Sample Compression, Support Vectors, and Generalization in Deep Learning

Christopher Snyder, *Student Member, IEEE,* and Sriram Vishwanath, *Senior Member, IEEE*

***Abstract*—Not only are Deep Neural Networks (DNNs) black box models, but also we frequently conceptualize them as such. We lack good interpretations of the mechanisms linking inputs to outputs. Therefore, we find it difficult to analyze in human-meaningful terms (1) what the network learned and (2) whether the network learned. We present a hierarchical decomposition of the DNN discrete classification map into logical (AND/OR) combinations of intermediate (True/False) classifiers of the input. Those classifiers that can not be further decomposed, called atoms, are (interpretable) linear classifiers. Taken together, we obtain a logical circuit with linear classifier inputs that computes the same label as the DNN. This circuit does not structurally resemble the network architecture, and it may require many fewer parameters, depending on the configuration of weights. In these cases, we obtain simultaneously an interpretation and generalization bound (for the original DNN), connecting two fronts which have historically been investigated separately. Unlike compression techniques, our representation is *exact*. We motivate the utility of this perspective by studying DNNs in simple, controlled settings, where we obtain superior generalization bounds despite using only combinatorial information (e.g. no margin information). We demonstrate how to "open the black box" on the MNIST dataset. We show that the learned, internal, logical computations correspond to semantically meaningful (unlabeled) categories that allow DNN descriptions in plain English. We improve the generalization of an already trained network by interpreting, diagnosing, and replacing components *within* the logical circuit that is the DNN.**

I. Introduction

Deep Neural Networks (DNNs) are among the most widely studied and applied models, in part because they are able to achieve state-of-the-art performance on a variety of tasks such as predicting protein folding, object recognition, playing chess. Each of these domains was previously the realm of many disparate, setting-specific, algorithms. The underlying paradigm of Deep Learning (DL) is, by contrast, relatively similar across these varied domains. This suggests that the advantages of DL may be relevant in a variety of future learning applications rather than being restricted to currently-known settings.

The philosophy of investigating deep learning has typically focused upon keeping experimental parameters as realistic as possible. A key advantage enabled by this realism is that the insights from each experiment are immediately transferable to settings of interest. However, this approach comes with an important disadvantage: Endpoints from realistic experiments can be extremely noisy and complicated functions of variables of interest, even for systems with simple underlying rules. Newton's laws are simple, but difficult to discover except in the most controlled of settings.

The goal of our study is to understand the relationship between generalization error and network size. We seek to clarify why DNN architectures that can potentially fit all possible training labels are able to generalize to unseen data. Specifically, we would like to understand why increasing the capacity of a DNN (through increasing the number of layers and parameters) is not always accompanied by an increase in test error. To this end we study fully-connected, Gaussian-initialized, unregularized, binary classification DNNs trained with gradient descent to minimize cross-entropy loss on 2-dimensional data [1]. Even in such simple settings, generalization is not yet well-understood (as bounds can be quite large for deep networks), and our goal is to take an important step in that direction.

In adopting a minimalist study of this generalization phenomenon, the view taken in this paper is aligned with that expressed by Ali Rahimi in the NIPS2017 "Test of Time Award" talk: "This is how we build knowledge. We apply our tools on simple, easy to analyze setups; we learn; and, we work our way up in complexity... Simple experiments — simple theorems are the building blocks that help us understand more complicated systems."

—Rahimi (2017)

Our contributions are:

1) We give an intuitive, visual explanation for generalization using experiments on simple data. We show that prior knowledge about the training data can imply regularizing constraints on the image of gradient descent independently of the architecture. We observe this effect is most pronounced at the decision boundary.
2) We represent exactly a DNN classification map as a logical circuit with many times fewer parameters, depending on the data complexity.
3) We demonstrate that our logical transformation is useful both for interpretation and improvement of trained DNNs. On the MNIST dataset we translate a network "into plain English". We improve the test accuracy of an already trained DNN by debugging and replacing *within the logical circuit* of the DNN a particular intermediate computation that had failed to generalize.
4) We give a formal explanation for generalization of deep networks on simple data using classical VC bounds for learning Boolean formulae. Our bound is favorable to

Chris Snyder and Sriram Vishwanath are with the Department of Electrical and Computer Engineering, University of Texas, Austin USA e-mail: 22csnyder@gmail.com, sriram@utexas.edu

[1]Though the generalization of DNNs has been attributed in part to SGD, dropout, batch normalization, weight sharing (e.g. CNNs), etc., none of these are strictly necessarily to exhibit the apparent paradox we describe.

state of the art bounds that use more information (e.g. margin). Our bounds are extremely robust to increasing depth.

II. Setting and Notation

In this paper we study binary classification ReLU fully connected deep neural networks $\mathcal{N} : \mathbb{R}^{n_0} \mapsto \mathbb{R}$, that assign input x label $y \in \{False, True\}$ according to the value $[\mathcal{N}(x) \geq 0]$. This network has d hidden layers, each of width n_l, indexed by $l = 1, \ldots, d$. We reserve the index $0[d+1]$ for the input[output] space, so that $n_{d+1} = 1$. Our ReLU nonlinearities, $R(x)_i = \max\{0, x_i\}$, are applied coordinate-wise, interleaving the affine maps defined by weights $A^{(l)} \in \mathbb{R}^{n_{l+1} \times n_l}, b^{(l)} \in \mathbb{R}^{n_{l+1}}$. These layers compute recursively

$$\mathcal{N}^{(l+1)}(x) \triangleq b^{(l+1)} + A^{(l)} R(\mathcal{N}^{(l)}(x)).$$

Here, we include the non-layer indices 0 and $d+1$ to address the input, $x = \mathcal{N}^{(0)}(x)$, and the output, $\mathcal{N}(x) = \mathcal{N}^{(d+1)}(x)$, respectively.

For a particular input, x, each neuron occupies a binary "state" according to the sign of its activation. The set of inputs for which the activation of a given neuron is identically 0 comprises a "neuron state boundary"(NSB), of which we consider the decision boundary to be a special case by convention. We can either group these states by layer or all together to designate either the layer state, $\sigma^l(x) \in \{0,1\}^{n_l}$, or the network state, $\bar{\sigma}(x) = (\sigma^1(x), \ldots, \sigma^d(x))$, respectively.

We consider our training set, $\{(x_1, y_1), \ldots, (x_m, y_m)\}$, to represent samples from some distribution, $\mathcal{D}$. We define generalization error of the network to be the difference between the fraction correctly classified in the finite training set and in the overall distribution. We define training as the process that assigns parameters the final value of unregularized gradient descent on cross-entropy loss.

III. Insights From a Controlled Setting

A. Finding the Right Question

Note that one is *not* always guaranteed small generalization error. There are many settings where DNNs under-perform and have high generalization error. For our purposes, it suffices to recall that when the inputs and outputs $(X, Y) \sim \mathcal{D}$ are actually independent, e.g., $Y|X \sim Bernoulli(1/2)$, neural networks still obtain zero empirical risk, which implies the generalization error can be arbitrarily bad in the worst case (Zhang et al., 2017). From this, we can conclude that making either an explicit or implicit assumption about the dataset, the data, or both, is *strictly necessary* and unavoidable. At the very least, one must make an assumption which rules out random labels with high probability.

Notice that the only procedural distinction between a DNN that will generalize and one that will memorize is the dataset. Those network properties capable of distinguishing learning from memorizing, e.g., Lipschitz constant or margin, must therefore arise as secondary characteristics. They are functions of the dataset the network is trained on.

We want clean descriptions of DNN functions that generalize. By the above discussion, these are the DNNs that inherit some regularizing property from the training data through the gradient descent process. What sort of architecture agnostic language allows for succinct descriptions of trained DNNs exactly when we make some strong assumption about the training set?

B. A Deep Think on Simple Observations

We find in our experiments that DNNs of any architecture trained on linearly classifiable data are almost always linear classifiers (Fig 1).

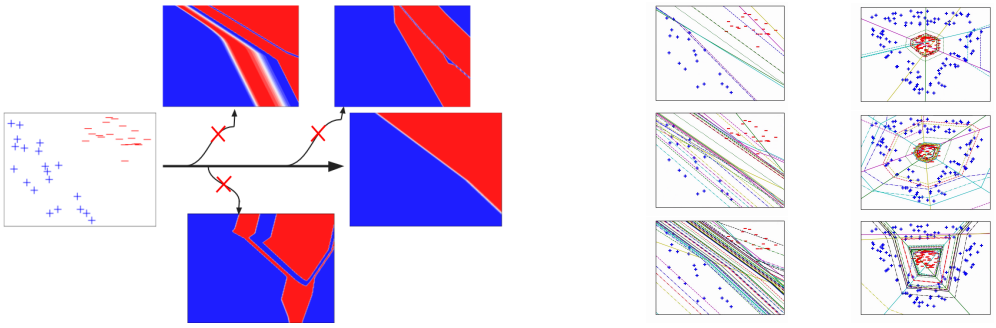
Is this interesting? Let us consider: though our network has enormous capacity, in this fixed setting of linearly separable data, the deep network behaves as though it has no more capacity than a linear model. When we discuss capacity of a class a functions, we ordinarily consider a hypothesis class consisting of networks indexed over all possible values of weights (or perhaps in a unit ball), since no such restrictions are explicitly built into in the learning algorithm. For a large architecture, such as ArchIII, this hypothesis class consists of a tremendous diversity of decision boundaries that fit the data. However, here we observe only a subset of learners: Not every configuration of weights nor every hypothesis is reachable by training with gradient descent on linearly classifiable data. Consider a learning the DNN weights corresponding to the 9 layer network, ArchIII. The VCDim of such hypotheses indexed by every possible weight assignment is $1e6$, which is unhelpfully large. But, have we measured the capacity of the correct class? If we instead use the class reachable by gradient descent, then data assumptions, which are in some form necessary, by constraining the inputs to our learning algorithm in turn restrict our hypothesis class. Linear separability is a particularly strong data assumption which reduces our the VC dimension of our hypothesis class from $1e6$ to 3. We conclude:

To ensure generalization of unregularized DNN learners, not only are data assumptions *necessary*, but also strong enough assumptions on the training data are themselves *sufficient* for generalization.

In Figures 1b, 1c, we see that a DNN with more parameters learns a more complicated *function* but not a more complicated *classifier*. For example, the number of linear regions does seem to scale with depth for fixed dataset. However, instead of intersecting the decision boundary or one another, these additional NSBs form redundant onion-like structures parallel to the decision boundary.

Since we have argued that learning guarantees in this setting are essentially equivalent to training data guarantees, capacity measures on the learned network $\mathcal{N}$ that imply generalization must somehow reflect the regularity of the data that was originally trained on. Conversely, the factors not determined by the training data structure should *not* factor into the capacity measure. For example, we desire bounds which do not grow with depth.

A capacity measure on $\mathcal{N}(x)$ that is determined entirely by restricting $\mathcal{N}$ to a neighborhood of its decision boundary accomplishes both such goals. The effect of the data is captured because the geometry of this boundary closely mirrors the that of the training data in arrangement and complexity. Consider also that behavior of $\mathcal{N}$ at the decision boundary is still is



(a) Regularizing Effect of Linearly Separable Data (b) DataI (c) DataII

Fig. 1: Structural organization of the decision boundary(DB) and NSBs (where each neuron changes from "on" to "off") of trained DNNs as the data (Fig. 1a) and architecture complexity varies (Figs.1b, 1c). In Fig. 1a, *if and only if* we include additional noisy training data to the linearly classifiable DataI can we avoid learning an (essentially) linear classifier. Regularity is not tied to data fit but data structure: all 4 ArchIII classifiers have 0 training error and vanishing loss on the same original data. In columns (1b,1c) we plot all NSBs(different linestyle[color] distinguishes NSBs of neurons in different[the same] hidden layer) and the DB(dotted). We see that for fixed dataset, increasing the architecture size (moving down a column) does not qualitatively change the learned DB. Additional layers may add more NSBs, but these organize during training in redundant, parallel shells that do not make the DB more complex. Only those NSBs that intersect the DB influence the DB and cause it to bend. Not only is the number of intersections between the DB and NSBs minimized, but also they separate from one another during training, as if by some (regularizing) "repulsive force", most readily apparent in row 2 col 1b (and in the Supplemental animations), that repels the NSBs from the decision boundary. There are several sources of relevant additional information for this figure. The Appendix contains Figures 8 and 9, which are useful to quickly visually appreciate the architectures, ArchI,II,III, and training data, DataI,II,III, that we use throughout. It also contains Figure 5, which along with the Supplemental Animations, elaborate on these NSB diagrams in number, kind, and size. 8 and 9

sufficient to determine each input classification and therefore the generalization analysis is unchanged. We claim restricting $\mathcal{N}$ to near the decision boundary destroys the architecture information used to parameterize $\mathcal{N}$. More specifically, only the existence of neurons whose NSB intersects the decision boundary can be inferred from observation of inputs x and outputs $\mathcal{N}(x)$ near the boundary. For example, this restriction is the same both for a linear classifier and for a deep network that learns a linear classifier with 50 linear regions (as in Fig. 1b).

IV. Opening the Black Box through Deep Logical Circuits

The key idea is to characterize the decision boundary of the DNN by writing the discrete valued classifier $x \mapsto \mathcal{N}(x) \geq 0$ as a *logical combination* of linear classifiers. It will turn out that the most economical descriptions will be hierarchical, so that the DNN classifier will be composed of Boolean combinations of intermediate classifiers. These intermediate classifiers identify higher order features useful for the learned task. The final result will be a logical circuit which produces the same binary label as the DNN classifier on all inputs. Finding a circuit that is "simple" is our key to both interpretability and generalization bounds.

One such example is shown in Figure 2. We show that our method translates a 9 hidden layer DNN classification map into an OR combination of just 6 linear classifiers. To emphasize, our representation *is* the DNN. It applies to all inputs: training, test, and adversarial alike.

When we train networks on the MNIST dataset, the learned circuit is more complicated, but we can still understand "role" of the intermediate classifiers within the circuit. By probing the internal circuitry with training and validation inputs, we can interpret the role of the components by cross-referencing with semantic categories (perhaps provided by a domain expert). *A priori*, there is no reason why this should be possible: The high level features a DNN learns as useful for this task are not obliged to be those that humans identify. However we see experimentally extremely encouraging evidence for this. When we group digits $0-4$ and $5-9$ into binary targets for classification, the DNN virtually always learns individual digits as intermediate steps within the logical circuit (Figure 3). For space, only those circuit components closest to the output are shown. A more involved circuit study is available in Figure 6.

The dichotomy presented is that Fig 3a demonstrates the importance of our method to interpretability, and Fig 3b, to improving generalization. Although interpretability and generalization are usually studied separately, understanding "what $\mathcal{N}$ has learned" is actually very closely related to understanding "what $\mathcal{N}$ has memorized". In fact, one of the takeaways from Figure 3 is that the *mechanism of memorization* itself can have interpretation. In Figure 3b we exploit such an interpretation to improve generalization error by "repairing" the defect.

To clarify,

V. A Theory of DNNs as Logical Hierarchies

In this section, starting with any fixed DNN classifier, we show how to construct, simplify, measure complexity of,

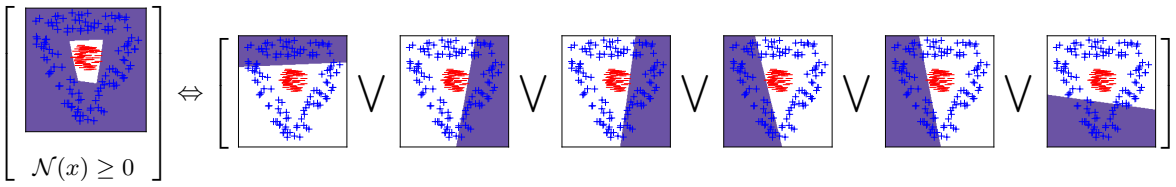


Fig. 2: The logical representation of the classifier learned by the ArchIII network to classify the DataII data (shaded region classified True). Our algorithm outputs the RHS: the rules the DNN uses to assign a positive label. For at least one of the images on the right, the input must lie in the blue region. These rules are not apparent by inspection of the $\sim 7e5$ network parameters.

and derive generalization bounds for an equivalent logical circuit. These bounds apply to the original DNN. We show they compare favorably with traditional norm based capacity measures.

A. Boolean Conversion: Notation and Technique

In our theory, we designate μ and τ as special characters with dual roles and identical conventions (all rules established for one holds for the other). We consider the symbols, μ, τ, to represent binary vectors that index by default over all binary vectorsand implicitly promote to diagonal binary matrices, $Diag(\mu), Diag(\tau)$, for purposes of matrix multiplication. For matrices, M, we define $(M_{\pm})_{i,j} = \max\{0, \pm M_{i,j}\}$. To demonstrate, we have for $d = 1$: $\mathcal{N}(x) = b^{(1)} + \max_{\mu} A_{+}^{(1)}\mu(b^{(0)} + A^{(0)}x) - \max_{\tau} A_{-}^{(1)}\tau(b^{(0)} + A^{(0)}x)$. In fact, we may write this as a MinMax or a MaxMin formulation by commuting, $-\max_{\tau} = \min -\tau$, and factoring out the Max and Min in either order. Our primary tool to relate to Boolean formulations is the following.

Proposition 1. *Let $f : \mathcal{A} \times \mathcal{B} \mapsto \mathbb{R}$. Then we have the following logical equivalence:*

$$\left[\max_{\alpha\in\mathcal{A}} \min_{\beta\in\mathcal{B}} f(\alpha, \beta) \geq 0\right] \Longleftrightarrow \bigvee_{\alpha\in\mathcal{A}} \bigwedge_{\beta\in\mathcal{B}} \left[f(\alpha, \beta) \geq 0\right]$$

We classify network states, $\bar{\Sigma} = \bar{\Sigma}_{+} \cup \bar{\Sigma}_{-}$, in terms of the output sign, $\bar{\Sigma}_{\pm} = \{\bar{\sigma}(x) | \pm \mathcal{N}(x) \geq 0\}$. We use $\bar{\Sigma}_0 = \bar{\Sigma}_{+} \cap \bar{\Sigma}_{-}$ for those states at the boundary. For $J \subset [d]$, we define $\bar{\Sigma}^J[\bar{\Sigma}_0^J]$ to be the projection of $\bar{\Sigma}[\bar{\Sigma}_0]$ onto the coordinates indexed by J. As a shorthand, we understand the symbols $\bar{\mu}^{[k]} = (\mu^1, \ldots, \mu^k)$ and $\bar{\mu} = \bar{\mu}^{[d]} = (\mu^1, \ldots, \mu^d)$ to be equivalent in any context they appear together.

Define for every τ, μ, a linear function of x, $P^{(1)}(\mu, \tau, x) = b^{(1)} + A_{+}^{(1)}\mu^1(b^{(0)} + A^{(0)}x) - A_{-}^{(1)}\tau^1(b^{(0)} + A^{(0)}x)$, called the "Net Operand". We have $[\mathcal{N}(x) \geq 0] \Leftrightarrow \vee_{\mu} \wedge_{\tau}[P^{(1)}(\mu, \tau, x) \geq 0]$. To generalize to more layers, we can recursively define:

$$P^{(l+1)}(\bar{\mu}^{[l+1]}, \bar{\tau}^{[l+1]}, x) = A_{+}^{(l+1)}\mu^{l+1}P^{(l)}(\bar{\mu}^{[l]}, \bar{\tau}^{[l]}, x) - A_{-}^{(l+1)}\tau^{l+1}P^{(l)}(\bar{\tau}^{[l]}, \bar{\mu}^{[l]}, x) + b^{(l+1)}.$$

One can derive by substitution that $P^{(d)}(\bar{\sigma}(x), \bar{\sigma}(x), x) = \mathcal{N}(x)$. This choice of $\bar{\mu} = \bar{\tau} = \bar{\sigma}(x)$ will always be a saddle point solution to Eqn 1 in the following theorem.

Theorem 1. *Let $P^{(d)}$ be the net operand for any fully-connected ReLU network, $\mathcal{N}$. Then,*

$$\mathcal{N}(x) = \max_{\mu^d} \min_{\tau^d} \cdots \max_{\mu^1} \min_{\tau^1} P^{(d)}(\bar{\mu}, \bar{\tau}, x) \tag{1}$$

$$\left[\mathcal{N}(x) \geq 0\right] \Leftrightarrow \bigvee_{\mu^d} \bigwedge_{\tau^d} \cdots \bigvee_{\mu^1} \bigwedge_{\tau^1} \left[P^{(d)}(\bar{\mu}, \bar{\tau}, x) \geq 0\right] \tag{2}$$

Notice that we can derive the second line (2) from the first (1) by recursive application of Proposition 1. Since we index over all binary states, the number of terms in our decomposition (Eqn 2) is extremely large. Though (it turns out) we may simply matters considerably by indexing instead over network states, $\bar{\Sigma}$. The next Theorem says that when the right hand side(RHS) of Eqn. 1 is indexed by only those states realized at the decision boundary, $\bar{\Sigma}_0$, the RHS still agrees with $\mathcal{N}(x)$ in *sign*, but necessarily numerical value. Thus they are equivalent classifiers.

Theorem 2. *Let $\mathcal{N}$ be a fully-connected ReLU network with net operand, $P^{(d)}$, and boundary states, $\bar{\Sigma}_0$. Then,*

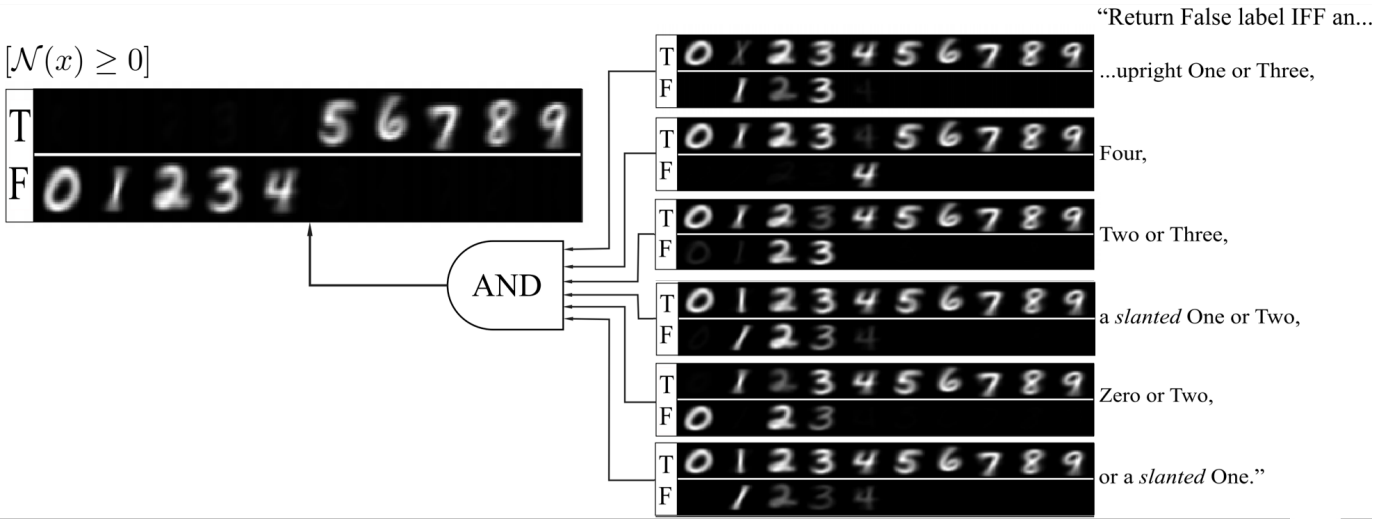
$$[\mathcal{N}(x) \geq 0] \Leftrightarrow \bigvee_{\mu^d \in \bar{\Sigma}_0^d} \bigwedge_{\tau^d \in \bar{\Sigma}_0^d} \bigvee_{\{\mu^{d-1} | (\mu^{d-1}, \mu^d) \in \bar{\Sigma}_0^{d-1,d}\}} \cdots \bigvee_{\{\mu^1 | \bar{\mu} \in \bar{\Sigma}_0\}} \bigwedge_{\{\tau^1 | \bar{\tau} \in \bar{\Sigma}_0\}} \left[P^{(d)}(\bar{\mu}, \bar{\tau}, x) \geq 0\right] \tag{3}$$

The proofs for both Theorems 1 and 2 can be found in the Appendix VIII-F. We also include explicit pseudocode, "Network Tree Algorithm" 2 (in Appendix VIII-E) for constructing our Logical Circuit from $\bar{\Sigma}_0$. Somehow, we find the actual python implementation more readable, which we have included in the supplemental named "network_tree_decompositionpy". We use this file to generate the readout in Figure 10d (Appendix VIII-D) to provide tangible, experimental support for the validity of our conversion algorithm.

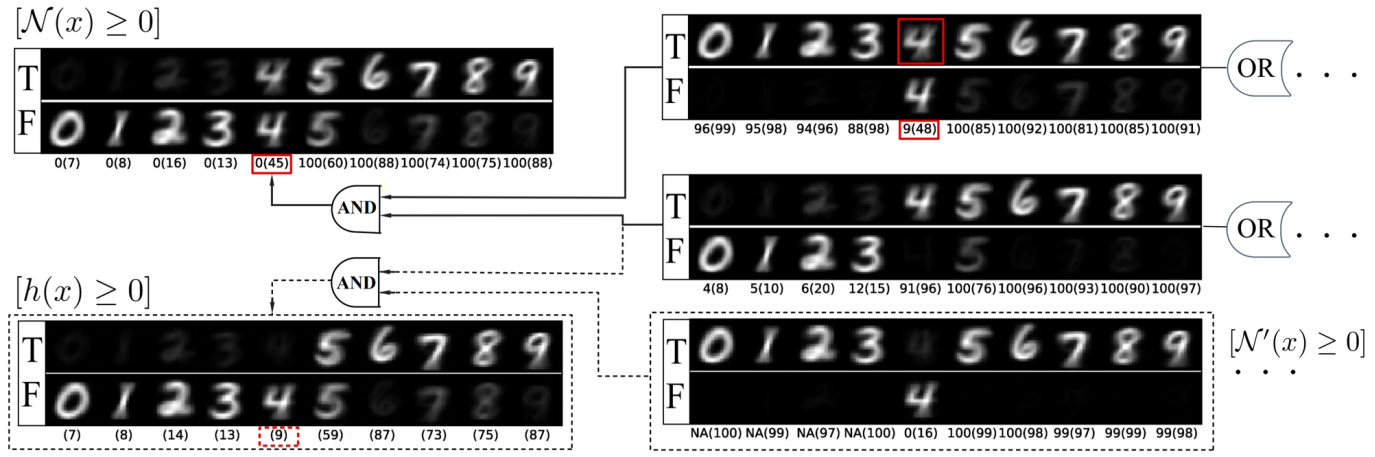
B. Formalizing Capacity for Logical Circuits

We repurpose the following theorem used by (Bartlett et al., 2017a) for ReLU networks data-independent VC dimension bounds.

Theorem 3. *(Theorem 17 in (Goldberg and Jerrum, 1995)): Let k,n be positive integers and $f : \mathbb{R}^n \times \mathbb{R}^k \mapsto \{0, 1\}$ be a function that can be expressed as a Boolean formula containing*



(a) Trained DNN with a Concise "English" Description



(b) A circuit with localized memorization

Fig. 3: Selected subsets of the logical circuits corresponding to binary classification DNNs trained on the MNIST dataset. Each 2×10 array represents a different binary classifier within the network circuit, which assigns True or False to every input image. In Fig. 3a[Fig. 3b] training[test] images of number l contribute in the l^{th} column to the brightness of either top or the bottom row of every array. The choice of row corresponds to whether corresponding classifier outputs True or False. The diagram reads right to left along solid arrows terminating in the leftmost array corresponding to the DNN binary output $[\mathcal{N}(x) \geq 0]$. The training objective only explicitly distinguishes $0-4$ from $5-9$. Yet, we see that the intermediate logical computations the network learns delineate semantically meaningful subcategories. In Fig. 3a, the DNN internal logic even admits a description *in plain English*. We show in Figure 3b how the internal logical circuitry *within the DNN* can be tweaked to improve generalization. Connected with solid lines, we see a network that has overfit badly $(1.0, 0.78$ train and test accuracy). The percentage [in parenthesis] under each column indicates how that training[test] digit is assigned True. We see that the intermediate classifier (middle right) struggles to separate Four from the positive labels. A second classifier (top right) is dedicated to learning to identify Fours as False, allowing the network to fit the training data. However, by comparing training and test performance, we can see that these Fours are not learned but memorized: As shown in solid rectangles, the intermediate and DNN classifier, respectively, assign True to $9\%[48\%]$ and $0\%[45\%]$ of the Fours in the training[test] set, accounting for the bulk of the generalization error! In practice, this network would be discarded and retrained from scratch. Since we now have access to the internal logic of the network, we are instead able to surgically replace the memorizing component. The first step we have done implicitly: we use *domain knowledge* to interpret the component function as "excluding Fours". We then train a second network, we call a "prosthetic", learning $[\mathcal{N}'(x) \geq 0]$ (with the same settings and data), to label 4 as False and $5-9$ as True. We can then excise the memorizing component, replacing its role in the logical circuit with the prosthetic (bottom right) to obtain a new classifier consisting of the three classifiers connected by dotted lines. The classifier we engineer ($[h(x) \geq 0]$ bottom left) does better on Fours, $45\% \rightarrow 9\%$ classified True (dotted rectangle) and has higher test accuracy overall $(.78 \rightarrow .83)$.

s distinct atomic predicates where each atomic predicate is a polynomial inequality or equality in $k+n$ variables of degree at most d. Let $\mathcal{F} = \{f(\cdot, w) : w \in \mathbb{R}^k\}$. Then VCDim$(\mathcal{F}) \leq 2k\log_2(8eds)$.

As a short hand, we refer to any Boolean formula satisfying the premises in the above theorem as class (k, s, d). If we consider (for fixed $\bar{\Sigma}_0$) the complexity of learning the weights defining the linear maps in Eqn. 2, Jerrum's Theorem tells us that we are primarily concerned with the number of parameters being learned. Fortunately, we only pay a learning penalty for those weights distinguishable by neuron activations in $\bar{\Sigma}_0$. For example, within the same layer, a single neuron is sufficient model any collection of neurons which are always "on" or "off" simultaneously at the decision boundary (even if this is false elsewhere). In general, we can restrict to a subset of $r_l = rank(\bar{\Sigma}_0^l)$ representative neurons without sacrificing expressivity at the boundary. We can additionally delete entire layers when $r_l = 1$. We use $\bar{r} \triangleq (r_0, r_1, \ldots, r_d)$ to group the dimensions of the reduced architecture into a single vector. Note that when $\mathcal{N}$ is a linear classifier, then $\bar{r}$ is a vector of all 1s. $r_l = 1$ at every layer.

Finally, we define $\Phi(\mathcal{N}) : \mathbb{R}^k \times \mathbb{R}^{n_0}$ to be the Boolean function in Eqn 2 corresponding to the *reduced* network, whose depth we also overload as d, and take k to be the number of parameters on which the formula depends. The formula has $s = |\bar{\Sigma}_0|^2$ inequalities. The explicit calculations for determining $k, s, d, \bar{r}$ are described Function $MinimalDescrip$ in Algorithm 1 in Appendix VIII-E. The following is automatic given the discussion so far.

Theorem 4. *Let $\mathcal{N} : \mathbb{R}^{n_0} \mapsto \mathbb{R}$ be a fully-connected ReLU network. Suppose the Boolean formula, $\Phi(\mathcal{N})$, is of class (k, s, d). Define the hypothesis class $\mathcal{H}_{\Phi(\mathcal{N})} \triangleq \{x \mapsto \Phi(\mathcal{N})(w, x) | w \in \mathbb{R}^k\}$. Then*

1) $x \mapsto [\mathcal{N}(x) \geq 0] \in \mathcal{H}_{\Phi(\mathcal{N})}$

2) VCDim$(\mathcal{H}_{\Phi(\mathcal{N})}) \leq 2k\log_2(8esd)$

Of course, this bound only applies to the learned DNN if the hypothesis class $\mathcal{H}_{\Phi(\mathcal{N})}$ is implied in advance. To address (informally) the capacity for a single classifier, $\mathcal{N}$, we define $VC^{Bool}_{k,d,s}(\mathcal{N}) \triangleq 2k\log_2(8esd)$ to be the complexity of learning the parameters of the $(k, s, d)-$Boolean formula representing $\mathcal{N}$. This is an upper bound for the smallest complexity over formulae Φ and classes $\mathcal{H}_{\Phi(\mathcal{N})}$ containing $\mathcal{N} \geq 0$ as a member. In Figure 4, we train $\mathcal{N}$ to classify samples in DataIII and compare qualitatively our capacity measure, $VC^{Bool}_{k,d,s}(\mathcal{N})$, with those of other well-known approaches as we vary the network size and training duration and depth. We compare with methods which appear at first glance to make use of additional information—that of scale, norm, and margin—which should in principle produce tighter bounds.

And yet, even though we do not take advantage of margin, we enjoy a comfortable edge over other comparable methods. Under all conditions, our bound seems to be orders of magnitude smaller than these other (well-respected) bounds. So, what is going on? In fact, it is *our* bound that is advantaged by using more (between-layer) information!.

We revisit the observation that a very deep DNN trained on linearly separable data is a linear classifier. We think that this simple characterization should somehow be accessible to our capacity measure through the weights. Linearly separable data represents, to us, the simplest, plausible, real-world proving ground for models of DNN generalization error. The methods with which we compare bound the distortion applied by each layer in terms of a corresponding weight matrix norm and accumulate the result. We should like our method to "realize" that the DNN classifier is linear, but this can not be discovered by scoring each layer. In fact, having an efficient Boolean representation is a *global* property that is sensitive to the relative configuration of weights across all layers. It is not information that is contained in the weight norms used by other methods, which destroy weight-sign information, among other properties, on which linearity of the classifier depends. We would even suggest that our notion of regularity is "more nuanced" in the sense that whether a layer is well-behaved only makes sense to talk about within the context of the overall network.

Returning to Figure 4b, we observe that we our bound is relatively stable with respect to increasing architecture size and depth. This behavior is instructive in its distinction from that of uniform (data-independent) VC dimension bounds, VC^{NoData}, which depend on the architecture alone. That these bounds produce unreasonably large, vacuous bounds for over-parameterized models is widely known and often recited. Perhaps this notoriety has dissuaded combinatorial analyses of DNN complexity altogether. However, our results demonstrate that the vast majority of the bloat in these VC^{NoData} bounds can be attributed to a lack of strong data assumptions and not to its combinatorial nature. When we compare against our own (also combinatorial) measure, $VC^{Bool}_{k,d,s}$, in Table I we observe that $VC^{Bool}_{k,d,s}$ produces bounds that are orders of magnitude smaller. We account for this discrepancy as follow: While VC^{NoData} yields weak bounds on generalization that always apply, $VC^{Bool}_{k,d,s}$ instead produces strong bounds that apply only when the data is nice. These bounds are smaller because the set of DNNs achievable by gradient descent on nice data is much more regular, and of smaller VC dimension. We explore this comparison in more depth in Appendix VIII-A.

Lastly, we offer some perspectives connecting our generalization studies to building better models in the future. There are many descriptions of complexity for DNNs. What makes ours a "good" one? All are equally valid in the sense that *every* one of them can prescribe some sufficiently strong regularity condition that will provably close the gap between training and test error. But, perhaps we should be more ambitious. We actually want to decrease model capacity *while also* retaining the ability to fit those patterns "typical" of real world data. While this second property is critical, it is also completely unclear how to guarantee, even analyze, or even define unambiguously. We surmise that since our capacity measure $VC^{Bool}_{k,d,s}$ seems empirically to be *already* minimized when the data is sufficiently structured, we can hope (and plausibly hypothesize) that those patterns that can be learned efficiently by a function class where $VC^{Bool}_{k,d,s}$ is controlled *explicitly* will not differ from those suited to unregularized DNNs, where we expect the structured nature of real world

data to *implicitly* regulate $VC^{Bool}_{k,d,s}$ already.

VI. Related Work

Our discussion of the role the data plays in generalization is perhaps most similar to Arpit et al. (2017). Many authors have studied the number of linear regions of a DNN before, usually focusing on a 2D slice or path through a the data (Serra et al., 2018; Raghu et al., 2017; Arora et al., 2018a), optionally including study of how these regions change with training (Hanin and Rolnick, 2019; Novak et al., 2018) or an informal proxy for network "complexity" (Zhang et al., 2018; Novak et al., 2018)

Formal approaches to explain generalization of DNNs fall into either "direct" or "indirect" categories. By direct, we mean that the bounds apply exactly to the trained learner, not to an approximation or stochastic counterpart. Ours falls under this category, so these are the bounds we compare to, including (Neyshabur et al., 2015; Bartlett et al., 2017b; Neyshabur et al., 2017), which we compare to in Fig. 4. While our approach relies on bounding possible training labelings (VCdim), these works all rely on having small enough weight norm compared to output margin.

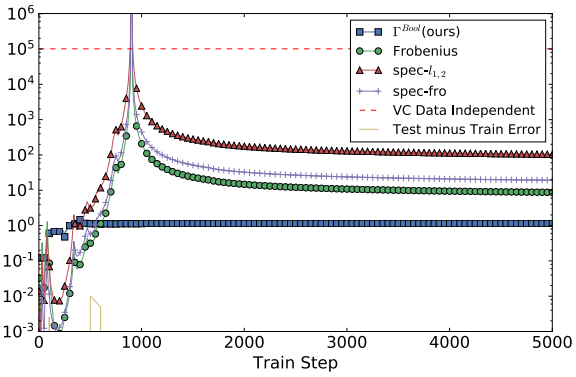
Indirect approaches analyze either a compressed or stochastic version of the DNN function. For example, PAC-Bayes analysis (McAllester, 1999) of neural networks (Langford and Caruana, 2002; Dziugaite and Roy, 2017) produces uniform generalization bounds over distributions of classifiers which scale with the divergence from some prior over classifiers. Recently, Valle-Pérez et al. (2019) produced promising such PAC-Bayes bounds, but they rely on an assumption that training samples the zero-error region uniformly, as well as some approximations of the prior marginal likelihood. Interestingly, they also touch on descriptional complexity in the appendix, which is thematically similar to our approach, but do not seem to have an algorithm to produce such a description. Another popular approach is to study a DNN through its compression (Arora et al., 2018b)(Zhou et al., 2018). Unlike our approach, which studies an equivalent classifier, these bounds apply only to the compressed version.

VII. Conclusions

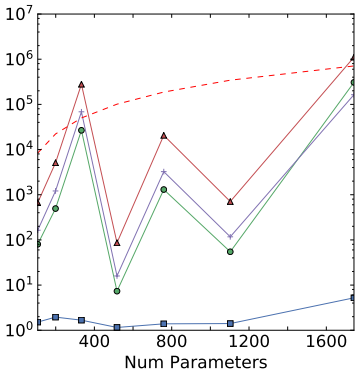
The motivation for our investigation was to describe regularity from the viewpoint of "monotonicity". Suppose that during training, the activations of a neuron in a lower layer separate the training data. While the specifics of gradient descent can be messy, there is no "reason" to learn anything other than a monotonic relationship (as we move in the input space) between the activations of that neuron, intermediate neurons in later layers, and the output. Two neurons related in this manner necessarily share discrete information about their state. The same is true of any tuple whose corresponding set of NSBs have empty intersection. We showed that NSBs adopt non-intersecting, onion-like structures, implying that very few measurements of network state are sufficient to determine the output label with a linear classifier. The "reason" VC^{NoData} produces such pessimistic bounds is because in the worst case, every binary value of $\bar{\sigma}(x)$ is required to determine $\mathcal{N}(x) \geq 0$ up to linear classifier. We expect structure in the data to reduce capacity by excluding these worst cases. For linearly separable data, the the learned DNN classifier depends on *no entry* of $\bar{\sigma}(x)$.

As a result, we have produced a powerful method for analyzing, interpreting, and improving DNNs. A deep network is a black box model for learning, but it need not be treated as such by those who study it. Our logical circuit formulation requires no assumptions and seems extremely promising for introspection and discussion of DNNs in many applications.

Whether our approach can be extended or adapted to other datasets is an pressing question for future research. An important and particularly difficult open question (precluding such an investigation presently) is the efficient determination of $\bar{\Sigma}_0$ (or even $\bar{\Sigma}$) analytically given the network weights. Such an algorithm seems prerequisite to bring deep logical circuit analysis to bear on datasets of higher dimension where we can no longer grid search.



(a) Capacity vs Training Step



(b) Capacity vs Num Parameters

Fig. 4: Qualitative comparisons of bounds on the generalization error for networks trained on DataIII during training (Fig. 4a) and as additional layers are added (Fig. 4b). Though our bound is in terms of VC dimension only, we compare favorably with other bounds that additionally use margin. Interestingly, the spike in capacity that occurs around 1000 training steps is not reflected in our bound, but captured by others. Thus, our method may be blind to some interesting training dynamics, for example, a massive shift in the relative alignment of weight vectors that leaves the intersection system of neuron state boundaries unchanged. The empirical phenomenon of depth-invariant generalization error is consistent with the behavior of our bound (Fig. 4b). These trends are representative of all 9 experiments (Figure 7).

Frobenius: $\frac{1}{m}\frac{1}{\gamma^2}\prod_{l=1}^{d}\|A^{(l)}\|_F^2$ (Neyshabur et al., 2015)

spec-$l_{1,2}$: $\frac{1}{m}\frac{1}{\gamma^2}\prod_{l=1}^{d}\|A^{(l)}\|_2^2\sum_{l=1}^{d}\frac{\|A^{(l)}\|_{1,2}^2}{\|A^{(l)}\|_2^2}$ (Bartlett et al., 2017b)

spec-fro: $\frac{1}{m}\frac{1}{\gamma^2}\prod_{l=1}^{d}\|A^{(l)}\|_2^2\sum_{l=1}^{d}h_l\frac{\|A^{(l)}\|_F^2}{\|A^{(l)}\|_2^2}$ (Neyshabur et al., 2017)

Γ^{Bool}(ours): $\min_{k,s,d}\sqrt{\frac{VC^{Bool}_{k,s,d}(\mathcal{N})}{m}}$.

References:

Ali Rahimi. NIPS2017 Test-of-time award presentation - YouTube, 2017. URL https://www.youtube.com/watch?v=x7psGHgatGM.

Chiyuan Zhang, Samy Bengio, Google Brain, Moritz Hardt, Benjamin Recht, Oriol Vinyals, and Google Deepmind. Understanding Deep Learning Requires Rethinking Generalization. *ICLR*, 2017. URL https://arxiv.org/pdf/1611.03530.pdf.

Peter L. Bartlett, Nick Harvey, Chris Liaw, and Abbas Mehrabian. Nearly-tight VC-dimension and pseudodimension bounds for piecewise linear neural networks. *ArXiv e-prints*, mar 2017a. URL http://arxiv.org/abs/1703.02930.

Paul W Goldberg and Mark R Jerrum. Bounding the Vapnik-Chervonenkis Dimension of Concept Classes Parameterized by Real Numbers. *Machine Learning*, 18:131–148, 1995. URL https://link.springer.com/content/pdf/10.1007/BF00993408.pdf.

Behnam Neyshabur, Ryota Tomioka, and Nathan Srebro. Norm-Based Capacity Control in Neural Networks. *Proceeding of the 28th Conference on Learning Theory (COLT)*, 40:1–26, 2015. URL http://proceedings.mlr.press/v40/Neyshabur15.pdf.

Peter L. Bartlett, Dylan J. Foster, and Matus J. Telgarsky. Spectrally-normalized margin bounds for neural networks. *NIPS*, pages 6241–6250, 2017b. URL http://papers.nips.cc/paper/7204-spectrally-normalized-margin-bounds-for-neural-networks.

Behnam Neyshabur, Srinadh Bhojanapalli, David Mcallester, and Nathan Srebro. Exploring Generalization in Deep Learning. *NIPS*, 2017. URL https://arxiv.org/pdf/1706.08947.pdf.

Devansh Arpit, Stanisław Jastrzębski, Nicolas Ballas, David Krueger, Emmanuel Bengio, Maxinder S. Kanwal, Tegan Maharaj, Asja Fischer, Aaron Courville, Yoshua Bengio, and Simon Lacoste-Julien. A closer look at memorization in deep networks, 2017. URL https://dl.acm.org/citation.cfm?id=3305406.

Thiago Serra, Christian Tjandraatmadja, and Srikumar Ramalingam. Bounding and Counting Linear Regions of Deep Neural Networks. *arxiv e-prints*, 2018. URL https://arxiv.org/pdf/1711.02114.pdf.

Maithra Raghu, Ben Poole, Jon Kleinberg, Surya Ganguli, and Jascha Sohl Dickstein. On the Expressive Power of Deep Neural Networks. *Proceedings of the 34th International Conference on Machine Learning*, 2017. URL https://arxiv.org/pdf/1606.05336.pdf.

Raman Arora, Amitabh Basu, Poorya Mianjy, and Anirbit Mukherjee. Understanding Deep Neural Networks with Rectified Linear Units. *ICLR*, 2018a. URL https://arxiv.org/pdf/1611.01491.pdf.

Boris Hanin and David Rolnick. Complexity of Linear Regions in Deep Networks. *arxiv e-prints*, 2019. URL https://arxiv.org/pdf/1901.09021.pdf.

Roman Novak, Yasaman Bahri, Daniel A. Abolafia, Jeffrey Pennington, and Jascha Sohl-Dickstein. Sensitivity and Generalization in Neural Networks: an Empirical Study. *ICLR*, feb 2018. URL http://arxiv.org/abs/1802.08760.

Liwen Zhang, Gregory Naitzat, and Lek-Heng Lim. Tropical Geometry of Deep Neural Networks. *ArXiv e-prints*, 2018. URL https://arxiv.org/pdf/1805.07091v1.pdf.

David A. McAllester. Some PAC-Bayesian Theorems. *Machine Learning*, 37(3):355–363, 1999. ISSN 08856125. doi: 10.1023/A:1007618624809. URL http://link.springer.com/10.1023/A:1007618624809.

John Langford and Rich Caruana. (Not) Bounding the True Error. In T G Dietterich, S Becker, and Z Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, pages 809–816. MIT Press, 2002. URL http://papers.nips.cc/paper/1968-not-bounding-the-true-error.pdfhttps://papers.nips.cc/paper/1968-not-bounding-the-true-error.

Gintare Karolina Dziugaite and Daniel M. Roy. Computing Nonvacuous Generalization Bounds for Deep (Stochastic) Neural Networks with Many More Parameters than Training Data. *UAI*, mar 2017. URL http://arxiv.org/abs/1703.11008.

Guillermo Valle-Pérez, Chico Q. Camargo, and Ard A. Louis. Deep learning generalizes because the parameter-function map is biased towards simple functions. *ICLR*, may 2019. URL http://arxiv.org/abs/1805.08522.

Sanjeev Arora, Rong Ge, Behnam Neyshabur, and Yi Zhang. Stronger

generalization bounds for deep nets via a compression approach. *arXiv pre-print*, feb 2018b. URL http://arxiv.org/abs/1802.05296.

Wenda Zhou, Victor Veitch, Morgane Austern, Ryan P Adams, and Peter Orbanz. Compressibility and Generalization in Large-Scale Deep Learning. *Arxiv*, 2018. URL https://arxiv.org/pdf/1804.05862.pdf.