Sparsest Models Elude Pruning: An Exposé of Pruning's Current Capabilities

Stephen Zhang [1] **Vardan Papyan** [1]

Abstract

Pruning has emerged as a promising approach for compressing large-scale models, yet its effectiveness in recovering the sparsest of models has not yet been explored. We conducted an extensive series of 485,838 experiments, applying a range of state-of-the-art pruning algorithms to a synthetic dataset we created, named the Cubist Spiral. Our findings reveal a significant gap in performance compared to ideal sparse networks, which we identified through a novel combinatorial search algorithm. We attribute this performance gap to current pruning algorithms' poor behaviour under overparameterization, their tendency to induce disconnected paths throughout the network, and their propensity to get stuck at suboptimal solutions, even when given the optimal width and initialization. This gap is concerning, given the simplicity of the network architectures and datasets used in our study. We hope that our research encourages further investigation into new pruning techniques that strive for true network sparsity.

1. Introduction

The burgeoning complexity of state-of-the-art deep learning models has made their training and deployment prohibitively expensive. To counteract this increasing demand for resources, model compression has become increasingly important in optimizing the computational efficiency of these networks. Among these techniques, a popular and proven option is network pruning which induces sparsity in the model parameters (Hoefler et al., 2021).

Whilst pruning is effective, it is difficult to assess how close current pruning algorithms are to obtaining the sparsest of models due to the complexity of the datasets and models used. For the same reason, analyzing and interpreting pruning's effects on trained models has been challenging which has allowed for potential shortcomings to go unnoticed.

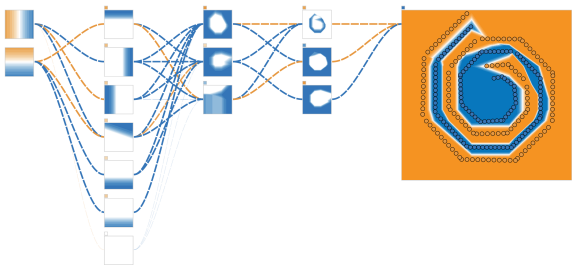


Figure 1. **Sparse Model Visualization.** Visualization of a sparse model, discovered through our combinatorial search algorithm, trained on the Cubist Spiral dataset. The first two squares on the left denote the input variables, while the final, larger square depicts the output from the classifier. The intermediate squares reveal post-activation states which are connected by edges, corresponding to entries in weight matrices. At the top of each square, there is a tiny square that is colored according to the bias of the corresponding neuron. Blue is used to represent a positive value, orange a negative value, and white – a value of zero.

This paper aims to scrutinize how closely various pruning algorithms approach the ideal, sparsest network, defined as the model with the fewest nonzero parameters that can achieve a specific target accuracy, and reveal the true efficacy of current pruning techniques.

1.1. Method Overview

To achieve our goals, we engineer the following tools that will be the basis for our analysis:

Cubist Spiral A synthetic dataset named the *Cubist Spiral*, depicted in Figure 2b. The simplicity inherent in the dataset leads to interpretable sparse models that are amenable to visualization and analysis.

Combinatorial Search A novel combinatorial search algorithm that searches across model sparsity masks for an optimal and maximally sparse model. Diverging from existing naive benchmarks such as random pruning, where the pruned weights are selected randomly, our algorithm leverages structured sparsity to perform

[1]Department of Mathematics, University of Toronto, Toronto, Canada. Correspondence to: Stephen Zhang <stephenn.zhang@mail.utoronto.ca>.

Proceedings of the 41^{st} International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

an efficient exploration across sparsity masks for the model.

Sparse Model Visualization A visualization tool, similar to TensorFlow Playground (Smilkov et al., 2017), designed for inspecting sparse models by graphically representing their non-zero subnetwork. An example of a model visualization is shown in Figure 1.

1.2. Contributions

Through an empirical study (code available on GitHub), we uncover the following deficiencies:

Algorithms Fail to Get Sparsest Model There exists a disparity between the achievable outcomes and the current capabilities of pruning techniques in terms of recovering a sparsest model.

Overparameterization Impedes Pruning Unstructured pruning techniques are unable to adequately perform structured pruning resulting in a deterioration of their performance under overparameterization.

Pruning Fails Under Optimal Conditions Pruning is unable to recover the sparsest sparsity masks for the model even when provided with the optimal width and initialization.

Through our visualization, we show:

Disconnected Paths Pruning algorithms are unable to correctly align the parameters of consecutive layers resulting in disconnected paths. This leads to an inflated number of nonzero parameters that are not contributing to the expressivity of the network.

Pruning Algorithms Foregoes Sparsity Pruned networks can be further pruned after training without harming model performance.

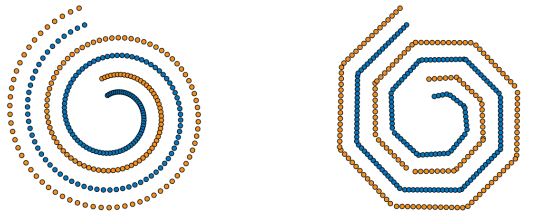
2. Background: Pruning Algorithms

Pruning algorithms are commonly classified into two main categories: structured and unstructured. In unstructured pruning, individual weights are pruned, whereas structured pruning operates at a higher level by pruning entire filters or channels (Wen et al., 2016; Li et al., 2017; Luo et al., 2017).

Beyond structured and unstructured, pruning algorithms can also be classified into the following three categories based on their pruning strategies.

2.1. Dense to Sparse

This category encompasses the methods of Optimal Brain Damage by LeCun et al. (1989) and its successor, Optimal Brain Surgeon by Hassibi & Stork (1992); Hassibi et al. (1993), which are seminal works not only in dense-to-sparse pruning but in pruning in general. More recently, magnitude-based pruning approaches have proven to be extremely effective (Han et al., 2015), leading to state-of-the-art methods being developed such as Gradual Magnitude Pruning (GMP) by Zhu & Gupta (2018) and the Lottery Ticket Hypothesis (LTH) by Frankle & Carbin (2018). These techniques typically start with a dense network configuration and implement pruning either progressively during the training process or upon its completion. While they offer resource savings during the inference stage, they do not reduce resource utilization during the training phase.



(a) Spiral dataset (b) Cubist Spiral dataset

Figure 2. Comparative view of spiral datasets.

2.2. Pruning at Initialization

Representative methods in this category include Gradient Signal Preservation (GraSP) by Wang et al. (2019), Prospect Pruning (ProsPr) by Alizadeh et al. (2021), Single-shot Network Pruning (SNIP) by Lee et al. (2019), Iterative Synaptic Flow Pruning (SynFlow) by Tanaka et al. (2020) and Iter SNIP and FORCE by de Jorge et al. (2021). In contrast to the previous category, these algorithms involve pruning neural networks at the initialization stage, followed by training the already-pruned models. This approach is beneficial as it conserves resources both during the training and inference phases, assuming the initial pruning overhead is negligible.

2.3. Sparse to Sparse

Sparse Evolutionary Training (SET) by Mocanu et al. (2018) was the pioneer algorithm in this category. Subsequently, several other algorithms have been introduced, such as Deep-R by Bellec et al. (2018), Sparse Networks From Scratch (SNFS) by Dettmers & Zettlemoyer (2019), and Dynamic Sparse Reparameterization (DSR) by Mostafa & Wang (2019). The Rigged Lottery (RigL) by Evci et al. (2020a) has emerged as a state-of-the-art method in this group. Distinguishing itself from other categories, this approach initiates with a sparsely connected neural network and maintains the total number of parameters while dynamically altering the nonzero connections throughout training.

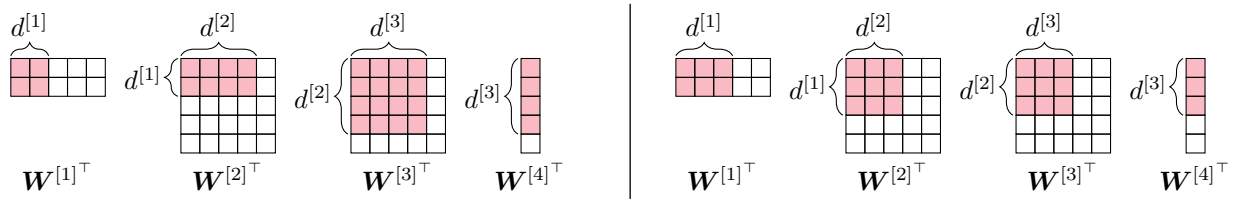


Figure 3. **First Phase.** Two structured sparsity masks that would be tested by the first phase of the combinatorial search. It is always the first $d^{[\ell-1]}$ columns and $d^{[\ell]}$ rows that are nonzero inside the masks, denoted by the light red squares. If both sets of masks reach the target accuracy, the set of masks on the right will be utilized by the second phase as it contains fewer nonzeros.

3. Methodology

3.1. Network

The objective of this study is to evaluate the effectiveness of pruning algorithms in identifying the sparsest possible network. Finding it requires a combinatorial search which is only practical for smaller network architectures, due to its complexity.

We therefore train four-layer Multilayer Perceptrons (MLPs) with ReLU activation functions, which take as input two coordinates and predicts a class label.

All combinatorial search experiments are done on MLPs of width 16. The pruning algorithms, on the other hand, are run on MLPs of varying widths:

$$\{3, 4, 5, 6, 7, 8, 16, 32, 64, 128, 256\},$$

to examine the impact of overparameterization on their efficacy.

3.2. Dataset

The simplicity of the architecture calls for a simple dataset as well. We opt for the classical synthetic spiral dataset, notable for its non-linear separability. To better suit sparse modeling techniques, we have adapted the spiral by straightening its naturally curved edges. This modification gives rise to what we call the *Cubist Spiral* dataset, a nod to the Cubism art movement that emphasized the use of minimal geometric shapes when depicting objects of interest. The classical spiral and its Cubist counterpart are juxtaposed in Figure 2.

We pick $50,000$ points spaced evenly along the spiral divided equally between the two classes. This deliberate choice of a large training set stems from our desire to separate any issues related to generalization when evaluating the efficacy of pruning algorithms.

3.3. Combinatorial Search

The combinatorial search is encapsulated in a function which obtains as input the width of the network D and a desired target accuracy ρ and returns a list of model sparsity masks for the MLP. The function involves two phases.

First Phase: Structured Sparsity The first phase performs a grid search over the number of neurons in each layer that span over the set $\{1, 2, ..., D\}$. We denote the number of neurons in layer ℓ as $d^{[\ell]}$ with $d^{[0]} = 2$ and $d^{[4]} = 1$. For each neuron configuration, a four-layer MLP is randomly initialized and masked such that only the first $d^{[\ell-1]}$ columns and $d^{[\ell]}$ rows in layer $\boldsymbol{W}^{[\ell]}$ are nonzero. The MLP is then trained and a final accuracy is computed. Given the results from all the trainings, the configuration that achieves the desired target accuracy with the fewest nonzeros is selected. A schematic showing how phase one operates is displayed above in Figure 3.

Second Phase: Unstructured Sparsity The second phase iterates over a list of unstructured sparsity masks for each weight matrix.[1] For weight $\boldsymbol{W}^{[\ell]}$, this list is generated by the function $\texttt{EligibleMasks}(d^{[\ell-1]}, d^{[\ell]})$ where $d^{[\ell-1]}$ and $d^{[\ell]}$ are determined based on the optimal configuration established in the first phase.

To ensure that the combinatorial search is done efficiently, $\texttt{EligibleMasks}(d^{[\ell]}, d^{[\ell-1]})$ ensures that each mask for the weight is confined to the rows and columns essential for fulfilling the neuron configuration. Furthermore, each required row and column contains at least one nonzero element. This assumption is grounded in the notion that the neuron configuration, as determined in the first phase of the search, is inherently minimal.

$\texttt{EligibleMasks}(d^{[\ell]}, d^{[\ell-1]})$ further optimizes the combinatorial search by eliminating masks that are functionally identical but differ merely by permutations of channels. The symmetry is broken by selecting from all row permutations the specific arrangement that results in a sequentially decreasing count of nonzero elements. If two rows have an equal count of nonzeros, the algorithm converts the binary vector representations of these masks into their decimal equivalents and arranges them in descending order. A schematic for phase two is depicted below in Figure 4.

[1] The combinatorial search iterates only over the masks of the weight matrices. As for the biases, we assign a value of zero to the i-th bias entry if and only if the i-th row of the weight matrix in that layer is zero in the mask.

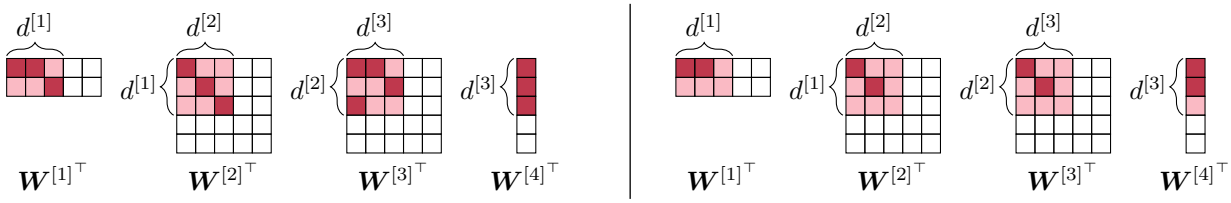


Figure 4. **Second Phase.** A schematic illustration of the second phase of the combinatorial search. **Left:** A set of unstructured sparsity masks that would be tested in the second phase, generated by utilizing the minimal structured sparsity masks found in the first phase. The dark red squares denote the nonzero entries in the unstructured sparsity masks and the light red squares denote the nonzero entries of the minimal structured sparsity masks found in the first phase. **Right:** An ineligible mask containing rows and columns without at least one nonzero element, which does not fully utilize the minimal number of neurons identified in the first phase.

This process provides a list of sparsity masks for each weight matrix which the combinatorial search then combines to form a list of masks for the model. These masks are then applied to models that are randomly initialized prior to any training. The algorithm for the combinatorial search is detailed in Algorithm 1 in the Appendix.

3.4. Selection of Pruning Algorithms

We benchmark the following pruning algorithms: GMP, LTH, GraSP, SNIP, SynFlow, Iter SNIP, FORCE, ProsPr, and RigL. We depict in Table 1 below each technique's categorization along with the FLOPS required to prune and train an MLP of width 16. We prune multiple models with different budgets of nonzeros for the weights. The budgets are specifically chosen to be centered around the range where the combinatorial search can reconstruct the spiral. Further details are provided in Appendix B and C.

Pruning Techniques	Pruning at Initialization		Dynamic Sparse Training	Dense Training Required	FLOPS Required
	One Shot	Iterative			
LTH				✓	$3.4 \cdot 10^9$
Dense Training				✓	$1.5 \cdot 10^9$
GMP				✓	$5.3 \cdot 10^8$
RigL			✓		$2.1 \cdot 10^8$
GraSP	✓				$2.0 \cdot 10^8$
ProsPr	✓				$1.9 \cdot 10^8$
SNIP	✓				$1.9 \cdot 10^8$
Iter SNIP		✓			$1.9 \cdot 10^8$
FORCE		✓			$1.9 \cdot 10^8$
SynFlow		✓			$1.9 \cdot 10^8$

Table 1. Table depicting the categorization and FLOPS required for each pruning technique that was tested in our experiments.

The aforementioned pruning techniques do not include bias parameters in the pruning process. To ensure that the comparison to the combinatorial search is fair, entries of the bias are masked based on whether the corresponding column in the succeeding weight matrix is fully pruned, i.e., $b_i^{[l]}$ is masked if and only if $\boldsymbol{W}_{:,i}^{[l+1]} = 0$. The bias corresponding to the classifier layer always remains fully dense.

3.5. Initialization Experiments

The combinatorial search trains models on a large number of model masks, where each trained model starts at a different initialization of the parameters. One might speculate that the success of the combinatorial search could be tied to the initialization rather than the actual mask of the parameters.

To study the effect of the parameter initialization on the success of the pruning algorithms, we pick the best initialization from the combinatorial search – the one that led to the sparsest model for a given target accuracy, ρ – and use it to initialize the pruning experiments. If a good initialization is all that is needed for successful pruning, then the pruning algorithms should succeed and be able to match the combinatorial search.

We equalize the comparison with the combinatorial search by running another round of the combinatorial search, but this time using the most successful initialization from the first combinatorial search to account for giving the pruned models the initialization. This also serves as a sanity check to verify whether the initialization is advantageous compared to a typical random initialization.

3.6. Optimization

We train the model parameters for 50 epochs using stochastic gradient descent (SGD) with momentum 0.9 and a batch size of 128. Parameters outside of the determined model mask are constrained to be zero. A weight decay is applied for all experiments and set to $5\text{e}{-4}$. For the pruning experiments, learning rates $\{0.05, 0.1, 0.2\}$ are used while for the combinatorial search, only $\{0.05, 0.1\}$ are used. We also utilize three learning rate schedulers: constant learning rate, a cosine annealing scheduler, and a decay of 0.1 applied at epochs 15 and 30.

4. Combinatorial Search Results

4.1. Phase One of the Combinatorial Search

Preliminary experiments, which involve running the first phase of the combinatorial search with varying target sparsities, reveal three categories of model performance:

1. **Below** 95% **Accuracy:** Models in this group were unable to even approximately reconstruct the spiral.

2. **Between** 95% **and** 99.5% **Accuracy:** Models within this range partially reconstructed the spiral as a sparse combination of polygons. However, they fell short of complete accuracy due to misclassification of certain minor segments of the spiral.

3. **Above** 99.5% **Accuracy:** Models surpassing 99.5% accuracy demonstrated essentially perfect reconstruction of the spiral.

Based on this categorization, we rerun the combinatorial search twice; once with the target accuracy of $\rho_1 = 95\%$ and a second time with $\rho_2 = 99.5\%$. Figure 5 below contains a scatter plot of all the models trained in the first phase of the search.

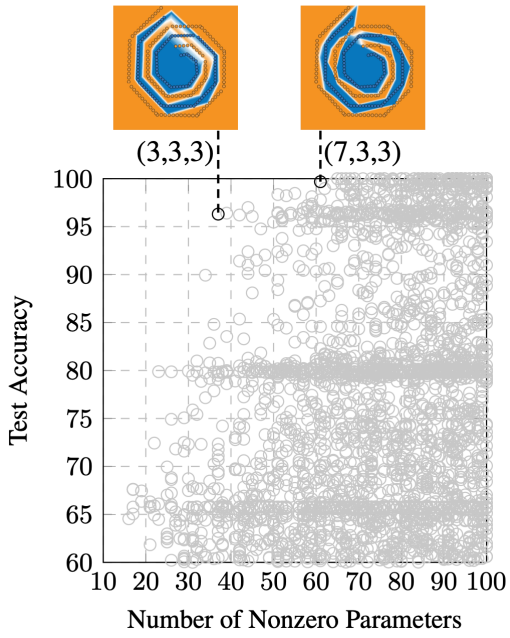


Figure 5. **Phase One of Combinatorial Search.** Scatter plot with each point corresponding to a different model that was trained with a different structured mask. Two models are highlighted – the sparsest achieving above 95% accuracy (where the number of neurons in each layer is 3,3,3) and the sparsest achieving above 99.5% accuracy (where the number of neurons in each layer is 7,3,3) – accompanied by their corresponding reconstructions of the spiral.

4.2. Phase Two of the Combinatorial Search

Phase two of the algorithm provides a total of 25,992 model masks to try for the 95% target accuracy and 266,004,066 model masks for the 99.5% target accuracy. Due to computational limits, we check only a subset of size 63,208 of the possible model masks for the 99.5% target accuracy. Details on the subset are given in Appendix D

For $\rho_1 = 95\%$, the combinatorial search found the model presented in Figure 6 below. We refer to this benchmark model as BENCH-95.

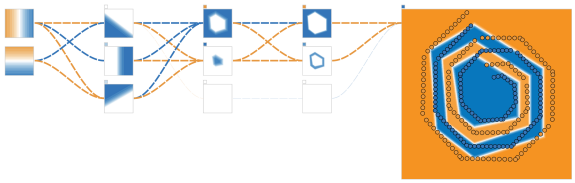


Figure 6. The minimal model found through the second phase of the combinatorial search that achieved over 95% accuracy. The model has 30 nonzero parameters and an accuracy of 96.04%.

For $\rho_2 = 99.5\%$, the combinatorial search found the model presented in Figure 7 below. We refer to this benchmark model as BENCH-995.

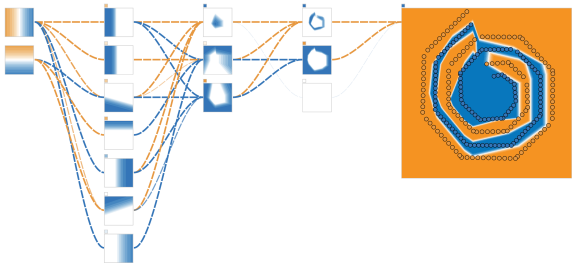


Figure 7. The minimal model found through the second phase of the combinatorial search that achieved over 99.5% accuracy. The model has 45 nonzero parameters and an accuracy of 99.59%.

4.3. Analysis of Sparse Models

Several observation can be deduced from the minimal models presented in Figures 6 and 7:

Selective Connectivity Both models exhibit selective connectivity and do not form connections with every neuron in the preceding layer. This suggests a more refined and efficient architectural design of the network.

Edges to Spiral The initial layers predominantly capture the spiral's edges. As we move deeper into the network, these edges are progressively integrated, forming polygonal shapes. In the last layers, these polygons are subtracted from one another to, roughly, reconstruct the spiral structure.

Suboptimal Sparsity The bottom neurons in the second and third layer of the model in Figure 6, and the bottom neuron in the third layer of the model in Figure 7, can be pruned to obtain a sparser model without significantly impacting the prediction of either model.

Hence, although the models are sparse, they can be further pruned. We comment on this further in Sections 5.2 and 7.2.

5. Pruning Algorithms Versus Combinatorial Search

Given the results of the combinatorial search, we run the pruning experiments with the following budgets of nonzero weights:

$$\{15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 30, 33, 37, 40, 44, 50, 53, 55, 57, 60, 65\}.$$

Figure 8 below shows the results.

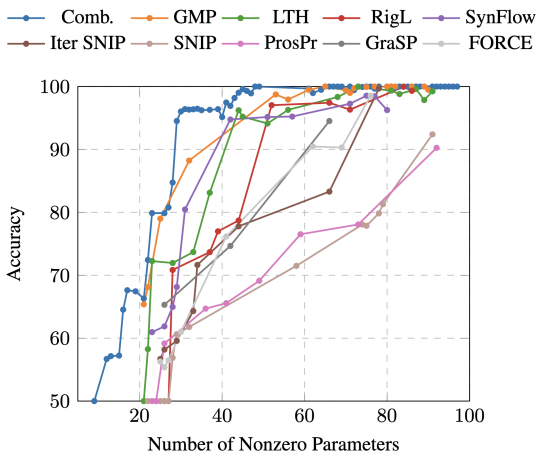


Figure 8. **Suboptimality of Pruning Algorithms.** The accuracy versus the number of nonzero parameters after training a four-layer, 16-width MLP on the Cubist Spiral dataset. A large gap is present across the board between the combinatorial search (Comb.) and the pruning algorithms, both in the $95\% - 99.5\%$ accuracy range and for accuracies above 99.5%. The Pareto frontiers are manually extracted from a scatter plot, shown in Figure 20 in Appendix F, that contains the accuracies of every pruning experiment.

5.1. Suboptimality of Pruning Algorithms

All pruning techniques suffer greatly in the $95\% - 99.5\%$ regime relative to the combinatorial search. In particular, the combinatorial search achieves above 95% accuracy with just 30 nonzeros. The second best is the dense-to-sparse method LTH requiring 44 nonzeros to reach the accuracy threshold. The sparse-to-sparse method RigL reaches the threshold at 52 nonzeros, while the sparsest pruning at initialization method that reaches the accuracy threshold is SynFlow with 51 nonzeros.

For accuracies above 99.5%, we again see a gap between pruning and the combinatorial search. The combinatorial search obtains above 99.5% accuracy with 45 nonzeros. The second-best method is GMP, which achieves a similar level of accuracy with 61 nonzeros, while RigL reaches this accuracy threshold with 84 nonzeros. The only pruning-at-initialization method that could reach the threshold within the tested nonzero budgets is Iter SNIP with 78 nonzeros and 99.69% accuracy.

5.2. Visualization of Pruned Models

To gain further insight as to why pruned models are struggling to match the combinatorial search, we visualize failed models generated by various pruning methods in Figures 9 and 10 below and comment on some key observations. Further examples are provided in Appendix H.

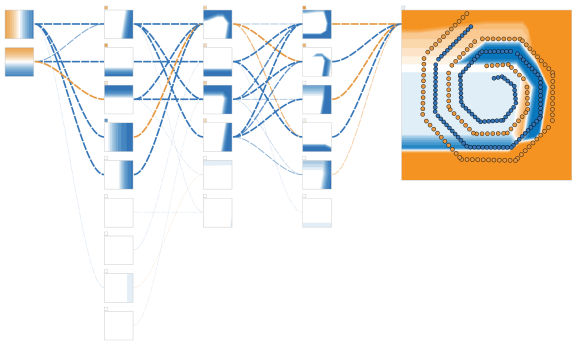


Figure 9. **ProsPr Suboptimality.** An example of a model, found through ProsPr, with 59 nonzero parameters that attains an accuracy of 76.52%. The nonzero parameters attached to the bottom four neurons in layer one, bottom two neurons in layer two, and bottom neuron in layer three form multiple disconnected paths in the network.

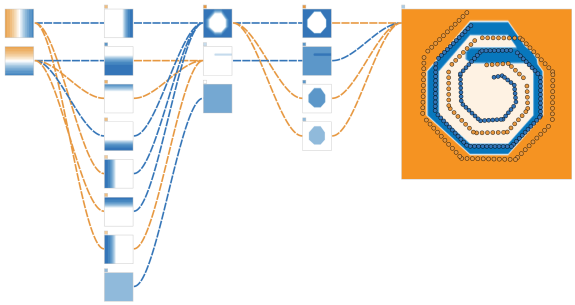


Figure 10. **GMP Suboptimality.** An example of a model, found through GMP, with 38 nonzero parameters that attains an accuracy of 84.03%. The bottom neurons in layers two and three form a disconnected path. Magnitude-wise, no weights appear to be prunable.

Disconnected Paths Current pruning algorithms are unable to properly align the weights between consecutive lay-

ers leading to *disconnected paths*. This occurs when there is a path in the network that is either disconnected from the model input or output. Nonzero parameters in a disconnected path do not contribute to the expressiveness of the network and inflate the number of nonzeros in the model.

Suboptimal Sparsity Similar to the models found by the combinatorial search, the model depicted in Figure 9 is foregoing a lot of sparsity that could be attained by magnitude pruning the model after training. This in part is due to the sub-optimal nature of current pruning algorithms requiring the final nonzero budget to be determined prior to any pruning or training being done.

6. Impact of Overparameterization on Pruning

In theory, overparameterization should be beneficial to pruning as it increases the number of combinatorial options for sparsity masks from which to find the optimal mask for a sparse model. Contrary to this belief, Figure 11 below shows the results from the experiments measuring the impact of overparameterization on the success of pruning algorithms.

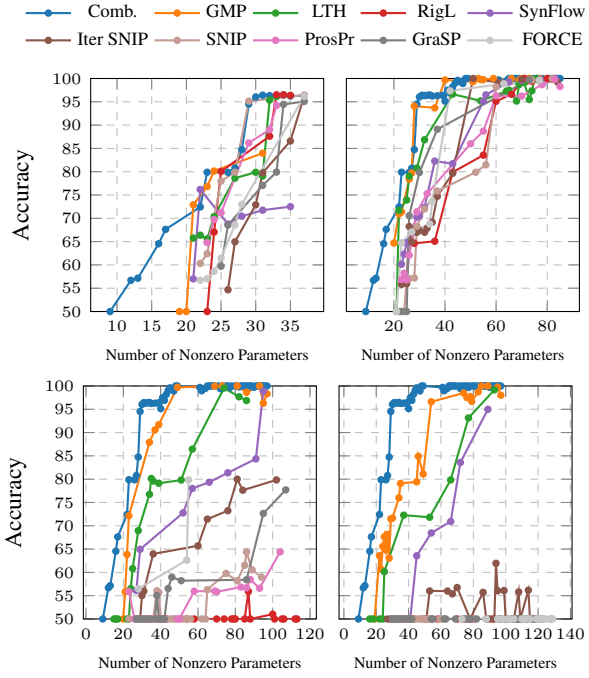


Figure 11. **Overparameterization Impedes Pruning Algorithms.** The accuracy versus the number of nonzero parameters after training four-layer MLPs of varying widths on the Cubist Spiral dataset. From left to right then top to bottom, the subplots correspond to MLPs of width 3, 7, 64, and 256. More width variations are detailed in Appendix G. The models obtained by the combinatorial search are included for reference.

6.1. Overparameterization Hinders Pruning

Figure 11 shows that overparameterization harms the performance of most pruning techniques and that at width 256, all pruning algorithms are largely failing. Furthermore, reducing the network width to 3 increases the performance of current pruning algorithms indicating that current unstructured pruning approaches are inadequately performing structured pruning. In Appendix L, we further prove that overparameterization leads to more disconnected paths for pruning methods that utilize a random mask at initialization, like RigL.

6.2. Optimal Width Limitations

Even when given the optimal width identified by the combinatorial search, the pruning methods are still unable to consistently match the accuracies that were shown to be empirically possible through the combinatorial search. Out of a total of 18,954 experiments, only two instances of pruning were able to match or beat the combinatorial search: SNIP with 29 nonzeros and an accuracy of 95.14% and GMP with 40 nonzeros and an accuracy of 99.68%. Both models were provided with the optimal widths of 3 and 7 respectively.

7. Impact of Initialization

In this section, we follow the experimental protocol detailed in Section 3.5, using the unmasked initialization of BENCH-995. Analogous experiments for BENCH-95 are included in Appendix A.

7.1. Combinatorial Search Using Optimal Initialization

Running another round of the combinatorial search[2] but this time training all models from the initialization of BENCH-995 recovers a sparser model that is visualized in Figure 12 below. Due to the neuron configuration found (6,3,2), increasing the MLP width beyond 16 would not lead to the combinatorial search finding sparser solutions. Further explanation can be found in Appendix J.

7.2. Pruning after Training is Insufficient

Figure 7 shows that pruning BENCH-995 obtained from the first combinatorial search after training will lead to a sparser model with a neuron configuration of $(7, 3, 2)$ and 42 nonzeros. However, the second combinatorial search reveals a model that has a neuron configuration of $(6, 3, 2)$ containing *less* nonzeros, 38, than what would be obtained by pruning BENCH-995. This indicates that magnitude pruning after training is insufficient to find a minimal model.

[2]Similar to the the (7,3,3) case, we do not perform an exhaustive combinatorial search over all the generated sparsity masks but rather only a subset. Details in Appendix D.

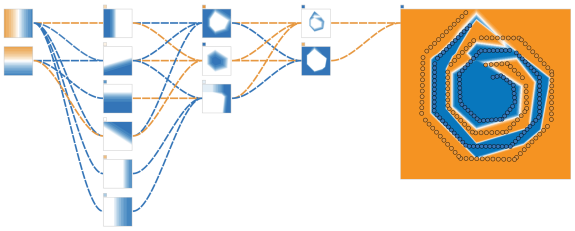


Figure 12. The minimal model found through the second run of the combinatorial search that achieved over 99.5% accuracy. The number of neurons in each layer is (6,3,2). The model has 38 nonzero parameters and an accuracy of 99.70%.

7.3. Pruning Fails with Optimal Initialization

Using the BENCH-995 initialization, Figure 13 below captures pruning's inability to recover a minimal sparsity mask for the model despite being given an ideal initialization. We can see that none of the pruning techniques are able to recover the minimal sparsity masks that the combinatorial search is able to find – even when provided with the optimal width by masking the rows and columns of each layer down to the largest width of the weight matrices of the sparsest model found by the combinatorial search.

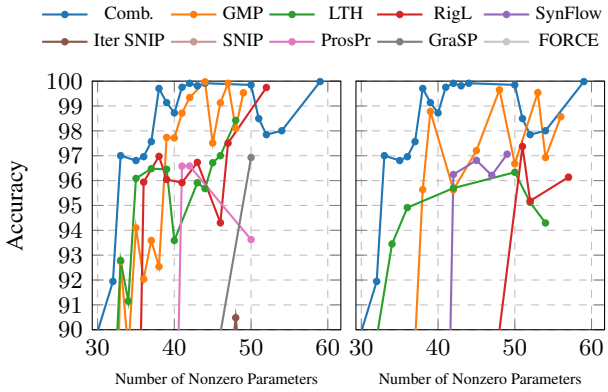
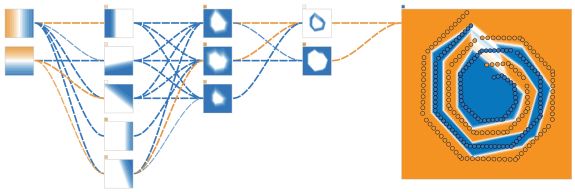


Figure 13. **Pruning Fails Under Optimal Conditions.** Models obtained by the combinatorial search and models that were pruned starting from the initialization of BENCH-995. The pruned models in the left subplot were given the optimal structured sparsity mask of width 6 determined by the combinatorial search. The subplot on the right depicts models that were pruned directly from width 16.

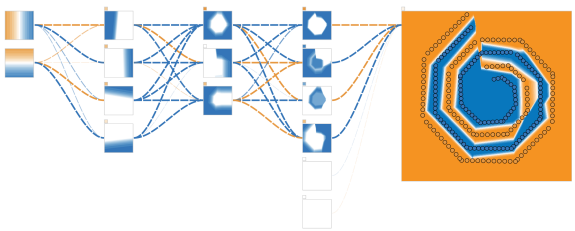
Depicted in Figure 14 are models that were pruned with GMP and RigL that fail to match the model identified by the combinatorial search.

8. Limitations of the Combinatorial Search

Synthetic Datasets and Small Models The high cost associated with performing the combinatorial search restricts our experiments to only synthetic datasets and small models.



(a) An example of a model, found through GMP, with 39 nonzero parameters that attains an accuracy of 97.73%.



(b) An example of a model, found through RigL, with 52 nonzero parameters that attains an accuracy of 99.74%. There appear to be roughly eight nonzero parameters that could be pruned after training which would still lead to a model that contains more nonzeros than the one recovered by the combinatorial search depicted in Figure 12.

Figure 14. **Pruning Algorithms Fail Under Optimal Conditions.** Despite being provided with the BENCH-995 initialization and the optimal width of 6, the pruned models depicted above still fall short of matching the models recovered by the combinatorial search.

Still, the shortcomings that are already being exhibited by pruning techniques in such a simplistic task should raise concerns and inquiries into pruning's current efficacy. While it is possible that the deficiencies of pruning algorithms observed do not extend to more complicated datasets and larger models, generally speaking, we would not expect an algorithm that does not work in a simple setting to work in a more complicated one.

No Guarantee of Sparsest While our combinatorial search can find a model that is sparser than any of the models found by pruning, there is no guarantee that the combinatorial search finds the optimally sparse model. Rather the combinatorial search only provides a lower bound on the sparsity that is attainable for the four-layer MLP yet is not achieved by existing pruning algorithms – the sparsest models elude pruning.

9. Related Works

Sparse Representations and Compressed Sensing This work is predicated on the assumption that pruning algorithms ought to be able to identify the sparsest model. It is natural to question why such an assumption is even feasi-

ble. The rationale stems from both empirical and theoretical works in the fields of sparse representations and compressed sensing where it is known that, within the framework of linear models, if the underlying sparse solution is sufficiently sparse, then pruning algorithms will recover it. For further details, refer to (Donoho, 2006; Elad, 2010; Candes & Tao, 2005; Tropp, 2006; 2004) and the cited literature.

Strong Tickets Sparked by interest in the lottery ticket hypothesis (Frankle & Carbin, 2018), numerous works have shown that, with high probability, there exists a subnetwork that can attain competitive performance within a sufficiently overparameterized randomly initialized network, called a *strong lottery ticket* (Malach et al., 2020; Ramanujan et al., 2020; Orseau et al., 2020). Our experiments reveal that, while the probability of identifying strong lottery tickets increases with the network's width, the effectiveness of current pruning methods in fact diminishes.

Random Pruning In line with our work, previous works have also benchmarked existing pruning techniques with naive pruning methods like random pruning (Liu et al., 2022; Gale et al., 2019). What sets our work apart is that random pruning, much like existing pruning techniques, remains susceptible to disconnected paths. This poses a challenge in achieving the recovery of a maximally sparse model, especially at high levels of overparameterization. Our approach to the combinatorial search guarantees that misalignment between weights is impossible making the search significantly more efficient in finding a minimal model.

Elucidating Pruning Several recent studies have expressed concerns about the current state of pruning, particularly with inconsistent benchmarking. Both Liu et al. (2023) and Blalock et al. (2020) proposed benchmarks for pruning, the former proposing SMC-Bench and the latter proposing ShrinkBench. Frankle et al. (2021) assessed several pruning at initialization techniques and remarked how they all perform similarly and are struggling to prune effectively at initialization. Evci et al. (2020b) showed that networks that were pruned at initialization have poor gradient flow leading to significantly worse generalization. For structured pruning, Liu et al. (2019) observed that the common pipeline of fine-tuning the pruned model is, at best, comparable to just training the model from scratch and encouraged a more careful evaluation of structured pruning. We differentiate ourselves from prior works by comparing pruning against the sparsest of models, enabling us to underscore fundamental issues inherent in current pruning methods.

Plant 'n' Seek In Fischer & Burkholz (2022), the authors handcraft sparse networks to solve synthetic problems and plant them within a larger randomly initialized network. They find that current pruning techniques are unable to extract the sparse subnetwork from the larger network either at initialization or after training. Our work, on the other hand, does not require handcrafting sparse subnetworks and all training starts from a completely random initialization. This experimental setup is more representative of the standard pruning paradigm, where model sizes might not be large enough for strong tickets to exist at initialization with high probability.

Disconnected Paths The tendency of pruning techniques to induce disconnected paths has previously been observed in prior works (Frankle et al., 2021; Vysogorets & Kempe, 2023; Pham et al., 2023). Both Frankle et al. (2021) and Vysogorets & Kempe (2023) propose measuring *effective sparsity*, which accounts for the disconnected paths when assessing sparsity. In Pham et al. (2023), the authors found that the ratio of the number of connected paths to the number of active neurons in the model is crucial for the success of pruning (Node-Path Balancing Principle) and introduced a novel pruning method that maximizes both quantities.

Pruning and Layer-Collapse It has been shown that current pruning techniques can inadvertently prune an entire layer at higher sparsity rates, effectively turning every path in the network into a disconnected one (Hayou et al., 2021). In Lee et al. (2020), the authors showed that an initialization that preserves layerwise dynamical isometry can assist in preventing this while Tanaka et al. (2020) proposed the pruning technique SynFlow as a solution. Figure 11 confirms that SynFlow is more robust to overparameterization compared to other pruning techniques but still fallible. Our work highlights that the problem is currently manifesting itself even at lower sparsity rates through disconnected paths and is more prevalent than just the catastrophic case where an entire layer is pruned.

10. Conclusion

We provided a comprehensive assessment of state-of-the-art pruning algorithms against the backdrop of ideal sparse networks obtained from a novel combinatorial search. Our findings reveal that current pruning algorithms fail to attain achievable sparsity levels – even when given the optimal width and initialization. We associate this discrepancy with unstructured pruning's inadequacy at performing structured pruning, their failure to benefit from overparameterization, and their tendency to induce disconnected paths while also foregoing sparsity. Despite the simplicity of the dataset and network architectures employed in our study, we believe that the issues highlighted in our work are only exacerbated at larger scales and we hope that our methods and findings will be of assistance for future forays into the development of new pruning techniques.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

Acknowledgements

We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC). This research was enabled in part by support provided by Compute Ontario (https://www.computeontario.ca) and the Digital Research Alliance of Canada (https://alliancecan.ca/en).

References

Alizadeh, M., Tailor, S. A., Zintgraf, L. M., van Amersfoort, J., Farquhar, S., Lane, N. D., and Gal, Y. Prospect pruning: Finding trainable weights at initialization using meta-gradients. In *International Conference on Learning Representations*, 2021.

Bellec, G., Kappel, D., Maass, W., and Legenstein, R. Deep rewiring: Training very sparse deep networks. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=BJ_wN01C-.

Blalock, D. W., Ortiz, J. J. G., Frankle, J., and Guttag, J. V. What is the state of neural network pruning? In Dhillon, I. S., Papailiopoulos, D. S., and Sze, V. (eds.), *MLSys*. mlsys.org, 2020. URL http://dblp.uni-trier.de/db/conf/mlsys/mlsys2020.html#BlalockOFG20.

Candes, E. J. and Tao, T. Decoding by linear programming. *IEEE transactions on information theory*, 51(12):4203–4215, 2005.

de Jorge, P., Sanyal, A., Behl, H., Torr, P., Rogez, G., and Dokania, P. K. Progressive skeletonization: Trimming more fat from a network at initialization. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=9GsFOUyUPi.

Dettmers, T. and Zettlemoyer, L. Sparse networks from scratch: Faster training without losing performance. *CoRR*, abs/1907.04840, 2019. URL http://arxiv.org/abs/1907.04840.

Donoho, D. L. Compressed sensing. *IEEE Transactions on information theory*, 52(4):1289–1306, 2006.

Elad, M. *Sparse and redundant representations: from theory to applications in signal and image processing*, volume 2. Springer, 2010.

Evci, U., Gale, T., Menick, J., Castro, P. S., and Elsen, E. Rigging the lottery: Making all tickets winners. In *International Conference on Machine Learning*, pp. 2943–2952. PMLR, 2020a.

Evci, U., Ioannou, Y. A., Keskin, C., and Dauphin, Y. N. Gradient flow in sparse neural networks and how lottery tickets win. *CoRR*, abs/2010.03533, 2020b. URL https://arxiv.org/abs/2010.03533.

Fischer, J. and Burkholz, R. Plant 'n' seek: Can you find the winning ticket? In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=9n9c8sf0xm.

Frankle, J. and Carbin, M. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2018.

Frankle, J., Dziugaite, G. K., Roy, D., and Carbin, M. Pruning neural networks at initialization: Why are we missing the mark? In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=Ig-VyQc-MLK.

Gale, T., Elsen, E., and Hooker, S. The state of sparsity in deep neural networks. *CoRR*, abs/1902.09574, 2019. URL http://arxiv.org/abs/1902.09574.

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

Hassibi, B. and Stork, D. Second order derivatives for network pruning: Optimal brain surgeon. *Advances in neural information processing systems*, 5, 1992.

Hassibi, B., Stork, D. G., and Wolff, G. J. Optimal brain surgeon and general network pruning. In *IEEE international conference on neural networks*, pp. 293–299. IEEE, 1993.

Hayou, S., Ton, J.-F., Doucet, A., and Teh, Y. W. Robust pruning at initialization. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=vXj_ucZQ4hA.

Hoefler, T., Alistarh, D., Ben-Nun, T., Dryden, N., and Peste, A. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *Journal of Machine Learning Research*, 22(241):1–124, 2021. URL http://jmlr.org/papers/v22/21-0366.html.

LeCun, Y., Denker, J., and Solla, S. Optimal brain damage. *Advances in neural information processing systems*, 2, 1989.

Lee, N., Ajanthan, T., and Torr, P. Snip: single-shot network pruning based on connection sensitivity. In *International Conference on Learning Representations*. Open Review, 2019.

Lee, N., Ajanthan, T., Gould, S., and Torr, P. H. S. A signal propagation perspective for pruning neural networks at initialization. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=HJeTo2VFwH.

Li, H., Kadav, A., Durdanovic, I., Samet, H., and Graf, H. P. Pruning filters for efficient convnets. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=rJqFGTslg.

Liu, S., Chen, T., Chen, X., Shen, L., Mocanu, D. C., Wang, Z., and Pechenizkiy, M. The unreasonable effectiveness of random pruning: Return of the most naive baseline for sparse training. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=VBZJ_3tz-t.

Liu, S., Chen, T., Zhang, Z., Chen, X., Huang, T., JAISWAL, A. K., and Wang, Z. Sparsity may cry: Let us fail (current) sparse neural networks together! In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=J6F3lLg4Kdp.

Liu, Z., Sun, M., Zhou, T., Huang, G., and Darrell, T. Rethinking the value of network pruning. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=rJlnB3C5Ym.

Luo, J.-H., Wu, J., and Lin, W. Thinet: A filter level pruning method for deep neural network compression. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

Malach, E., Yehudai, G., Shalev-Schwartz, S., and Shamir, O. Proving the lottery ticket hypothesis: Pruning is all you need. In III, H. D. and Singh, A. (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 6682–6691. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/malach20a.html.

Mocanu, D. C., Mocanu, E., Stone, P., Nguyen, P. H., Gibescu, M., and Liotta, A. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature Communications*, 9(1):2383, June 2018. ISSN 2041-1723. doi: 10.1038/s41467-018-04316-3. URL https://doi.org/10.1038/s41467-018-04316-3.

Mostafa, H. and Wang, X. Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization. In Chaudhuri, K. and Salakhutdinov, R. (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 4646–4655. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/mostafa19a.html.

Orseau, L., Hutter, M., and Rivasplata, O. Logarithmic pruning is all you need. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H. (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 2925–2934. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/1e9491470749d5b0e361ce4f0b24d037-Paper.pdf.

Pham, H., Ta, T.-A., Liu, S., Xiang, L., Le, D. D., Wen, H., and Tran-Thanh, L. Towards data-agnostic pruning at initialization: What makes a good sparse mask? In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=xdOoCWCYaY.

Ramanujan, V., Wortsman, M., Kembhavi, A., Farhadi, A., and Rastegari, M. What's hidden in a randomly weighted neural network? In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 11890–11899, 2020. doi: 10.1109/CVPR42600.2020.01191.

Smilkov, D., Carter, S., Sculley, D., Viégas, F. B., and Wattenberg, M. Direct-manipulation visualization of deep networks. *arXiv preprint arXiv:1708.03788*, 2017.

Tanaka, H., Kunin, D., Yamins, D. L., and Ganguli, S. Pruning neural networks without any data by iteratively conserving synaptic flow. *Advances in neural information processing systems*, 33:6377–6389, 2020.

Tropp, J. A. Greed is good: Algorithmic results for sparse approximation. *IEEE Transactions on Information theory*, 50(10):2231–2242, 2004.

Tropp, J. A. Just relax: Convex programming methods for identifying sparse signals in noise. *IEEE transactions on information theory*, 52(3):1030–1051, 2006.

Vysogorets, A. and Kempe, J. Connectivity matters: Neural network pruning through the lens of effective sparsity. *Journal of Machine Learning Research*, 24(99):1–23, 2023. URL http://jmlr.org/papers/v24/22-0415.html.

Wang, C., Zhang, G., and Grosse, R. Picking winning tickets before training by preserving gradient flow. In *International Conference on Learning Representations*, 2019.

Wen, W., Wu, C., Wang, Y., Chen, Y., and Li, H. Learning structured sparsity in deep neural networks. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, pp. 2082–2090, Red Hook, NY, USA, 2016. Curran Associates Inc. ISBN 9781510838819.

Zhou, H., Lan, J., Liu, R., and Yosinski, J. Deconstructing lottery tickets: Zeros, signs, and the supermask. In *Advances in Neural Information Processing Systems*, 2019.

Zhu, M. H. and Gupta, S. To prune, or not to prune: Exploring the efficacy of pruning for model compression, 2018. URL https://openreview.net/forum?id=S1lN69AT-.

A. BENCH-95 Initialization Experiments

We run the same experiments as detailed in Section 7 but using the BENCH-95 initialization.

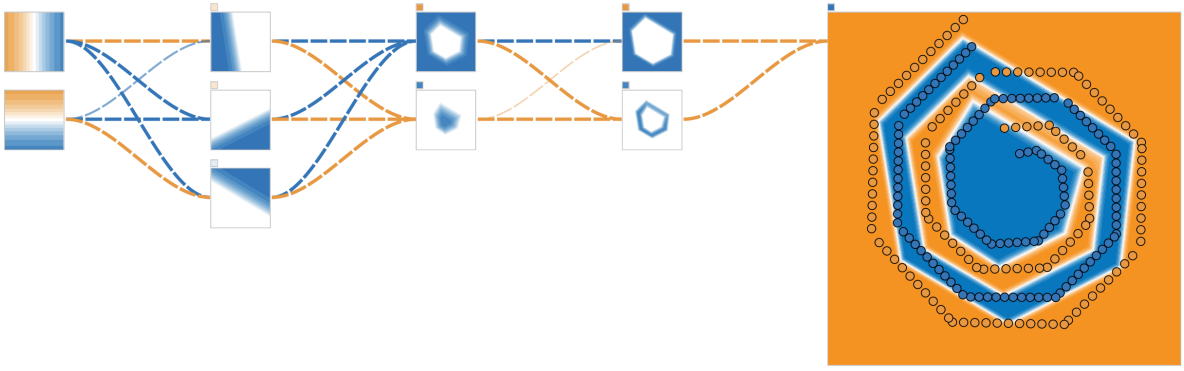


Figure 15. Benchmark Model with 26 nonzeros that was trained using the BENCH-95 initialization. Attains 96.47% accuracy.

Starting from the fixed initialization of BENCH-95, the combinatorial search is now able to identify a minimal neuron configuration of (3,2,2) and a benchmark model with 26 nonzeros that attains 96.47% accuracy.

Figure 16 below depicts models obtained by the combinatorial search and models that were pruned starting from the BENCH-95 initialization.

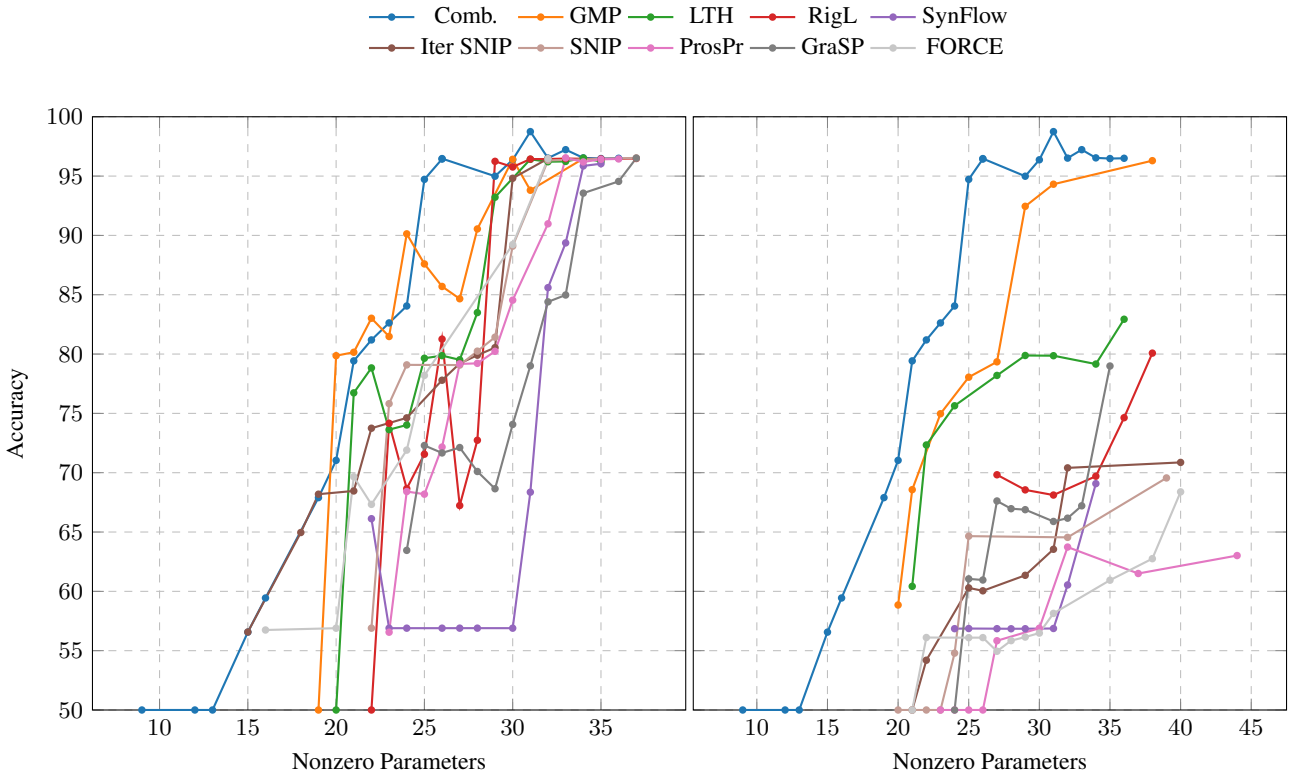


Figure 16. Analogous plot to Figure 13 but models are trained from the BENCH-95 initialization. The left subplot shows models that were provided with the optimal width of 3 via a structured sparsity mask. The right subplot shows models that were directly pruned from a width of 16. The scatter plot used to generate the Pareto frontiers in this plot are depicted in Figure 23.

B. Hyperparameters for Pruning Algorithms

To enforce sparsity in the models, we base our implementation on code from: https://github.com/facebookresearch/open_lth.

B.1. Hyperparameters for GMP

We use the cubic decay schedule detailed in (Zhu & Gupta, 2018) with 199 pruning steps spread out evenly across the 50 epochs of training. We base our implementation of magnitude pruning on code from: https://github.com/facebookresearch/open_lth.

B.2. Hyperparameters for LTH

We divide the training into five different blocks of 50 epochs of training (i.e. 250 epochs total). Between each block, we prune $p\%$ of the weights where p is chosen so that the final sparsity is reached. Then, we rewind the model back to initialization along with the learning rate scheduler. This ensures that the hyperparameters are consistent with the rest of the experiments.

B.3. Hyperparameters for RigL

We use the ERK distribution to determine the sparsity for each layer. For the update schedule, we utilize the hyperparameters: $\Delta T = 200, \alpha = 0.3$, f_{decay} to be cosine annealing, and we stop updating the mask 75% through training. We base our implementation of RigL on code from: https://github.com/verbose-avocado/rigl-torch and the implementation of the ERK distribution on code from: https://github.com/google-research/rigl.

B.4. Hyperparameters for ProsPr

We use the momentum and learning rate used in training to calculate the meta-gradients. We also perform three training steps to calculate the meta-gradients. We base our implementation of ProsPr on code from: https://github.com/mil-ad/prospr.

B.5. Hyperparameters for GraSP

We use the hyperparameters and base our implementation of GraSP on code from: https://github.com/alecwangcq/GraSP.

B.6. Hyperparameters for Iter SNIP and FORCE

We set the number of iterations to be 10 using the exponential decay schedule and just one batch to compute the average saliency per iteration. We base our implementations of Iter SNIP and FORCE on code from: https://github.com/naver/force.

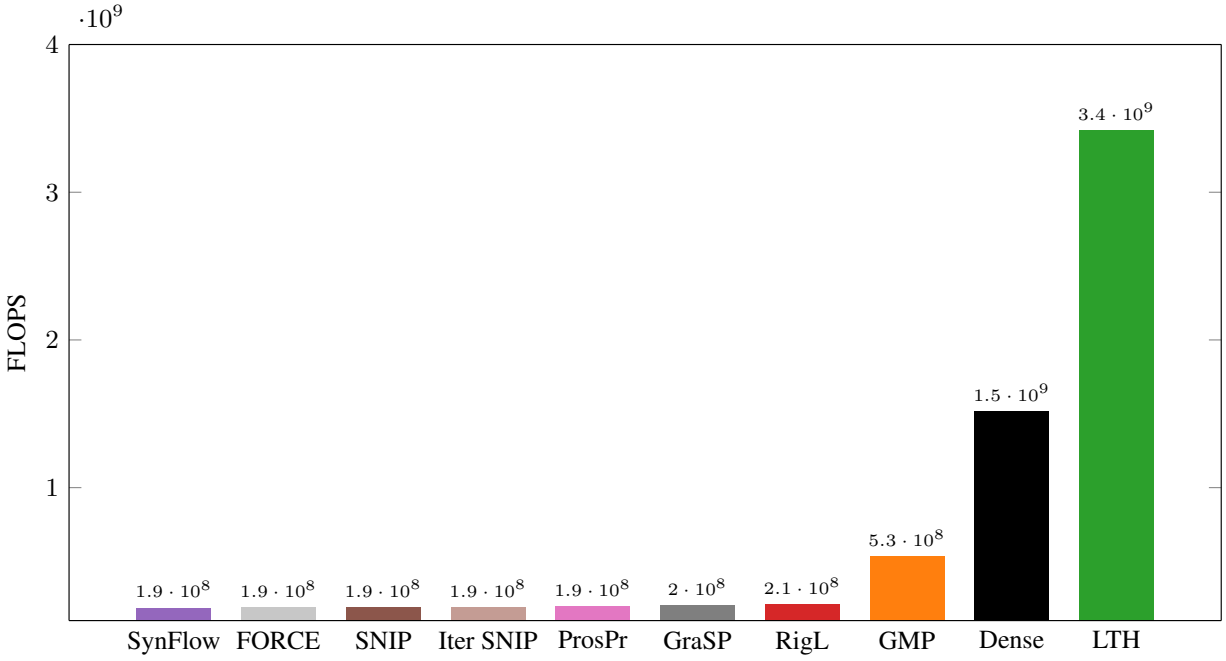
B.7. Hyperparameters for SynFlow

We set the number of iterations to 100 with an exponential pruning schedule. We base our implementation of SynFlow on code from: https://github.com/ganguli-lab/Synaptic-Flow.

B.8. Implementation of SNIP

We base our implementation of SNIP on code from: https://github.com/mil-ad/snip.

C. FLOPS Measurements



The FLOPS required for each pruning technique as depicted in Table 1 are measured by pruning and training an MLP of width 16 with 55 nonzero weights. We measure the FLOPS by computing the number of floating operations utilized by the nonzero parameters (including biases) in the model throughout pruning and training. We base our implementation on code from: `https://github.com/simochen/model-tools`

D. Selection of Subsets of Unstructured Masks for Combinatorial Search

We restrict the unstructured masks from two fronts. The first front is by restricting the masks of $\boldsymbol{W}^{[2]}$ to a subset of all possible masks. This is done through the for loop on line 3 in Algorithm 1 and instead of choosing all eligible unstructured masks, we simply choose the first three for the (7,3,3) configuration that are deemed eligible. For the (6,3,2) configuration, we choose the first six eligible unstructured masks. The second front is by restricting to the set of unstructured masks for the model to those that contain fewer nonzeros than a certain amount. This amount was set to 49 for the (7,3,3) configuration and 45 for the (6,3,2) configuration. We also run these experiments with a single learning rate of 0.05.

E. Dependence on Initialization

The combinatorial search does not achieve the same sparse network if different initializations are provided. A similar dependence of the mask on initialization has previously been observed in Frankle & Carbin (2018) and studied in Zhou et al. (2019). In the latter, the authors demonstrated that the mask only depends on the sign pattern of the initialization. Inspired by such observations, we ran exploratory experiments that indicated otherwise, at least in the case of the combinatorial search. As highlighted in Evci et al. (2020b), the lack of consistency across initializations might be attributed to poor gradient flow and overall challenges associated with training a sparse network from scratch.

F. Scatter Plots

The scatter plots that were used to generate the Pareto frontiers in Figures 8, 11, and 13 are depicted in Figures 20, 21, and 22 respectively.

G. Overparameterization: More Widths

We include the plots corresponding to widths 4,5,6,8, 32, and 128 in Figure 17. The scatter plots that were used to generate the Pareto frontiers in these plots are shown in Figure 24.

H. Additional Playground Visualizations

See Figures 18, 19 to see more visualizations of models that were obtained from pruning a width 16, four-layer MLP that was randomly initialized.

I. Technical Details of Sparse Model Visualization

The visualization tool operates in two parallel threads. The first thread uses PyTorch to produce the input variables, post-activation states, and model output by inputting a 512×512 grid of evenly spaced points in the square $[-2.25, 2.25] \times [-2.25, 2.25]$. It saves these tensors as well as the model parameters as global variables. The second thread runs a Flask application that visualizes these tensors using a blend of HTML and JavaScript. Specifically, the squares representing the input variables, post-activation states, and model output are illustrated using HTML Canvases for efficiency. Meanwhile, the connections denoting the weight entries are visualized with the JavaScript library D3, through Bezier curves with two control points.

J. Increasing Width Beyond 16

In Section 7, the first phase of the combinatorial search identified a structured sparsity mask using the width 16 MLP, comprising 50 nonzero parameters. Suppose we conduct a subsequent combinatorial search with an increased width of 17, which uncovers a solution not attainable at width 16. In this scenario, the structured sparsity mask would necessitate a minimum of 57 nonzeros. This requirement breaks down as follows: the first layer would need $2 \times 1 + 1$ parameters, the second layer $1 \times 1 + 1$ parameters, the third layer $1 \times 17 + 17$ parameters, and the classifier layer $17 \times 1 + 1$ parameters, cumulatively resulting in a total of 57 parameters.

K. Detailed Description of Combinatorial Search Algorithm

The first loop in the function $\textsc{EligibleMasks}(d_{in}, d_{out})$ (annotated by line 1) evaluates every conceivable quantity of nonzero elements in the weight matrix symbolized by n. Since each row and column contains at least one nonzero element, the minimal count of nonzero elements in the layer is determined by $\min = \max(d_{in}, d_{out})$. It's also self-evident that this count cannot exceed the product $\max = d_{in} \cdot d_{out}$.

The second loop (annotated by line 2) explores all eligible counts of nonzero elements in each row of the weight, labeled as k_i. This exploration ensures that the sum of nonzero elements across different rows, k_i, equates to the total number of nonzero elements in the entire weight matrix, n.

The third and concluding nested loop (annotated by line 3) peruses all potential masks of size k_i for each row in the weight matrix, across all rows.

The innermost if statement (annotated by line 4) selects from all row permutations the specific arrangement that results in a sequentially decreasing count of nonzero elements. If two rows have an equal count of nonzeros, the algorithm converts the binary vector representations of these masks into their decimal equivalents and arranges them in descending order based on these decimal values. Finally, it does the final check to verify that the mask is devoid of zero columns and if deemed as an eligible mask, the mask is padded with zeros so that its dimensions match the weight matrix.

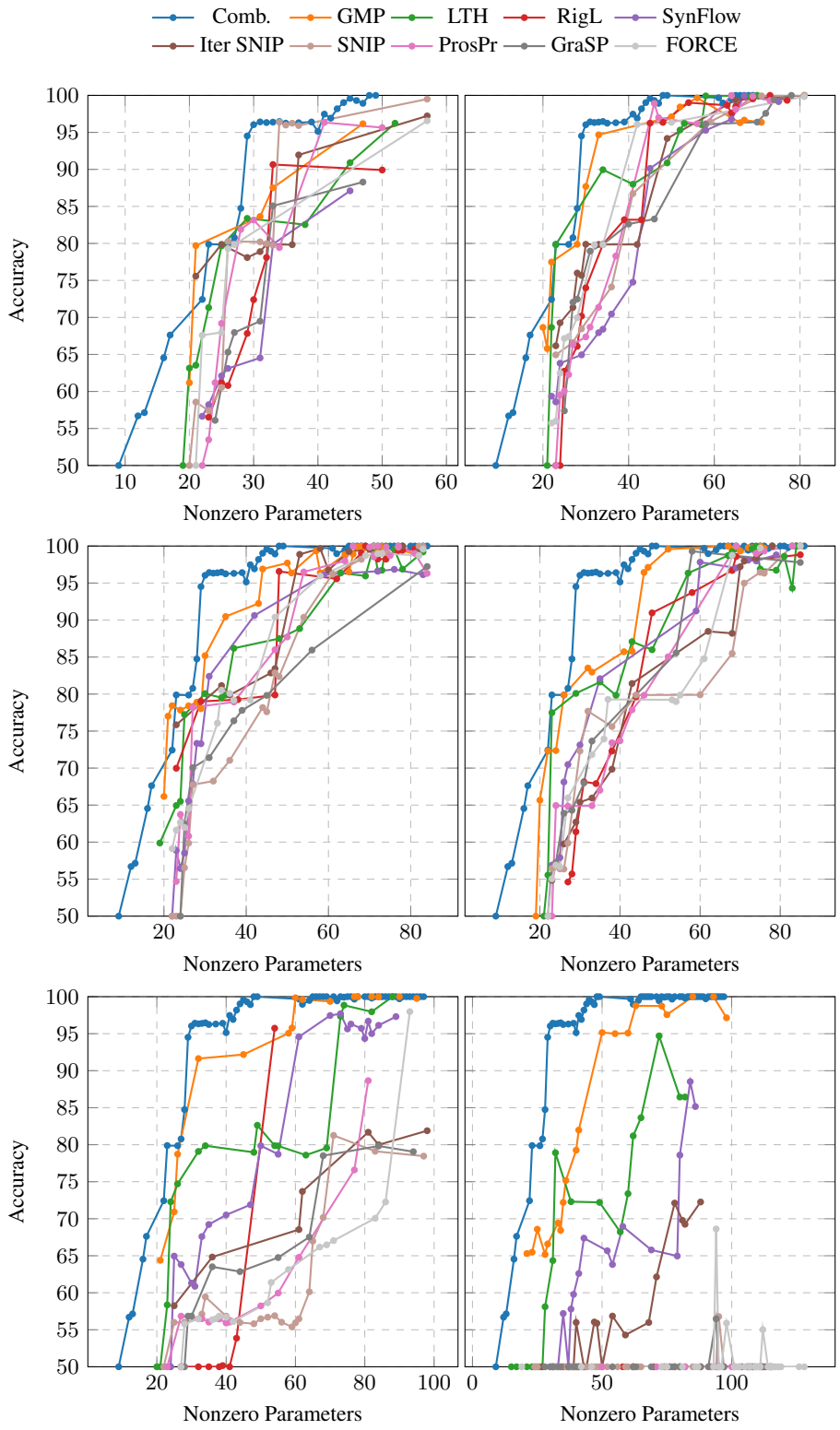
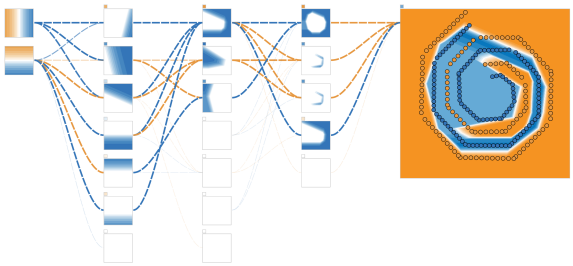
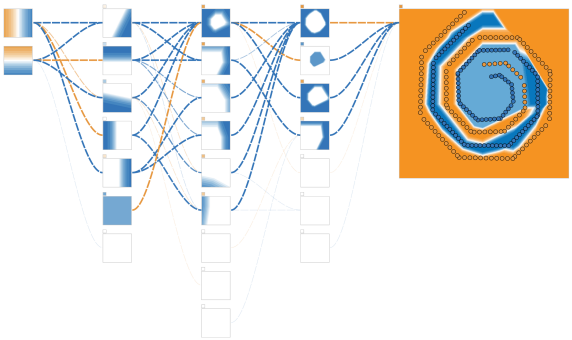


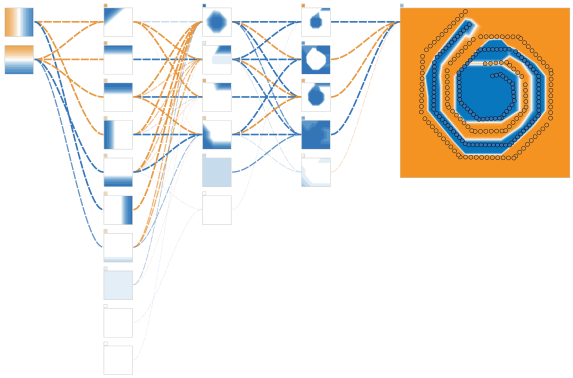
Figure 17. **More Widths** The accuracy versus the number of nonzero parameters after training four-layer MLPs of varying widths on the Cubist Spiral dataset. From left to right then top to bottom, the subplots correspond to MLPs of width 4, 5, 6, 8, 32, and 128.



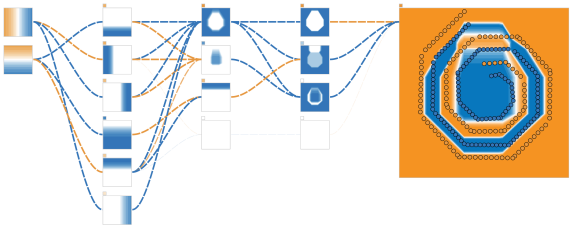
(a) Model attained using FORCE. 62 nonzeros and 90.47% accuracy. Observe there are a significant number of small magnitude weights.



(b) Model attained using GraSP. 66 nonzeros and 94.52% accuracy. Significant number of disconnected paths.

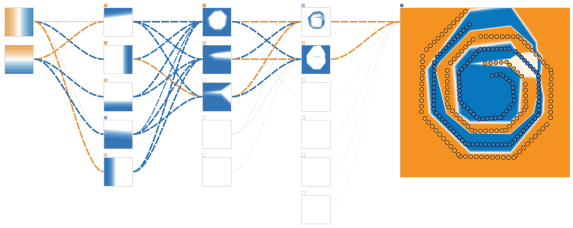


(c) Model attained using Iter SNIP. 78 nonzeros and 99.69% accuracy. Significant number of disconnected paths.

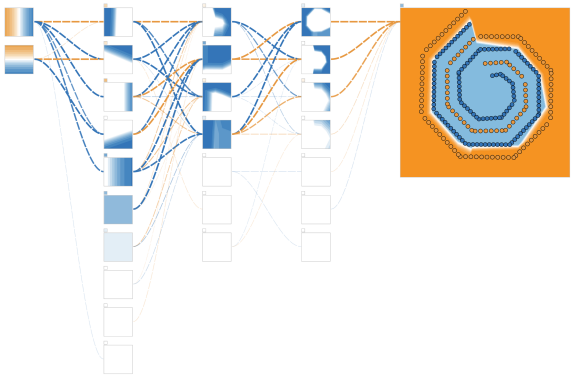


(d) Model attained using LTH. 44 nonzeros and 96.23% accuracy.

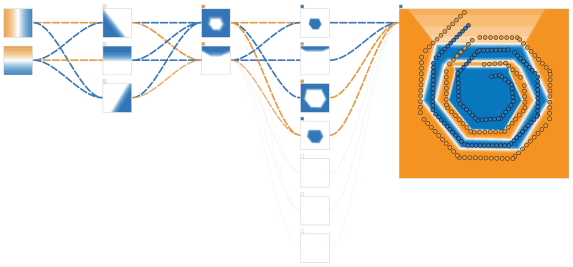
Figure 18. Visualizations of four-layer MLPs of width 16 that were attained by FORCE, GraSP, Iter SNIP, and LTH. While all pruning algorithms suffer from disconnected paths to some degree, some do appear to be more robust than others to the issue.



(a) Model attained using RigL. 52 nonzeros and 97.04% accuracy. Significant number of disconnected paths.



(b) Model attained using SNIP. 79 nonzeros and 81.30% accuracy. Significant number of disconnected paths.



(c) Model attained using SynFlow. 42 nonzeros and 94.76% accuracy. Presence of small magnitude weights.

Figure 19. Various visualizations of four-layer MLPs of width 16 that were attained by RigL, SNIP, SynFlow.

Algorithm 1: Combinatorial Search

Input: Desired Accuracy, ρ, and maximal number of channels per layer, D.
Output: The set of all possible model masks S.

Function COMBINATORIALSEARCH (ρ, D) **:**

// Phase One: Loop through all possible numbers of neurons in each layer.
$N \leftarrow \{\}$
// Two-dimensional input and one-dimensional output.
$d^{[0]} = 2, d^{[4]} = 1$
for $(d^{[1]}, d^{[2]}, d^{[3]}) \in \{1, \ldots, D\}^3$ **do**
  $\theta \leftarrow \texttt{MLP}(\text{depth}{=}4, \text{width}{=}D)$
  // Mask layers according to neuron configuration.
  for $\ell \in \{1, 2, 3, 4\}$ **do**
    $\boldsymbol{W}^{[\ell]}[:,\ d^{[\ell]} : D] = 0, \quad \boldsymbol{W}^{[\ell]}[d^{[\ell-1]} : D,\ :\ ] = 0, \quad \boldsymbol{b}^{[\ell]}[d^{[\ell]} : D] = 0$
  if ACCURACY$(\theta) > \rho$ **then**
    $N \leftarrow N \cup \{(d^{[1]}, d^{[2]}, d^{[3]})\}$
// Find successful configuration that minimizes model nonzeros.
$d^{[1]}, d^{[2]}, d^{[3]} \leftarrow \arg\min_{(d^{[1]}, d^{[2]}, d^{[3]}) \in N} (2 * d^{[1]} + d^{[1]}) + (d^{[1]} * d^{[2]} + d^{[2]}) + (d^{[2]} * d^{[3]} + d^{[3]}) + (d^{[3]} * 1 + 1)$
// Phase Two: Generate eligible masks for each layer.
for $\ell \in \{1, 2, 3, 4\}$ **do**
  $\text{supp}^{[\ell]} \leftarrow$ ELIGIBLEMASKS$(d^{[\ell-1]}, d^{[\ell]})$
return $\{(\boldsymbol{s}^{[1]}, \boldsymbol{s}^{[2]}, \boldsymbol{s}^{[3]}, \boldsymbol{s}^{[4]}) \mid \boldsymbol{s}^{[1]} \in \text{supp}^{[1]}, \boldsymbol{s}^{[2]} \in \text{supp}^{[2]}, \boldsymbol{s}^{[3]} \in \text{supp}^{[3]}, \boldsymbol{s}^{[4]} \in \text{supp}^{[4]}\}$

Function ELIGIBLEMASKS $(d^{[in]}, d^{[out]})$ **:**

supp $\leftarrow \{\}$
// Calculate min and max possible nonzeros for each layer.
$\min = \max(d^{[in]}, d^{[out]})$
$\max = d^{[in]} \cdot d^{[out]}$
// Loop through all nonzero counts for the layer's weights.
1 **for** $n \in \{\min, \ldots, \max\}$ **do**
  // Loop over all row-wise nonzero distributions.
2 **for** $(k_1, \ldots, k_{d^{[out]}}) \in \left\{\boldsymbol{k} \in \{1, 2, \ldots, d^{[in]}\}^{d^{[out]}} \mid \sum_{i=1}^{d^{[out]}} k_i = n\right\}$ **do**
    // $\mathcal{B}(k) = \left\{v \in \{0,1\}^{d^{[in]}} \mid \sum_{i=1}^{d^{[in]}} v_i = k\right\}$
    // Loop over all possible masks for each row in the layer.
3 **for** $\boldsymbol{s}_1 \in \mathcal{B}(k_1) \wedge \ldots \wedge \boldsymbol{s}_{d^{[out]}} \in \mathcal{B}(k_{d^{[out]}})$ **do**
4 **if** ELIGIBLE(STACK$(\boldsymbol{s}_1, \ldots, \boldsymbol{s}_{d^{[out]}})$) **then**
        supp $\leftarrow$ supp $\cup \{$PADWITHZEROS$(\boldsymbol{s}_1, \ldots, \boldsymbol{s}_{d^{[out]}})\}$
return supp

Function ELIGIBLE $(\boldsymbol{S})$ **:**

if $\boldsymbol{S}$ *contains zero columns* **then**
  return False
// Ensure non-increasing nonzeros across the rows.
for $i \in \{1, \ldots,$ ROWS$(\boldsymbol{S}) - 1\}$ **do**
  if $\|\boldsymbol{S}_{i,:}\|_0 > \|\boldsymbol{S}_{i+1,:}\|_0$ **then**
    return False
  else if $\|\boldsymbol{S}_{i,:}\|_0 = \|\boldsymbol{S}_{i+1,:}\|_0$ **then**
    if BINARYTODECIMAL$(\boldsymbol{S}_{i,:}) >$ BINARYTODECIMAL$(\boldsymbol{S}_{i+1,:})$ **then**
      return False
return True

L. Overparameterization Leads to Disconnected Paths

Theorem L.1. *Consider an L-layer multilayer perceptron with weights $\boldsymbol{W}^{[1]}, ..., \boldsymbol{W}^{[L]}$ where $\boldsymbol{W}^{[1]} \in \mathbb{R}^{d\times w}, \boldsymbol{W}^{[2]}, ..., \boldsymbol{W}^{[L-1]} \in \mathbb{R}^{w\times w}, \boldsymbol{W}^{[L]} \in \mathbb{R}^{w\times C}$. Suppose the model is randomly pruned such that n_ℓ nonzero weights remain in $\boldsymbol{W}^{[\ell]}$. Assume that $L \geq 4$ and $w > \max_{\ell\in\{2,...,L-2\}}(n_\ell + n_{\ell+1})$. Then the probability that there is no connected path in the model tends to one when the width of the model goes to infinity.*

Proof: For the proof, we will use the notation that $\boldsymbol{W}^{[\ell]}$ denotes the unpruned weight matrix, $\boldsymbol{M}^{[\ell]}$ denotes the mask generated by randomly pruning layer $\boldsymbol{W}^{[\ell]}$, and $\tilde{\boldsymbol{W}}^{[\ell]} = \boldsymbol{M}^{[\ell]} \odot \boldsymbol{W}^{[\ell]}$ denotes the pruned weight matrix. It is clear that if there are no connected paths between the two layers $\tilde{\boldsymbol{W}}^{[\ell]}$ and $\tilde{\boldsymbol{W}}^{[\ell+1]}$, there are no connected paths in the whole pruned network.

Let $A_k^{[\ell]}$ denote the set of masks satisfying:

$$A_k^{[\ell]} = \{\boldsymbol{M}^{[\ell]} \in \mathbb{R}^{w\times w} \ : \ \boldsymbol{M}^{[\ell]}_{1,\cdot} \neq 0, ..., \boldsymbol{M}^{[\ell]}_{k,\cdot} \neq 0, \boldsymbol{M}^{[\ell]}_{k+1,\cdot} = 0, ..., \boldsymbol{M}^{[\ell]}_{w,\cdot} = 0\}$$

For $2 \leq \ell \leq L-2$, we can express the probability that there are no connected paths between $\tilde{\boldsymbol{W}}^{[\ell]}$ and $\tilde{\boldsymbol{W}}^{[\ell+1]}$ as follows:

$$\sum_{k=1}^{n_l} \binom{w}{k} \mathbb{P}(\text{No Connected Paths between } \tilde{\boldsymbol{W}}^{[\ell]} \text{ and } \tilde{\boldsymbol{W}}^{[\ell+1]} | \boldsymbol{M}^{[\ell]} \in A_k^{[\ell]}) \cdot \mathbb{P}(\boldsymbol{M}^{[\ell]} \in A_k^{[\ell]})$$

For there to be no connected paths between $\tilde{\boldsymbol{W}}^{[\ell]}$ and $\tilde{\boldsymbol{W}}^{[\ell+1]}$, that means that all $n_{\ell+1}$ nonzero entries in $\boldsymbol{M}^{[\ell+1]}$ must lie in the remaining $w-k$ columns that are not aligned with the k nonzero rows of $\boldsymbol{M}^{[\ell]}$. Thus, we find that

$$\mathbb{P}(\text{No Connected Paths between } \tilde{\boldsymbol{W}}^{[\ell]} \text{ and } \tilde{\boldsymbol{W}}^{[\ell+1]} | \boldsymbol{M}^{[\ell]} \in A_k^{[\ell]}) = \sum_{r=1}^{n_{\ell+1}} \frac{\binom{w-k}{r} \cdot |A_r^{[\ell+1]}|}{\binom{w^2}{n_{\ell+1}}}$$

Combining everything together, we get that

$$\mathbb{P}(\text{No Connected Paths between } \tilde{\boldsymbol{W}}^{[\ell]} \text{ and } \tilde{\boldsymbol{W}}^{[\ell+1]}) = \sum_{k=1}^{n_\ell} \left[\frac{\binom{w}{k} \cdot |A_k^{[\ell]}|}{\binom{w^2}{n_\ell}} \cdot \sum_{r=1}^{n_{\ell+1}} \frac{\binom{w-k}{r} \cdot |A_r^{[\ell+1]}|}{\binom{w^2}{n_{\ell+1}}} \right]$$

To see that this term is tending to one as $w \to \infty$, notice that

$$\sum_{k=1}^{n_\ell} \binom{w}{k} \cdot |A_k^{[\ell]}| = \binom{w^2}{n_\ell}$$

Utilizing this, we can bound the probability as follows:

$$\begin{aligned}
\sum_{k=1}^{n_\ell} \left[\frac{\binom{w}{k} \cdot |A_k^{[\ell]}|}{\binom{w^2}{n_\ell}} \cdot \sum_{r=1}^{n_{\ell+1}} \frac{\binom{w-k}{r} \cdot |A_r^{[\ell+1]}|}{\binom{w^2}{n_{\ell+1}}} \right] &= \sum_{k=1}^{n_\ell} \left[\frac{\binom{w}{k} \cdot |A_k^{[\ell]}|}{\binom{w^2}{n_\ell}} \cdot \sum_{r=1}^{n_{\ell+1}} \frac{\binom{w}{r} \cdot \prod_{j=0}^{k-1} \left(\frac{w-j-r}{w-j}\right) \cdot |A_r^{[\ell+1]}|}{\binom{w^2}{n_{\ell+1}}} \right] \\
&\geq \left(\frac{w - n_\ell + 1 - n_{\ell+1}}{w} \right)^{n_\ell - 1} \sum_{k=1}^{n_\ell} \left[\frac{\binom{w}{k} \cdot |A_k^{[\ell]}|}{\binom{w^2}{n_\ell}} \cdot \sum_{r=1}^{n_{\ell+1}} \frac{\binom{w}{r} \cdot |A_r^{[\ell+1]}|}{\binom{w^2}{n_{\ell+1}}} \right] \\
&= \left(\frac{w - n_\ell + 1 - n_{\ell+1}}{w} \right)^{n_\ell - 1}
\end{aligned}$$

arriving at the following set of inequalities:

$$\left(\frac{w - n_\ell + 1 - n_{\ell+1}}{w} \right)^{n_\ell - 1} \leq \mathbb{P}(\text{No Connected Paths between } \tilde{\boldsymbol{W}}^{[\ell]} \text{ and } \tilde{\boldsymbol{W}}^{[\ell+1]}) \leq 1$$

Applying squeeze theorem, we get that the probability is tending to one as $w \to \infty$.

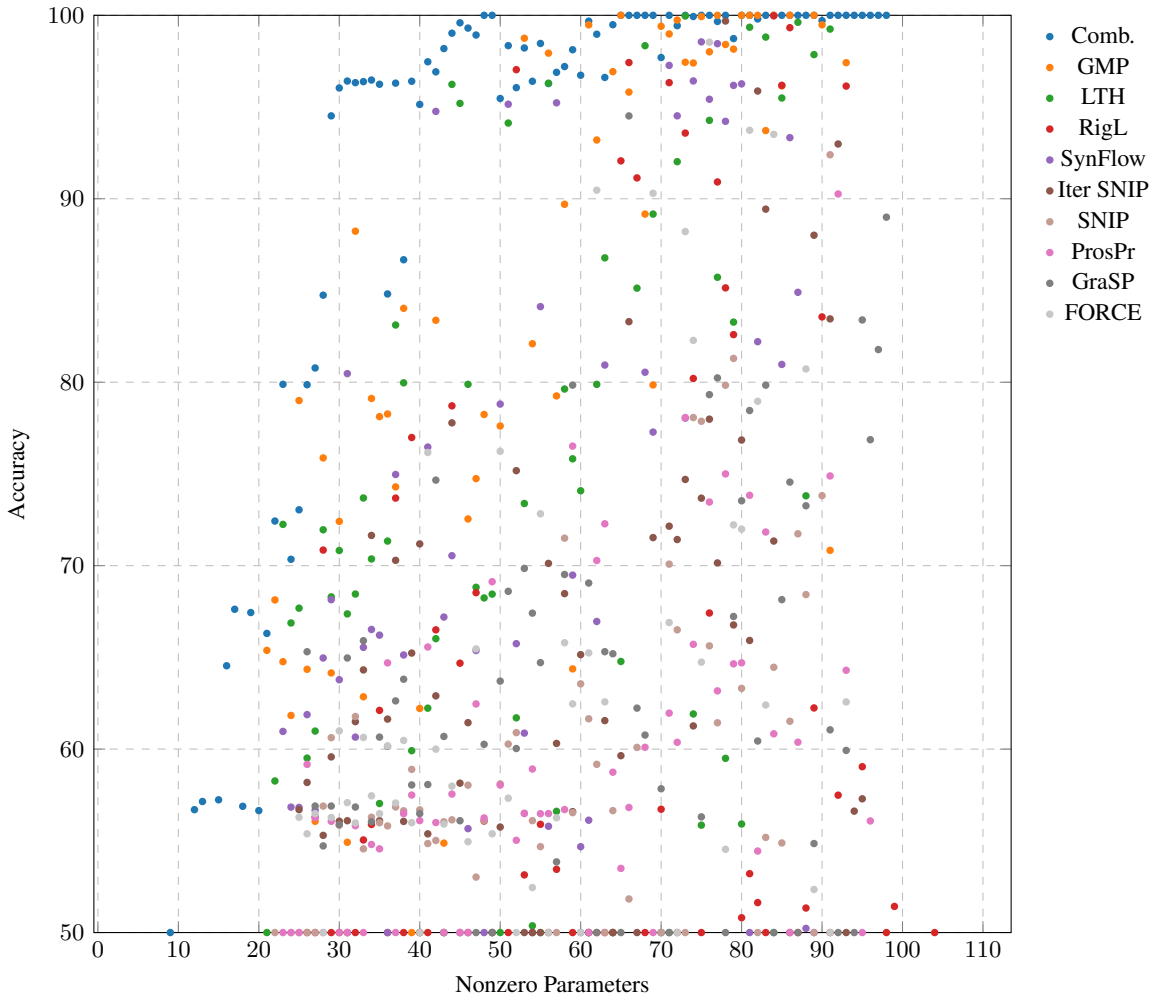


Figure 20. Scatter plots containing the best performing run for each pruning approach at a given number of nonzeros in the model. Pareto frontiers depicted in Figure 8 are interpolated from the datapoints depicted in this plot.

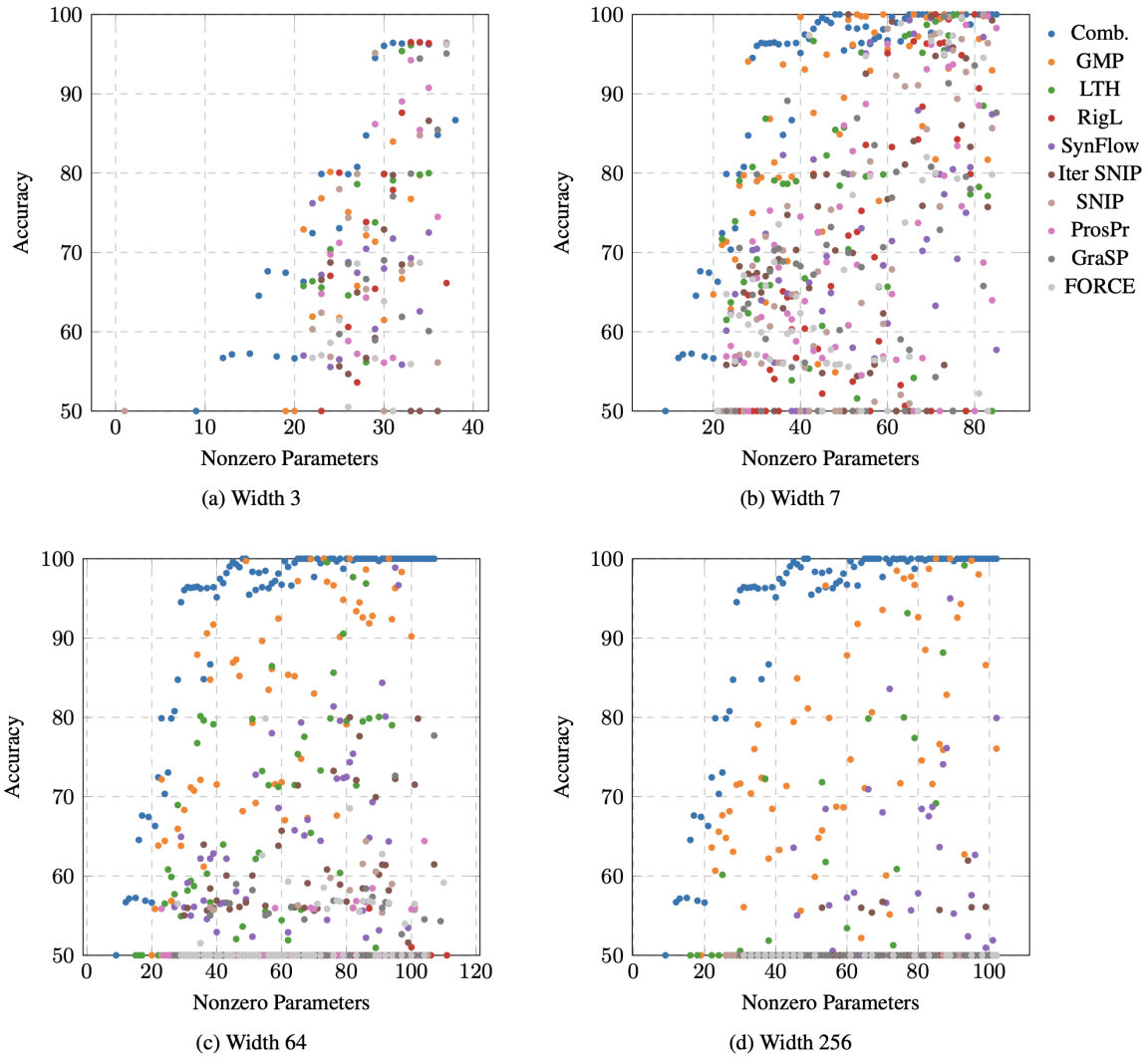
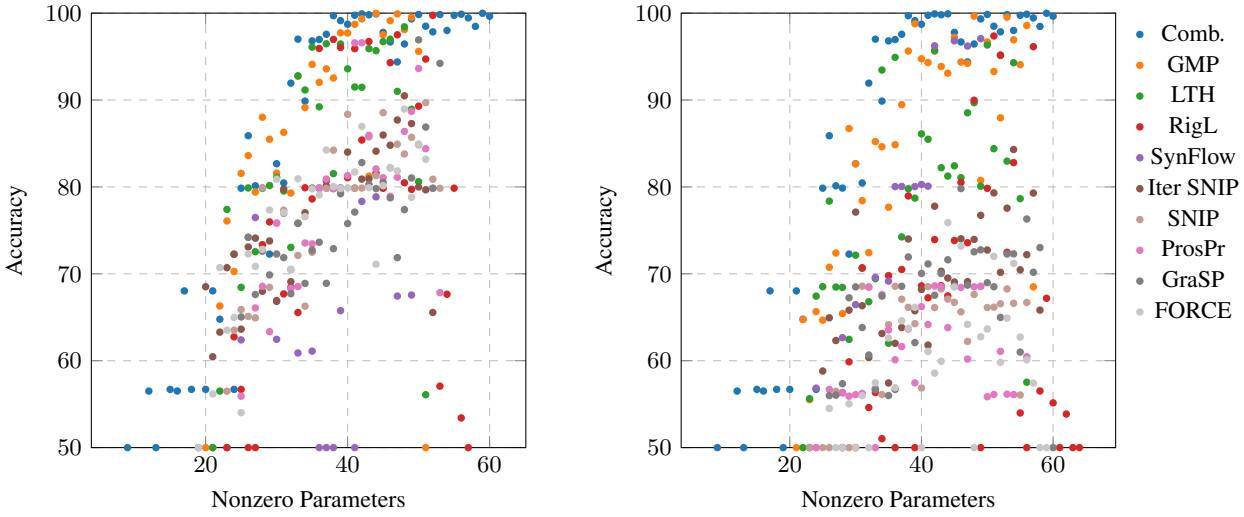


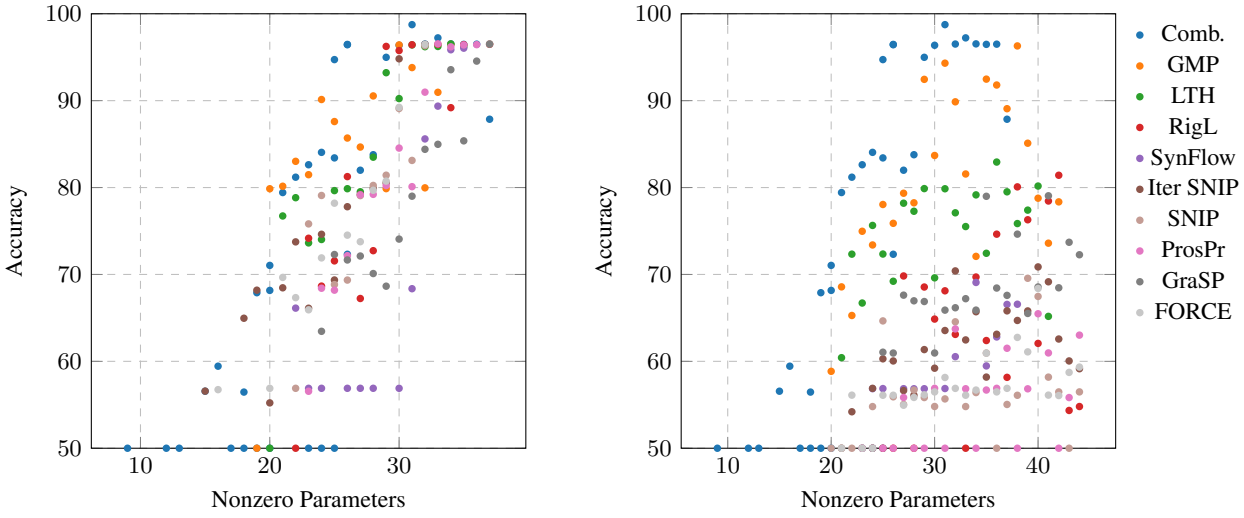
Figure 21. Scatter plots containing the best performing run for each pruning approach at a given number of nonzeros in the model. Pareto frontiers depicted in Figure 11 are interpolated from the datapoints depicted in this plot.



(a) Models Given Optimal Width of 6

(b) Width 16

Figure 22. Scatter plots containing the best performing run for each pruning approach at a given number of nonzeros in the model. Pareto frontiers depicted in Figure 13 are interpolated from the datapoints depicted in this plot.



(a) Models Given Optimal Width of 3

(b) Width 16

Figure 23. Scatter plots containing the best performing run for each pruning approach at a given number of nonzeros in the model. Pareto frontiers depicted in Figure 16 are interpolated from the datapoints depicted in this plot.

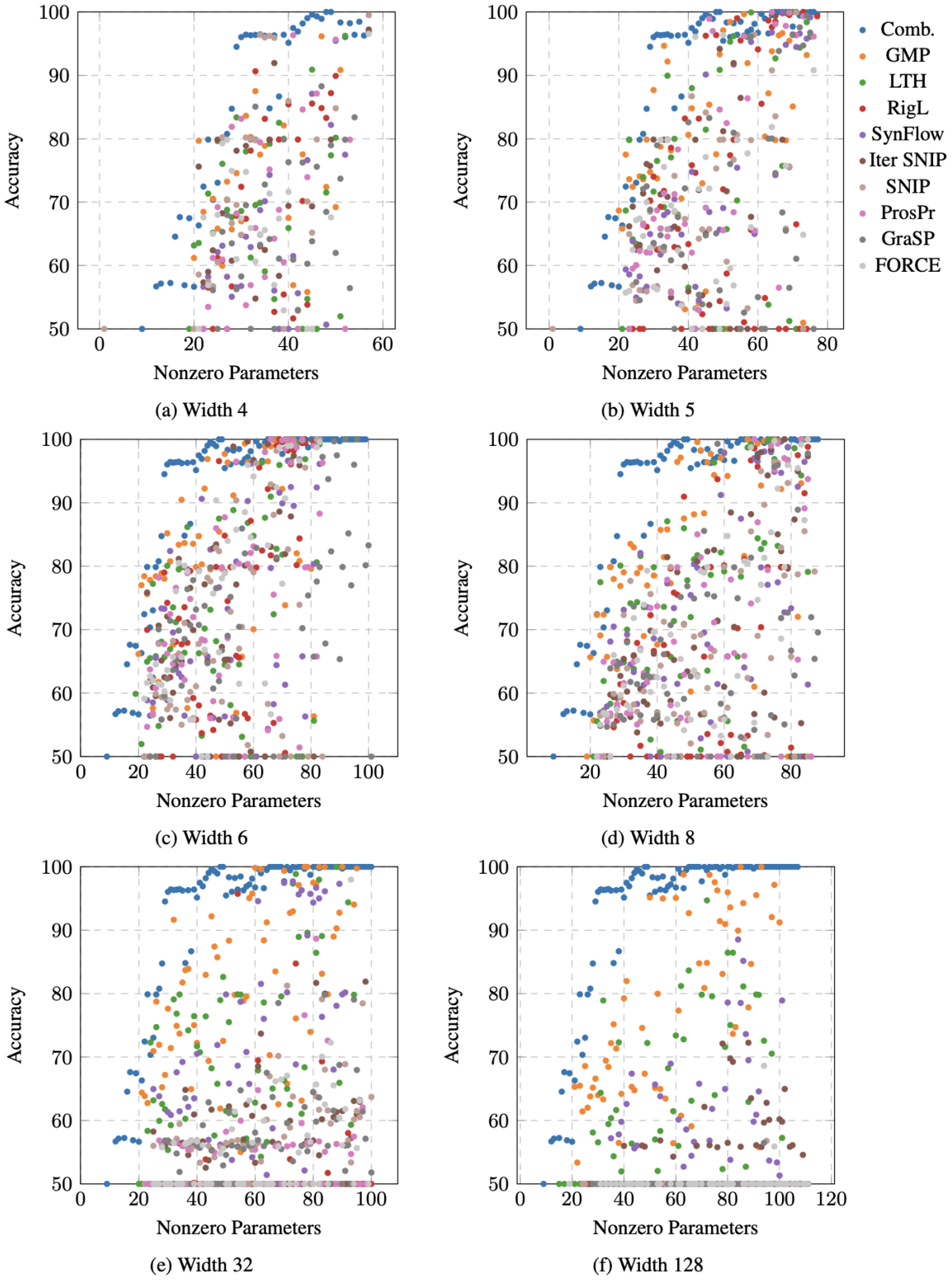


Figure 24. Scatter plots containing the best performing run for each pruning approach at a given number of nonzeros in the model. Pareto frontiers depicted in Figure 17 are interpolated from the datapoints depicted in this plot.