GETTING AWAY WITH MORE NETWORK PRUNING: FROM SPARSITY TO GEOMETRY AND LINEAR REGIONS

Anonymous authors
Paper under double-blind review

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. What could explain such a drop? In this work, we explore how sparsity may affect the geometry of the linear regions defined by a neural network and consequently reduce its expected maximum number of linear regions. We observe that sparsity affects accuracy in pruned neural networks similarly to how it affects the number of linear regions as well as — and more so — our proposed upper bound on that number. Conversely, we find out that selecting the sparsity on each layer to maximize our bound very often improves accuracy in comparison to using the same sparsity across all layers, thereby providing us guidance on where to prune.

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 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 obtain 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. (2013) have shown that the parameters of neural networks can be redundant, the smoother loss landscape of larger neural networks leads to better training convergence (Li et al., 2018; Sun et al., 2020). Curiously, Jin et al. (2022) argue that pruning also smooths the 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 a primary concern in the literature.

Hoefler et al. (2021) illustrate that a moderate amount of pruning typically improves accuracy while further pruning may lead to a substantial decrease in accuracy, whereas Liebenwein et al. (2021) show that this tolerable amount of pruning depends on the task for which the network is trained. In terms of what to prune, Blalock et al. (2020) observe that most approaches consist of either

(i) removing parameters with the smallest absolute value (Hanson & Pratt, 1988; Mozer & Smolensky, 1989; Janowsky, 1989; Han et al., 2015; 2016; Li et al., 2017; Frankle & Carbin, 2019; Elesedy et al., 2020; Gordon et al., 2020; Tanaka et al., 2020; Liu et al., 2021); or

(ii) removing parameters with smallest expected impact on accuracy (LeCun et al., 1989; Hassibi & Stork, 1992; Hassibi et al., 1993; Lebedev & Lempitsky, 2016; Molchanov et al., 2017; Dong et al., 2017; Yu et al., 2018; Zeng & Urtasun, 2018; Baykal et al., 2019; Lee et al., 2019; Wang et al., 2019; Liebenwein et al., 2020; Wang et al., 2020; Xing et al., 2020; Singh & Alistarh, 2020; Yu et al., 2022), to which we can add the special case of exact compression (Serra et al., 2020; Sourek & Zelezny, 2021; Serra et al., 2021; Ganev & Walters, 2022).

While the extensive work on this topic has helped us prune more with a lesser impact on accuracy, fairness studies recently debuted by Hooker et al. (2019) have focused instead on the impact of pruning on recall — the ability of a network to correctly identify samples as belonging to each class.

Recall tends to be more severely affected for classes and features that are underrepresented in the dataset (Hooker et al., 2019; Paganini, 2020; Hooker et al., 2020), which Tran et al. (2022) attribute to differences across such groups in gradient norms and Hessian matrices of the loss function. Good et al. (2022) have shown that such recall distortions may also occur in balanced datasets, however in a more nuanced form: a moderate amount of pruning leading to comparable accuracy reduces differences in recall, whereas an excessive amount of pruning leading to lower accuracy increases differences in recall. Hence, avoiding a loss in accuracy due to pruning is also relevant for fairness.

In fact, one question keeps pushing this area: **how can we get away with more network pruning?**

Before we get there to 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 in which the neurons have piecewise linear activations — and therefore relate to its ability to classify more complex data. We have discovered that a neural network can be a factored representation of functions that are substantially more complex than the activation function of each neuron. These networks may represent a piecewise linear function in which the number of pieces — or linear regions — grows polynomially on the width and exponentially on the depth of the network (Pascanu et al., 2014; Montúfar et al., 2014).

While many papers have shown that the right choice of parameters may lead to an astronomical number of linear regions (Montúfar et al., 2014; Telgarsky, 2015; Arora et al., 2018; Serra et al., 2018), the maximum number of linear regions can be affected by narrow layers (Montúfar, 2017), the number of possible combinations of active neurons (Serra et al., 2018), and the parameters of the network (Serra & Ramalingam, 2020). Despite the exponential growth in depth, Serra et al. (2018) observe that a shallow network may in some cases yield more linear regions among architectures with the same number of neurons. Furthermore, the number of linear regions in some cases relates to the accuracy of the networks (Serra et al., 2018). Curiously, however, Hanin & Rolnick (2019a;b) have shown that the typical initialization of neural networks is unlikely to yield the expressive number of linear regions that have been reported elsewhere. Whereas most of the literature is focused on fully-connected feedforward networks using the Rectified Linear Unit (ReLU) (Nair & Hinton, 2010; Glorot et al., 2011) as the activation function, extensions of these results have been presented for convolutional networks by Xiong et al. (2020) and for maxout networks Goodfellow et al. (2013) by Montúfar et al. (2014), Serra et al. (2018), Tseran & Montúfar (2021), and Montúfar et al. (2021).

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 (Cybenko, 1989; Funahashi, 1989; Hornik et al., 1989). Their results were later extended to rectifier networks (Yarotsky, 2017) — with ReLU as the activation function of each neuron — and subsequently focused on approximations in which the networks have limited width but arbitrarily large depth (Lu et al., 2017; Hanin & Sellke, 2017). In comparison to universal approximation results, we may argue that the theory of linear regions tells us what piecewise linear functions are possible to represent — and consequently what other functions can be approximated with them — given limited resources translated in terms of the number of layers and the width of each layer.

Likewise, another question seems to remain open here: **can we use the number of linear regions to improve accuracy if trained models are typically much less expressive in practice?**

Now that you have read about both sides of the coin, you may have guessed where we are heading.

We posit that these two topics — network pruning and the theory of linear regions — can be used in conjunction. Namely, that the theory of linear regions can be used to guide us on how to prune neural networks. In order to get there, we must first address the paradox in our second question. As observed by Hanin & Rolnick (2019a), perturbing the parameters of networks designed to maximize the number of linear regions, such as the one by Telgarsky (2015), leads to a sudden drop on the number of linear regions. We interpret that as a matter on the distribution of the number of linear regions: if by perturbing these constructions we get much smaller values, we can infer that the numbers obtained with these constructions are telling us something about the long tail of the distribution. However, if certain architectural choices lead to much larger numbers of linear regions, 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 to networks that are more likely to perform well in practice.

That brings us to a crucial 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 which can be obtained by a neural network.

In summary, this paper presents the following contributions:

(i) We prove an upper bound on the expected number of linear regions over the possible ways in which the weight matrices might be pruned in a feedforward network, which specializes the bound in Serra et al. (2018) for the case in which the weight matrix is sparsified (Section 3).

(ii) We propose a method to count the number of linear regions on subspaces of input with arbitrary dimension in order to compare upper bounds and the number of linear regions of pruned networks, which provides a generalized approach to methods involving unidimensional Hanin & Rolnick (2019a) and bidemensional (Hanin & Rolnick, 2019b) inputs (Section 4).

(iii) We present computational experiments showing that (1) pruning affects the number of linear regions and the upper bound proposed very similarly; (2) pruning also affects the accuracy of the networks in a similar way; and (3) pruning more from one layer and less from another in order to improve the upper bound also leads to improvements in accuracy (Section 5).

In addition, we present background notation in Section 2 and final remarks in Section 6.

2 NOTATION

In this paper, we study the linear regions defined by the fully-connected layers of feedforward networks. For simplicity, we assume in our notation that the entire network consists of such layers. 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. Likewise, we abstract the fact that the 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 $i \in \mathbb{N}_l = \{1, 2, \dots, n_l\}$. All of those neurons have a ReLU activation function. 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 consider the neuron to be *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. When the perturbations are sufficiently small, the neurons that are active and inactive for the original input remain in the same state for the perturbed input. If we fix the neurons in their current active or inactive state, then the neural network acts as a linear function.

If we consider every configuration of active and inactive neurons, then the network corresponds to 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.

In what follows, we consider some building blocks leading to a bound for pruned neural networks.

The Geometric Perspective, Part 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 of the neural network. 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 next how we regard inputs on the hyperplane ($\boldsymbol{W}_i^l \boldsymbol{h}^{l-1} + \boldsymbol{b}_i^l = 0$).

The Geometric Perspective, Part 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 (Montúfar et al., 2014). In other words, that there would be one region corresponding to each possible combination of neurons being active or inactive in layer l. However, the maximum number of regions defined in such a way would depend 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, would be limited to at most $\sum_{d=0}^{n_{l-1}} \binom{n_l}{d}$ (Zaslavsky, 1975). 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 — not to mention other factors affecting this bound that will be discussed in part III.

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.

The Geometric Perspective, Part III: 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} (Raghu et al., 2017). However, the input space after the first layer can be much smaller than that. Within each linear region defined by the first $l-1$ layers that is then further partitioned by layer l, the output 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 $\boldsymbol{M}_i^l = \boldsymbol{0}$ and $\boldsymbol{d}_i^l = 0$ otherwise. If layer $l+1$ or any subsequent layer has more than n_l neurons, that would not imply that the dimension of the image of the transformation for 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 $\min\{n_0, n_1, \ldots, n_l\}$ (Montúfar, 2017). In fact, this dimension 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 (Serra et al., 2018).

The Geometric Perspective, Part IV: The Effect of Parameters There are some ways in which 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. That would imply that the hyperplane arrangement partitions the space in a way that could also be done if the dimension of the space was the same as the rank of the weight matrix (Serra et al., 2018). For example, if all activation hyperplanes are parallel to one another and thus the rank of the weight matrix is 1, then n_l neurons would not be able to partition the input space into more than $n_l + 1$ regions. Second, consider the case in which a neuron is stable, meaning that is always active or always inactive for any valid input (Tjeng et al., 2019). Not only that would affect the dimension of the output, but also the effective number of activation hyperplanes, since the activation hyperplane corresponding to a stable neuron has no inputs to one of its sides (Serra & Ramalingam, 2020).

The Algebraic Perspective: 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 in many different ways. First, a neuron may become stable if $\boldsymbol{W}_i^l = 0$, since the bias term alone defines 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 $\frac{n!}{\binom{n^2}{n}}$ of the cases if we assume every possible position for those n parameters in the n^2 different positions. Hence, we may except some rank deficiency in the weight matrix even if we do not prune as much. Third, the rank of submatrices on the columns may decrease even if the the rank of the weight matrix matches the number of columns. 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 proceed by proposing an expected upper bound on the number of linear regions. 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 the example in the previous paragraph. 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) it is stronger than the third effect — rank deficiency on submatrices — and do not require as strict assumptions. Curiously, a bound based on the third effect is somewhat more sophisticated to derive. We present that bound and discuss its limitations in Appendix B.

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

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

We refer the reader to Appendix A for the proof of Theorem 1. 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 5.

4 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 (Serra et al., 2018). Among other things, these formulations have also been used for network verification Cheng et al. (2017), embedding the relationship between inputs and outputs of a network into optimization problems Say et al. (2017); Delarue et al. (2020); Bergman et al. (2022), identifying stable neurons Tjeng et al. (2019) to facilitate adversarial robustness verification Xiao et al. (2019) as well as network compression (Serra et al., 2020; 2021), and producing counterfactual explanations (Kanamori et al., 2021). Moreover, several studies have analyzed and improved such formulations Fischetti & Jo (2018); Anderson et al. (2019); Botoeva et al. (2020); Serra & Ramalingam (2020); Anderson et al. (2020); Serra et al. (2021).

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 $\boldsymbol{g}^l_i$ and $\boldsymbol{h}^l_i$ as well as with the neuron activation $\boldsymbol{z}^l_i$:

$$\boldsymbol{W}^l_i \boldsymbol{h}^{l-1} + \boldsymbol{b}^l_i = \boldsymbol{g}^l_i \tag{1}$$

$$(\boldsymbol{z}^l_i = 1) \rightarrow \boldsymbol{h}^l_i = \boldsymbol{g}^l_i \tag{2}$$

$$(\boldsymbol{z}^l_i = 0) \rightarrow \boldsymbol{g}^l_i \leq 0 \tag{3}$$

$$(\boldsymbol{z}^l_i = 0) \rightarrow \boldsymbol{h}^l_i = 0 \tag{4}$$

$$\boldsymbol{h}^l_i \geq 0 \tag{5}$$

$$\boldsymbol{z}^l_i \in \{0,1\} \tag{6}$$

The indicator constraints (2)–(4) can be converted to linear inequalities (Bonami et al., 2015).

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 simplifying assumption allowing us to assume that a neuron can be either active ($\boldsymbol{z}^l_i = 1$) or inactive ($\boldsymbol{z}^l_i = 0$) when the preactivation output is zero ($\boldsymbol{g}^l_i = 0$). 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:

$$\max f \tag{7}$$

$$\text{s.t.} (1) - (6) \qquad \forall l \in \mathbb{L}, i \in \mathbb{N}_l \tag{8}$$

$$(\boldsymbol{z}^l_i = 1) \rightarrow f \leq \boldsymbol{g}^l_i \qquad \forall l \in \mathbb{L}, i \in \mathbb{N}_l \tag{9}$$

$$(\boldsymbol{z}^l_i = 0) \rightarrow f \leq -\boldsymbol{g}^l_i \qquad \forall l \in \mathbb{L}, i \in \mathbb{N}_l \tag{10}$$

$$\boldsymbol{h}^0 \in \mathbb{X} \tag{11}$$

We note that constraint (10) has not been used in prior work, where it is assumed that the neuron is inactive when $\boldsymbol{g}^l_i = 0$ Serra et al. (2018); Serra & Ramalingam (2020). 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.

Finally, we extend this formulation for counting linear regions on a subspace of the input. This form of counting has been introduced by (Hanin & Rolnick, 2019a) for 1-dimensional inputs and later extended by (Hanin & Rolnick, 2019b) 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:

$$\boldsymbol{h}^0 = \boldsymbol{p}^0 + \sum_{i=1}^{S} \alpha_i (\boldsymbol{p}^i - \boldsymbol{p}^0) \tag{12}$$

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, such as $\boldsymbol{p}^0$, could be the origin as before. However, that is not necessary.

5 Computational Experiments

We have carried out computational experiments aimed at assessing the following items:

(1) if there is a connection between network accuracy and the number of linear regions;

(2) if this connection also translates to the upper bound from Theorem 1; and

(3) if the upper bound from Theorem 1 can guide us on how much to prune from each layer.

Datasets, training details, pruning algorithms, and hyperparameters For each architecture, we used 30 different neural networks. Depending on the experiment, these networks were trained on the datasets MNIST (LeCun et al., 1998), Fashion (Xiao et al., 2017), or CIFAR-10 (Krizhevsky, 2009). Besides metrics discussed later, we measured the mean network accuracy before pruning, which corresponded to a density $p = 1$, as well as for another seven values of p that were aimed at gradually degrading the accuracy toward random guessing, which corresponded to accuracy 10% accuracy in those datasets. 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. These networks were pruned using Layerwise Magnitude Pruning (LMP) — i.e., removing the same proportion of parameters with smallest absolute from each layer. LMP worked better under the extreme sparsities used than Global Magnitude Pruning (GMP) — i.e., choosing the parameters with smallest absolute regardless of the layer — and also prevented the network from getting disconnect with GMP when p was too small. Our implementation is derived from the ShrinkBench framework Blalock et al. (2020). We have opted for magnitude-based pruning due to its simplicity, popularity, and frequent use as a component of more sophisticated pruning algorithms.

Upper bound calculation For each architecture and density p used, we calculated the upper bound by firstly estimating the probability distribution for the rank of the weight matrix in each layer — i.e., the probabilities of the form $P(k|R, C, S)$ in Theorem 1. For that purpose, 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%.

Choice of density per layer after pruning Besides pruning the networks with the same density p in each feedforward layer, we have also chosen different densities to each layer in order to increase the upper bound value while still pruning the same number of parameters. We consider varying the density from pruning as much as possible from the first layer to pruning as much as possible from the second layer. From preliminary experimentation, we observed that (i) the upper bound can be reasonably approximated by a quadratic function; and (ii) pruning more from the first layer in the settings considered tends to be more advantageous. Hence, we use both extremes mentioned above in addition to density p in both layers to interpolate the upper bound and then we evaluate the local maximum of the interpolation. If that local maximum is not pruning more from the first layer, we sample densities for the first layer uniformly from the minimum density all the way to p.

Linear region counting Given the large cost of counting linear regions in neural networks, we opted to work toward items 1 and 2 with smaller neural networks in order to hopefully establish a connection between the upper bound and the performance of the network. For that purpose, we used networks trained on MNIST with 2 feedforward layers having 20 neurons each. We counted the number of linear regions in these networks by restricting the input to the affine space defined by 2, 3, and 5 sample points randomly chosen from the dataset, which respectively define subspaces with dimension at most 1, 2, and 4. Hence, we compared those numbers of linear regions with the upper bound when the dimension of the input corresponds to the dimension of the affine subspace. In prior work, the number of linear regions in subspaces is restricted to at most 3 samples and thus dimension 2 (Hanin & Rolnick, 2019b). While our approach allows counting on subspaces of arbitrary size, we focus on subspaces of smaller dimension to keep the runtimes manageable. We contrast the mean number of linear regions on subspaces with the mean accuracy under uniform pruning in Figure 1 and with layer densities improving the bound in Figure 5 (Appendix C).

Connecting upper bounds and sparsity We worked toward item 3 using the networks from the first experiment as well as networks with 2 feedforward layers having 100, 200, and 400 neurons each and an adaptation of the LeNet architecture (LeCun et al., 1998). In the case of LeNet, the

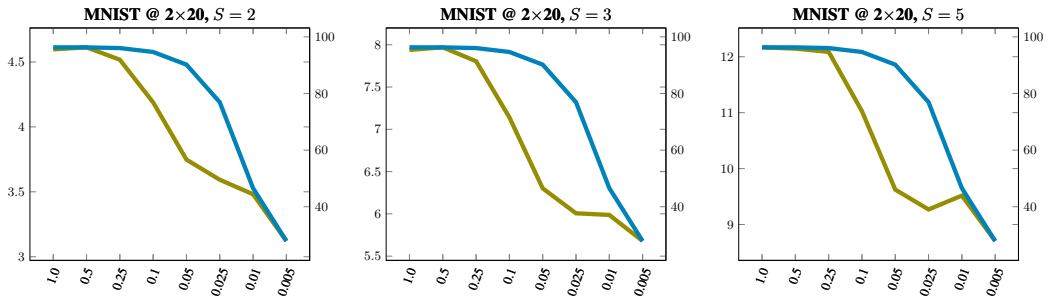

Figure 1: Comparison between the mean accuracy (blue curve; right y axis) and the log base 2 of the mean number of linear regions on the affine subspace defined by $S = 2$, 3, or 5 sample points for 30 models (olive curve) when the same density p is used in both layers.

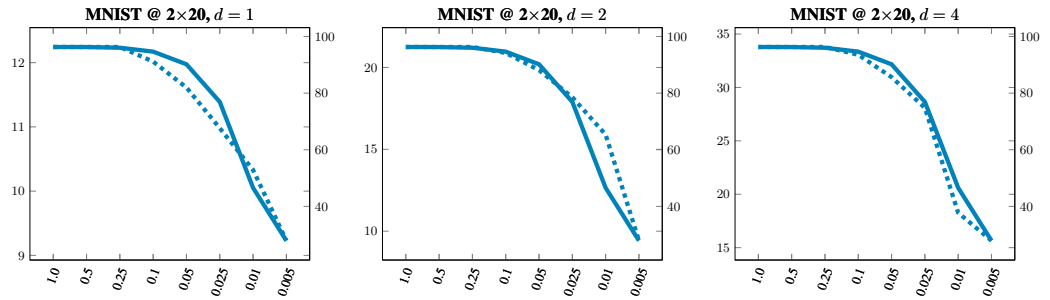

Figure 2: Comparison between the mean accuracy (continuous blue curve; right y axis) for 30 models and the log base 2 of the bound from Theorem 1 (dashed blue curve) for input dimension $d = 1$, 2, and 4 (equivalent to $S = 2$, 3, and 5) when the same density p is used in both layers.

bounds and the pruning only consider the 2 feedforward layers immediately preceding the output layer. We have trained these networks on all the datasets considered. For the smaller networks used previously, we contrast the upper bound on subspaces with the mean accuracy under uniform pruning in Figure 2 and with layer densities improving the bound in Figure 6 (Appendix C). For the larger networks, we contrast the mean accuracy under uniform pruning with the mean accuracy with layer densities improving the upper bound and illustrate the difference — or accuracy improvement — between both in Figure 3; and we contrast the upper bounds in both cases in Figure 7 (Appendix C).

For consistency across plots, dashed lines are used for upper bounds, blue is used for uniform pruning and orange is used for nonuniform pruning, and olive is used for the number of linear regions.

6 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. 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 in order to achieve a chosen weight 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, which is a case in which Figure 7 (Appendix C) shows that the upper bound can be considerably improved

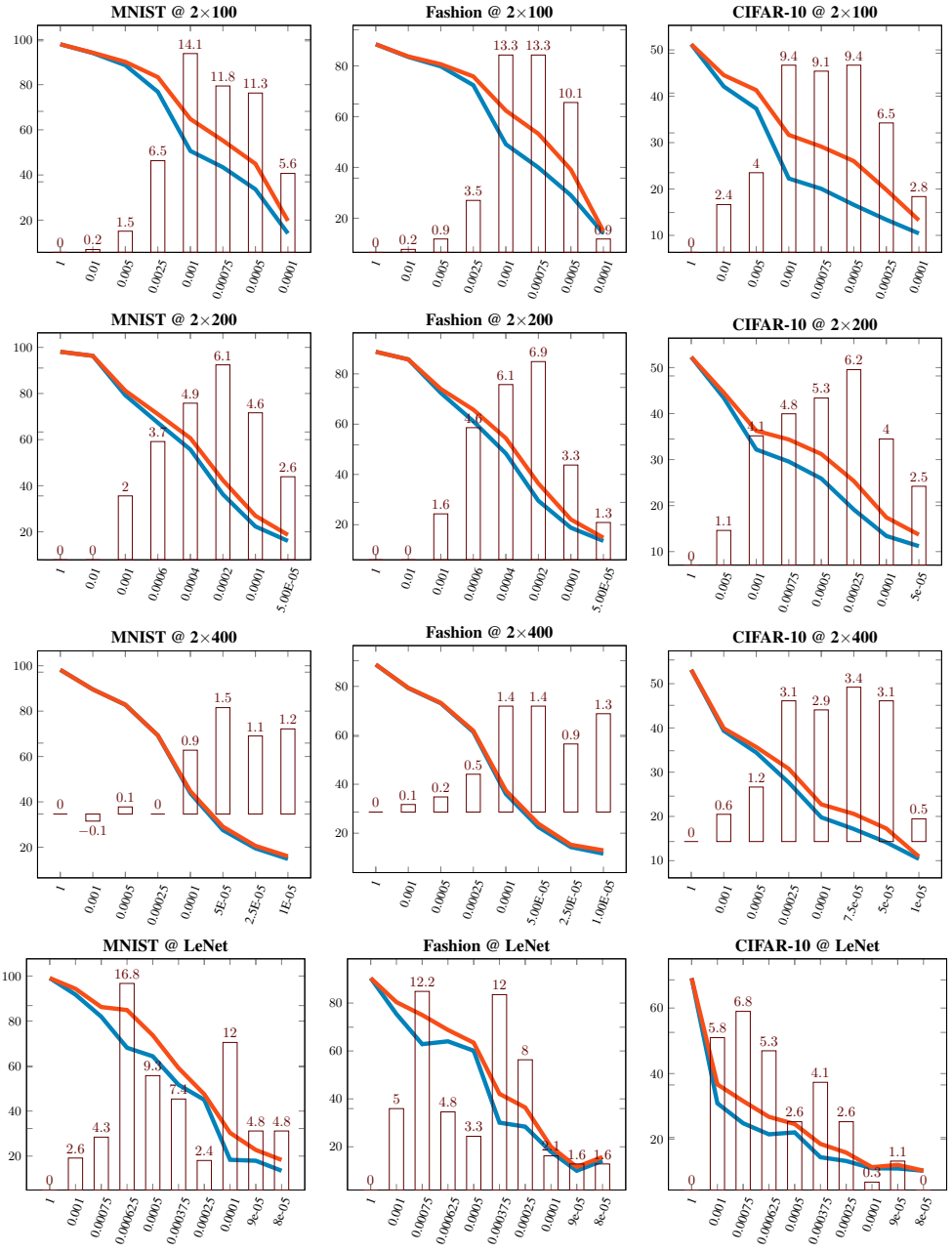

Figure 3: Comparison between mean accuracy (y-axis) for 30 models trained and then pruned with a density p of preserving each parameter of the feedforward layers (x-axis) when (i) the same density is used in all layers (blue curve); and (ii) the density is chosen to increase the bound from Theorem 1 (orange curve). The difference between them is shown in a column chart (maroon bars).

by moving away from uniform pruning. 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 in future work we intend to investigate how to also make decisions about pruning convolutional filters. We observe from Figure 5 that the number of linear regions on subspaces nevertheless increases when we prune to increase the upper bound. Hence, the potential number of linear regions can guide us on pruning more from neural networks with less impact on the accuracy.

REFERENCES

R. Anderson, J. Huchette, C. Tjandraatmadja, and J. Vielma. Strong mixed-integer programming formulations for trained neural networks. In *IPCO*, 2019.

R. Anderson, J. Huchette, W. Ma, C. Tjandraatmadja, and J. Vielma. Strong mixed-integer programming formulations for trained neural networks. *Mathematical Programming*, 2020.

R. Arora, A. Basu, P. Mianjy, and A. Mukherjee. Understanding deep neural networks with rectified linear units. In *ICLR*, 2018.

C. Baykal, L. Liebenwein, I. Gilitschenski, D. Feldman, and D. Rus. Data-dependent coresets for compressing neural networks with applications to generalization bounds. In *ICLR*, 2019.

D. Bergman, T. Huang, P. Brooks, A. Lodi, and A. Raghunathan. JANOS: An integrated predictive and prescriptive modeling framework. *INFORMS Journal on Computing*, 2022.

D. Blalock, J. Ortiz, J. Frankle, and J. Guttag. What is the state of neural network pruning? In *MLSys*, 2020.

P. Bonami, A. Lodi, A. Tramontani, and S. Wiese. On mathematical programming with indicator constraints. *Mathematical Programming*, 2015.

E. Botoeva, P. Kouvaros, J. Kronqvist, A. Lomuscio, and R. Misener. Efficient verification of relu-based neural networks via dependency analysis. In *AAAI*, 2020.

C. Cheng, G. Nührenberg, and H. Ruess. Maximum resilience of artificial neural networks. In *ATVA*, 2017.

G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 1989.

A. Delarue, R. Anderson, and C. Tjandraatmadja. Reinforcement learning with combinatorial actions: An application to vehicle routing. In *NeurIPS*, 2020.

M. Denil, B. Shakibi, L. Dinh, M. Ranzato, and N. Freitas. Predicting parameters in deep learning. In *NeurIPS*, 2013.

X. Dong, S. Chen, and S. Pan. Learning to prune deep neural networks via layer-wise optimal brain surgeon. In *NeurIPS*, 2017.

B. Elesedy, V. Kanade, and Y. W. Teh. Lottery tickets in linear models: An analysis of iterative magnitude pruning, 2020.

M. Fischetti and J. Jo. Deep neural networks and mixed integer linear optimization. *Constraints*, 2018.

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

K.-I. Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural Networks*, 2(3), 1989.

I. Ganev and R. Walters. Model compression via symmetries of the parameter space. 2022. URL `https://openreview.net/forum?id=8MN_GH4Ckp4`.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *AISTATS*, 2011.

A. Good, J. Lin, H. Sieg, M. Ferguson, X. Yu, S. Zhe, J. Wieczorek, and T. Serra. Recall distortion in neural network pruning and the undecayed pruning algorithm. In *NeurIPS*, 2022.

I. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio. Maxout networks. In *ICML*, 2013.

M. Gordon, K. Duh, and N. Andrews. Compressing BERT: Studying the effects of weight pruning on transfer learning. In *Rep4NLP Workshop*, 2020.

S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. In *NeurIPS*, 2015.

S. Han, H. Mao, and W. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. In *ICLR*, 2016.

B. Hanin and D. Rolnick. Complexity of linear regions in deep networks. In *ICML*, 2019a.

B. Hanin and D. Rolnick. Deep relu networks have surprisingly few activation patterns. In *NeurIPS*, 2019b.

B. Hanin and M. Sellke. Approximating continuous functions by ReLU nets of minimal width. *arXiv*, 1710.11278, 2017.

S. Hanson and L. Pratt. Comparing biases for minimal network construction with back-propagation. In *NeurIPS*, 1988.

B. Hassibi and D. Stork. Second order derivatives for network pruning: Optimal Brain Surgeon. In *NeurIPS*, 1992.

B. Hassibi, D. Stork, and G. Wolff. Optimal brain surgeon and general network pruning. In *IEEE International Conference on Neural Networks*, 1993.

T. Hoefler, D. Alistarh, T. Ben-Nun, N. Dryden, and A. Peste. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *arXiv*, 2102.00554, 2021.

S. Hooker, A. Courville, G. Clark, Y. Dauphin, and A. Frome. What do compressed deep neural networks forget? *arXiv*, 1911.05248, 2019.

S. Hooker, N. Moorosi, G. Clark, S. Bengio, and E. Denton. Characterising bias in compressed models. *arXiv*, 2010.03058, 2020.

K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5), 1989.

S. Janowsky. Pruning versus clipping in neural networks. *Physical Review A*, 1989.

T. Jin, D.M. Roy, M. Carbin, J. Frankle, and G.K. Dziugaite. On neural network pruning's effect on generalization. In *NeurIPS*, 2022.

K. Kanamori, T. Takagi, K. Kobayashi, Y. Ike, K. Uemura, and H. Arimura. Ordered counterfactual explanation by mixed-integer linear optimization. In *AAAI*, 2021.

Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

V. Lebedev and V. Lempitsky. Fast ConvNets using group-wise brain damage. In *CVPR*, 2016.

Y. LeCun, J. Denker, and S. Solla. Optimal brain damage. In *NeurIPS*, 1989.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998.

N. Lee, T. Ajanthan, and P. Torr. SNIP: Single-shot network pruning based on connection sensitivity. In *ICLR*, 2019.

H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. Graf. Pruning filters for efficient convnets. In *ICLR*, 2017.

Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. In *NeurIPS*, 2018.

L. Liebenwein, C. Baykal, H. Lang, D. Feldman, and D. Rus. Provable filter pruning for efficient neural networks. In *ICLR*, 2020.

L. Liebenwein, C. Baykal, B. Carter, D. Gifford, and D. Rus. Lost in pruning: The effects of pruning neural networks beyond test accuracy. In *MLSys*, 2021.

S. Liu, T. Chen, X. Chen, Z. Atashgahi, L. Yin, H. Kou, L. Shen, M. Pechenizkiy, Z. Wang, and D. C. Mocanu. Sparse training via boosting pruning plasticity with neuroregeneration. In *NeurIPS*, 2021.

Z. Lu, H. Pu, F. Wang, Z. Hu, and L. Wang. The expressive power of neural networks: A view from the width. In *NeurIPS*, 2017.

P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz. Pruning convolutional neural networks for resource efficient inference. In *ICLR*, 2017.

G. Montúfar. Notes on the number of linear regions of deep neural networks. In *SampTA*, 2017.

G. Montúfar, R. Pascanu, K. Cho, and Y. Bengio. On the number of linear regions of deep neural networks. In *NeurIPS*, 2014.

G. Montúfar, Y. Ren, and L. Zhang. Sharp bounds for the number of regions of maxout networks and vertices of Minkowski sums, 2021.

M. Mozer and P. Smolensky. Using relevance to reduce network size automatically. *Connection Science*, 1989.

V. Nair and G. Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010.

M. Paganini. Prune responsibly. *arXiv*, 2009.09936, 2020.

R. Pascanu, G. Montúfar, and Y. Bengio. On the number of response regions of deep feedforward networks with piecewise linear activations. In *ICLR*, 2014.

M. Raghu, B. Poole, J. Kleinberg, S. Ganguli, and J. Dickstein. On the expressive power of deep neural networks. In *ICML*, 2017.

B. Say, G. Wu, Y. Zhou, and S. Sanner. Nonlinear hybrid planning with deep net learned transition models and mixed-integer linear programming. In *IJCAI*, 2017.

T. Serra and S. Ramalingam. Empirical bounds on linear regions of deep rectifier networks. In *AAAI*, 2020.

T. Serra, C. Tjandraatmadja, and S. Ramalingam. Bounding and counting linear regions of deep neural networks. In *ICML*, 2018.

T. Serra, A. Kumar, and S. Ramalingam. Lossless compression of deep neural networks. In *CPAIOR*, 2020.

T. Serra, X. Yu, A. Kumar, and S. Ramalingam. Scaling up exact neural network compression by ReLU stability. In *NeurIPS*, 2021.

S. P. Singh and D. Alistarh. WoodFisher: Efficient second-order approximation for neural network compression. In *NeurIPS*, 2020.

G. Sourek and F. Zelezny. Lossless compression of structured convolutional models via lifting. In *ICLR*, 2021.

R. Sun, D. Li, S. Liang, T. Ding, and R. Srikant. The global landscape of neural networks: An overview. *IEEE Signal Processing Magazine*, 37(5):95–108, 2020.

H. Tanaka, D. Kunin, D. Yamins, and S. Ganguli. Pruning neural networks without any data by iteratively conserving synaptic flow. In *NeurIPS*, 2020.

M. Telgarsky. Representation benefits of deep feedforward networks. 2015.

V. Tjeng, K. Xiao, and R. Tedrake. Evaluating robustness of neural networks with mixed integer programming. In *ICLR*, 2019.

C. Tran, F. Fioretto, J.-E. Kim, and R. Naidu. Pruning has a disparate impact on model accuracy. In *NeurIPS*, 2022.

H. Tseran and G. Montúfar. On the expected complexity of maxout networks. In *NeurIPS*, 2021.

C. Wang, R. Grosse, S. Fidler, and G. Zhang. EigenDamage: Structured pruning in the Kronecker-factored eigenbasis. In *ICML*, 2019.

C. Wang, G. Zhang, and R. Grosse. Picking winning tickets before training by preserving gradient flow. In *ICLR*, 2020.

H. Xiao, K. Rasul, and R. Vollgraf. Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms. *arXiv*, 1708.07747, 2017.

K. Xiao, V. Tjeng, N. Shafiullah, and A. Madry. Training for faster adversarial robustness verification via inducing ReLU stability. *ICLR*, 2019.

X. Xing, L. Sha, P. Hong, Z. Shang, and J. Liu. Probabilistic connection importance inference and lossless compression of deep neural networks. In *ICLR*, 2020.

H. Xiong, L. Huang, M. Yu, L. Liu, F. Zhu, and L. Shao. On the number of linear regions of convolutional neural networks. In *ICML*, 2020.

D. Yarotsky. Error bounds for approximations with deep ReLU networks. *Neural Networks*, 94, 2017.

R. Yu, A. Li, C. Chen, J. Lai, V. Morariu, X. Han, M. Gao, C. Lin, and L. Davis. NISP: Pruning networks using neuron importance score propagation. In *CVPR*, 2018.

X. Yu, T. Serra, S. Ramalingam, and S. Zhe. The combinatorial brain surgeon: Pruning weights that cancel one another in neural networks. In *ICML*, 2022.

T. Zaslavsky. *Facing up to arrangements: face-count formulas for partitions of space by hyperplanes*. American Mathematical Society, 1975.

W. Zeng and R. Urtasun. MLPrune: Multi-layer pruning for automated neural network compression. 2018.

A PROOF OF THEOREM 1

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, 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 Serra et al. (2018). 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 ($|S^l| = j$):

$$R(l, d) = \begin{cases} \sum_{j=0}^{\min\{n_L,d\}} \binom{n_L}{j} & \text{if } l = L, \\ \sum_{j=0}^{n_l} N_{n_l,d,j} R(l + 1, min\{j, d\}) & \text{if } 1 \leq l \leq L - 1 \end{cases} \tag{13}$$

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 Serra et al. (2018), $\sum_{j=0}^{n_l} N_{n_l,d,j} \leq \sum_{j=0}^{\min\{n_l,d\}} \binom{n_l}{j}$. 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:

$$R(l, d) = \begin{cases} \sum_{j=0}^{\min\{n_L,d\}} \binom{n_L}{j} & \text{if } l = L, \\ \sum_{j=0}^{\min\{n_l,d\}} \binom{n_l}{j} R(l + 1, min\{n_l - j, d\}) & \text{if } 1 \leq l \leq L - 1 \end{cases} \tag{14}$$

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 $l \in \{1, \ldots, L - 1\}$, 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. □

B ANOTHER BOUND BASED ON SPARSITY

The next result considers the effect of pruning on the dimension of the output of feedforward layers.

Theorem 2. *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, then $R(l, d)$ for $l = L$ is at most*

$$\sum_{j=0}^{\min\{n_L,d\}} \binom{n_L}{j}$$

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

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

Proof. We start from the recurrence in equation (14) in the proof of Theorem 1.

If the weight matrix is sufficiently sparse, having $n_l - j$ neurons active in a given linear region may not necessarily imply that the dimension of the image for that linear region is $n_l - j$ due to rank deficiency of the submatrix. In other words, given a weight matrix W^l for layer l with n_l rows and n_{l-1} columns, a submatrix on $n_l - j$ of its rows may not necessarily have rank $n_l - j$. While we keep the base case of $l = L$ the same as in (14), we refine the recurrence for $l \in \{1, \ldots, L - 1\}$ to

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

Now we reformulate the recurrence for a single occurrence of $R(l+1, \min\{k, d\})$ for each value of k while changing the meaning of the indices j and k as illustrated in Figure 4:

$$R(l, d) = \sum_{j=0}^{\min\{n_l,d\}} \binom{n_l}{j} \sum_{k=0}^{n_l-j} P(k|R = n_l - j, C = n_{l-1}) R(l+1, \min\{k, d\}) \tag{15}$$

$$= \sum_{j=0}^{n_l} R(l+1, \min\{n_l - j, d\}) \sum_{k=0}^{\min\{j,d\}} \binom{n_l}{k} P(n_l - j|R = n_l - k, C = n_{l-1}). \tag{16}$$

□

Now that recurrence (16) in Theorem 2 has a similar shape to the one in equation (14), and we can compare them if we flip the terms of the later. Namely, for the case in which $1 \leq l \leq L - 1$, we have

$$R(l, d) = \sum_{j=0}^{\min\{n_l,d\}} R(l+1, min\{n_l - j, d\}) \binom{n_l}{j}$$

in equation (14).

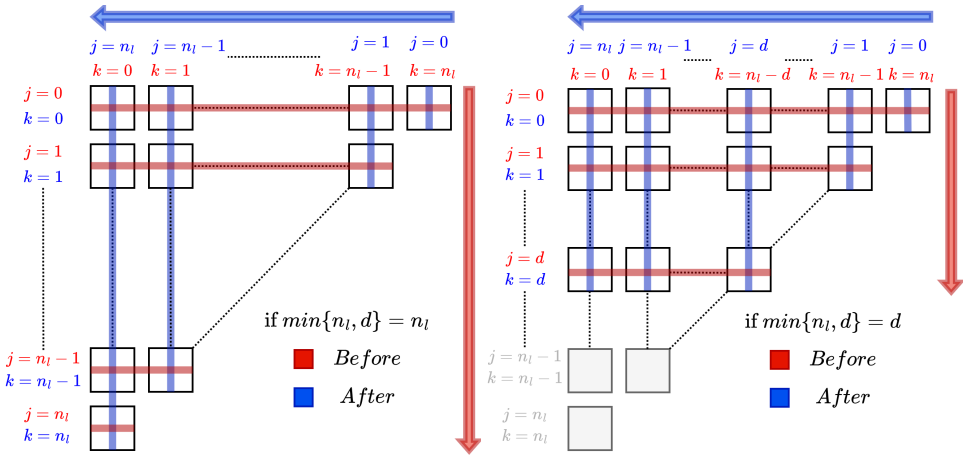

Figure 4: Mapping between the summation indices j and k of recurrence (15) in red and of recurrence (16) in blue for the cases in which (i) $\min\{n_l, d\} = n_l$ (left) and (ii) $\min\{n_l, d\} = d$ (right).

Note that each term $R(l+1, \min\{n_l - j, d\})$ in (16) is multiplied by $\sum_{k=0}^{j} \binom{n_l}{k} P(n_l - j | R = n_l - k, C = n_{l-1})$ rather than $\binom{n_l}{j}$ in (14). Hence, it would be tempting to consider a ratio between the two as an indicator of how the dimension of the image of a linear region in layer l affects the number of linear regions going forward. However, note that j goes up to n_l in the expression for R^I rather than up to $\min\{n_l, d\}$ as before for R, which captures cases in which the weight submatrix considered is rank deficient. In other words, there are additional recurrence terms with lower input dimension than those considered before.

One issue with this bound is the assumption of independence between the probabilities for different submatrices. Due to the fact that the decreases in the maximum number of linear regions due to this bound are not as expressive as those obtained with Theorem 1, we focused on attention to that Theorem instead.

C ADDITIONAL PLOTS

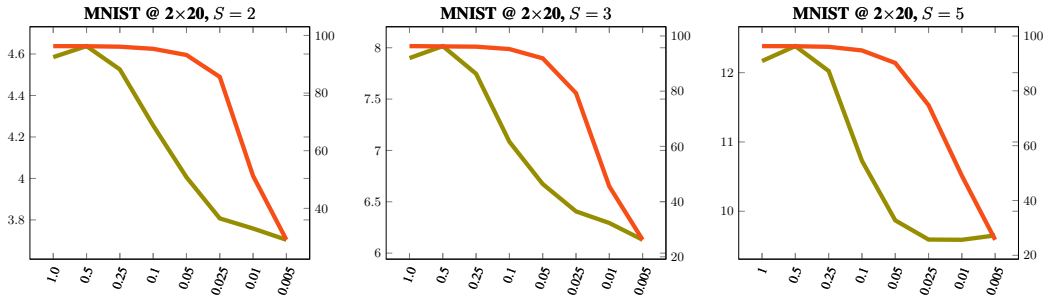

Figure 5: Comparison between the mean accuracy (orange curve; right y axis) and the log base 2 of the mean number of linear regions on the affine subspace defined by $S = 2$, 3, or 5 sample points for 30 models (olive curve; left y axis) with a layer density increasing the bound from Theorem 1.

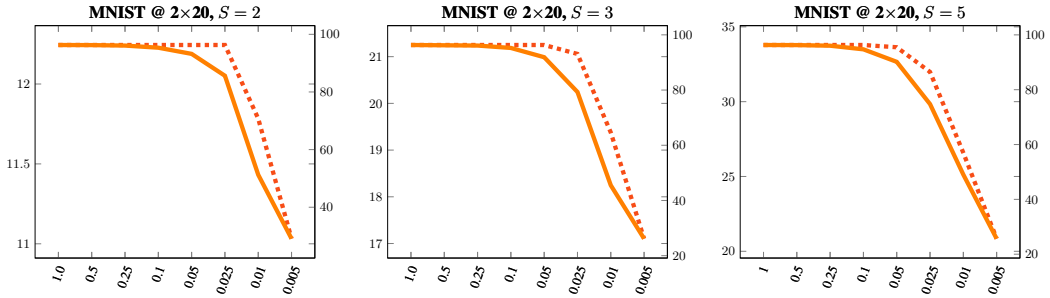

Figure 6: Comparison between the mean accuracy (continuous orange curve; right y axis) for 30 models and the log base 2 of the bound from Theorem 1 (dashed orange curve) for input dimension $d = 1$, 2, and 4 (equivalent to $S = 2$, 3, and 5) with a layer density increasing the bound from Theorem 1.

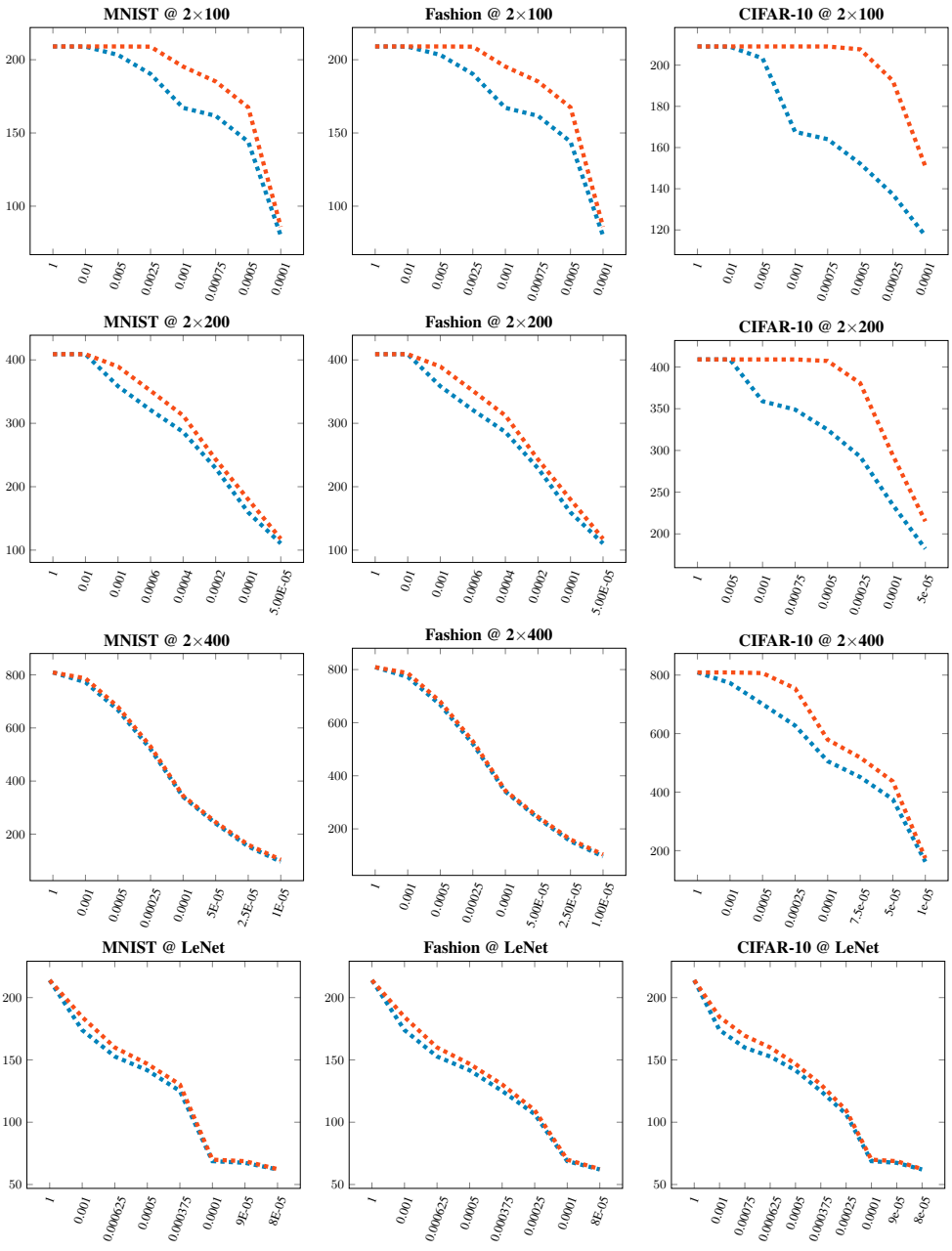

Figure 7: Comparison between the log base 2 of the bound from Theorem 1 given the density p of preserving each parameter of the feedforward layers (x-axis) when (i) the same density is used in both layers (blue curve); and (ii) the density is chosen to increase the bound (orange curve).