{aligned}$$

Define $y \coloneqq c \odot m$ and substitute to obtain:

$$\begin{aligned} \underset{y \in \mathbb{R}^d}{\text{maximize}} \quad & (v/c)^T y \\ \text{subject to} \quad & 0 \preceq y \preceq c, \\ & \mathbb{1}^T y = k. \end{aligned}$$

Now eliminate the upper bound constraint by introducing additional variables y' to give:

$$\begin{aligned} \underset{y, y' \in \mathbb{R}^d}{\text{maximize}} \quad & (v/c)^T y \\ \text{subject to} \quad & y \succeq 0,\ y' \succeq 0, \\ & y + y' = c, \\ & \mathbb{1}_d^T y = k. \end{aligned}$$

Finally, we rewrite this problem in the form of a standard OT problem by introducing the variable $Y \coloneqq [y,\, y']$ and the cost matrix $C \coloneqq [-v/c,\, 0]$ to yield:

$$\begin{aligned} \underset{Y \in \mathbb{R}^{d \times 2}_+}{\text{minimize}} \quad & \sum_{ij} C_{ij} Y_{ij} \\ \text{subject to} \quad & Y \mathbb{1}_2 = c, \\ & \mathbb{1}_d Y = [k,\, \mathbb{1}_d^T c - k], \end{aligned}$$

where we obtain the second column constraint by combining $y + y' = c$ and $\mathbb{1}_d^T y = k$.

B Derivation of Soft Top-k Forward Pass (Algorithm 4)

Problem 3 is an entropy regularized optimal transport problem with cost matrix $C \coloneqq [-v/c, 0] \in \mathbb{R}^{d \times 2}$, row marginals $c \in \mathbb{R}^d$, and column marginals $[k, \mathbb{1}_d^T c - k]$. By Lemma 2 in [11], the optimal solution to this problem can be written in the form $\mathrm{diag}(\exp \nu) \exp(-\beta C) \mathrm{diag}(\exp[\mu, \mu'])$, with dual variables $\mu, \mu' \in \mathbb{R}$ and $\nu \in \mathbb{R}^d$. Moreover, we can compute a sequence of iterates converging to an optimal collection of dual variables using Sinkhorn iteration.

Note that the dual variables μ, μ', and ν are only unique up to an additive constant: for any μ, μ', ν, and δ, we have $\mathrm{diag}(\exp \nu) \exp(-\beta C) \mathrm{diag}(\exp[\mu, \mu']) = \mathrm{diag}(\exp(\nu + \delta)) \exp(-\beta C) \mathrm{diag}(\exp[\mu - \delta, \mu' - \delta])$. We use this degree of freedom to fix $\mu' = 0$.

This yields the following Sinkhorn updates:

$$\begin{aligned} \mu_{t+1} &= \log k - \log \sum_{i=1}^{d} \exp(-\beta C_{i1} + \nu_{t,i}) \\ &= \log k - \log \sum_{i=1}^{d} \exp(\beta v_i / c_i + \nu_{t,i}), \\ \nu_{t+1} &= \log c - \log(\exp(-\beta C_{\cdot 1} + \mu_t) + \exp(-\beta C_{\cdot 2} + \mu'_t)) \\ &= \log c - \log(\exp(\beta v / c + \mu_t) + \mathbb{1}_d), \end{aligned}$$

as stated in Algorithm 4.

C Derivation of Soft Top-k Backward Pass (Algorithm 5)

The gradient of the loss with respect to the input values v of the soft top-k function follows from Theorem 3 of [42] with some algebraic manipulation. We restate the theorem below with the notation used in this paper. In the following, let $\bar{q} \coloneqq q_{:-1}$ denote the vector q with the last entry removed, and let $\bar{Y} \coloneqq Y_{\cdot,:-1}$ denote the matrix Y with the last column removed.

Theorem [42]. Let the solution $Y \in \mathbb{R}^{N \times M}$ of the entropy regularized optimal transport problem with cost matrix C, row marginals p and column marginals q be given by:

$$Y_{ij} = \exp(-\beta C_{ij} + \nu_i^* + \mu_j^*),$$

where $\nu^* \in \mathbb{R}^N$, and $\mu^* \in \mathbb{R}^M$ are optimal dual variables.
Then $\frac{d\nu^*}{dC}$ and $\frac{d\mu^*}{dC}$ are given by:

$$\begin{bmatrix} \frac{d\nu^*}{dC} \\ \frac{d\mu^*}{dC} \end{bmatrix} = \begin{bmatrix} -H^{-1}D \\ \mathbf{0} \end{bmatrix},$$

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

$$D_{\ell ij} = \beta \begin{cases} \delta_{\ell i} Y_{ij}, & \ell = 1, \dots, N \\ \delta_{(\ell - N)j} Y_{ij}, & \ell = N+1, \dots, N+M-1 \end{cases},$$

$$H^{-1} = -\begin{bmatrix} (\mathrm{diag}(p))^{-1} + (\mathrm{diag}(p))^{-1}\bar{Y}\mathcal{K}^{-1}\bar{Y}^T(\mathrm{diag}(p))^{-1} & -(\mathrm{diag}(p))^{-1}\bar{Y}\mathcal{K}^{-1} \\ -\mathcal{K}^{-1}\bar{Y}^T(\mathrm{diag}(p))^{-1} & \mathcal{K}^{-1} \end{bmatrix},$$

$$\mathcal{K} = \mathrm{diag}(\bar{q}) - \bar{Y}^T(\mathrm{diag}(p))^{-1}\bar{Y}.$$

Specializing this theorem to our setting with $N = d$ and $M = 2$, and recalling that the soft top-k mask values are given by $m_i = Y_{i1}/c_i$, we obtain:

$$H^{-1} = \frac{1}{\mathcal{K}} \begin{bmatrix} -\mathcal{K}\mathrm{diag}(1/c) - mm^T & m \\ m^T & -1 \end{bmatrix},$$

$$\mathcal{K} = k - \mathbb{1}^T(m^2 \odot c).$$

Therefore:

$$\begin{bmatrix} \frac{d\nu^*}{dC_{\cdot 1}} \\ \frac{d\mu^*}{dC_{\cdot 1}} \end{bmatrix} = \frac{\beta}{\mathcal{K}} \begin{bmatrix} \mathcal{K}\mathrm{diag}(m) + m(m^2 \odot c)^T - m(m \odot c)^T \\ -(m^2 \odot c)^T + (m \odot c)^T \end{bmatrix}.$$

Taking the derivative of the mask with respect to the first column of the cost matrix, we have:

$$\begin{aligned} \frac{dm}{dC_{\cdot 1}} &= -\beta \mathrm{diag}(m) + m\frac{d\mu^*}{dC_{\cdot 1}} + \mathrm{diag}(m)\frac{d\nu^*}{dC_{\cdot 1}} \\ &= -\beta\left(\mathrm{diag}(m \odot (\mathbb{1} - m)) - \frac{1}{\mathcal{K}}(m \odot (\mathbb{1} - m))(m \odot (\mathbb{1} - m) \odot c)^T\right). \end{aligned}$$

Finally, using our definition of the cost matrix as $C_{\cdot 1} = -v/c$ and applying the loss gradient $g \coloneqq \left(\frac{dL}{dm}\right)^T$, we obtain:

$$\left(\frac{dL}{dv}\right)^T = \left(\frac{dL}{dm}\frac{dm}{dv}\right)^T = \beta m \odot (\mathbb{1} - m) \odot \left(\frac{g}{c} - \frac{1}{\mathcal{K}}\mathbb{1}^T(g \odot m \odot (\mathbb{1} - m))\right),$$

as desired.

D Training Details

In Tables 4 and 5, we detail the hyperparameters used for our training runs. These hyperparameters were derived from `https://github.com/pytorch/vision/blob/96d1fecf/references/classification/README.md`.

Table 4: ResNet-50 training hyperparameters

Hyperparameter	Value
optimizer	Nesterov accelerated gradient method ($\text{momentum} = 0.9$)
max. learning rate	1.0
min. learning rate	0.001
learning rate warmup epochs	5
learning rate decay schedule	cosine
batch size	4096
weight decay	10^{-4} (0.0 for bias and normalization parameters)
label smoothing	0.1
data augmentation	random crops, random horizontal flips
input resolution	224×224
sparsity annealing schedule	linear from 1 to target sparsity at epoch fraction 0.2
β annealing schedule	linear from 1 to $\beta_{\max}$ at epoch fraction 0.8
Sinkhorn max. iterations	100
Sinkhorn tolerance ϵ	0.01

Table 5: ViT-B/16 training hyperparameters

Hyperparameter	Value
optimizer	AdamW ($\beta = (0.9, 0.999), \epsilon = 10^{-8}$)
max. learning rate	0.003
min. learning rate	0.0
learning rate warmup epochs	10% of total epochs
learning rate decay schedule	cosine
batch size	4096
weight decay	0.3 (0.0 for bias and normalization parameters)
label smoothing	0.1
data augmentation	random crops, random horizontal flips, RandAugment (ops $= 2$, magnitude $= 9$)
mixup α	0.2
CutMix α	1.0
gradient L_2 norm clip	1.0
input resolution	224×224
exponential moving averaging	false
sparsity annealing schedule	linear from 1 to target sparsity at epoch fraction 0.2
β annealing schedule	linear from 1 to $\beta_{\max}$ at epoch fraction 0.8
Sinkhorn max. iterations	100
Sinkhorn tolerance ϵ	0.01

E FLOP Measurements

Due to differences in the computational cost associated with individual parameters, the sparsity fraction does not map 1-to-1 to the fraction of FLOPs required for inference. Tables 6 and 7 give FLOP costs for our sparse models as a percentage of the FLOP cost of the corresponding dense model. We performed our FLOP measurements using the open source tool available at `https://github.com/sovrasov/flops-counter.pytorch/`.

We count multiply and add operations as one FLOP each. We remark that there exists some inconsistency in the literature regarding this convention, with some prior work using multiply-accumulate (MAC) counts and FLOP counts interchangeably. To convert the base FLOP counts listed below for ResNet-50 and ViT-B/16 to MACs, we can simply divide the given counts by 2.

In Table 6, the ResNet-50 FLOP counts for Top-KAST are slightly higher than those for Spartan, possibly due to the exclusion of the input convolutional layer from pruning in the case of Top-KAST.[1] In particular, this demonstrates that the accuracy improvements seen in Spartan-trained models over those trained using Top-KAST do not correlate with an increase in their FLOP costs.

Table 6: Percentage FLOP costs of sparse ResNet-50 models relative to the FLOP cost of a dense ResNet-50. The cost of a dense ResNet-50 model is 8.24 GFLOPs.

		Sparsity				
Method	**Epochs**	**80%**	**90%**	**95%**	**97.5%**	**99%**
Top-KAST	100	28.43 $\pm$0.28	17.11 $\pm$0.88	11.21 $\pm$0.29	7.82 $\pm$0.08	-
	200	27.80 $\pm$0.31	17.66 $\pm$0.07	11.75 $\pm$0.12	7.93 $\pm$0.13	-
	400	28.06 $\pm$0.12	18.05 $\pm$0.28	11.40 $\pm$0.16	7.74 $\pm$0.19	-
Spartan	100	24.68 $\pm$0.07	14.48 $\pm$0.11	8.67 $\pm$0.14	5.04 $\pm$0.11	2.53 $\pm$0.04
	200	24.37 $\pm$0.60	14.37 $\pm$0.09	8.43 $\pm$0.06	4.97 $\pm$0.07	2.59 $\pm$0.04
	400	23.97 $\pm$0.31	14.20 $\pm$0.08	8.56 $\pm$0.10	5.07 $\pm$0.06	2.50 $\pm$0.03

Table 7: Percentage FLOP costs of 90% sparse ViT-B/16 models at 224×224 input resolution relative to the FLOP cost of a dense ViT-B/16 model. The cost of a dense ViT-B/16 model is 35.19 GFLOPs.

		Sparsity Structure		
Method	**Epochs**	**Unstructured**	**16×16 blocks**	**32×32 blocks**
Top-KAST	100	13.42 $\pm$0.00	12.76 $\pm$0.00	12.76 $\pm$0.00
	300	13.41 $\pm$0.00	12.76 $\pm$0.00	12.76 $\pm$0.00
Spartan	100	13.40 $\pm$0.00	12.76 $\pm$0.00	12.76 $\pm$0.00
	300	13.43 $\pm$0.00	12.76 $\pm$0.00	12.76 $\pm$0.00

F FLOP-Sensitive Pruning

We demonstrate FLOP-sensitive pruning with Spartan on ResNet-50 using the following cost model: assign a cost of 1 to each parameter of a fully connected layer, and a cost of N^2 to each parameter of a convolutional layer where the output has size $N \times N$ along its spatial dimensions. We evaluate two valuation functions: $v_1(c_i, \theta_i) = c_i|\theta_i|$ and $v_{0.5}(c_i, \theta_i) = \sqrt{c_i}|\theta_i|$. v_1 results in the same pruning order as in standard pruning, but with a FLOP budget constraint instead of the usual sparsity budget. $v_{0.5}$ assigns a lower value to the parameters of convolutional layers, and results in networks where the parameters of convolutional layers are preferentially pruned. We use $\beta_{\max} = 10$ for v_1 and $\beta_{\max} = 160$ for $v_{0.5}$ to compensate for the relatively smaller scale of the normalized values v/c in the soft top-k forward pass (Algorithm 4).

Table 8 gives the top-1 accuracy, FLOP percentage, and sparsity percentage for each of these valuation functions. Spartan yields models with identical FLOP percentages of 5.76%, which is slightly higher than the budgeted value of 5%—this discrepancy is due to the additional cost of the normalization layers and activation functions in the network. Most notably, there is a substantial difference in the sparsity percentages realized by these valuation functions. As expected, $v_{0.5}$ preferentially sparsifies the parameters of convolutional layers and yields denser fully connected layers, resulting in lower sparsity overall.

Table 8: Results of FLOP-sensitive pruning experiments on ImageNet-1K with ResNet-50 models.

	Accuracy %	**FLOP %**	**Sparsity %**
v_1	73.88	5.76	95.94
$v_{0.5}$	74.13	5.76	89.73

[1] This setup for Top-KAST follows the protocol used by Jayakumar et al. [24]

G Additional Experiments

In Table 9, we compare Spartan against two additional variants of the Top-KAST baseline: Top-KAST with the Erdos-Renyi-Kernel (ERK) sparsity distribution, and with pruning applied to the parameters of all convolutional and fully-connected layers with the exception of bias terms (prune all). Top-KAST (excl. input conv.) denotes the Top-KAST variant used in the experiments presented in the main text, where we exclude the input convolutional layer from pruning. We find that there is some small variation in the measured top-1 validation accuracies, but our conclusion that Spartan improves generalization at higher levels of sparsity is unchanged.

Table 9: Comparison between Spartan and additional variants of the Top-KAST baseline on ImageNet-1K with ResNet-50 models.

		Sparsity	
Method	**Epochs**	**90%**	**95%**
Top-KAST (ERK)	100	75.98	73.63
	200	77.06	75.08
	400	77.48	75.62
Top-KAST (prune all)	100	75.74	73.72
	200	76.77	75.07
	400	77.44	75.77
Top-KAST (excl. input conv.)	100	75.48 ± 0.15	73.51 ± 0.16
	200	76.84 ± 0.11	75.20 ± 0.11
	400	77.37 ± 0.07	75.90 ± 0.04
Spartan	100	76.17 ± 0.10	74.68 ± 0.24
	200	77.06 ± 0.13	75.92 ± 0.01
	400	77.40 ± 0.06	76.48 ± 0.20

H Learned Sparsity Patterns

We observe a qualitative difference in the distribution of per-layer sparsities between ViT-B/16 models trained with unstructured sparsity and those trained with block structured sparsity (Figure 5). In particular, the output projections of self-attention layers under block structured pruning are significantly more dense in the later blocks of the network relative to unstructured pruning. The reasons for this difference are not immediately clear to us, and we leave further investigation of this phenomenon to future work.

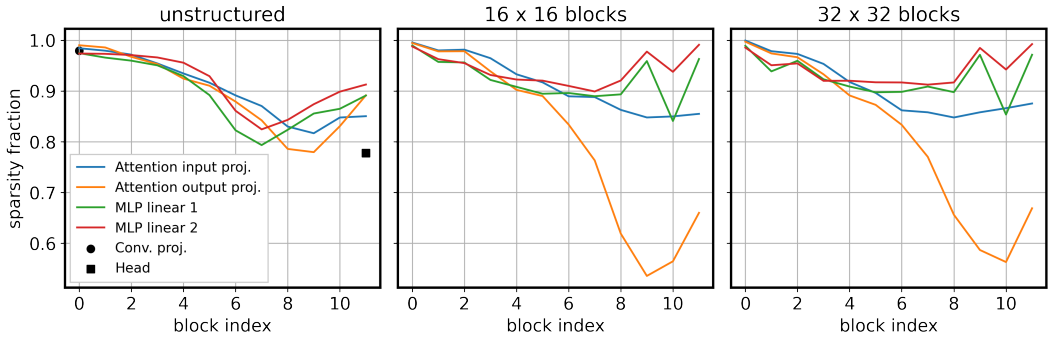

Figure 5: Per-layer sparsities of ViT-B/16 models trained with Spartan.

Block structured pruning produces coherent sparsity patterns in ViT-B/16 models. In Figure 6, we visualize the magnitudes of the weight matrices corresponding to the input projection of each

self-attention layer in a ViT-B/16 model trained with Spartan using 32×32 block structured pruning. This matrix maps vectors of dimension 768 to query, key, and value embedding vectors, each of dimension 768. We observe that the training process yields similar sparsity patterns in the query and key embedding submatrices, which correspond to the left and center panels in the visualization for each layer. This is an intuitively reasonable property since the self-attention layer computes inner products of the query and key embeddings in order to construct attention maps. We note that this symmetry emerges purely as a result of the optimization process; we did not incorporate any prior knowledge into Spartan regarding the role of particular entries of the weight matrices subject to sparsification.

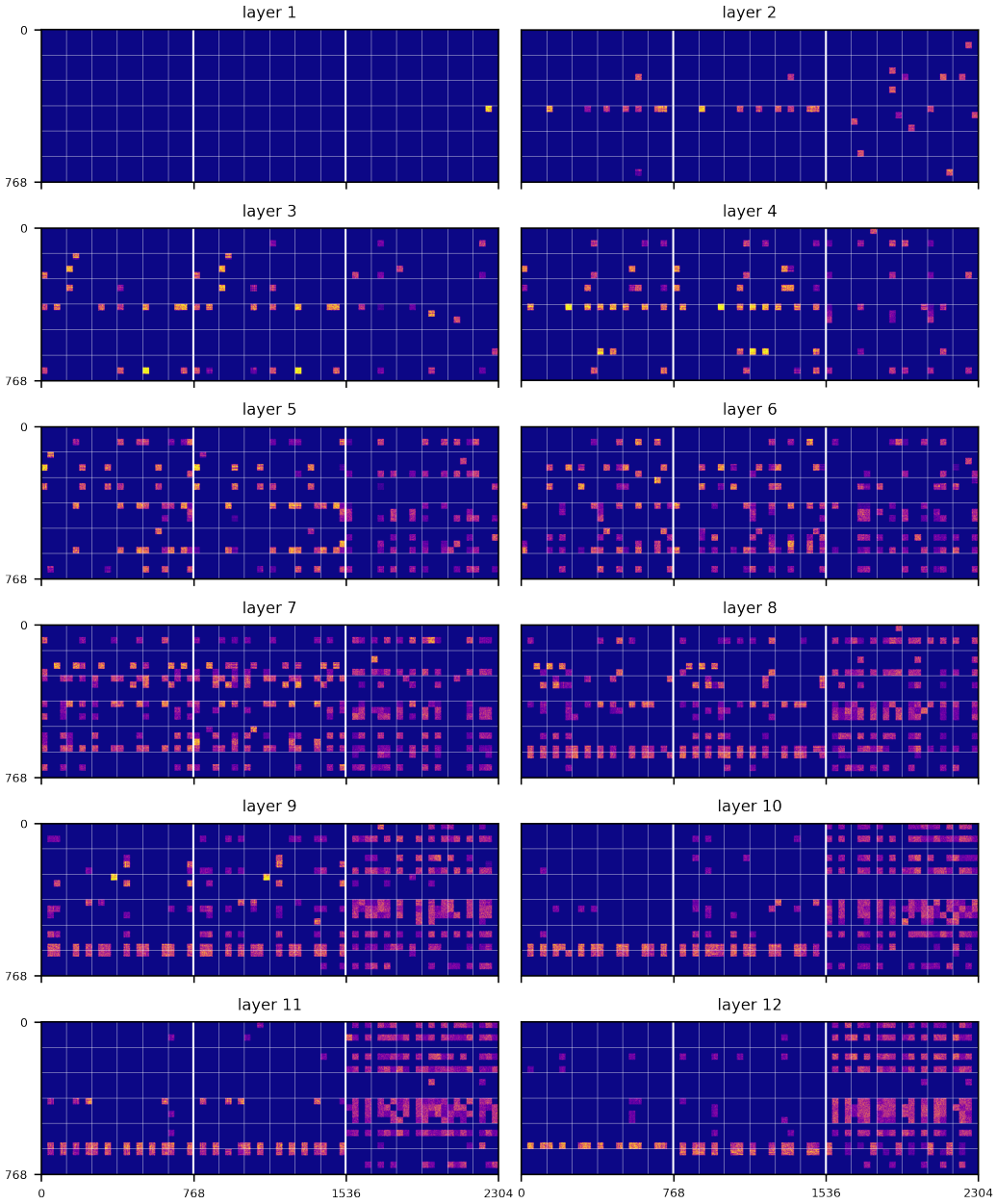

Figure 6: Weight magnitudes of the input projection matrices of each self-attention layer in a 32×32 block sparse ViT-B/16 network trained using Spartan.