Getting Away with More Network Pruning: From Sparsity to Geometry and Linear Regions

Junyang Cai*[1], Khai-Nguyen Nguyen*[1], Nishant Shrestha[1], Aidan Good[1], Ruisen Tu[1], Xin Yu[2], Shandian Zhe[2], and Thiago Serra[1]

[1] Bucknell University, United States
{jc092,nkn002,ns037,wag011,rt024,thiago.serra}@bucknell.edu
[2] University of Utah, United States
xin.yu@utah.edu,zhe@cs.utah.edu

Abstract. One surprising trait of neural networks is the extent to which their connections can be pruned with little to no effect on accuracy. But when we cross a critical level of parameter sparsity, pruning any further leads to a sudden drop in accuracy. This drop plausibly reflects a loss in model complexity, which we aim to avoid. In this work, we explore how sparsity also affects the geometry of the linear regions defined by a neural network, and consequently reduces the expected maximum number of linear regions based on the architecture. We observe that pruning affects accuracy similarly to how sparsity affects the number of linear regions and our proposed bound for the maximum number. Conversely, we find out that selecting the sparsity across layers to maximize our bound very often improves accuracy in comparison to pruning as much with the same sparsity in all layers, thereby providing us guidance on where to prune.

Keywords: Model complexity · Network pruning · Solution counting.

1 Introduction

In deep learning, there are often good results with little justification and good justifications with few results. Network pruning exemplifies the former: we can easily prune half or more of the connections of a neural network without affecting the resulting accuracy, but we may have difficulty explaining why we can do that. The theory of linear regions exemplifies the latter: we can theoretically design neural networks to express very nuanced functions, but we may end up obtaining much simpler ones in practice. In this paper, we posit that the mysteries of pruning and the wonders of linear regions can complement one another.

When it comes to pruning, we can reasonably argue that reducing the number of parameters improves generalization. While Denil et al. [12] show that the parameters of neural networks can be redundant, it is also known that the smoother loss landscape of larger neural networks leads to better training convergence [45,66]. Curiously, Jin et al. [36] argue that pruning also smooths the

*Equal contribution.

loss function, which consequently improves convergence during fine tuning — the additional training performed after pruning the network. However, it remains unclear to what extent we can prune without ultimately affecting accuracy, which is an important concern since a machine learning model with fewer parameters can be deployed more easily in environments with limited hardware.

The survey by Hoefler et al. [31] illustrates that a moderate amount of pruning typically improves accuracy while further pruning may lead to a substantial decrease in accuracy, whereas Liebenwein et al. [46] show that this tolerable amount of pruning depends on the task for which the network is trained. In terms of what to prune, another survey by Blalock et al. [6] observes that most approaches consist of either removing parameters with the smallest absolute value [28,54,35,24,23,44,16,14,22,67,48]; or removing parameters with smallest expected impact on the output [42,29,30,40,50,13,79,82,4,43,72,47,73,76,64,80], to which we can add the special case of exact compression [60,65,63,18].

While most work on this topic has helped us prune more with a lesser impact on accuracy, fairness studies recently debuted by Hooker et al. [32] have focused instead on the impact of pruning on recall — the ability of a network to correctly identify samples as belonging to a certain class. Recall tends to be more severely affected by pruning in classes and features that are underrepresented in the dataset [32,56,33], which Tran et al. [70] attribute to differences across such groups in gradient norms and Hessian matrices of the loss function. In turn, Good et al. [20] showed that such recall distortions may also occur in balanced datasets, but in a more nuanced form: moderate pruning leads to comparable or better accuracy while reducing differences in recall, whereas excessive pruning leads to lower accuracy while increasing differences in recall. Hence, avoiding a significant loss in accuracy due to pruning is also relevant for fairness.

Overall, network pruning studies have been mainly driven by one question: **how can we get away with more network pruning?** Before we get there with our approach, let us consider the other side of the coin in our narrative.

When it comes to the theory of linear regions, we can reasonably argue that the number of linear regions may represent the expressiveness of a neural network — and therefore relate to its ability to classify more complex data. We have learned that a neural network can be a factored representation of functions that are substantially more complex than the activation function of each neuron. This theory is applicable to networks in which the neurons have piecewise linear activations, and consequently the networks represent a piecewise linear function in which the number of pieces — or linear regions — may grow polynomially on the width and exponentially on the depth of the network [57,52]. When the activation function is the Rectified Linear Unit (ReLU) [55,19], each linear region corresponds to a different configuration of active and inactive neurons. For geometric reasons that we discuss later, not every such configuration is feasible.

The study of linear regions bears some resemblance to universal approximation results, which have shown that most functions can be approximated to arbitrary precision with sufficiently wide neural networks [10,17,34]. These results were extended in [78] to the currently more popular ReLU activation and later focused on networks with limited width but arbitrarily large depth [49,27]. In comparison to universal approximation, the theory of linear regions tells us what piecewise linear functions are possible to represent — and thus what other functions can be approximated with them — in a context of limited resources translated as both the number of layers and the width of each layer.

Most of the literature is focused on fully-connected feedforward networks using the ReLU activation function, which will be our focus on this paper as well. Nevertheless, there are also adaptations and extensions of such results for convolutional networks by [77] and for maxout networks [21] by [52,62,71,53].

Several papers have shown that the right choice of parameters may lead to an astronomical number of linear regions [52,68,3,62], while other papers have shown that the maximum number of linear regions can be affected by narrow layers [51], the number of active neurons across different linear regions [62], and the parameters of the network [61]. Despite the exponential growth in depth, Serra et al. [62] observe that a shallow network may in some cases yield more linear regions among architectures with the same number of neurons. Whereas the number of linear regions among networks of similar architecture relates to the accuracy of the networks [62], Hanin and Rolnick [25,26] show that the typical initialization and subsequent training of neural networks is unlikely to yield the expressive number of linear regions that have been reported elsewhere.

These contrasting results lead to another question: **is the network complexity in terms of linear regions relevant to accuracy if trained models are typically much less expressive in practice?** Now that you have read both sides of our narrative, you may have guessed where we are heading.

We posit that these two topics — network pruning and the theory of linear regions — can be combined. Namely, that the latter can guide us on how to prune neural networks, since it can be a proxy to model complexity.

But we must first address the paradox in our second question. As observed by Hanin and Rolnick [25], perturbing the parameters of networks designed to maximize the number of linear regions, such as the one by Telgarsky [68], leads to a sudden drop on the number of linear regions. Our interpretation is that every architecture has a probability distribution for the number of linear regions. If by perturbing these especially designed constructions we obtain networks with much smaller numbers, we may infer that these constructions correspond to the tail of that distribution. However, if certain architectural choices lead to much larger numbers of linear regions at best, we may also conjecture that the entire distribution shifts accordingly, and thus that even the ordinary trained network might be more expressive if shaped with the potential number of linear regions in mind. Hence, we conjecture the architectural choices aimed at maximizing the number of linear regions may lead better performing networks.

That brings us to a gap in the literature: to the best of our understanding, there is no prior work on how network pruning affects the number of linear regions. We take the path that we believe would bring the most insight, which consists of revisiting — under the lenses of sparsity – the factors that may limit the maximum number of linear regions based on the neural network architecture.

In summary, this paper presents the following contributions:

(i) We prove an upper bound on the expected number of linear regions over the ways in which weight matrices might be pruned, which refines the bound in [62] to sparsified weight matrices (Section 3).
(ii) We introduce a network pruning technique based on choosing the density of each layer for increasing the potential number of linear regions (Section 4).
(iii) We propose a method based on Mixed-Integer Linear Programming (MILP) to count linear regions on input subspaces of arbitrary dimension, which generalizes the cases of unidimensional [25] and bidimensional [26] inputs; this MILP formulation includes a new constraint in comparison to [62] for correctly counting linear regions in general (Section 5).

2 Notation

In this paper, we study the linear regions defined by the fully-connected layers of feedforward networks. For simplicity, we assume that the entire network consists of such layers and that each neuron has a ReLU activation function, hence being denoted as a *rectifier network*. However, our results can be extended to the case in which the fully-connected layers are preceded by convolutional layers, and in fact our experiments show their applicability in that context. We also abstract the fact that fully-connected layers are often followed by a softmax layer.

We assume that the neural network has an input $\boldsymbol{x} = [x_1 \; x_2 \; \dots \; x_{n_0}]^T$ from a bounded domain $\mathbb{X}$ and corresponding output $\boldsymbol{y} = [y_1 \; y_2 \; \dots \; y_m]^T$, and each hidden layer $l \in \mathbb{L} = \{1, 2, \dots, L\}$ has output $\boldsymbol{h}^l = [h_1^l \; h_2^l \dots h_{n_l}^l]^T$ from neurons indexed by $i \in \mathbb{N}_l = \{1, 2, \dots, n_l\}$. Let $\boldsymbol{W}^l$ be the $n_l \times n_{l-1}$ matrix where each row corresponds to the weights of a neuron of layer l, $\boldsymbol{W}_i^l$ the i-th row of $\boldsymbol{W}^l$, and $\boldsymbol{b}^l$ the vector of biases associated with the units in layer l. With $\boldsymbol{h}^0$ for $\boldsymbol{x}$ and $\boldsymbol{h}^{L+1}$ for $\boldsymbol{y}$, the output of each unit i in layer l consists of an affine function $g_i^l = \boldsymbol{W}_i^l \boldsymbol{h}^{l-1} + \boldsymbol{b}_i^l$ followed by the ReLU activation $h_i^l = \max\{0, g_i^l\}$. We denote the neuron *active* when $h_i^l = g_i^l > 0$ and *inactive* when $h_i^l = 0$ and $g_i^l < 0$. We explain later in the paper how we consider the special case in which $h_i^l = g_i^l = 0$.

3 The Linear Regions of Pruned Neural Networks

In rectifier networks, small perturbations of a given input produce a linear change on the output before the softmax layer. This happens because the neurons that are active and inactive for the original input remain in the same state if the perturbation is sufficiently small. Hence, as long as the neurons remain in their current active or inactive states, the neural network acts as a linear function.

If we consider every configuration of active and inactive neurons that may be triggered by different inputs, then the network acts as a piecewise linear function. The theory of linear regions aims to understand what affects the achievable number of such pieces, which are also known as linear regions. In other words, we are interested in knowing how many different combinations of active and inactive neurons are possible, since they make the network behave differently for inputs that are sufficiently different from one another.

Many factors may affect such number of combinations. We consider below some building blocks leading to an upper bound for pruned networks.

(i) The Activation Hyperplane: Every neuron has an input space corresponding to the output of the neurons from the previous layer, or to the input of the network if the neuron is in the first layer. For the i-th neuron in layer l, that input space corresponds to $\boldsymbol{h}^{l-1}$. The hyperplane $\boldsymbol{W}_i^l \boldsymbol{h}^{l-1} + \boldsymbol{b}_i^l = 0$ defined by the parameters of the neuron separate the inputs in $\boldsymbol{h}^{l-1}$ into two half-spaces. Namely, the inputs that activate the neuron in one side ($\boldsymbol{W}_i^l \boldsymbol{h}^{l-1} + \boldsymbol{b}_i^l > 0$) from those that do not activate the neuron in the other side ($\boldsymbol{W}_i^l \boldsymbol{h}^{l-1} + \boldsymbol{b}_i^l < 0$). We discuss in (iii) how we regard inputs on the hyperplane ($\boldsymbol{W}_i^l \boldsymbol{h}^{l-1} + \boldsymbol{b}_i^l = 0$).

(ii) The Hyperplane Arrangement: With every neuron in layer l partitioning $\boldsymbol{h}^{l-1}$ into two half-spaces, our first guess could be that the intersections of these half-spaces would lead the neurons in layer l to partition $\boldsymbol{h}^{l-1}$ into a collection of 2^{n_l} regions [52]. In other words, that there would be one region corresponding to every possible combination of neurons being active or inactive in layer l. However, the maximum number of regions defined in such a way depends on the number of hyperplanes and the dimension of space containing those hyperplanes. Given the number of activation hyperplanes in layer l as n_l and assuming for now that the size of the input space $\boldsymbol{h}^{l-1}$ is n_{l-1}, then the number of linear regions defined by layer l, or N_l, is such that $N_l \leq \sum_{d=0}^{n_{l-1}} \binom{n_l}{d}$ [81]. Since $N_l \ll 2^{n_l}$ when $n_{l-1} \ll n_l$, we note that this bound can be much smaller than initially expected — and that does not cover the other factors discussed in (iv), (v), and (vi).

(iii) The Boundary: Before moving on, we note that the bound above counts the number of full-dimensional regions defined by a collection of hyperplanes in a given space. In other words, the activation hyperplanes define the boundaries of the linear regions and within each linear region the points are such that either $\boldsymbol{W}_i^l \boldsymbol{h}^{l-1} + \boldsymbol{b}_i^l > 0$ or $\boldsymbol{W}_i^l \boldsymbol{h}^{l-1} + \boldsymbol{b}_i^l < 0$ with respect to each neuron i in layer l. Hence, this bound ignores cases in which we would regard $\boldsymbol{W}_i^l \boldsymbol{h}^{l-1} + \boldsymbol{b}_i^l = 0$ as making the neuron inactive when $\boldsymbol{W}_i^l \boldsymbol{h}^{l-1} + \boldsymbol{b}_i^l \geq 0$ for any possible input in $\boldsymbol{h}^{l-1}$, and vice-versa when $\boldsymbol{W}_i^l \boldsymbol{h}^{l-1} + \boldsymbol{b}_i^l \leq 0$, since in either case the linear region defined with $\boldsymbol{W}_i^l \boldsymbol{h}^{l-1} + \boldsymbol{b}_i^l = 0$ would not be full-dimensional and would actually be entirely located on the boundary between other full-dimensional regions.

(iv) Bounding Across Layers: As we add depth to a neural network, every layer of the network breaks each linear region defined so far in even smaller pieces with respect to the input space $\boldsymbol{h}^0$ of the network. One possible bound would be the product of the bounds for each layer l by assuming the size of the input space to be n_{l-1} [58]. That comes with the assumption that every linear region defined by the first $l-1$ layers can be further partitioned by layer l in as many linear regions as possible. However, this partitioning is going to be more detailed in some linear regions than in others because their input space might be very different. The output of a linear region in layer l is defined by a linear transformation with rank at most n_l. The linear transformation would be $\boldsymbol{h}^l = \boldsymbol{M}^l \boldsymbol{h}^{l-1} + \boldsymbol{d}^l$, where $\boldsymbol{M}_i^l = \boldsymbol{W}_i^l$ and $\boldsymbol{d}_i^l = \boldsymbol{b}_i^l$ if neuron i of layer l is active in the linear region and $\boldsymbol{M}_i^l = \mathbf{0}$ and $\boldsymbol{d}_i^l = 0$ otherwise. Hence, the output from a linear region is the composite of the linear transformations in each layer. If layer $l+1$ or any subsequent layer has more than n_l neurons, that would not imply that the dimension of the image from any linear region is greater than n_l since the output of any linear region after layer l is contained in a space with dimension at most $\min\{n_0, n_1, \dots, n_l\}$ [51]. In fact, the dimension the of image is often much smaller if we consider that the rank of each matrix $\boldsymbol{M}_i^l$ is bound by how many neurons are active in the linear region, and that in only one linear region of a layer we would see all neurons being active [62].

(v) The Effect of Parameters: The value of the parameters may also interfere with the hyperplane arrangement. First, consider the case in which the rank of the weight matrix is smaller than the number of rows. For example, if all activation hyperplanes are parallel to one another and thus the rank of the weight matrix is 1. No matter how many dimensions the input space has, this situation is equivalent to drawing parallel lines in a plane. Hence, n_l neurons would not be able to partition the input space into more than $n_l + 1$ regions. In general, it is as if the dimension of the space being partitioned were equal to the rank of the weight matrix [62]. Second, consider the case in which a neuron is stable, meaning that this neuron is always active or always inactive for any valid input [69]. Not only that would affect the dimension of the image because a stably inactive neuron always outputs zero, but also the effective number of activation hyperplanes: since the activation hyperplane associated with a stable neuron has no inputs to one of its sides, it does not subdivide any linear region [61].

(vi) The Effect of Sparsity: When we start making parameters of the neural network equal to zero through network pruning, we may affect the number of linear regions due to many factors. First, some neurons may become stable. For example, neuron i in layer l becomes stable if $\boldsymbol{W}_i^l = 0$, i.e., if that row of parameters only has zeros, since the bias term alone ends up defining if the neuron is active ($\boldsymbol{b}_i^l > 0$) or inactive ($\boldsymbol{b}_i^l < 0$). That is also likely to happen if only a few parameters are left, such as when all the remaining weights and the bias are all either positive or negative, since the probability of all parameters having the same sign increases significantly as the number of parameters left decrease if we assume that parameters are equally likely to be positive or negative. Second, the rank of the weight matrix $\boldsymbol{W}^l$ may decrease with sparsity. For example, let us suppose that the weight matrix has n rows, n columns, and that there are only n nonzero parameters. Although it is still possible that those n parameters would all be located in distinct rows and columns to result in a full-rank matrix, that would only occur in $\dfrac{n!}{\binom{n^2}{n}}$ of the cases if we assume every possible arrangement for those n parameters in the n^2 different positions. Hence, we should expect some rank deficiency in the weight matrix even if we do not prune that much. Third, the rank of submatrices on the columns may decrease even if the weight matrix is full row rank. This could happen in the typical case where the number of columns exceeds the number of rows, such as when the number of neurons decreases from layer to layer, and in that case we could replace the number of active neurons with the rank of the submatrix on their columns for the dimension of the output from each linear region in order to obtain a tighter bound.

Based on the discussion above, we propose an expected upper bound on the number of linear regions over the possible sparsity patterns of the weight matrices. We use an expected bound rather than a deterministic one to avoid the unlikely scenarios in which the impact of sparsity is minimal, such as in the previous example with n parameters leading to matrix with rank n. This upper bound considers every possible sparsity pattern in the weight matrix as equally probable, which is an assumption that aligns with random pruning and does not seem to be too strict in our opinion. For simplicity, we assume that every weight of the network has a probability p of not being pruned; or, conversely, a probability $1-p$ of being pruned. We denote p as the network *density*.

Moreover, we focus on the second effect of sparsity — through a decrease on the rank of the weight matrix — for two reasons: (1) it subsumes part of the first effect when an entire row becomes zero; and (2) we found it to be stronger than the third effect in preliminary comparisons with a bound based on it.

Theorem 1. *Let $R(l,d)$ be the expected maximum number of linear regions that can be defined from layer l to layer L with the dimension of the input to layer l being d; and let $P(k|R,C,S)$ be the probability that a weight matrix having rank k with R rows, C columns, and probability S of each element being nonzero. With p_l as the probability of each parameter in $\boldsymbol{W}^l$ from remaining in the network after pruning — the layer density, then $R(l,d)$ for $l=L$ is at most*

$$\sum_{k=0}^{n_L} P(k|R=n_L, C=n_{L-1}, S=p_L) \sum_{j=0}^{\min\{k,d\}} \binom{n_L}{j}$$

and $R(l,d)$ for $1 \leq l \leq L-1$ is at most

$$\sum_{k=0}^{n_l} P(k|R=n_l, C=n_{l-1}, S=p_l) \sum_{j=0}^{\min\{k,d\}} \binom{n_l}{j} R(l+1, \min\{n_l - j, d, k\}).$$

Proof. We begin with a recurrence on the number of linear regions similar to the one in [62]. Namely, let $R(l,d)$ be the maximum number of linear regions that can be defined from layer l to layer L with the dimension of the input to layer l being d, and let $N_{n_l,d,j}$ be the maximum number of regions from partitioning a space of dimension d with n_l activation hyperplanes such that j of the corresponding neurons are active in the resulting subspaces ($|S^l| = j$):

$$R(l,d) = \begin{cases} \sum_{j=0}^{\min\{n_L,d\}} \binom{n_L}{j} & \text{if } l = L, \\ \sum_{j=0}^{n_l} N_{n_l,d,j} R(l+1, min\{j,d\}) & \text{if } 1 \leq l \leq L-1 \end{cases} \quad (1)$$

Note that the base case of the recurrence directly uses what we know about the number of linear regions given the number of hyperplanes and the dimension of the space. That bound also applies to $\sum_{j=0}^{n_l} N_{n_l,d,j}$ in the other case from the recurrence. Based on Lemma 5 from [62], $\sum_{j=0}^{n_l} N_{n_l,d,j} \leq \sum_{j=0}^{\min\{n_l,d\}} \binom{n_l}{j}$. Some of these linear regions will have more neurons active than others. In fact, there are at most $\binom{n_l}{j}$ regions with $|S^l| = j$ for each j. In resemblance to BC, we can thus assume that the largest possible number of neurons is active in each linear region defined by layer l for the least impact on the input dimension of the following layers. Since $\binom{n_l}{j} = \binom{n_l}{n_l-j}$, we may conservatively assume that $\binom{n_l}{0}$ linear regions have n_l active neurons, $\binom{n_l}{1}$ linear regions have $n_l - 1$ active neurons, and so on. That implies the following refinement of the recurrence:

$$R(l,d) = \begin{cases} \sum_{j=0}^{\min\{n_L,d\}} \binom{n_L}{j} & \text{if } l = L, \\ \sum_{j=0}^{\min\{n_l,d\}} \binom{n_l}{j} R(l+1, min\{n_l - j, d\}) & \text{if } 1 \leq l \leq L-1 \end{cases} \tag{2}$$

Note that there is a slight change on the recurrence call, by which j is replaced with $n_l - j$, given that we are working backwards from the largest possible number of active neurons n_l with $n_l - j$.

Finally, we account for the rank of the weight matrix upon sparsification. For the base case of $l = L$, we replace n_L from the end of the summation range with the rank k of the weight matrix $\boldsymbol{W}^L$, and then we calculate the expected maximum number of linear regions using the probabilities of rank k having any value from 0 to n_L as

$$\sum_{k=0}^{n_L} P(k|R = n_L, C = n_{L-1}) \sum_{j=0}^{\min\{k,d\}} \binom{n_L}{j},$$

which corresponds to the first expression in the statement. For the case in which $l \in \{1, \dots, L-1\}$, we similarly replace n_l from the end of the summation range with the rank k of the weight matrix $\boldsymbol{W}^l$, and then we calculate the expected maximum number of linear regions using the probabilities of rank k having any value from 0 to n_l as

$$\sum_{k=0}^{n_l} P(k|R = n_l, C = n_{l-1}) \sum_{j=0}^{\min\{k,d\}} \binom{n_l}{j} R^H(l+1, \min\{n_l - j, d, k\}),$$

which corresponds to the second expression in the statement. ■

Please note that the probability of the rank of a sparse matrix is not uniform when the probability of the sparsity patterns is uniform. We discuss how to compute the former from the later as one of the items in Section 6.

4 Pruning Based on Linear Regions

Based on Theorem 1, we devise a network pruning strategy for maximizing the number of linear regions subject to the total number of parameters to be pruned. For a global density p reflecting how much should be pruned, we may thus choose a density p_l for each layer l, some of which above and some of which below p if we do not prune uniformly. We illustrate below the simpler case of pruning two hidden layers and not pruning the connections to the output layer, which is the setting used in our experiments. We focused on two layers because there is only one degree of freedom in that case: for any density p_1 that we choose, the density p_2 is implied by p_1 and by the global density p. When there are more layers involved, trying to optimize the upper bound becomes more challenging. If the effect is not as strong, it could be due to issues solving this nonlinear optimization problem rather than with the main idea in the paper.

When pruning two layers, the relevant dimensions for us are the input size n_0 and the layer widths n_1 and n_2. Assuming the typical setting in which $n_0 > n_1 = n_2$, the maximum rank of both weight matrices is limited by the number of rows (n_1 for $\boldsymbol{W}^1$ and n_2 for $\boldsymbol{W}^2$). However, the greater number of columns in $\boldsymbol{W}^1$ (n_0) implies that we should expect the rank of $\boldsymbol{W}^1$ to be greater if $p_1 = p_2$, whereas preserving more nonzero elements in $\boldsymbol{W}^2$ by pruning a little more from $\boldsymbol{W}^1$ may change the probabilities for $\boldsymbol{W}^2$ with little impact on those for $\boldsymbol{W}^1$. In some of our experiments, the second layer actually has more parameters than the first, meaning that we need to consider $p_1 > p_2$ instead of $p_1 < p_2$.

From preliminary experimentation, we indeed observed that (i) pruning more from the layer with more parameters tends to be more advantageous in terms of maximizing the upper bound; and also that (ii) the upper bound can be reasonably approximated by a quadratic function. Hence, we use the extremes consisting of pruning as much as possible from each of the two layers, say $\overline{p}_1$ and $\overline{p}_2$, in addition to the uniform density p in both layers to interpolate the upper bound. If that local maximum of the interpolation is not pruning more from the layer l with more parameters, we search for the density p_l that improves the upper bound the most by uniformly sampling densities from p all the way to $\overline{p}_l$.

5 Counting Linear Regions in Subspaces

Based on the characterization of linear regions in terms of which neurons are active and inactive, we can count the number of linear regions defined by a trained network with a Mixed-Integer Linear Programming (MILP) formulation [62]. Among other things, these formulations have also been used for network verification [9], embedding the relationship between inputs and outputs of a network into optimization problems [59,11,5], identifying stable neurons [69] to facilitate adversarial robustness verification [75] as well as network compression [60,63], and producing counterfactual explanations [37]. Moreover, several studies have analyzed and improved such formulations [15,2,8,61,1,63].

In these formulations, the parameters $\boldsymbol{W}^l$ and $\boldsymbol{b}^l$ of each layer $l \in \mathbb{L}$ are constant while the decision variables are the inputs of the network ($\boldsymbol{x} = \boldsymbol{h}^0 \in \mathbb{X}$), the ouputs before and after activation of each feedforward layer ($\boldsymbol{g}^l \in \mathbb{R}^{n_l}$ and $\boldsymbol{h}^l \in \mathbb{R}_+^{n_l}$ for $l \in \mathbb{L}$), and the state of the neurons in each layer ($\boldsymbol{z}^l \in \{0,1\}^{n_l}$ for $l \in \mathbb{L}$). By mapping these variables according to the parameters of the network, we can characterize every possible combination of inputs, outputs, and activation states as distinct solutions of the MILP formulation. For each layer $l \in \mathbb{L}$ and neuron $i \in \mathbb{N}_l$, the following constraints associate the input $\boldsymbol{h}^l$ with the outputs $\boldsymbol{g}_i^l$ and $\boldsymbol{h}_i^l$ as well as with the neuron activation $\boldsymbol{z}_i^l$:

$$\boldsymbol{W}_i^l \boldsymbol{h}^{l-1} + \boldsymbol{b}_i^l = \boldsymbol{g}_i^l \tag{3}$$
$$(\boldsymbol{z}_i^l = 1) \to \boldsymbol{h}_i^l = \boldsymbol{g}_i^l \tag{4}$$
$$(\boldsymbol{z}_i^l = 0) \to \boldsymbol{g}_i^l \leq 0 \tag{5}$$
$$(\boldsymbol{z}_i^l = 0) \to \boldsymbol{h}_i^l = 0 \tag{6}$$
$$\boldsymbol{h}_i^l \geq 0 \tag{7}$$
$$\boldsymbol{z}_i^l \in \{0,1\} \tag{8}$$

The indicator constraints (4)–(6) can be converted to linear inequalities [7].

We can use such a formulation for counting the number of linear regions based on the number of distinct solutions on the binary vectors $\boldsymbol{z}^l$ for $l \in \mathbb{L}$. However, we must first address the implicit simplifying assumption allowing us to assume that a neuron can be either active ($\boldsymbol{z}_i^l = 1$) or inactive ($\boldsymbol{z}_i^l = 0$) when the preactivation output is zero ($\boldsymbol{g}_i^l = 0$) in (3)–(8). We can do so by maximizing the value of a continuous variable that is bounded by the preactivation output of every active neuron and the negated preactivation output of every inactive neuron. In other words, we count the number of solutions on the binary variables for the solutions with positive value for the following formulation:

$$\begin{aligned}
\max \quad & f & & \qquad (9)\\
\text{s.t.} \quad & (3) - (8) & \forall l \in \mathbb{L}, i \in \mathbb{N}_l & \qquad (10)\\
& (\boldsymbol{z}_i^l = 1) \to f \leq \boldsymbol{g}_i^l & \forall l \in \mathbb{L}, i \in \mathbb{N}_l & \qquad (11)\\
& (\boldsymbol{z}_i^l = 0) \to f \leq -\boldsymbol{g}_i^l & \forall l \in \mathbb{L}, i \in \mathbb{N}_l & \qquad (12)\\
& \boldsymbol{h}^0 \in \mathbb{X} & & \qquad (13)
\end{aligned}$$

We note that constraint (12) has not been used in prior work, where it is assumed that the neuron is inactive when $\boldsymbol{g}_i^l = 0$ [62,61]. However, its absence makes the counting of linear regions incompatible with the theory used to bound the number of linear regions, which assumes that only full-dimensional linear regions are valid. Hence, this represents a small correction to count all the linear regions.

Finally, we extend this formulation for counting linear regions on a subspace of the input. This form of counting has been introduced by [25] for 1-dimensional inputs and later extended by [26] to 2-dimensional inputs. Although far from the upper bound, the number of linear regions can still be very large even for networks of modest size, which makes the case for analyzing how neural networks partition subspaces of the input. In prior work, 1 and 2-dimensional inputs have been considered as the affine combination of 1 and 2 samples with the origin, and a geometric algorithm is used for counting the number of linear regions defined. We present an alternative approach by adding the following constraint to the MILP formulation above in order to limit the inputs of the neural network:

$$\boldsymbol{h}^0 = \boldsymbol{p}^0 + \sum_{i=1}^{S} \alpha_i (\boldsymbol{p}^i - \boldsymbol{p}^0) \tag{14}$$

Where $\{\boldsymbol{p}^i\}_{i=0}^S$ is a set of $S+1$ samples and $\{\alpha_i\}_{i=1}^S$ is a set of S continuous variables. One of these samples, say $\boldsymbol{p}^0$, could be chosen to the be origin.

6 Computational Experiments

We ran computational experiments aimed at assessing the following items:

(1) if accuracy after pruning and the number of linear regions are connected;
(2) if this connection also translates to the upper bound from Theorem 1; and
(3) if that bound can guide us on how much to prune from each layer.

Our experiments involved models trained on the datasets MNIST [41], Fashion [74], CIFAR-10 [38], and CIFAR-100 [38]. We used multilayer perceptrons having 20, 100, 200, and 400 neurons in each of their 2 fully-connected layers (denoted as 2×20, 2×100, 2×200, and 2×400), and adaptations of the LeNet [41] and AlexNet [39] architectures. For each choice of dataset and architecture used, we trained and pruned 30 models. Only the fully-connected layers were pruned. In the case of LeNet and AlexNet, we considered the output of the last convolutional layer as the input for upper bound calculations, as if their respective dimensions were 400×128×84 and 1024×4096×4096. We removed the weights with smallest absolute value (magnitude pruning), using either the same density p on each layer or choosing different densities while pruning the same number of parameters in total. We discuss other experimental details later. The source code is available at `https://github.com/caidog1129/getting_away_with_network_pruning`.
Experiment 1: We compared the mean accuracy of networks that are pruned uniformly according to their network density with the number of linear regions on subspaces defined by random samples from the datasets (Figure 1) as well as with the upper bound with input dimensions matching those subspaces (Figure 2). We used a simpler architecture (2 × 20) to keep the number of linear regions small enough to count and a simpler dataset (MNIST) to obtain models with good accuracy. In this experiment, we observe that indeed the number of linear regions drops with network density and consequently with accuracy. However, the most relevant finding is that the upper bound also drops in a similar way, even if its values are much larger. This finding is important because it is actionable: if we compare the upper bound resulting from different pruning strategies, then we may prefer a pruning strategy that leads to a smaller drop in the upper bound. Moreover, it is considerably cheaper to work with the upper bound since we do not need to train neural networks and neither count their linear regions.

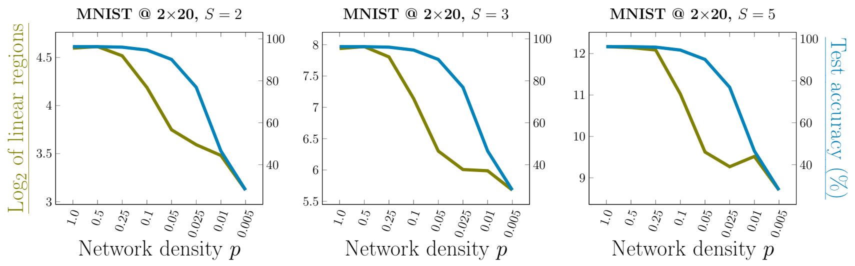

Fig. 1. Comparison between mean number of linear regions on the affine subspace defined by $S = 2$, 3, or 5 sample points (olive curve) and mean test accuracy (blue curve; right y axis) with the same density p used to prune both layers of the networks.

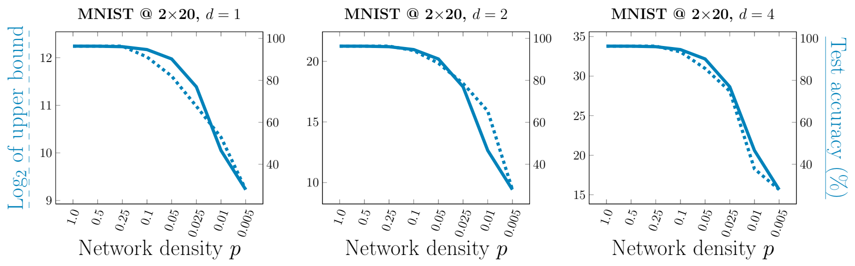

Fig. 2. Comparison between the upper bound from Theorem 1 (dashed blue curve) for input dimension $d = 1$, 2, and 4 (equivalent to $S = 2$, 3, and 5) and mean test accuracy (continuous blue curve; right y axis) for the same networks and densities from Figure 1.

Experiment 2: We compared using the same density p in each layer with using per layer densities as described in Section 4. We evaluated the simpler datasets (MNIST, Fashion, and CIFAR-10) on the simpler architectures (multilayer perceptrons and LeNet) in Figure 3, where every combination of dataset and architecture is tested to compare accuracy gain across network sizes and datasets. We set aside the most complex architecture (AlexNet) and the most complex datasets (CIFAR-10 and CIFAR-100) in Figure 4. In this experiment, we observe that pruning the fully-connected layers differently and oriented by the upper bound indeed leads to more accurate networks[‡]. The difference between the pruning strategies is noticeable once the network density starts impacting the network accuracy. We intentionally evaluated network densities leading to very different accuracies and all the way to a complete deterioration of network performance, and we notice that the gain is consistent across all of them them. If the number of parameters is similar across fully-connected layers, such as in the case of 2 × 400, we notice that the gain is smaller because more uniform densities are better for the upper bound. Curiously, we also observe a relatively greater gain with our pruning strategy for CIFAR-10 on multilayer perceptrons.

[‡]Plots for the upper bounds can be found at `https://arxiv.org/abs/2301.07966`.

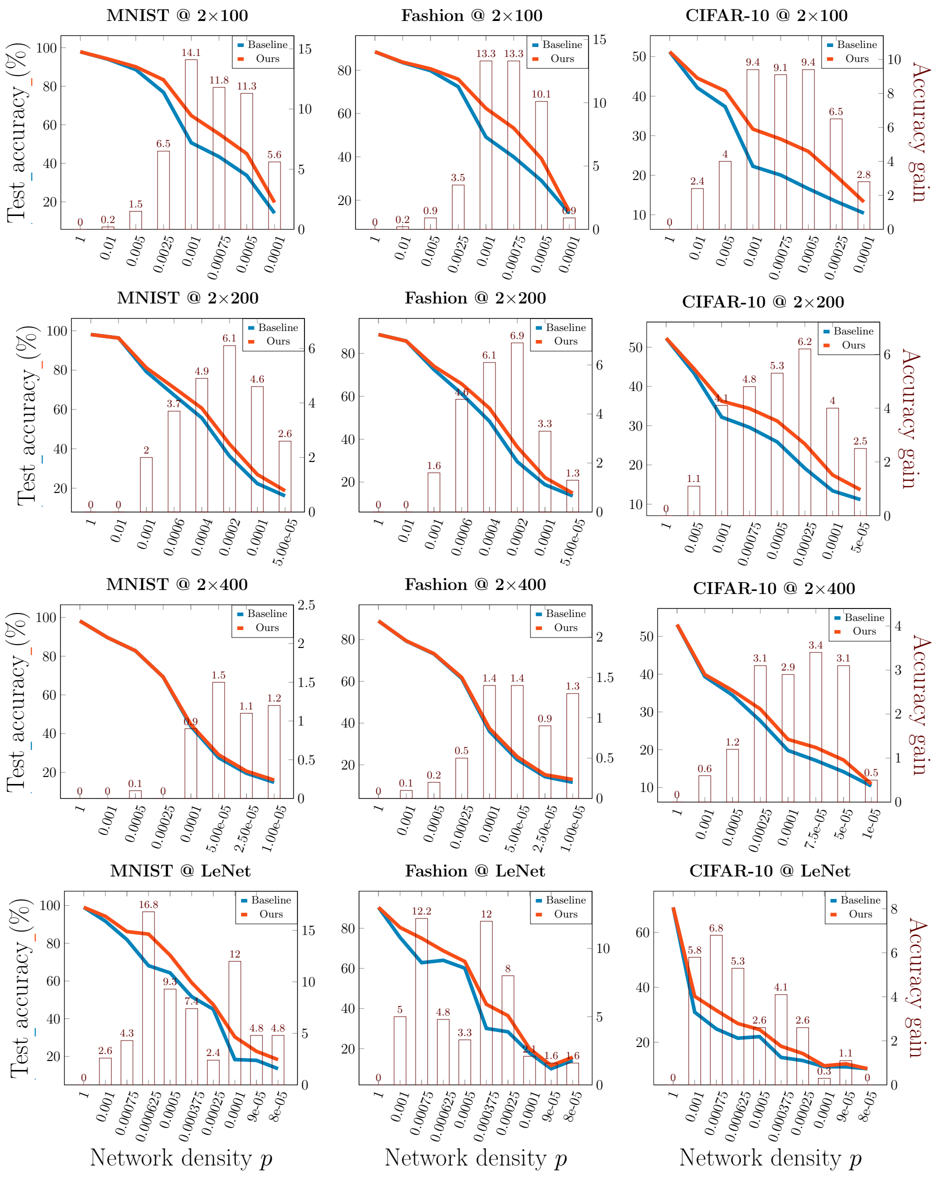

Fig. 3. Comparison between the mean test accuracy as fully-connected layers are pruned using the baseline method and our method with each network density p. In the **baseline** method, the same density is used in all layers (blue curve). In **our method**, layer densities are chosen to maximize the bound from Theorem 1 while pruning the same number of parameters (orange curve). The **accuracy gain** from using our method instead of the baseline is shown in the scaled columns (**maroon** bars; right y axis). Each column refers to a dataset among MNIST, Fashion, and CIFAR-10. Each row refers to an architecture among multilayer perceptrons (2×100, 2×200, and 2×400) and LeNet. We test every combination of dataset and architecture.

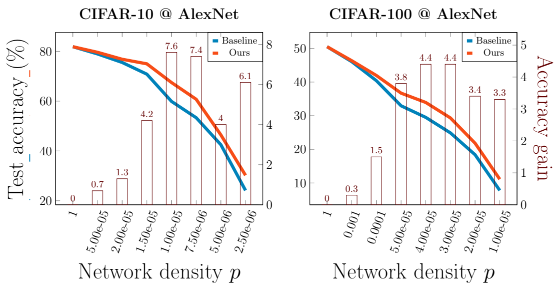

Fig. 4. Comparison between mean test accuracy for the same strategies as in Figure 3 for the AlexNet architecture, in which we test the datasets CIFAR-10 and CIFAR-100.

Additional Details: Each network was trained for 15 epochs using stochastic gradient descent with batch size of 128 and learning rate of 0.01, pruned, and then fine-tuned with the same hyperparameters for another 15 epochs. We have opted for magnitude-based pruning due to its simplicity, popularity, and frequent use as a component of more sophisticated pruning algorithms [6,16]. Our implementation is derived from the ShrinkBench framework [6]. In the baseline that we used, we opted for removing a fixed proportion of parameters from each layer (layerwise pruning) to avoid disconnecting the network, which we observed to happen under extreme sparsities if the parameters with smallest absolute value were mostly concentrated in one of the layers. We measured the mean network accuracy before pruning, which corresponds to network density $p = 1$, as well as for another seven values of p. In the experiments in Figure 3, the choices of p were aimed at gradually degrading the accuracy toward random guessing, which corresponds to accuracy 10% accuracy in those datasets with 10 balanced classes (MNIST, Fashion, and CIFAR-10). In the experiments with AlexNet in Figure 4, we aimed for a similar decay in performance.

Upper Bound Calculation: Estimating the probabilities $P(k|R, C, S)$ in Theorem 1 is critical to calculate the upper bound. For multilayer perceptrons and LeNet, we generated a sample of matrices with the same shape as the weight matrix for each layer and in which every element is randomly drawn from the normal distribution with mean 0 and standard deviation 1. These matrices were randomly pruned based on the density p, which may have been the same for every layer or may varied per layer as discussed later, and then their rank was calculated. We first generated 50 such matrices for each layer, kept track the minimum and maximum rank values obtained, $\min_r$ and $\max_r$, and then generated more matrices until the number of matrices generated was at least as large as $(\max_r - \min_r + 1) * 50$. For example, 50 matrices are generated if the rank is always the same, and 500 matrices are generated if the rank goes from 11 to 20. Finally, we calculated the probability of each possible rank based on how many times that value was observed in the samples. For example, if 10 out of 500 matrices have rank 11, then we assumed a probability of 2% for the rank of the matrix to be 11. For AlexNet, the time required for sampling is considerably longer. Hence, we resorted to an analytical approximation which is faster but possibly not as accurate. For an $m \times n$ matrix, $m \leq n$, with density p, the probability of all the elements being zero in a given row is $(1-p)^n$. We can overestimate the rank of the matrix as the number of rows with nonzero elements, which then corresponds to a binominal probability distribution with m independent trials having each a probability of success given by $1-(1-p)^n$. For 2×100, calculating the upper bound takes 15-20 seconds with sampling and 0.5-1 second with the analytical approximation. For 2×400, we have 10-20 minutes vs. 20 seconds. For AlexNet, the analytical approximation takes 20 minutes.

7 Conclusion

In this work, we studied how the theory of linear regions can help us identify how much to prune from each fully-connected feedforward layer of a neural network. First, we proposed an upper bound on the number of linear regions based on the density of the weight matrices when neural networks are pruned. We observe from Figure 2 that the upper bound is reasonably aligned with the impact of pruning on network accuracy. Second, we proposed a method for counting the number of linear regions on subspaces of arbitrary dimension. In prior work, the counting of linear regions in subspaces is restricted to at most 3 samples and thus dimension 2 [26]. We observe from Figure 1 to the number of linear regions is also aligned with the impact of pruning on network accuracy — although not as accurately as the upper bound. Third, and most importantly, we leverage this connection between the upper bound and network accuracy under pruning to decide how much to prune from each layer subject to an overall network density p. We observe from Figure 3 that we obtain considerable gains in accuracy across varied datasets and architectures by pruning from each layer in a proportion that improves the upper bound on the number of linear regions rather than pruning uniformly. These gains are particularly more pronounced when the number of parameters differs across layers. Hence, the gains are understandably smaller when the width of the layers increases (from 100 to 200 and 400) but greater when the size of the input increases (from 784 for MNIST and Fashion to 3,072 for CIFAR-10 with a width of 400). We also obtain positive results with pruning fully connected layers of convolutional networks as illustrated with LeNet and AlexNet, and in future work we intend to investigate how to also make decisions about pruning convolutional filters. Althought we should not discard the possibility of a confounding factor affecting both accuracy and linear regions, our experiments indicate that the potential number of linear regions can guide us on pruning more from neural networks with less impact on accuracy.

Acknowledgments We would like to thank Christian Tjandraatmadja, Anh Tran, Tung Tran, Srikumar Ramalingam, and the anonymous reviewers for their advice and constructive feedback. Junyang Cai, Khai-Nguyen Nguyen, Nishant Shrestha, Aidan Good, Ruisen Tu, and Thiago Serra were supported by the National Science Foundation (NSF) award IIS 2104583. Xin Yu and Shandian Zhe were supported by the NSF CAREER award IIS 2046295.

References

1. Anderson, R., Huchette, J., Ma, W., Tjandraatmadja, C., Vielma, J.: Strong mixed-integer programming formulations for trained neural networks. Mathematical Programming (2020)
2. Anderson, R., Huchette, J., Tjandraatmadja, C., Vielma, J.: Strong mixed-integer programming formulations for trained neural networks. In: IPCO (2019)
3. Arora, R., Basu, A., Mianjy, P., Mukherjee, A.: Understanding deep neural networks with rectified linear units. In: ICLR (2018)
4. Baykal, C., Liebenwein, L., Gilitschenski, I., Feldman, D., Rus, D.: Data-dependent coresets for compressing neural networks with applications to generalization bounds. In: ICLR (2019)
5. Bergman, D., Huang, T., Brooks, P., Lodi, A., Raghunathan, A.: JANOS: An integrated predictive and prescriptive modeling framework. INFORMS Journal on Computing (2022)
6. Blalock, D., Ortiz, J., Frankle, J., Guttag, J.: What is the state of neural network pruning? In: MLSys (2020)
7. Bonami, P., Lodi, A., Tramontani, A., Wiese, S.: On mathematical programming with indicator constraints. Mathematical Programming (2015)
8. Botoeva, E., Kouvaros, P., Kronqvist, J., Lomuscio, A., Misener, R.: Efficient verification of relu-based neural networks via dependency analysis. In: AAAI (2020)
9. Cheng, C., Nührenberg, G., Ruess, H.: Maximum resilience of artificial neural networks. In: ATVA (2017)
10. Cybenko, G.: Approximation by superpositions of a sigmoidal function. Mathematics of Control, Signals and Systems (1989)
11. Delarue, A., Anderson, R., Tjandraatmadja, C.: Reinforcement learning with combinatorial actions: An application to vehicle routing. In: NeurIPS (2020)
12. Denil, M., Shakibi, B., Dinh, L., Ranzato, M., Freitas, N.: Predicting parameters in deep learning. In: NeurIPS (2013)
13. Dong, X., Chen, S., Pan, S.: Learning to prune deep neural networks via layer-wise optimal brain surgeon. In: NeurIPS (2017)
14. Elesedy, B., Kanade, V., Teh, Y.W.: Lottery tickets in linear models: An analysis of iterative magnitude pruning (2020)
15. Fischetti, M., Jo, J.: Deep neural networks and mixed integer linear optimization. Constraints (2018)
16. Frankle, J., Carbin, M.: The lottery ticket hypothesis: Finding sparse, trainable neural networks. In: ICLR (2019)
17. Funahashi, K.I.: On the approximate realization of continuous mappings by neural networks. Neural Networks **2**(3) (1989)
18. Ganev, I., Walters, R.: Model compression via symmetries of the parameter space (2022), `https://openreview.net/forum?id=8MN_GH4Ckp4`
19. Glorot, X., Bordes, A., Bengio, Y.: Deep sparse rectifier neural networks. In: AISTATS (2011)
20. Good, A., Lin, J., Sieg, H., Ferguson, M., Yu, X., Zhe, S., Wieczorek, J., Serra, T.: Recall distortion in neural network pruning and the undecayed pruning algorithm. In: NeurIPS (2022)
21. Goodfellow, I., Warde-Farley, D., Mirza, M., Courville, A., Bengio, Y.: Maxout networks. In: ICML (2013)
22. Gordon, M., Duh, K., Andrews, N.: Compressing BERT: Studying the effects of weight pruning on transfer learning. In: Rep4NLP Workshop (2020)

23. Han, S., Mao, H., Dally, W.: Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. In: ICLR (2016)
24. Han, S., Pool, J., Tran, J., Dally, W.: Learning both weights and connections for efficient neural network. In: NeurIPS (2015)
25. Hanin, B., Rolnick, D.: Complexity of linear regions in deep networks. In: ICML (2019)
26. Hanin, B., Rolnick, D.: Deep relu networks have surprisingly few activation patterns. In: NeurIPS (2019)
27. Hanin, B., Sellke, M.: Approximating continuous functions by ReLU nets of minimal width. arXiv **1710.11278** (2017)
28. Hanson, S., Pratt, L.: Comparing biases for minimal network construction with back-propagation. In: NeurIPS (1988)
29. Hassibi, B., Stork, D.: Second order derivatives for network pruning: Optimal Brain Surgeon. In: NeurIPS (1992)
30. Hassibi, B., Stork, D., Wolff, G.: Optimal brain surgeon and general network pruning. In: IEEE International Conference on Neural Networks (1993)
31. Hoefler, T., Alistarh, D., Ben-Nun, T., Dryden, N., Peste, A.: Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. arXiv **2102.00554** (2021)
32. Hooker, S., Courville, A., Clark, G., Dauphin, Y., Frome, A.: What do compressed deep neural networks forget? arXiv **1911.05248** (2019)
33. Hooker, S., Moorosi, N., Clark, G., Bengio, S., Denton, E.: Characterising bias in compressed models. arXiv **2010.03058** (2020)
34. Hornik, K., Stinchcombe, M., White, H.: Multilayer feedforward networks are universal approximators. Neural Networks **2**(5) (1989)
35. Janowsky, S.: Pruning versus clipping in neural networks. Physical Review A (1989)
36. Jin, T., Roy, D., Carbin, M., Frankle, J., Dziugaite, G.: On neural network pruning's effect on generalization. In: NeurIPS (2022)
37. Kanamori, K., Takagi, T., Kobayashi, K., Ike, Y., Uemura, K., Arimura, H.: Ordered counterfactual explanation by mixed-integer linear optimization. In: AAAI (2021)
38. Krizhevsky, A.: Learning multiple layers of features from tiny images. Tech. rep., University of Toronto (2009)
39. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. Communications of the ACM **60**(6), 84–90 (2017)
40. Lebedev, V., Lempitsky, V.: Fast ConvNets using group-wise brain damage. In: CVPR (2016)
41. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE (1998)
42. LeCun, Y., Denker, J., Solla, S.: Optimal brain damage. In: NeurIPS (1989)
43. Lee, N., Ajanthan, T., Torr, P.: SNIP: Single-shot network pruning based on connection sensitivity. In: ICLR (2019)
44. Li, H., Kadav, A., Durdanovic, I., Samet, H., Graf, H.: Pruning filters for efficient convnets. In: ICLR (2017)
45. Li, H., Xu, Z., Taylor, G., Studer, C., Goldstein, T.: Visualizing the loss landscape of neural nets. In: NeurIPS (2018)
46. Liebenwein, L., Baykal, C., Carter, B., Gifford, D., Rus, D.: Lost in pruning: The effects of pruning neural networks beyond test accuracy. In: MLSys (2021)
47. Liebenwein, L., Baykal, C., Lang, H., Feldman, D., Rus, D.: Provable filter pruning for efficient neural networks. In: ICLR (2020)

48. Liu, S., Chen, T., Chen, X., Atashgahi, Z., Yin, L., Kou, H., Shen, L., Pechenizkiy, M., Wang, Z., Mocanu, D.C.: Sparse training via boosting pruning plasticity with neuroregeneration. In: NeurIPS (2021)
49. Lu, Z., Pu, H., Wang, F., Hu, Z., Wang, L.: The expressive power of neural networks: A view from the width. In: NeurIPS (2017)
50. Molchanov, P., Tyree, S., Karras, T., Aila, T., Kautz, J.: Pruning convolutional neural networks for resource efficient inference. In: ICLR (2017)
51. Montúfar, G.: Notes on the number of linear regions of deep neural networks. In: SampTA (2017)
52. Montúfar, G., Pascanu, R., Cho, K., Bengio, Y.: On the number of linear regions of deep neural networks. In: NeurIPS (2014)
53. Montúfar, G., Ren, Y., Zhang, L.: Sharp bounds for the number of regions of maxout networks and vertices of Minkowski sums (2021)
54. Mozer, M., Smolensky, P.: Using relevance to reduce network size automatically. Connection Science (1989)
55. Nair, V., Hinton, G.: Rectified linear units improve restricted boltzmann machines. In: ICML (2010)
56. Paganini, M.: Prune responsibly. arXiv **2009.09936** (2020)
57. Pascanu, R., Montúfar, G., Bengio, Y.: On the number of response regions of deep feedforward networks with piecewise linear activations. In: ICLR (2014)
58. Raghu, M., Poole, B., Kleinberg, J., Ganguli, S., Dickstein, J.: On the expressive power of deep neural networks. In: ICML (2017)
59. Say, B., Wu, G., Zhou, Y., Sanner, S.: Nonlinear hybrid planning with deep net learned transition models and mixed-integer linear programming. In: IJCAI (2017)
60. Serra, T., Kumar, A., Ramalingam, S.: Lossless compression of deep neural networks. In: CPAIOR (2020)
61. Serra, T., Ramalingam, S.: Empirical bounds on linear regions of deep rectifier networks. In: AAAI (2020)
62. Serra, T., Tjandraatmadja, C., Ramalingam, S.: Bounding and counting linear regions of deep neural networks. In: ICML (2018)
63. Serra, T., Yu, X., Kumar, A., Ramalingam, S.: Scaling up exact neural network compression by ReLU stability. In: NeurIPS (2021)
64. Singh, S.P., Alistarh, D.: WoodFisher: Efficient second-order approximation for neural network compression. In: NeurIPS (2020)
65. Sourek, G., Zelezny, F.: Lossless compression of structured convolutional models via lifting. In: ICLR (2021)
66. Sun, R., Li, D., Liang, S., Ding, T., Srikant, R.: The global landscape of neural networks: An overview. IEEE Signal Processing Magazine **37**(5), 95–108 (2020)
67. Tanaka, H., Kunin, D., Yamins, D., Ganguli, S.: Pruning neural networks without any data by iteratively conserving synaptic flow. In: NeurIPS (2020)
68. Telgarsky, M.: Representation benefits of deep feedforward networks (2015)
69. Tjeng, V., Xiao, K., Tedrake, R.: Evaluating robustness of neural networks with mixed integer programming. In: ICLR (2019)
70. Tran, C., Fioretto, F., Kim, J.E., Naidu, R.: Pruning has a disparate impact on model accuracy. In: NeurIPS (2022)
71. Tseran, H., Montúfar, G.: On the expected complexity of maxout networks. In: NeurIPS (2021)
72. Wang, C., Grosse, R., Fidler, S., Zhang, G.: EigenDamage: Structured pruning in the Kronecker-factored eigenbasis. In: ICML (2019)
73. Wang, C., Zhang, G., Grosse, R.: Picking winning tickets before training by preserving gradient flow. In: ICLR (2020)

74. Xiao, H., Rasul, K., Vollgraf, R.: Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms. arXiv **1708.07747** (2017)
75. Xiao, K., Tjeng, V., Shafiullah, N., Madry, A.: Training for faster adversarial robustness verification via inducing ReLU stability. ICLR (2019)
76. Xing, X., Sha, L., Hong, P., Shang, Z., Liu, J.: Probabilistic connection importance inference and lossless compression of deep neural networks. In: ICLR (2020)
77. Xiong, H., Huang, L., Yu, M., Liu, L., Zhu, F., Shao, L.: On the number of linear regions of convolutional neural networks. In: ICML (2020)
78. Yarotsky, D.: Error bounds for approximations with deep ReLU networks. Neural Networks **94** (2017)
79. Yu, R., Li, A., Chen, C., Lai, J., Morariu, V., Han, X., Gao, M., Lin, C., Davis, L.: NISP: Pruning networks using neuron importance score propagation. In: CVPR (2018)
80. Yu, X., Serra, T., Ramalingam, S., Zhe, S.: The combinatorial brain surgeon: Pruning weights that cancel one another in neural networks. In: ICML (2022)
81. Zaslavsky, T.: Facing up to arrangements: face-count formulas for partitions of space by hyperplanes. American Mathematical Society (1975)
82. Zeng, W., Urtasun, R.: MLPrune: Multi-layer pruning for automated neural network compression (2018)