SEARCH SPACES FOR NEURAL MODEL TRAINING

Anonymous authors
Paper under double-blind review

ABSTRACT

While larger neural models are pushing the boundaries of what deep learning can do, often more weights are needed to train models rather than to run inference for tasks. This paper seeks to understand this behavior using search spaces – adding weights creates extra degrees of freedom that form new paths for optimization (or wider search spaces) rendering neural model training more effective. We then show how we can augment search spaces to train sparse models attaining competitive scores across dozens of deep learning workloads. They are also are tolerant of structures targeting current hardware, opening avenues for training and inference acceleration. Our work encourages research to explore beyond massive neural models being used today.

1 INTRODUCTION

In the area of deep learning, increasing the size of neural models has led to dramatic advances (Hestness et al., 2017; Kaplan et al., 2020; Henighan et al., 2020), motivating training of hundreds of billions of parameters (Brown et al., 2020; Fedus et al., 2021). However, the growing size of models incur higher memory costs and runtimes, hindering training or inference tasks that can be tackled using current hardware (Thompson et al., 2020). Nevertheless, larger models have been shown to train faster (Li et al., 2020) and better Kaplan et al. (2020) which drives their adoption.

An area of research that seeks to compensate the increasing costs of model size is sparsity (Hoefler et al., 2021), which moves weights to zero so they can be discarded from storage or computations. Interestingly, neural models are capable of performing tasks on sizes smaller than they were trained for (Narang et al., 2017; Renda et al., 2020). While sparsity emerges as a promising option to reduce inference costs for overparameterized neural models (Luo et al., 2017; Allen-Zhu et al., 2019), training observes limited success (Lee et al., 2019; Frankle & Carbin, 2019; Wang et al., 2020; Tanaka et al., 2020; Frankle et al., 2021; Bellec et al., 2018; Mocanu et al., 2018; Dettmers & Zettlemoyer, 2019; Evci et al., 2020a; Jayakumar et al., 2020). This raises an interesting question – why do larger neural models train better – that remains to be understood. Being able to run inference but not training using fewer weights points to a problem with search, or how models navigate search spaces throughout training, rather than model capacity (LeCun et al., 1990).

This paper explains why deep learning tasks improve with the size of search spaces being explored during training. We find that adding weights provides extra degrees of freedom that form new paths of optimization and facilitate the search for neural models. Using this understanding, we propose a series of steps to augment search spaces of sparse models during training to approximate the behavior of larger models. We then show this methodology achieves competitive results on dozens of deep learning workloads, even when satisfying constraints needed to accelerate training and inference using Sparse Tensor Cores (Mishra et al., 2021) in NVIDIA GPUs.

The paper is organized as follows. In Section 2 we use search spaces to describe reasons more weights are needed for training. We then devise approximations to augment their size when training sparse models in Section 3. Section 4 shows sparse models constructed in this fashion perform competitively across a plethora of deep learning workloads and can target current hardware accelerators Mishra et al. (2021) for training and inference. In Section 5 we relate to prior research. Section 6 concludes with directions for future work.

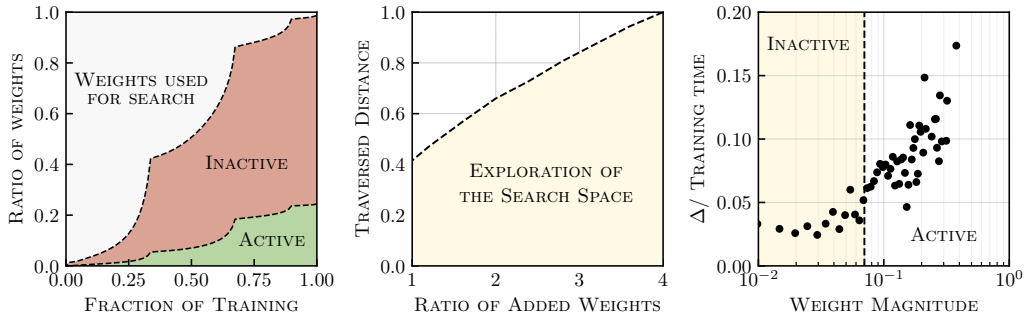

Figure 1: Investigations into the role of search spaces for ResNet50. Left: Fraction of weights whose magnitudes remains above (active set) or below (inactive set) a threshold until the end of training. The threshold is chosen such that one-fourth of the weights remain for inference tasks. Middle: Cumulative distance (normalized by its maximum) that neural models traverse in search space as a function of the number of weights added to the active set. Right: The time (Δ) it takes for weights to uncorrelate (normalized by the total number of training steps) as a function of magnitudes obtained after training. Points correspond to the median Δ over weight bins sampled from a single neural layer. Appendix A covers data for a broader span of neural models and tasks.

2 A PRIMER ON SEARCH SPACES

We begin by understanding what role search spaces (or the set of all possible weight combinations that a model can take) have in neural model training. We show neural weights needed for inference are discovered throughout training with help from added weights (or weights that can be discarded after training). More specifically, we observe adding weights help neural models better explore search spaces to discover promising combinations for inference. While they do not learn meaningful representations, added weights form alternate paths for optimization that enable training to escape critical points. By combining these observations, we provide intuition that more weights are needed for training because they augment search spaces allowing neural models to train more effectively.

2.1 DISCOVERING WEIGHTS NEEDED FOR INFERENCE

A popular technique to reduce inference costs is to discard weights of a neural model whose magnitudes fall below a certain threshold after training (typically denoted as sparsity), since they contribute the most to defining the behavior of a neural model (Jayakumar et al., 2020). Neural models constructed in this fashion (or sparse models) can achieve the same accuracy as larger models when trained from the same initialization (Frankle & Carbin, 2019) or with additional training steps (Mishra et al., 2021). Thus, we can assume neural weights retained after training are critical for inference, as opposed to removed weights which are not necessary.

The question then arises whether weights relevant for inference can be determined earlier in the training process. For this purpose, we examine how weights evolve throughout training, classifying them into two types of behavior. We define a set of weights as active at a particular point in time if their magnitudes remain above a chosen threshold from that point until the end of training, representing weights that are important for inference. Conversely, the set of weights whose magnitudes remain below the threshold from that point until the end of training are inactive, after which they play no role in inference tasks. There are also the remaining weights that constantly move above and below the threshold. While other combinations of weights that can contribute to inference may exist, this is one instance that is known to produce good accuracy (Frankle & Carbin, 2019).

Figure 1 (left) shows how sets evolve over the course of training when using a threshold that maintains a quarter of the weights for inference. In early training, most weights switch between either side of the threshold as neural models are still learning, eventually stabilizing into active or inactive sets. Neural weights relevant for inference (or form the active set) are not prebaked (Frankle et al., 2021), but rather discovered as the model learns. As a result, we can presume all weights including

ones that are eventually discarded (or added weights) play a crucial role in determining what will be used for inference.

2.2 EXPLORING SEARCH SPACES WITH MODEL SIZE

An important aspect of training is how well can neural models navigate search spaces to discover better weight combinations for inference tasks. Given added weights help uncover weights relevant for inference, we posit they must play an important role in exploring search spaces. We characterize exploration with distances that neural models undertake throughout training, which are measured using cumulative increments $\sum_t \sum_w |w(t+1) - w(t)|$ over successive time steps t across convolutions and linear layers, excluding other neural layers.

Figure 1 (middle) illustrates the distance traversed (or exploration of the search space) as a function of weights added to a model that can successfully perform inference tasks (assume a model one-fourth its original size) but requires more weights to train. We observe distance increases with the number of weights being added, therefore larger models can navigate search spaces more effectively.

Based on the above, we reason using more weights during training expands the size of search spaces, making it easier to discover combinations of weights that are needed for inference. More precisely, adding weights helps neural models explore new regions of the search space, escaping critical points that often lead to poor task accuracy in smaller models (Choromanska et al., 2015). However, the exact function of added weights (which can be later discarded for inference) is discussed below.

2.3 ROLES OF WEIGHTS USING SIMILARITIES

To better understand the role of weights during training, we look at how similar are their values over time using correlations. We can interpret correlations as the degree with which weights in a neural model learn: weights that get reinforced (or learn) throughout training are correlated to previous values and exhibit temporal patterns, and conversely weights that have random changes (or don't learn) are uncorrelated and have no dependence over time.

We measure the similarity between a pair of series x and y through the Pearson correlation coefficient Rodgers & Nicewander (1988): $\rho_{x,y} \equiv E[(x-\mu_x)(y-\mu_y)]/\sigma_x\sigma_y$, where μ is the mean, σ the standard deviation, and $E[\cdot]$ the expected value. Coefficients of value $+1$ denote two series have identical trends, 0 indicates the series are random, and -1 represents series with opposite behavior. We treat each individual weight as a time series $w \in \{w_1, w_2, \ldots, w_t\}$ and compute Pearson coefficients between windows $x \in \{w_1, w_2, \ldots, w_{t-\tau}\}$ and $y \in \{w_\tau, w_{\tau+1}, \ldots, w_t\}$, representing the similarity between the current weights and their values at some future time τ. Since correlations naturally decay with τ (temporal patterns are less likely to persist over long durations), an important metric is the time Δ after which weights no longer correlate to their future values ($\rho = 0$).

Figure 1 (right) illustrates the fraction of total training after which weights become completely uncorrelated (Δ) as a function of their magnitudes obtained after training. While weights belonging to the active set exhibit correlations that persist across a significant portion of training, weights forming the inactive set have short-term correlations and behave randomly over small periods. We conjecture long-term correlations of large weights are characteristic of learning, as they signify repeated reinforcement along the weight direction, whereas the near-random motion of small weights suggests they don't learn useful representations and can be removed during inference.

Since weights that move into the inactive set are absent of correlations across long periods of training, we interpret their crucial role during training as follows: short-term interactions enable training to take different paths for optimization, which facilitates escape from bad saddle points (Kawaguchi, 2016) or high error plateaus (Dauphin et al., 2014) that can slow down learning. In later sections, we also discover their learned values can be periodically destroyed throughout training without affecting accuracy. Thus, we reasonably conclude that added weights removed during inference do not learn meaningful representations, but rather their short-term behavior is responsible for widening the search space.

2.4 AN INTUITIVE EXPLANATION ABOUT SEARCH SPACES

Based on the data above, we construct the hypothesis that "weights removable at inference time represent extra degrees of freedom that are needed to augment search spaces for training neural models effectively." This explains the prevailing observation that neural models with reduced sizes, which perform well at inference tasks, would underperform during training.

For a more precise formulation, assume neural models are parameterized by n. In this scenario, training traverses through n-dimensional search spaces to minimize the loss function, leveraging all n weights in the process. Conversely, models trained with dn weights, where $d \in (0, 1)$, navigate search spaces that are constrained to dn dimensions. Operating on reduced spaces makes training more susceptible to critical points, limiting weight combinations that neural models can explore during training. Since weights removable at inference observe short-term interactions, they do not learn representations but rather augment search spaces allowing training to be more effective.

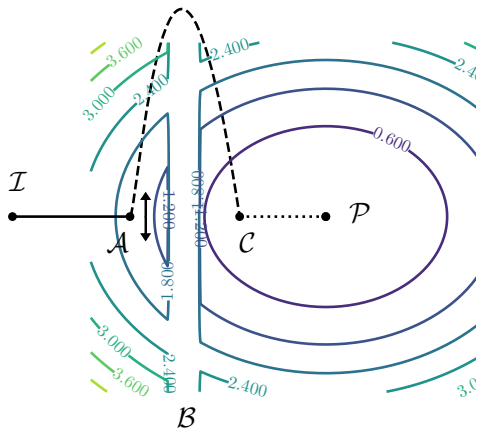

Figure 2: Training trajectories for one- and two-parameter neural models on a 2D convex landscape with a minimum $\mathcal{P}$ and barrier $\mathcal{B}$.

Figure 2 illustrates an example. Let's assume we are training neural models in a two-dimensional loss landscape of convex shape with a minimum at $\mathcal{P}$ and a vertical barrier $\mathcal{B}$. A one-parameter model that can move left or right during training will converge prematurely at $\mathcal{A}$ if initialized at $\mathcal{I}$ such that $\mathcal{B}$ obstructs its path to $\mathcal{P}$ (solid curve). Adding another weight augments the search space so training can take new possible paths along the loss landscape to escape $\mathcal{B}$ (dashed curve). After reaching $\mathcal{C}$, training can discard the added weight and converge on $\mathcal{P}$ (dotted curve). A similar generalization can be made for higher dimensions, where adding weights helps training escape saddle points. This behavior is believed to be a crucial aspect for neural model training with backpropagation.

3 RECOMMENDATIONS FOR SPARSE TRAINING

While search spaces are important for training, their sizes are often made smaller for sparse models. Building on understanding from previous sections, we uncover the following recommendations to approximate wider search spaces when training sparse models:

(1) Rewiring weights that participate in training allows sparse models to explore different regions of the search space.

(2) Gradient updates for weights that do not participate in training encourages alternate paths for optimization.

(3) Inducing exploitation (stop rewiring after some amount of training steps, reset non-participating weights to zero or regularize them) reduces noise from gradient accumulations.

Appendix C summarizes possible methods for expanding the size of search spaces during neural model training using these three simple steps.

3.1 REWIRING OF NEURAL WEIGHTS

Sparse models are commonly trained by rewiring (or sampling) a subset of weights from a neural model according to some criteria (magnitude, sign, gradients, etc), allowing them to learn which weights are needed for inference. Some works rewire weights every training iteration (Bellec et al., 2018; Wortsman et al., 2019; Raihan & Aamodt, 2020; Jayakumar et al., 2020; Zhou et al., 2021),

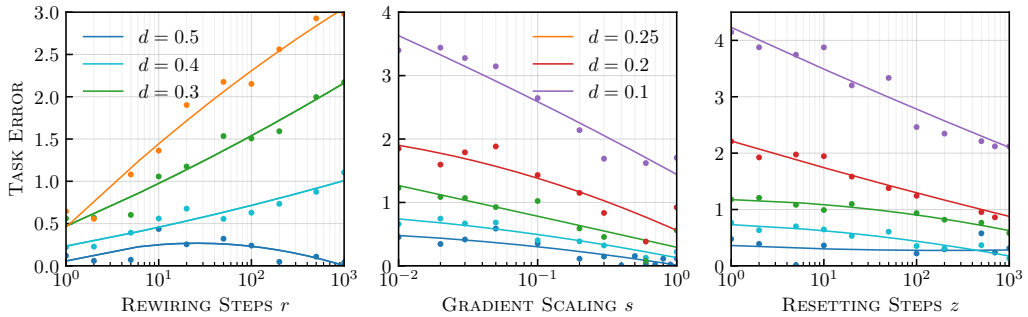

Figure 3: Investigations into various aspects of training sparse models using ResNet50. Left: Task error (or accuracy difference between regular and sparse models) as a function of rewiring steps r. Middle: Task error as a function of scaling factor s applied to gradients of weights that do not participate in training. Right: Task error where non-participating weights are reset to zero every z training steps. We consider sparse models of different sizes d denoting the ratio of weights being used. Lines represent polynomial fits of sample points. Appendix B covers data for a broader span of neural models and tasks.

while others rewire every hundreds of training steps (Evci et al., 2020a) or after an entire pass through the data (Mocanu et al., 2018; Dettmers & Zettlemoyer, 2019). Since there are no clear reasons behind these choices, an important question becomes how often must weights be rewired so neural models can learn.

We study the effects rewiring rates have on accuracy when training sparse models of various sizes d, where d denotes the fraction of weights being used. Figure 3 (left) plots the task error (or accuracy difference between regular and sparse models) as a function of the number of training steps r taken between rewirings. We find rewiring frequency does not matter when sparsity is low or moderate ($d \sim 0.5$), but errors increase with r as models get sparser ($d \ll 0.5$). Therefore, weights should be rewired frequently (within a few training steps) to avoid losing accuracy.

We can use search spaces to explain why training improves with more frequent rewiring. A sparse model is made of weights that participate in training (are used in forward and backward propagations) and those that do not participate (are set to zero during training). Search spaces will expand only when weights are rewired: neural models always operate on reduced spaces if participating weights remain unchanged throughout training ($r \to \infty$), and expand exactly once after going through the entire data when they are rewired every epoch. By swapping between weights that participate and do not participate, training can take different paths for optimization and approximate a free search space, which helps neural models escape critical points (Evci et al., 2020c).

3.2 Updates for weights not participating in training

Previously, we found neural weights not needed for inference form short-term interactions that make training more effective. Since sparse models lack weights that have similar roles, we can approximate this behavior using gradient updates for non-participating weights. Because non-participating weights do not contribute to the loss, their gradients determine whether another optimization path is better suited to traverse the region being explored in the loss landscape. Repeating gradient updates registers the importance of these paths over some period of training and can trigger rewiring when weights exceed a threshold, which allows training to explore different regions of the search space while operating on (or forward and backward propagating) a smaller set of weights.

We evaluate the importance of gradients updates on non-participating weights by reducing their contribution with a scale factor s. Figure 3 (middle) shows the task error as a function of the scale factor s using various sizes d. We observe error increases as $s \to 0$ (no gradients contribute when $s = 0$), which can be attributed to premature convergence when training lacks expressive power to explore different paths. Dampened gradient updates affect sparser models ($d \ll 0.5$), which are

sensitive to reduced spaces, more than larger models ($d \sim 0.5$), which may still have sufficient search. As a result, non-participating weights should be updated regularly to retain accuracy.

Gradient updates for weights that do not participate in training could mean they also learn representations, such that more capacity rather than search improves accuracy. We verify this by resetting non-participating weights to zero after every z training steps, thus removing any representations they might have learned. Figure 3 (right) shows the task error as a function of z. Errors are the highest when values are reset every iteration ($z = 1$), which is equivalent to training on reduced spaces (weights can still be rewired but we no longer encourage certain ones to participate in training). Conversely, error decreases as weights are reset less frequently and saturates after sufficient training steps ($z \sim 1k$). Interestingly, this also represents the time it takes for added weights to uncorrelate, as shown in previous sections. The ability to remove information from non-participating weights reinforces the idea that they augment search spaces rather than model capacity.

While most of literature trains sparse models using fewer weights and gradients (Bellec et al., 2018; Mocanu et al., 2018; Dettmers & Zettlemoyer, 2019), more recent success has been found by updates gradients for weights that do not participate in training (Wortsman et al., 2019; Liu et al., 2020; Zhou et al., 2021; Hubara et al., 2021) as described above.

3.3 Balancing between exploration and exploitation

Deep learning like other optimization problems treads a delicate balance between exploration and exploitation. In early stages of training neural models explore search spaces using high learning rates, whereas late stages exploit specific regions using small learning rates. Training sparse neural models can also affect this balance, as fewer weights reduces the degrees of freedom and thus hinders exploration during training. On the other hand, gradient updates on non-participating weights introduces noise as any nonzero value will not reflect what is being used for training, which limits exploitation (training bounces around basins of attraction due to gradient noise).

Figure 4 shows task error degrades without proper exploration, by training smaller models with neural layers of reduced widths (No explore), or exploitation, by training sparse models delineated so far (No exploit). Therefore, we seek ways to induce exploitation when augmenting search spaces for more exploration. One course of action is to remove noise introduced by non-participating weights during late stages, so training can take steepest descents towards the minima. To achieve this we stop rewiring weights after sufficient training (Fix), such that non-participating weights can no longer contribute to the loss. Figure 4 shows this decreases error rates tremendously.

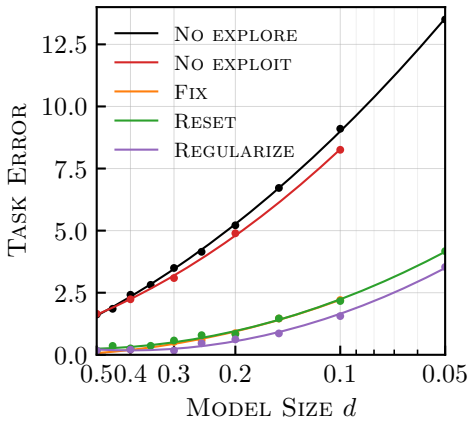

Figure 4: Task error after training ResNet50 using different exploration versus exploitation strategies as a function of model size d. Appendix B covers data for a broader span of neural models and tasks.

Another option is to draw analogies with added weights, which can be safely discarded after sufficient training since they do not learn representations over time. We reset non-participating weights to zero roughly every $1k$ training steps (or the time it takes for added weights to uncorrelate) in order to remove gradient noise that may trigger unnecessary rewiring, allowing training to exploit nearby regions. Figure 4 shows error decreases substantially when non-participating weights are either reset to zero (Reset) or regularized with a decay factor (Regularize) as suggested in (Zhou et al., 2021). Sparsity literature introduces alternative courses for inducing exploitation (Mocanu et al., 2018; Dettmers & Zettlemoyer, 2019; Evci et al., 2020a).

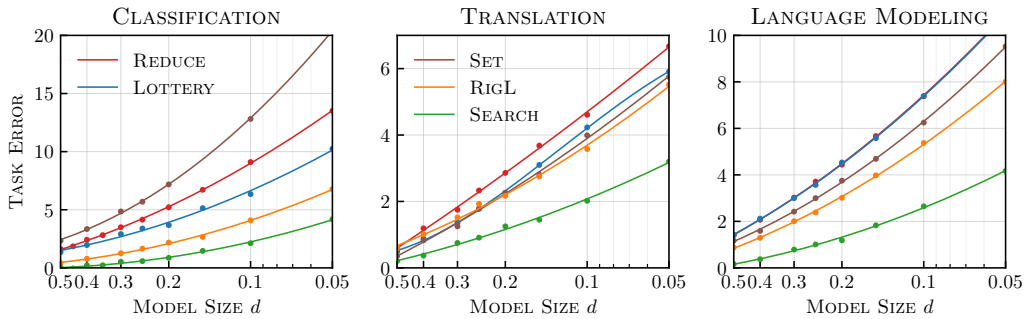

Figure 5: Task error (or accuracy difference between regular and sparse models) comparing various methods as a function of model size d. Left to right: ResNet50 (Classification), Transformer (Translation), Transformer-XL (Language Modeling). Appendix I covers data for a broader span.

4 Empirical Data

Using recommendations from the previous section, we are in position to train sparse models approximating wider search spaces. This section presents empirical evidence that wider search spaces can help sparse models achieve better accuracy for inference tasks. We first demonstrate that our strategy performs competitively against state-of-the-art sparsity research. Then, we explore sparse models are tolerant to sparsity structures that target hardware acceleration using Sparse Tensor Cores (Mishra et al., 2021).

4.1 Comparisons to sparsity research

Experiments are conducted across a wide breadth of deep learning tasks and neural architectures trained on large data sets, as detailed in Appendix E. We draw comparisons between full-sized models that have free search (Regular), smaller models consisting of neural layers with reduced widths (Reduce) that can run on existing dense hardware, and sparse models that approximate wider search spaces (Search). Appendix F shows where accuracy for the latter falls between the former two.

Table 1 lists accuracies for regular models and their differences for sparse models (adding the two numbers produces accuracies for sparse models) across various tasks as a function of d, where d denotes the ratio of weights being used. We find using our recommendations most sparse models can be halved in size ($d = 0.5$) without sacrificing any task accuracy, whereas training with a quarter of the weights ($d = 0.25$) often reduces accuracy by less than 1%. Some exceptions include efficient convolutional models and sparser models ($d = 0.1$) which may be constrained by capacity.

We also draw comparisons to sparsity literature. While Lottery (Frankle & Carbin, 2019) determines at initialization which weights participate in sparse models (Wang et al., 2020; Tanaka et al., 2020), SET Mocanu et al. (2018) and RigL (Mocanu et al., 2018) rewire them throughout training (see Appendix D). However, we do not consider methods that move weights across layers (Dettmers & Zettlemoyer, 2019; Liu et al., 2020) or adopt concepts similar to those explored in this paper (Zhou et al., 2021; Hubara et al., 2021). Figure 5 shows the task error (or difference in accuracy between regular and sparse models) for these, which indicates our strategy outperforms competing approaches on all tasks and model sizes by a wide margin.

4.2 Application on hardware accelerators

We have shown earlier that by using search spaces we can reduce the size of neural models while maintaining accuracy. However, such models cannot be accelerated on modern hardware with current matrix-math pipelines (Park et al., 2017; Gale et al., 2020) without imposing particular structures (or positions of weights used during training and inference). On the other hand, neural structures targeting hardware acceleration (e.g., removing blocks (Gray et al., 2017), channels or fil-

Table 1: Accuracies for regular models and their differences for sparse models of different sizes d.

Model	REGULAR	$d = 0.5$	$d = 0.25$	$d = 0.1$	Model	REGULAR	$d = 0.5$	$d = 0.25$	$d = 0.1$
ResNet18	70.30	+0.01	−0.92	−2.75	SqueezeNet V1	60.77	−0.88	−5.20	–
ResNet34	73.87	−0.20	−0.65	−2.27	MobileNet V2	71.53	−0.87	−0.87	–
ResNet50	76.71	−0.05	−0.59	−2.17	Stacked UNet-64	69.53	−1.41	−3.79	−8.19
ResNet101	77.50	−0.25	−0.61	−1.52	SSD-ResNet18	19.15	−0.81	−2.49	−5.55
ResNeXt50	77.68	+0.02	−0.49	−2.01	SSD-ResNet50	24.93	−0.52	−2.06	−5.31
ResNeXt101	79.27	+0.20	+0.19	–	Faster R-CNN	37.71	−0.25	−1.61	−5.51
WideResNet50	78.13	+0.13	−0.34	−1.06	Mask R-CNN	38.26	−0.34	−1.42	−4.80
WideResNet101	78.63	−0.12	+0.04	−1.00	Mask R-CNN	35.03	−0.88	−2.65	−7.44
InceptionV3	77.10	−0.08	−0.88	−3.25	Mask R-CNN 3×	40.78	−0.12	−1.12	−3.51
Xception	79.28	+0.04	−0.28	−1.26	Mask R-CNN 3×	37.05	−0.03	−0.81	−2.98
DenseNet121	75.46	−0.46	−1.75	−4.35	RetinaNet	36.48	−0.42	−2.42	−6.00
DenseNet161	78.77	+0.01	−0.86	−2.35	RPN	57.61	−0.16	−0.90	−2.56
DenseNet169	76.97	+0.04	−0.96	−3.23	DETR	39.90	+0.10	−0.60	−0.80
VGG11-BN	70.70	−0.33	−0.79	−2.24	Pix2PixHD*	68.83	−2.27	+3.02	+3.52
VGG16-BN	74.00	−0.25	−0.46	−1.75	Few-Shot Vid2Vid*	25.78	−0.52	+0.63	+4.52
VGG19-BN	74.88	+0.09	−0.48	−1.52	FAZE*	2.94	+0.04	−0.02	+0.04
DRN-C-26	75.22	−0.30	−0.74	−2.35	Vaswani Base	26.87	−0.68	−1.92	−3.62
DRN-C-42	76.78	−0.10	−0.62	−1.98	Vaswani Large	28.43	−0.09	−0.92	−2.12
DRN-A-50	78.30	−0.23	−0.74	−2.28	Levenshtein*	6.16	+0.11	+0.23	+0.45
DeiT Tiny	72.70	−2.81	−8.09	−16.49	GNMT	24.81	−0.12	+0.26	−0.15
DeiT Small	80.08	−1.53	−3.75	−8.30	XL Base*	22.88	+0.49	+2.04	+5.41
DeiT Base	81.95	−0.75	–	–	XL Large*	17.90	+0.16	+1.01	+2.65
ShuffleNetV2	68.44	−0.43	−1.44	–	BERT Base	87.66	−0.04	–	–
MNASNet V1	71.80	−1.09	−3.36	–	BERT Large	90.92	−0.02	–	–

* Lower means better (negative differences mean better)

ters (Wen et al., 2016; Li et al., 2017), layers (Michel et al., 2019)) often degrade accuracy. This apparent tradeoff between accuracy and performance (or speed) has hindered their adoption.

Therefore, we explore whether search spaces can make hardware-aware structures more amenable for deep learning. As a case study, we consider Sparse Tensor Cores introduced in NVIDIA Ampere GPU architecture (Mishra et al., 2021), which have twice the math throughput of regular matrix operations. The hardware expects a 2:4 sparsity structure that takes at least two values to be zero for each group of four values. Appendix H illustrates examples of 2:4 sparsity for inference (1D) and for training (2D).

Table 2 lists accuracy differences between sparse and regular models (positive values mean sparse performs better), where we extend learning rate schedules for some models compared to what is typically used in literature (see Appendix G for details). We find structured neural models can generally retain accuracy for tasks. Using 2:4 for training (2D) and inference (1D) roughly match accuracy of regular models, while neural models adopting coarser structures such as block sparsity (Narang et al., 2017; Gray et al., 2017) fail to benefit from search spaces and perform no better than smaller models (see Appendix H). Therefore, we can conclude 2:4 sparsity structures are particularly effective at approximating wider search spaces due to their finer granularity. This suggests possible

Table 2: Accuracies for regular models and their differences for 2:4 sparse models.

Model	REGULAR	2:4 1D	2:4 2D	Model	REGULAR	2:4 1D	2:4 2D
ResNet18	72.17	+0.00	−0.28	VGG19	74.88	+0.04	−0.21
ResNet34	75.14	+0.06	−0.27	Xception	79.28	+0.04	−0.11
ResNet50	77.67	+0.05	+0.09	DETR	39.90	−0.30	−0.40
ResNet101	78.94	−0.09	−0.31	Pix2PixHD*	68.83	−1.34	+0.68
InceptionV3	78.11	−0.11	−0.18	Few-Shot Vid2Vid*	26.06	−0.49	+0.04
ResNext50	78.36	−0.21	−0.19	FAZE*	2.49	−0.08	–
ResNext101	79.27	+0.28	+0.36	GNMT	24.81	+0.15	+0.09
WideResNet50	78.13	−0.07	−0.08	Transformer Large	28.43	−0.10	−0.39
WideResNet101	78.63	+0.08	−0.07	BERT Large	90.92	+0.03	−0.55
DRN C 26	77.66	−0.05	−0.11				

* Lower means better

avenues towards accelerating training (Hubara et al., 2021) as well as obtaining efficient models for inference (Mishra et al., 2021; Zhou et al., 2021).

5 RELATED WORK

While this paper presents new perspectives on the benefits of using more weights, overparameterization has been extensively explored for neural model training, ranging from theoretical studies of how sufficiently large yet contrived models can converge to global minima (Allen-Zhu et al., 2019; Zou & Gu, 2019) to empirical investigations using more realistic examples of how training explores loss landscapes (Goodfellow et al., 2015; Ardalani et al., 2019), albeit from different angles. More recently, dramatic advances have been made by training larger neural models (Brown et al., 2020; Fedus et al., 2021). It was further observed that accuracy scales as a function of model size (dubbed as scaling laws (Hestness et al., 2017; Kaplan et al., 2020; Henighan et al., 2020)) and larger models can converge to better accuracy in fewer training steps (Ardalani et al., 2019; Li et al., 2020).

Applying sparsity to reduce the size of neural models has been a topic of interest for the past three decades (LeCun et al., 1990; Hassibi & Stork, 1993; Reed, 1993; Castellano et al., 1997; Han et al., 2015; Jayakumar et al., 2020). Typically, sparse models are constructed by training much larger models (that are easier to train) and removing some of their weights either after training (Han et al., 2015; Mishra et al., 2021) or gradually alongside training (Narang et al., 2017; Zhu & Gupta, 2018; Gale et al., 2019). However, the above are only useful to reduce costs for inference.

For training acceleration, early works (Lee et al., 2019; Wang et al., 2020; Tanaka et al., 2020) sought to remove weights before training with limited success (Gale et al., 2019; Frankle et al., 2021). Surprisingly, it was shown sparse models can be trained when initialized the same way as a larger trained model (Frankle & Carbin, 2019). After this breakthrough, many works (Dettmers & Zettlemoyer, 2019; Evci et al., 2020a; Jayakumar et al., 2020) tried to dynamically learn sparse models by rewiring their weights throughout training (though some works were published earlier (Bellec et al., 2018; Mocanu et al., 2018)), which improved accuracy but not enough to match that of larger models. More recent advances (Wortsman et al., 2019; Liu et al., 2020; Savarese et al., 2020; Zhou et al., 2021; Hubara et al., 2021) introduced gradient updates for all weights in a neural model, including ones that do not participate in training, and observed better success.

Our explanations about what role adding more weights might have on training – they act as extra degrees of freedom which facilitate search for neural models – complement existing observations that larger models (which have more weights) train better and faster. We use this understanding to consolidate many ideas that have been circling around in sparsity literature but remain not clearly understood (how frequently must weights be rewired and why). Recent works converging in similar directions (Zhou et al., 2021; Hubara et al., 2021) or seeking better understanding (Liu et al., 2021) are conducted in tandem with our research.

6 CONCLUSION

We describe a simple reason more weights are needed for training than inference – adding more weights gives neural models extra degrees of freedom to augment search spaces (with new paths for optimization) for training neural models effectively (or escaping critical points). Our experiments uncover recommendations to approximate the behavior of larger models (or wider search spaces) when training sparse models and demonstrate that they work across dozens of deep learning workloads.

We believe these results open many questions. On the practical side, it may be interesting to consider how could this strategy be adopted to accelerate real-world training and inference workloads today, reducing the environmental impact from training very large models. On the theoretical side, we would like to understand better ways to approximate wider search spaces in cases where accuracy still suffers as well as how to address potential biases induced by sparse models which may affect applications. We hope that our results will spur further research on these unconventional architectures, which challenge the default choice held by massive neural models today.

REFERENCES

Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. A convergence theory for deep learning via over-parameterization. In *ICML*, 2019.

Newsha Ardalani, Joel Hestness, and Gregory Diamos. Empirically characterizing overparameterization impact on convergence. `https://openreview.net/forum?id=S1lPShAqFm`, 2019.

Guillaume Bellec, David Kappel, Wolfgang Maass, and Robert Legenstein. Deep rewiring: Training very sparse deep networks. In *ICLR*, 2018.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *NeurIPS*, 2020.

Giovanna Castellano, Anna Maria Fanelli, and Marcello Pelillo. An iterative pruning algorithm for feedforward neural networks. *IEEE Transactions Neural Networks*, 1997.

Anna Choromanska, Mikael Henaff, Michael Mathieu, Gerard Ben Arous, and Yann LeCun. The loss surfaces of multilayer networks. In *ICAIS*, 2015.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. Transformer-XL: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019.

Yann N Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *NeurIPS*. 2014.

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

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Utku Evci, Trevor Gale, Jacob Menick, Pablo Samuel Castro, and Erich Elsen. Rigging the lottery: Making all tickets winners. In *ICML*. 2020a.

Utku Evci, Yani A. Ioannou, Cem Keskin, and Yann Dauphin. Gradient flow in sparse neural networks and how lottery tickets win. *arXiv preprint arXiv:2010.03533*, 2020b.

Utku Evci, Fabian Pedregosa, Aidan Gomez, and Erich Elsen. The difficulty of training sparse neural networks. *arXiv preprint arXiv:1906.10732*, 2020c.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *arXiv preprint arXiv:2101.03961*, 2021.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *ICLR*, 2019.

Jonathan Frankle, Gintare K. Dziugaite, Daniel M. Roy, and Michael Carbin. The lottery ticket hypothesis at scale. *arXiv preprint arXiv:1903.01611*, 2019.

Jonathan Frankle, Gintare Karolina Dziugaite, Daniel Roy, and Michael Carbin. Pruning neural networks at initialization: Why are we missing the mark? In *ICLR*, 2021.

Trevor Gale, Erich Elsen, and Sara Hooker. The state of sparsity in deep neural networks. *arXiv preprint arXiv:1902.09574*, 2019.

Trevor Gale, Matei Zaharia, Cliff Young, and Erich Elsen. Sparse gpu kernels for deep learning. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2020.

I. J. Goodfellow, O. Vinyals, and A. M. Saxe. Qualitatively characterizing neural network optimization problems. In *ICLR*, 2015.

S. Gray, A. Radford, and D. P. Kingma. GPU kernels for block-sparse weights. `https://cdn.openai.com/blocksparse/blocksparsepaper.pdf`, 2017.

Jiatao Gu, Changhan Wang, and Junbo Zhao. Levenshtein transformer. In *NeurIPS*. 2019.

Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *NeurIPS*. 2015.

Babak Hassibi and David G. Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *NeurIPS*. 1993.

K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask R-CNN. In *ICCV*, 2017.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

Tom Henighan, Jared Kaplan, Mor Katz, Mark Chen, Christopher Hesse, Jacob Jackson, Heewoo Jun, Tom B. Brown, Prafulla Dhariwal, Scott Gray, Chris Hallacy, Benjamin Mann, Alec Radford, Aditya Ramesh, Nick Ryder, Daniel M. Ziegler, John Schulman, Dario Amodei, and Sam McCandlish. Scaling laws for autoregressive generative modeling. *arXiv preprint arXiv:2010.14701*, 2020.

Joel Hestness, Sharan Narang, Newsha Ardalani, Gregory Diamos, Heewoo Jun, Hassan Kianinejad, Md. Mostofa Ali Patwary, Yang Yang, and Yanqi Zhou. Deep learning scaling is predictable, empirically. *arXiv preprint arXiv:1712.00409*, 2017.

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. *arXiv preprint arXiv:2102.00554*, 2021.

Itay Hubara, Brian Chmiel, Moshe Island, Ron Banner, Seffi Naor, and Daniel Soudry. Accelerated sparse neural training: A provable and efficient method to find n:m transposable masks. *arXiv preprint arXiv:2102.08124*, 2021.

Siddhant M. Jayakumar, Razvan Pascanu, Jack W. Rae, Simon Osindero, and Erich Elsen. Top-KAST: Top-k always sparse training. In *NeurIPS*. 2020.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Kenji Kawaguchi. Deep learning without poor local minima. In *NeurIPS*. 2016.

Yann LeCun, John S. Denker, and Sara A. Solla. Optimal brain damage. In *NeurIPS*. 1990.

Namhoon Lee, Thalaiyasingam Ajanthan, and Philip Torr. SNIP: Single-shot network pruning based on connection sensitivity. In *ICLR*, 2019.

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. In *ICLR*. 2017.

Zhuohan Li, Eric Wallace, Sheng Shen, Kevin Lin, Kurt Keutzer, Dan Klein, and Joey Gonzalez. Train big, then compress: Rethinking model size for efficient training and inference of transformers. In *ICML*, 2020.

T. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. Focal loss for dense object detection. In *ICCV*, 2017.

Junjie Liu, Zhe Xu, Runbin Shi, Ray C. C. Cheung, and Hayden K.H. So. Dynamic sparse training: Find efficient sparse network from scratch with trainable masked layers. In *ICLR*, 2020.

Shiwei Liu, Lu Yin, Decebal Constantin Mocanu, and Mykola Pechenizkiy. Do we actually need dense over-parameterization? In-time over-parameterization in sparse training. In *ICML*. 2021.

Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: Single shot multibox detector. *arXiv preprint arXiv:1512.02325*, 2015.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. RoBERTa: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. In *ICCV*, 2017.

Paul Michel, Omer Levy, and Graham Neubig. Are sixteen heads really better than one? In *NeurIPS*, volume 32, 2019.

Asit Mishra, Jorge Albericio Latorre, Jeff Pool, Darko Stosic, Dusan Stosic, Ganesh Venkatesh, Chong Yu, and Paulius Micikevicius. Accelerating sparse deep neural networks. *arXiv preprint arXiv:2104.08378*, 2021.

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.

Sharan Narang, Eric Undersander, and Gregory Diamos. Block-sparse recurrent neural networks. *arXiv preprint arXiv:1711.02782*, 2017.

NVIDIA. Deep learning examples. `https://github.com/NVIDIA/DeepLearningExamples`, 2020a.

NVIDIA. Imaginaire. `https://github.com/NVlabs/imaginaire`, 2020b.

NVIDIA. Megatron-LM. `https://github.com/NVIDIA/Megatron-LM`, 2020c.

Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. fairseq: A fast, extensible toolkit for sequence modeling. In *NAACL-HLT*, 2019.

Jongsoo Park, Sheng Li, Wei Wen, Ping Tak Peter Tang, Hai Li, Yiran Chen, and Pradeep Dubey. Faster cnns with direct sparse convolutions and guided pruning. In *ICLR*, 2017.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An imperative style, high-performance deep learning library. In *NeurIPS*. 2019.

A Radford, K Narasimhan, T Salimans, and I Sutskever. Improving language understanding by generative pre-training. `https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/language-unsupervised/language_understanding_paper.pdf2`, 2018.

Md Aamir Raihan and Tor M. Aamodt. Sparse weight activation training. In *NeurIPS*. 2020.

R. Reed. Pruning algorithms-a survey. *IEEE Transactions on Neural Networks*, 1993.

Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *NeurIPS*. 2015.

Alex Renda, Jonathan Frankle, and Michael Carbin. Comparing rewinding and fine-tuning in neural network pruning. In *ICLR*, 2020.

Joseph Lee Rodgers and W. Alan Nicewander. Thirteen ways to look at the correlation coefficient. *The American Statistician*, 1988.

Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. MobileNetV2: Inverted residuals and linear bottlenecks. In *CVPR*, 2018.

Pedro Savarese, Hugo Silva, and Michael Maire. Winning the lottery with continuous sparsification. In *NeurIPS*, 2020.

Sohil Shah, Pallabi Ghosh, Larry S. Davis, and Tom Goldstein. Stacked U-Nets: A no-frills approach to natural image segmentation. *arXiv preprint arXiv:1804.10343*, 2018.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *CVPR*, 2015. URL `http://arxiv.org/abs/1409.4842`.

Hidenori Tanaka, Daniel Kunin, Daniel L. K. Yamins, and Surya Ganguli. Pruning neural networks without any data by iteratively conserving synaptic flow. In *NeurIPS*, 2020.

Neil C. Thompson, Kristjan Greenewald, Keeheon Lee, and Gabriel F. Manso. The computational limits of deep learning. *arXiv preprint arXiv:2007.05558*, 2020.

Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers distillation through attention. *arXiv preprint arXiv:2012.12877*, 2020.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*. 2017.

Chaoqi Wang, Guodong Zhang, and Roger Grosse. Picking winning tickets before training by preserving gradient flow. In *ICLR*, 2020.

Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Guilin Liu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. Video-to-video synthesis. In *NeurIPS*, 2018a.

Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. High-resolution image synthesis and semantic manipulation with conditional gans. In *CVPR*, 2018b.

Ting-Chun Wang, Ming-Yu Liu, Andrew Tao, Guilin Liu, Jan Kautz, and Bryan Catanzaro. Few-shot video-to-video synthesis. In *NeurIPS*, 2019.

Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *NeurIPS*. 2016.

Mitchell Wortsman, Ali Farhadi, and Mohammad Rastegari. Discovering neural wirings. In *NeurIPS*. 2019.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2. `https://github.com/facebookresearch/detectron2`, 2019.

Fisher Yu, Vladlen Koltun, and Thomas Funkhouser. Dilated residual networks. In *CVPR*, 2017.

A. Zhou, Y. Ma, J. Zhu, J. Liu, Z. Zhang, K. Yuan, W. Sun, , and H. Li. Learning n:m fine-grained structured sparse neural networks from scratch. In *ICLR*, 2021.

M. Zhu and S. Gupta. To prune, or not to prune: Exploring the efficacy of pruning for model compression. In *ICLR*. 2018.

Difan Zou and Quanquan Gu. An improved analysis of training over-parameterized deep neural networks. In *NeurIPS*, 2019.

A A PRIMER ON SEARCH SPACES

We expand our investigations on the role of search spaces in neural model training to cover more neural architectures and deep learning tasks.

Figure 6 shows the fraction of weights whose magnitudes remains above (active set) or below (inactive set) a threshold until the end of training, where the threshold is chosen such that one-fourth of the weights remain for inference tasks. Similar as observed in Section 2.1, we find neural weights that are needed for inference (or active set) are discovered throughout rather than early in training for other tasks as well.

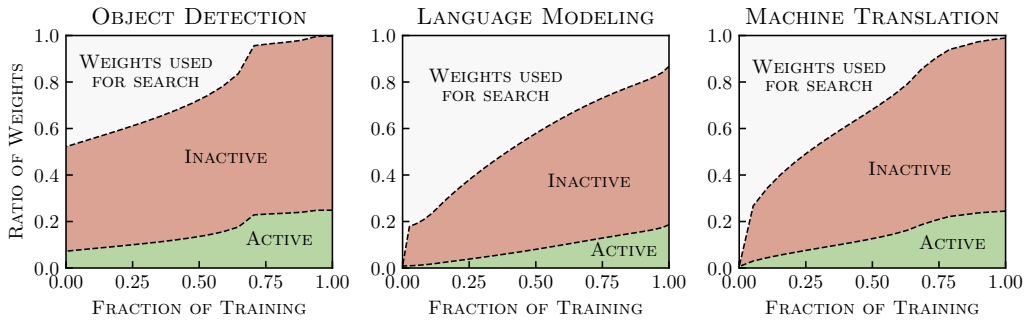

Figure 6: Same as Figure 1 (left). From left to right: Mask-RCNN (Object Detection), Transformer-XL (Language Modeling), GNMT (Machine Translation).

Figure 7 illustrates cumulative distances (normalized by the maximum) that neural models traverse in search space as a function of the number of weights being added to the active set. Besides vision tasks, we find adding weights also help neural models better explore search spaces for language tasks.

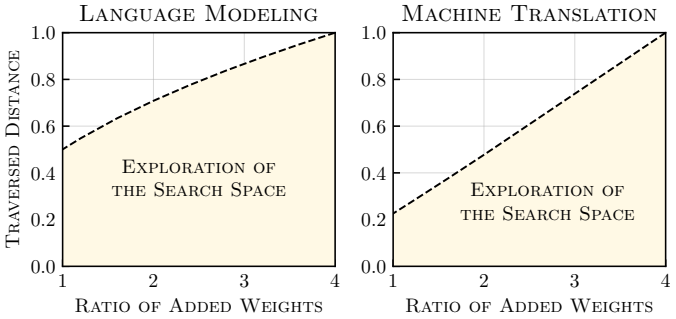

Figure 7: Same as Figure 1 (middle). From left to right: Transformer-XL (Language Modeling), GNMT (Machine Translation).

Figure 8 shows the fraction of time (Δ) it takes for weights to decorrelate as a function of magnitudes obtained after training. Since their correlations are short-termed, added weights (or weights that obtain small values) do not learn meaningful representations over time, rather they provide alternate paths for optimization during training to escape critical points.

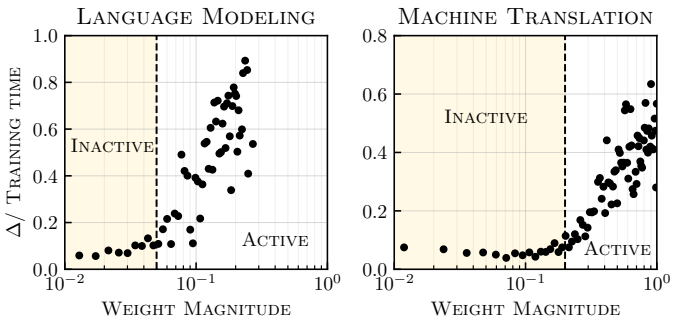

Figure 8: Same as Figure 1 (right). From left to right: Transformer-XL (Language Modeling), GNMT (Machine Translation).

Similar conclusions can be drawn as in Section 2, providing more evidence that the size of search spaces influences neural model training.

B Recommendations for training

We expand our investigations on how to approximate the behavior of larger neural models (or wider search spaces) when training sparse models, covering more neural architectures and deep learning tasks.

Figure 9 plots the task error as a function of rewiring steps r for sparse models of different sizes d, where d denotes the ratio of weights being used. We observe that error increases with less frequent rewiring ($r \to \infty$) for other vision tasks, since rewiring is related to how often the search space expands.

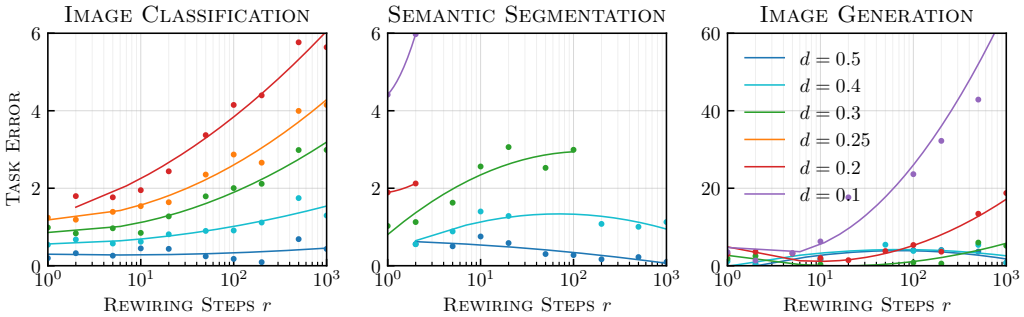

Figure 9: Same as Figure 3 (left). From left to right: InceptionV3 (Image Classification), Mask RCNN (Semantic Segmentation), Pix2PixHD (Image Generation).

Figure 10 shows the task error as a function of the scale factor s applied to gradient updates for non-participating weights. We observe the error increases with decreasing contributions of the gradients ($s \to 0$), which suggests updates to non-participating weights are also important during training for other tasks.

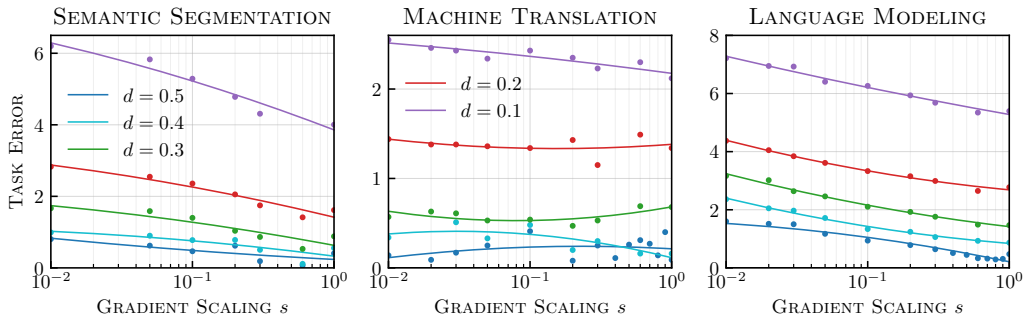

Figure 10: Same as Figure 3 (middle). From left to right: Mask R-CNN (Semantic Segmentation), Transformer (Machine Translation), Transformer-XL (Language Modeling).

Figure 11 demonstrates the task error as a function of the number of training steps z at which non-participating weights are reset to zero. Similar to results in Section 3.2, error rates saturate after sufficient training ($z \sim 1k$), which reinforces the idea that non-participating weights augment search spaces rather than model capacity.

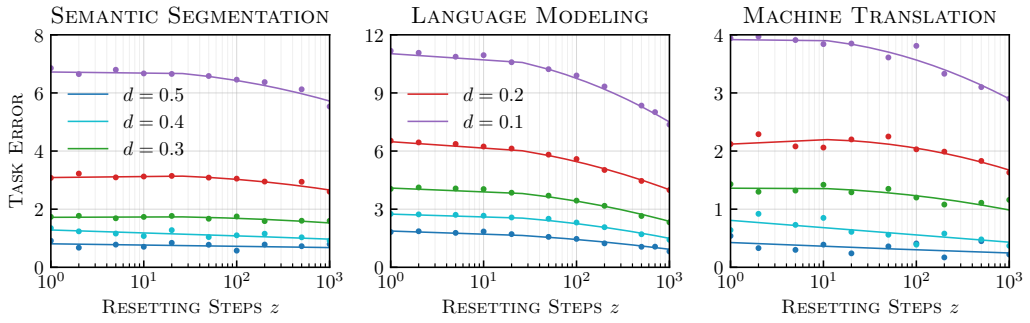

Figure 11: Same as Figure 3 (right). From left to right: Mask RCNN (Semantic Segmentation), Transformer-XL (Language Modeling), GNMT (Machine Translation).

Figure 12 compares various exploration and exploitation strategies for training sparse models, as described in Section 3.3. While task accuracy degrades with lack of proper exploration or exploitation, inducing exploitation by removing gradient noise from non-participating weights (Fix, Reset, Regularize) substantially decreases the error rates.

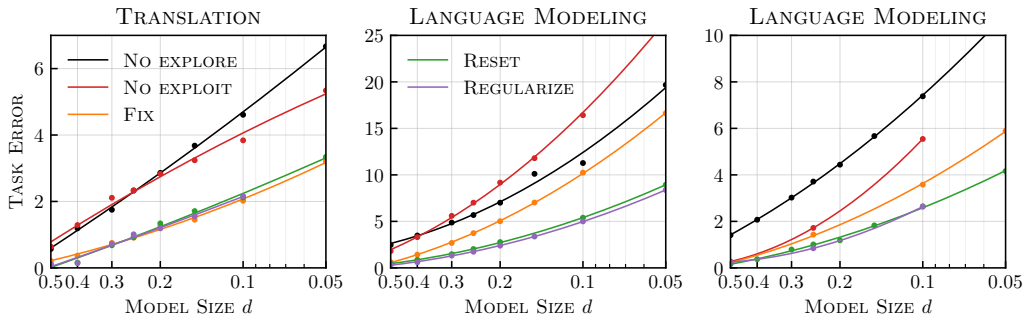

Figure 12: Same as Figure 4. From left to right: Transformer (Translation), Transformer-XL Base (Language Modeling), Transformer-XL Large (Language Modeling).

The above results across more deep learning workloads further validate recommendations put forth in Section 3 for training sparse models.

C Methodology

Algorithm 1 summarizes possible methods using recommendations described in the paper.

Algorithm 1 Train($\mathcal{D}, \gamma, d, v, z, \beta$)

```
Initialize neural weights w at random
for each training iteration t do
    Sample a mini batch of data D
    p ← w_i if w_i ≥ τ, where τ is chosen so that |p| = d        ▷ Get participating weights
    n ← w_i if w_i < τ                                           ▷ Get non-participating weights
    ℓ ← L(p, D)                                                  ▷ Forward pass
    ∂ℓ/∂w ← ∂L(p,D)/∂w                                           ▷ Backward pass
    w ← w + γ ∂ℓ/∂w                                              ▷ Optimizer step
    if t ≥ v then n ← 0                                          ▷ Fix
    if t mod z = 0 then n ← 0                                    ▷ Reset
    n ← n − βn                                                   ▷ Regularize
```

We rewire weights based on magnitude in order to preserve long-term correlations that represent learning. At each point in time, the top d-proportion of weights in each layer of a neural model participate in training, and the rest do not participate. We use the participating weights to compute loss and gradients, while optimizers perform gradient updates for both participating and non-participating weights (equivalent to a straight-through-estimator).

For example, the forward stage computes $x_{\ell+1} = p_\ell x_\ell$ recursively for an input x and participating weights p, which is then fed into the loss function L. The backward stage derives two sets of gradients: activation gradients from layer ℓ are passed to downstream layer $\ell - 1$ using $\partial L / \partial x_{\ell-1} = \partial L / \partial x \times p_\ell^T$, and weight gradients from layer ℓ are computed as $\partial L / \partial w_\ell = \partial L / \partial x_\ell \times x_{\ell-1}^T$, where w denotes all of the weights in a neural model.

Training observes no memory storage savings, since we perform gradient updates for all the weights. The equations above can also only accelerate forward stages, and parts of backward stages that compute activation gradients. For further acceleration, we can consider applying sparsity to parts of backward stages that compute weight gradients (Raihan & Aamodt, 2020).

We also note the methods above makes a tradeoff between exploration and exploitation. While approximating wider search spaces enhances more exploration, restricting non-participating weights enables exploitation. v, z, and β dictate how and when to switch between these two phases. These variables can either be automated or kept as default, and used either in isolation or combined.

D IMPLEMENTATION DETAILS

We design experiments on PyTorch (Paszke et al., 2019) using custom autograd functions for convolutions and linear layers. Our functions emulate sparsity using a binary tensor (or mask) that we multiply elementwise with the weights of each layer during forward and backward propagation. We determine masks based on weight magnitudes, as described earlier, after the optimizer step and before the next training iteration.

SEARCH. We approximate wider search spaces as described in the previous appendix. We find the best tradeoff between exploration and exploitation using variables v, z, and β. Stopping rewiring after half of training ($v = 0.5$) often provides the best results. z should be large enough to keep short-term correlations (on the order of $1k$ training steps). $\beta = 0.0002$ chosen in (Zhou et al., 2021) roughly matches our choice for z. All methods perform equally well with their optimal variables, so we can take different choices across workloads. For most experiments we adopt RESET and use FIX in a few select cases.

REDUCE. For smaller models, we reduce the widths of neural layers by a factor of $1/d$ to match the number of weights used in sparse models. In some cases, we approximate this behavior by applying sparsity once at initialization, which represents an upper bound since tensors retain their dimensionality.

LOTTERY. We construct lottery tickets by training neural models to completion (e.g., k steps) and computing masks based on their trained weights Frankle & Carbin (2019); Frankle et al. (2019). We initialize sparse models with the original initialization ($t = 0$) or after some amount of training ($t = \epsilon$), and train them for $k - t$ steps using the same hyperparameters. Following (Frankle et al., 2019), we choose ϵ between $k/10$ ad $k/100$ across various workloads. While larger values for ϵ can deliver better accuracy, this comes at the cost of training acceleration, also neural models are initialized closer to their solutions (Evci et al., 2020b).

SET AND RIGL. We rewire participating weights over time by removing their weakest values and either adding new ones randomly (Mocanu et al., 2018) or based on their gradients (Evci et al., 2020a). We initialize weights to zero when they become participating. The fraction f of weights to rewire decays linearly throughout training, which works as well as more complex schedules (such as cosine and inverse power). We choose the best initial value for f and rewiring frequency across various workloads.

E EXPERIMENTAL SETUP

E.1 IMAGE CLASSIFICATION

We train popular convolutional models like ResNets (He et al., 2016), VGG (Simonyan & Zisserman, 2015), Stacked U-Nets (Shah et al., 2018), Dilated Residual Networks (Yu et al., 2017), Inception (Szegedy et al., 2015), MobileNet (Sandler et al., 2018), as well as vision transformers like DeiT (Touvron et al., 2020). Training involves standard pipelines for image classification on ImageNet-2012 as described in literature and found in public code repositories. For most workloads, we adopt learning rates with linear warmups for the first 5 epochs, drop the learning rate by a factor of ten at epochs 30-60-80, and stop training after 90 epochs. A few select neural models (e.g., mobilenets and vision transformers), however, are trained for more epochs using linear or cosine schedules.

We measure model quality using top-1 classification accuracy. We apply sparsity to convolutions and linear layers with some exceptions: convolutions whose input channels are not divisible by 16 (e.g., first convolution layer, group and depthwise separable convolutions).

E.2 IMAGE SEGMENTATION AND DETECTION

Image segmentation and detection tasks include both regression and classification components. Popular detectors and segmentors are typically trained in two phases: first a backbone is trained for image classification, followed by the addition of model components that are trained for detection

or segmentation. Backbones are trained on ImageNet-2012, while downstream tasks are trained on COCO. We adapt training scripts and code from Detectron2 (Wu et al., 2019).

We train neural models such as regions with convolutional neural networks (R-CNN) variants (Ren et al., 2015; He et al., 2017), vanilla one-shot detectors (Liu et al., 2015), and with focal loss (Lin et al., 2017). Convolution and linear layers encountered in pretrained backbones are sparse, like for classification tasks. Detection and segmentation heads are also targeted.

E.3 Generative modeling

Generative Adversarial Networks (GANs) contain two subnetworks: a generative model and a discriminative model which combine regression and discrimination tasks during training. For image and video tasks, the generator model regresses pixel colors. We explore conditional GANs for super image-to-image translation and video-to-video synthesis using Imaginaire (NVIDIA, 2020b). We measure quality of generated outputs using the Frechet Inception Distance (FID). We experiment with generative neural models like Pix2PixHD (Wang et al., 2018b), Vid2Vid (Wang et al., 2018a), and FewShot-Vid2Vid (Wang et al., 2019), targeting convolution and linear layers.

E.4 Machine Translation

We explore transformer and recurrent neural models for language translation. All models are encoder-decoder style architectures trained for English to German (En-De) translation on WMT. We adapt model and training code from Fairseq (Ott et al., 2019) and NVIDIA Deep Learning Examples (NVIDIA, 2020a). We measure model quality using BLEU scores.

We experiment with GNMT (Wu et al., 2016) and transformer-based architectures (Vaswani et al., 2017; Gu et al., 2019). All linear layers are sparse, except for embeddings and vocabulary projections.

E.5 Language modeling

We consider recent advances in word-level language modeling using transformer decoder (left-to-right) or encoder (bi-directional) architectures (Radford et al., 2018; Devlin et al., 2018). We pretrain language models in an unsupervised fashion on WikiText-103 or Wikipedia corpus, and evaluate on downstream tasks that are zero shot or require additional finetuning. We train them using Megatron (NVIDIA, 2020c) and NVIDIA Deep Learning Examples (NVIDIA, 2020a). Model quality is measured in terms of perplexity or F1 score.

We train language models such as Transformer-XL Dai et al. (2019) and BERT Devlin et al. (2018). We make all linear layers sparse, except for embeddings, vocabulary projections, and classification heads for downstream tasks.

F Search capacity

It is also interesting to understand how well can our sparse models approximate wider search spaces. To this end, we measure where does accuracy of sparse models (Search) fall between accuracies of larger neural models with free search (Regular) and smaller ones that operate on reduced spaces (Reduce). We designate a metric $f = (\text{Regular} - \text{Search})/(\text{Regular} - \text{Reduce})$ to represents the capacity for search, where a value of one means accuracy matches that of free search, and zero implies accuracy is no better than without any additional search.

Figure 13 illustrates f as a function of size d for various tasks. For $d \sim 0.5$, sparse models are able to approximate wider search spaces found in larger models across all neural architectures and tasks. This helps explain why models with moderate amounts of sparsity are less sensitive to search constraints. While f decreases for sparser models ($d \ll 0.5$), they are still much more efficient than trivially smaller models (obtained by reducing widths of neural layers). Even at $d = 0.1$, most models can capture three-quarters of the possible search capacity.

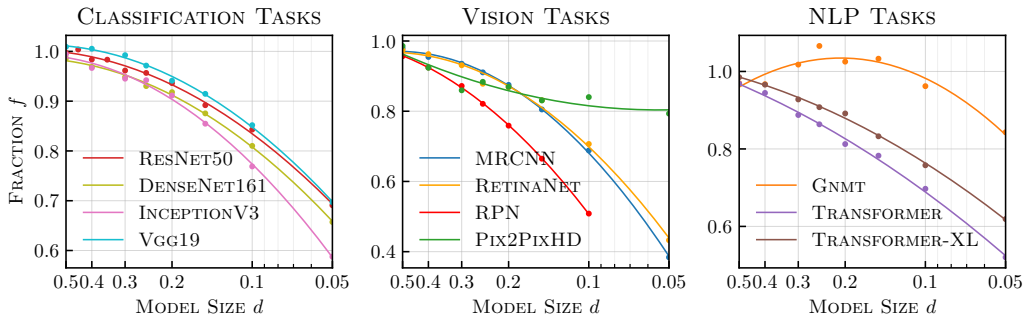

Figure 13: Fraction of accuracy that sparse models achieve between regular and smaller models as a function of model size d across various tasks and neural architectures. We measure this fraction as $f = (\text{Regular} - \text{Search})/(\text{Regular} - \text{Reduce})$ which represents capacity for search.

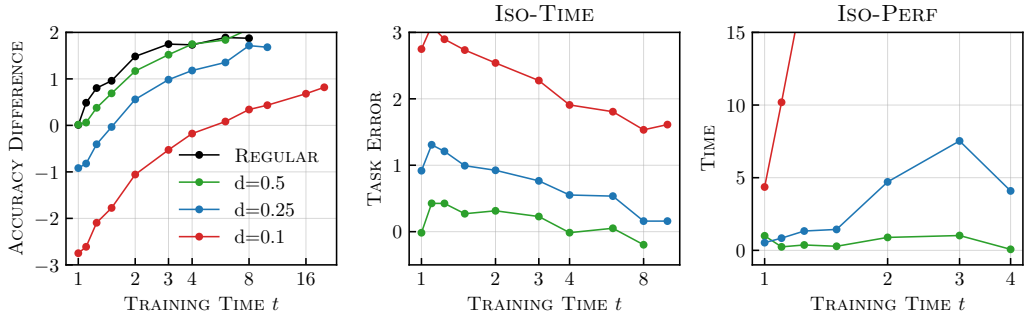

Figure 14: Investigations into the effects of sparsity on longer training of ResNet18. Left: Accuracy differences as a function of training time t (or ratio of steps to the original training schedule). Middle: Task error between regular and sparse models trained for the same duration. Right: Training time it takes for sparse models to match accuracy of regular models. Appendix G covers data for a broader set of neural models and tasks.

G Effects of longer training

Since conventional models can achieve higher accuracy when trained on larger data sets or with longer training schedules (Liu et al., 2019), another interesting direction is to explore the effects of sparsity on models that are trained to the limits of their capacity. We train neural models longer by extending their learning rate schedules after warmup , e.g. a schedule $1/t$ becomes $2/t$.

Figure 14 (left) plots accuracy differences as a function of training time t, which denotes the ratio of training steps to the original training schedule. The fact accuracy improves with more training suggests neural models are often not trained to capacity using conventional schedules. We observe sparse models achieve worse accuracy than regular models that are undertrained ($t \sim 1$), but can match accuracy when trained to capacity ($t \gg 1$).

Figure 14 (middle) illustrates task errors between regular and sparse models that have been trained for the same number of steps. We find error scales inversely with t, since more training gives neural models better chances to explore search spaces. Particularly, errors can reach zero with sufficient training (Evci et al., 2020a) for models that are not constrained by capacity ($d \geq 0.25$). Figure 14 (right) shows the time it takes for sparse models to recover accuracy of regular models is relatively short when $t \sim 1$, but significantly longer as accuracy saturates ($t \gg 1$). This time also increases for smaller model sizes, reminiscent of efforts that seek to undertrain massive neural models for efficiency (Kaplan et al., 2020; Li et al., 2020).

We expand our investigations on the effects of longer training to cover more neural architectures and deep learning task. Figure 15 illustrates accuracy deltas as a function of training time t. For vision tasks, we find sparse models of moderate sizes ($d \geq 0.25$) can match accuracy of regular models after sufficient training ($t \geq 3$). On the other hand, for language modeling, sparse models cannot match accuracy for any time t because regular models are already near capacity for the task at hand. Obviously, when using popular training schedules ($t \sim 1$), sparse models can be trained a bit longer to recover the lost accuracy.

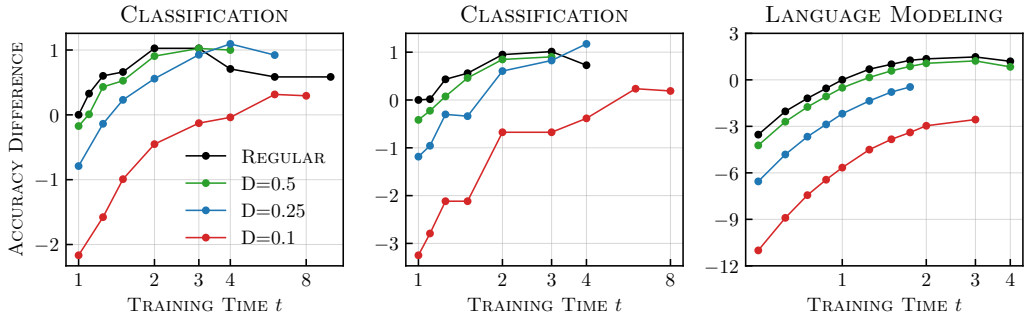

Figure 15: Same as Figure 14 (left). Left to right: ResNet50 (Classification), InceptionV3 (Classification), Transformer-XL (Language Modeling).

H Application on hardware accelerators

This appendix discusses various structures (or collection of weights in a sparse model) considered in the paper that are amenable for acceleration using modern matrix-math hardware.

H.1 Block sparsity

We first look at block sparsity (Narang et al., 2017; Gray et al., 2017), which removes blocks of contiguous elements (or weights) in a neural layer as shown in Figure 16. Block sparsity addresses common issues that are present for unstructured formats: indices for active blocks reduce storage overhead by a factor of the block size, blocks are stored contiguously in memory which reduces irregular memory accesses, and their computations can exploit faster matrix-math hardware, such as Tensor Cores in NVIDIA GPUs.

We construct block sparse structures by (1) partitioning a neural layer into a set of blocks, (2) aggregating elements in each block into a metric, and (3) removing blocks according to some criteria based on their metrics. While we remove blocks based on largest magnitude $\max_{i \in (1, b^2)} w_i$. other choices such as the p-norm $(\sum_i^{b^2} |w_i|^p)^{1/p}$ achieve similar results.

Table 3: Accuracies for regular models and their differences for block sparse models using different block sizes b.

Model	Regular	Reduce	$b = 1$	$b = 2$	$b = 4$	$b = 8$	$b = 16$	$b = 32$
Transformer-XL	22.88	−2.48	−0.82	−2.03	−2.35	−2.24	−2.36	−2.41
Transformer	28.43	−0.77	−0.21	−0.75	−1.10	−1.47	−1.98	−1.94
GNMT	24.81	−2.75	−0.15	−0.15	+0.02	−0.08	−0.15	+0.30
ResNet50	76.71	−1.63	−0.46	−0.99	−2.19	−2.79	–	–
Mask RCNN	35.03	−0.88	−0.28	−1.17	−2.43	−3.74	−4.41	−5.00

Because structures restrict the combination of weights that can be formed in a neural model, an important question is then for what block sizes b (if any) can sparse models approximate wider

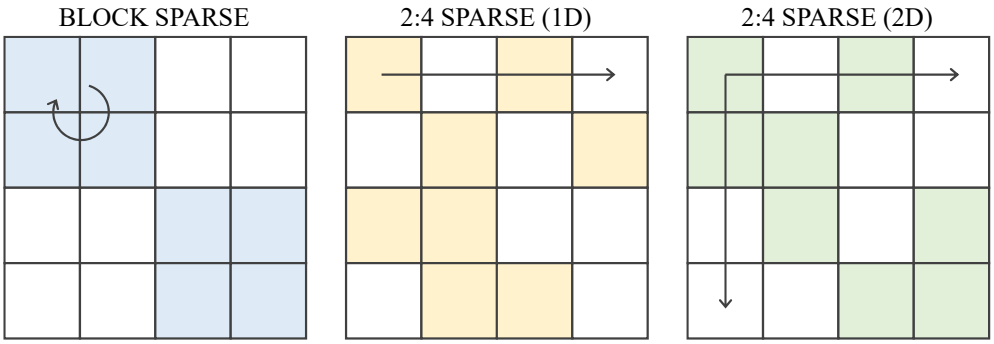

Figure 16: Block sparse, 2:4 1D, and 2:4 2D structures for a 4×4 neural layer. Blank cells represent weights that do not participate in training (are assumed zero) and colored cells denote weights that participate (have nonzero values). Arrows indicate direction along which structure is imposed.

search spaces. Table 3 lists accuracy differences between sparse and regular models for $d = 0.5$ using various block sizes. We find block sparse models fail to maintain accuracy, performing no better than smaller models with neural layers of reduced widths. Notably, accuracy deteriorates for all tasks and block sizes, including smaller blocks ($b \leq 4$) that are less amenable for hardware acceleration. In other words, block structures are too coarse for approximating wider search spaces. For example, different weights in a block may have different roles during training: a block that participates in training may contain weights that are not important, wheres a non-participating block may have weights that were crucial to keep. Both cases prevent sparse models from retaining weights over time that are relevant for search, and thus impacting accuracy.

H.2 2:4 SPARSITY

We next consider Sparse Tensor Cores (Mishra et al., 2021) introduced in NVIDIA Ampere GPU architecture which exploit 2:4 sparsity and have twice the math throughput of regular matrix units. Figure 16 shows that 2:4 sparsity mandates each group of four values must have at least two values that are zero. Typically, 2:4 is applied on weights w in the forward pass, $y = wx$. However, we can also apply 2:4 on weight transposes w^T for the backward pass, $\partial L/\partial x = \partial L/\partial y \times w^T$. We denote these two options as 2:4 1D that accelerates forward pass for inference Mishra et al. (2021), and 2:4 2D that accelerates both forward and backward passes for training.

The 2:4 sparsity structure must always be imposed along the inner dimension of dot products. For linear layers, we apply 2:4 on a $n \times k$ weight tensor along k or n (for forward or backward pass, respectively). For convolutions, we apply 2:4 on a $k \times c \times r \times s$ weight tensor along input channels c or $k \times r \times s$ (for forward or backward pass, respectively), where k denotes output channels, r and s are kernel dimensions.

We can satisfy 2:4 1D constraints by removing weights with lowest magnitudes. Since 2:4 2D constraints have no trivial solution, we seek to minimize the cumulative magnitude of the removed weights. In other words, for each 4×4 block in the tensor, we construct all possible combinations of 2:4 2D patterns, compute their 1-norm, and choose the structure that has the largest norm.

I COMPARISONS TO SPARSITY RESEARCH

We expand our comparisons to literature covering more neural architectures and deep learning tasks. Figure 17 illustrates the task error (or difference in accuracy between regular and sparse models) across various methods, neural architectures, and tasks. We find our strategy outperforms competing approaches in most cases with a few exceptions: we do not outperform lottery tickets in segmentation tasks because detectors and segmentors are trained with small learning rates which limit exploration. For generation tasks, our models are slightly worse for $d \sim 0.5$ (though noisy scores make it difficult to draw conclusions), but the asymptotic behavior of error rates as $d \to 0$ clearly indicates that our approach is superior.

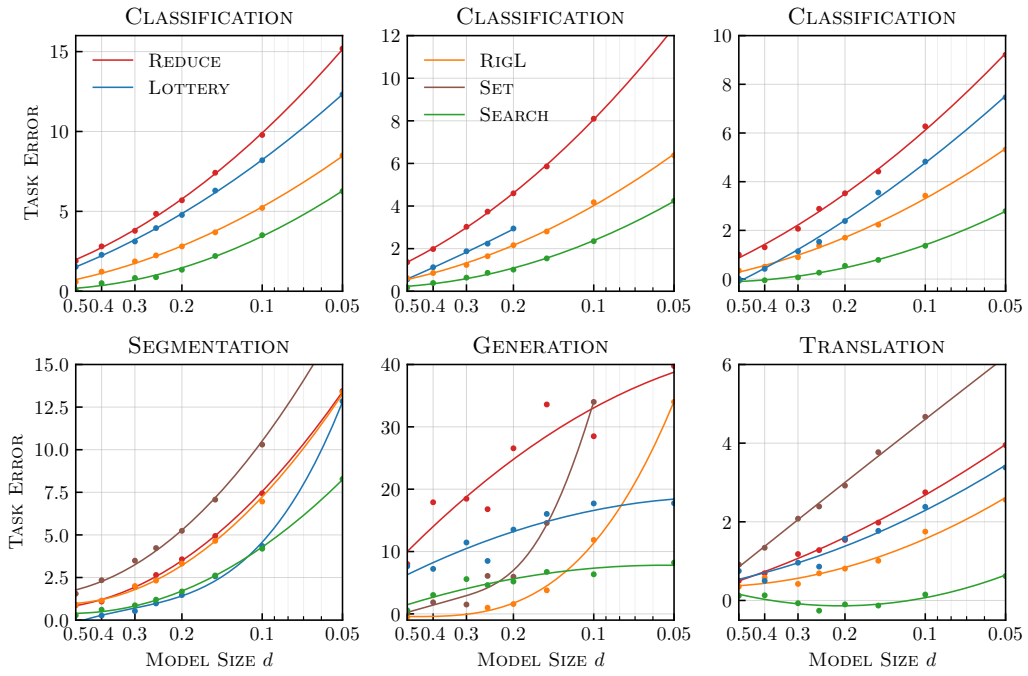

Figure 17: Same as Figure 5. Clockwise: InceptionV3 (Classification), DenseNet161 (Classification), VGG19 (Classification), GNMT (Translation), Pix2PixHD (Generation), Mask RCNN (Segmentation).