
# Sample Compression, Support Vectors, and Generalization in Deep Learning

| | |
|---|---|
| Journal: | *IEEE Transactions on Neural Networks and Learning Systems* |
| Manuscript ID | TNNLS-2019-I-10668 |
| Manuscript Type: | Special Issue: Structured Multi-output Learning: Modelling, Algorithm, Theory and Applications |
| Date Submitted by the Author: | 02-Jan-2019 |
| Complete List of Authors: | Snyder, Christopher; University of Texas. Austin, Department of Electrical and Computer Engineering<br>Vishwanath, Sriram; University of Texas, Austin, Electrical and Computer Engg |
| Keywords: | Deep Neural Networks, Sample Compression, Generalization, Support Vector Machines |
| | |

# Sample Compression, Support Vectors, and Generalization in Deep Learning

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

***Abstract*— Even though Deep Neural Networks (DNNs) are widely celebrated for their practical performance, they possess many intriguing properties related to depth that are difficult to explain both theoretically and intuitively. Understanding how weights in deep networks coordinate together across layers to form useful learners has proven challenging, in part because of the repeated composition of nonlinearities induced by depth. This paper presents a reparameterization of DNNs as a linear function of a feature map that is *locally independent* of the weights. This feature map transforms depth-dependencies into simple *tensor* products and maps each input to a discrete subset of the feature space. Then, using a max-margin assumption, the paper develops a *sample compression* representation of the neural network in terms of the discrete activation state of neurons induced by $s$ "support vectors". The papers shows that the number of support vectors $s$ relates with learning guarantees for neural networks through sample compression bounds, yielding a sample complexity of $\mathcal{O}(ns/\epsilon)$ for networks with $n$ neurons. Finally, the number of support vectors $s$ is found to have monotonic dependence on width, depth, and label noise for simple networks trained on the MNIST dataset.**

***Index Terms*—Deep Neural Networks, Sample Compression, Generalization**

## I. INTRODUCTION

Neural networks represent an intriguing class of models that have achieved state-of-the-art performance results for many machine learning tasks. Although neural networks have been studied for over half a century [1], the variations which have recently garnered interest are called "deep" neural networks (DNNs). Deep learning is characterized by stacking one layer after another and using the computational power of modern graphical processor units (GPUs) or custom processors/ASICs to train them. It is shown experimentally that such networks with more layers tend to generalize better [2] [3].

Thus, deep learning presents a scenario where the best performing models are also generally poorly understood. Improvements to our theoretical understanding of deep neural networks will aid in structured, principled approaches to the design, analysis, and use of such networks. An important step in understanding a model is to prove generalization bounds that agree with performance in practice. For DNNs, several attempts have been made in this direction based on margin, perturbation, PAC-Bayes, or complexity type approaches (see related work section for further details). In current literature, it is not uncommon to aim for bounds that are depth-independent, i.e. to find bounds that do not obviously get worse with depth. Although such bounds continue to be of significant interest, we ultimately desire bounds that improve with depth, and thus aid in explaining the performance of DNNs in practice.

In this paper, under assumptions presented in Section III-C, we recast leaky-ReLU type networks as an equivalent support vector machine (SVM) problem where the features correspond to paths through the network and the embedding map $\phi$ has local invariance to perturbation of the weights of the DNN. Though this embedding is non-trivial function of these weights, the induced kernel has a simple interpretation as the inner product of in the input space scaled by the number of shared paths in the network. This feature space has a tensor product decomposition into spaces corresponding to each layer that enables us to perform depth-dependent analysis.

Our main contributions can be summarized as:

1) We present a framework for recasting neural networks with two-piecewise-linear nonlinearities (such as ReLU) as an SVM problem where classification with the network is equivalent to linear classification in a particular tensor space. Here, the corresponding embedding of training points is insensitive to local perturbations of weights.
2) Under a max-margin assumption on the network within this new feature space, we define "network support vectors": those training samples that are mapped to support vectors under the learned embedding map. An important consequence of this max-margin assumption is that only *finitely many* neural network classifiers correspond to a set of network support vectors.
3) We show that the number of network support vectors, $s$, can be related theoretically to generalization and experimentally to network architecture. We use a sample compression variant of PAC-Bayes to prove in Theorem 1 a bound on the sample complexity of max-margin networks with $n$ neurons of $\mathcal{O}(ns)$. We show that the quantity, $ns$, experimentally decreases with depth (despite $n$ increasing with depth).

## II. RELATED WORK

There have been multiple, well-thought-out efforts to model, characterize and understand generalization error in DNNs. One well-studied direction is to impose a sufficiently small-norm condition on the neural network weights [4][5]. Since the weight norms induce a bound on the network's Lipschitz constant, one can connect this with insensitivity of the network output to input perturbations, either through a product of weight spectral norms [6] or through the norm of the network Jacobian

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

itself [7]. Instead of invariance to input perturbations, one can also consider the degree of invariance of the network dependence to weight perturbations [8]. A natural way to concretely relate such perturbation schemes to generalization error is through the means of probably approximately correct PAC-Bayes analysis as in [9] [10].

The general principle underlying PAC-Bayes analysis is to characterize (in bits, with respect to some prior) the degree of precision in specifying the final neural network weights in order to realize the observed training performance. Such PAC-Bayes based generalization bounds were applied successfully to the study of neural networks by [11] and more recently by [12], albeit for stochastic networks. Generally speaking, if multiple weights corresponding to a large neighborhood result in similar neural network behavior, then fewer bits are needed to specify these weights. Overall, insensitivity to weight perturbation is one potential manner to formalize the popular high-level idea that "flat minima generalize well" [13], [14].

In order to correctly reproduce the improvement in generalization observed in deep learning with each additional layer, the principle difficulty is that these approaches must make layer-wise considerations (either of each weight matrix or each layer-wise computation) that accumulate and grow the generalization bound as depth increases. Of course, it is possible to find suitable assumptions that control or mitigate this depth-dependent growth as in [4] or [15]. Given this challenge, other "network compression" type approaches that characterize the network function without addressing every individual parameter are gaining interest. For example, [16] analyzes the number of nonzero weights as a form of capacity control, while others have studied approximating a deep network by a "compressed" version with fewer nonzero weights [17] [18].

In this paper, we use a *sample* compression [19] representation approach for understanding neural network depth-dependence. We transform the neural network into a related SVM problem, then recover the network function from (suitably defined) support vectors. Sample compression characterizes PAC-learnable functions as those that can be recovered from (through some function of) a small enough subset of training samples [20]. Sample compression methods can be seen as a natural extension of the PAC-Bayes analysis present above, in which the prior also attributes a probability to each subset of training point indices to be used in recovering the classifier [21], [22].

## III. ON NEURAL NETWORKS AS SUPPORT VECTOR MACHINES

### A. Notation Definitions and Setting

In this paper, we consider the family of nonlinearities $\rho(x) = \beta x \mathbb{1}_{\{x<0\}}(x) + \gamma x \mathbb{1}_{\{x\geq 0\}}(x)$ for $\beta, \gamma \in \mathbb{R}$ for the neural network, which encompasses ReLU, LeakyReLU, and absolute value as examples. We will refer to these nonlinearities and the neural networks that use them as "ReLU-like". For vector arguments, $\rho$ is understood to be applied element-wise. We do not use biases. For integer $m$, we will use $[m]$ to mean the set $\{1, \ldots, m\}$.

Consider a neural network with $d$ (depth) hidden layers, width $\Omega$ neurons in layer $l$, $f$ input features, and $m$ training samples.

We will use $\mathcal{W} = \mathbb{R}^{\Omega} \times (\prod_{i=1}^{d-1} \mathbb{R}^{\Omega\times\Omega}) \times \mathbb{R}^{\Omega\times f}$ to denote the set of all possible weights within the neural network. Here $A^{(l)}_{i_{l+1},i_l}$ refers to the scalar weight from neuron $i_l$ in $l$ to neuron $i_{l+1}$ in layer $l+1$. We use $w$ to refer to *all of the weights collectively*, with $w = (A^{(d)}, \ldots, A^{(1)}, A^{(0)}) \in \mathcal{W}$. Each $w$ corresponds to a neural network mapping each $x \in \mathcal{X} \triangleq \mathbb{R}^f$ to $\mathbb{R}$ as follows:

$$\mathcal{N}(x,w) \triangleq \mathcal{N}_w(x) \triangleq A^{(d)}\rho(A^{(d-1)}\rho(\ldots(A^{(1)}\rho(A^{(0)}x)\ldots)), \tag{1}$$

We distinguish between $\mathcal{N}_w : \mathcal{X} \mapsto \mathbb{R}$, which returns scalar values, and the related classifier returning labels, $\mathcal{N}^{sign}_w \triangleq sign \circ \mathcal{N}_w : \mathcal{X} \mapsto \mathcal{Y}$. Here $\mathcal{Y} \triangleq \{-1, +1\}$ and $sign(\cdot)$ is a function returning the sign of its argument (defaulting to $+1$ for 0 input). For a data distribution $\mathcal{D}$ on $\mathcal{X} \times \mathcal{Y}$, the goal is to use a training set $S^{(m)} = \{(x^j, y^j)\}_{j=1}^m \sim \mathcal{D}^m$ to learn a set of weights $w$ so that $\mathcal{N}^{sign}_w$ has small probability of misclassification on additional samples drawn from $\mathcal{D}$.

We define a *path* (in a neural network) to be an element of $(\prod_{i=1}^{d}[\Omega]) \times [f]$, corresponding to a choice of 1 neuron per hidden layer and 1 input feature. Sometimes it is convenient to refer to these input features as neurons in layer $l = 0$. Thus, one says that the path $i_d, \ldots, i_1, i_0$ traverses neuron $i_l$ in layer $l = 0, 1, \ldots, d$.

Given a set of weights $w$, we define $\Lambda(w)$ to be the path-indexed vector with the product of weights along path $p$ in position $p$. Often we use $\bar{w}$ to shorten $\Lambda(w)$, and we use $\bar{w}_p$ or $\bar{w}_{i_d,\ldots,i_1,i_0}$ when we want to specify the path.

Sometimes a particular vector, $v$, with entries indexed by paths, depends linearly on the choice of neuron traversed in each particular layer, i.e., $v_{i_d,\ldots,i_1,i_0} = s^{(d)}_{i_d} s^{(d-1)}_{i_{d-1}} \cdots s^{(0)}_{i_0}$. In this case we say $v$ is the "tensor product" of $s^{(d)}, \ldots, s^{(0)}$, denoted

$$\left(\bigotimes_{l=0}^{d} s^{(l)}\right)_{i_d,\ldots,i_1,i_0} \triangleq (s^{(d)} \otimes s^{(d-1)} \otimes \cdots \otimes s^{(0)})_{i_d,\ldots,i_1,i_0} \triangleq s^{(d)} s^{(d-1)}_{i_{d-1}} \cdots s^{(0)}_{i_0}.$$

### B. A Reparameterization of the Network

Consider the set of all paths starting from some feature in the input and passing through one neuron per hidden layer of a ReLU neural network. Index these $f\Omega^d$ many paths by the coordinate tuple $(i_d, \ldots, i_1, i_0)$ to denote the path starting at feature $i_0$ in the input and passing through neuron $i_l$ in hidden layer $l$. Given a set of network weights $w$, we can define $\Lambda(w) = \bar{w} = \bar{w}_{i_d,\ldots,i_1,i_0}$, whose $(i_d, \ldots, i_1, i_0)^{th}$ coordinate is the product of weights along path $(i_d, \ldots, i_1, i_0)$. Inspired by [15], (who used a similar factorization without exploring the connections with support vector machines) we note that the output of a neural network can be viewed as a sum of contributions over paths

$$\mathcal{N}(x,w) = \sum_{p=(i_d,\ldots,i_1,i_0)} \sigma^{(d)}(x,w)_{i_d} \cdots \sigma^{(1)}(x,w)_{i_1} x_{i_0} \bar{w}_p$$

where $\sigma^{(l)}(x,w)$ is an indicator vector for which neurons in layer $l$ are active for input $x$ with weights $w$. For convenience, we also define $\bar{\sigma}(x,w) = \bar{\sigma}(x,w)_{i_d,\ldots,i_1} = \sigma^{(d)}(x,w)_{i_d} \cdots \sigma^{(1)}(x,w)_{i_1}$, which is also an indicator but over paths instead of neurons. The above summation over all tuples $(i_d,\ldots,i_1,i_0)$ can be interpreted as an inner product $\langle \phi(x,w), \bar{w} \rangle$ where

$$\phi(x,w)_{i_d,\ldots,i_1,i_0} = \sigma^{(d)}(x,w)_{i_d} \cdots \sigma^{(1)}(x,w)_{i_1} x_{i_0} \qquad (2)$$

is a $w$-parameterized family of embedding maps from the input to a feature space we denote as the "Path Space" $\mathcal{F}$, i.e., the set of all tensors assigning some scalar to each path index-tuple $i_d,\ldots,i_1,i_0$ with $i_l \in [b]$ for $l \in [d]$ and $i_0 \in [f]$. The neural network then is *almost* a kernel classifier in that the model only interacts with the input through inner products with a feature map $\phi(x,w)$, though unlike a SVM, the feature map has some dependence on $w$.

An important insight is that, over small regions of the weight space, our embedding $\phi(x_i,w)$ does not depend on $w$ for any of the finitely many training points. More concretely, suppose that none of the pre-nonlinearity activations of neurons in $\mathcal{N}$ are *identically* zero. Then for each training sample and each neuron pre-activation, we obtain an open ball about this pre-activation (excluding zero). Since the function from the weights to each pre-activation is continuous, the preimage of each ball in the weight space is open. The intersection of these (finitely many) preimages is an open set around the current network weights in which the feature space embedding of training samples (not necessarily test samples) is *independent* of our weights. Interestingly, this implies that over small regions of weights around $w$, say $B_\epsilon(w)$, we may parameterize our training outputs unambiguously by the product of weights over paths, $\bar{w} \in \Lambda(B_\epsilon(w))$, instead of the "usual" parameterization $w$. Note though that globally the relationship is not $1-1$.

Let us suppose, as in gradient based methods, that DNN training terminates at model weights $w$ if and only if some local stopping condition $C_{STOP}$ that depends only on $\mathcal{N} \restriction_{S^{(m)} \times B_\epsilon(w)}$ returns True. Since $\bar{w}$ is locally sufficient for $\mathcal{N} \restriction_{S^{(m)} \times B_\epsilon(w)}$, $\bar{w}$ is in turn locally sufficient for $C_{STOP}$. Therefore we may, without loss of generality, characterize the networks, $\mathcal{N}_w^{sign}$, for which $C_{STOP}$ returns true, by studying how networks outputs vary in neighborhoods of $\bar{w}$ (which has a linear relationship to model outputs) instead of in neighborhoods of $w$.

We further observe that if we use cross entropy loss, as in common with neural networks, the local loss landscape of models parameterized by $\Lambda(B_\epsilon(w)) \mapsto \mathcal{N} \restriction_{S^{(m)} \times B_\epsilon(w)}$ is exactly that of the loss landscape of logistic regression models on $\mathcal{F}$ with the same training data, feature map $\phi(\cdot,w)$, and parameters restricted to $\Lambda(B_\epsilon(w))$.

### C. Assumptions Made

Prior to detailing the assumptions made in this paper, we first highlight a compelling recent work on *unregularized* logistic regression for linearly separable problems in [23]. Here, the authors prove that gradient descent yields a sequence of classifiers whose normalized versions converge to the max margin solution. For example, the authors provide a theoretical basis for the increase in test accuracy and test loss during training even after the training accuracy is 100%. Note that this peculiar behavior is also common to neural networks [24]. Inspired by this connection, we make the following assumptions

**Assumption 1.** *Zero Training Error*
*The weights $w$ obtained from training on $S^{(m)}$ ensure $\mathcal{N}_w^{sign}$ correctly classifies every sample in $S^{(m)}$. Equivalently:*

$$\forall (x,y) \in S^{(m)} \quad y\langle \Lambda(w), x \rangle \geq 0$$

Note that, for zero training error, linear separability[1] of our embedded data, $\{(\phi(x^j,w),y^j)\}_{j=1}^m$, is strictly necessary. Motivated by analogy with maximum margin classifiers in logistic regression, we make the following second assumption on the network weights obtained by training on $S^{(m)}$:

**Assumption 2.** *Max-Margin*
*The training procedure returns weights $w$ such that up to positive scaling, $\Lambda(w)$ is the maximum margin classifier for the $w$-parameterized embedding $\{(\phi(x^j,w),y^j) : j \in [m]\}$. Equivalently, $w$ must satisfy the relation*

$$\Lambda(w) \in \arg\max_{\bar{v} \in \mathcal{F}} \min_{(x,y) \in S^{(m)}} \frac{y\langle \bar{v}, \phi(x,w) \rangle}{\|\bar{v}\|}$$

### D. Rationale For Assumptions Made

Of the two assumptions made in the paper, note that Assumption 1, of zero training error, is not uncommon for neural networks in practice [25]. Therefore, we do not discuss Assumption 1 in greater detail in this subsection, focusing more on the second assumption in this paper.

The value of Assumption 2 is more nuanced. We only seek to show that it is true in related simplified cases, concisely explains certain experimental phenomena, and allows theoretical tractability–not that it is strictly true in any particular setting. Assumption 2 then represents an *idealized* case, similar to that of "point mass" in physics or an "ideal gas" in chemistry or of "free market" in economics. Even though such ideals may be rare in practice, they enable us to derive analytical expressions that form good starting points for research in each specific domain. We have mentioned through analogy with logistic regression that, Assumption 2 is true in one particular setting; the max-margin condition is satisfied (even without a norm penalty) when the embedding map does not depend on weights. Now we present some experimental observations that are somewhat surprising and difficult to motivate without Assumption 2.

Consider training data $S^{(m)} \subset \mathbb{R}^2$ organized by label into the $1^{st}$ and $3^{rd}$ quadrants so that for each training datum $(x^j,y^j)$, we have that $y^j x^j$ is coordinate-wise positive. The weights obtained from training (without biases) with LeakyReLU nonlinearity on the described data are shown graphically in Figure 1 (More details and training data can be found in appendix VI-A, in particular Figures 7 8).

Thoughtful inspection of Figure 1 reveals that every path from the input to output traverses an even number of negative

[1] In fact we are guaranteed a separating hyperplane containing the origin

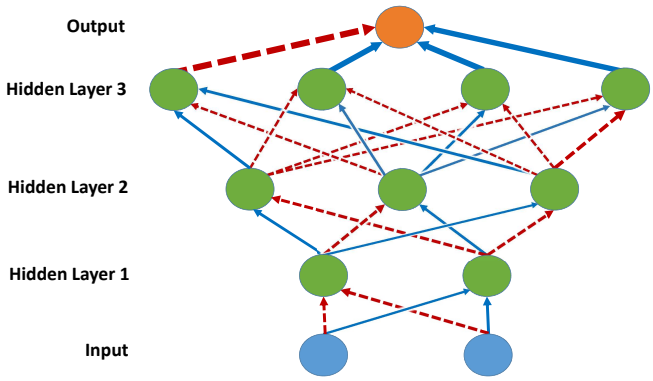


Fig. 1: Learned network weights after training on data with positive samples in the $1^{st}$ quadrant and negative samples in the $3^{rd}$ quadrant. Negative [positive] weights are represented by red dotted [blue solid] arrows [respectively]. Thicker arrows correspond to weights of larger magnitude. The finding is that each path from any input feature to the output contains an even number of red arrows (negative weights). This coordination of weight signs across layers is a striking feature of training that is implied by Assumption 2, but is not readily explained otherwise. (See Figure 7 for a detailed version of this figure annotated with weight values).

weights. We will show this is an immediate consequence of Assumption 2, but not clear to us through any other theoretical lens that we are aware of. Notice also that this insight is not nearly so apparent when weights are conceptually grouped by layer instead of across paths.

To explain the finding, observe that for any weights, $w$, the embedded training data, $y^j\phi(x^j, w)$, is coordinate-wise positive since $\bar{\sigma}(x, w)$ is always coordinate-wise positive. Since the max-margin classifier is in the positive linear combination of the $\{y^j\phi(x^j, w)\}_{j=1}^m$, by Assumption 2, $\bar{w}$ is coordinate-wise positive. Since each coordinate of $\bar{w}$ corresponds to a product of weights over a single path, this is equivalent to traversing an even number of negative weights.

Note that, without idealized assumptions such as Assumption 2, it is very difficult to build a framework that helps us gain an understanding of the problem, or the implications of its solution. In particular, Assumption 2 forms a starting point for a deeper theoretical understanding of neural networks, one that provides useful insights that can be employed towards a more general, overall theory for deep neural networks.

### *E. Network Support Vectors*

In this section we use Assumption 2 to extend the definition of support vectors to neural networks with zero training error. By the Representer Theorem [26], the max-margin condition on $\bar{w}$ in Assumption 2 implies that for some nonnegative scalars $\alpha_1, \ldots, \alpha_m$,

$$\bar{w} = \sum_{k=1}^{m} \alpha_k y^k \phi(x^k, w). \tag{3}$$

Analogous to classical SVMs, for a fixed set of weights $w$ achieving Assumption 2, we define the the subset $S^{(s)} \triangleq \{(x^k, y^k) : \alpha_k \neq 0\}$ of those training data points that correspond to nonzero $\alpha_k$ to be "network support vectors"(NSVs) or simply "support vectors" (SVs) when context is clear. We also use $S^{(m-s)} = S^{(m)} - S^{(s)}$ to denote the $m-s$ data which are not support vectors.

To gain an experimental understanding of these "support vectors", we train neural networks on a 2-class MNIST variant formed by grouping labels $0-4$ and $5-9$. We show that many qualitative properties of SVMs continue to hold true in this case when the embedding map is learned. We first determine network weights obtained from minimizing the neural network loss. Then, we define an embedding map $\phi(\cdot, w)$ using those weights. Finally, we train a SVM using the kernel as defined by $\langle \phi(x^i, w), \phi(x^j, w) \rangle$. The details of this experiment are presented in the Appendix VI-A.

As noted in these experiments, we determine that the behavior of the number of NSVs is qualitatively similar to what we might find in a conventional SVM setting. For example, we typically find $s/m \approx 0.15$. We find that every time we increase the number of training samples, $m$, and retrain the network from scratch, the net effect is that $s$ increases but $s/m$ asymptotically decreases to $0.1$ (Figure 2). This is entirely expected in the simplified setting with a fixed embedding map: additional samples can only decrease the margin, reducing the fraction of volume within the margin of the hyperplane. Thus, additional randomly selected samples are increasingly unlikely to be support vectors.

Given that the SVM model (with fixed embedding) is determined entirely by $S^{(s)}$, the model is said to have "memorized" the sample $(x, y) \in S^{(m)}$ iff $(x, y)$ is a support vector. We find that a similar notion holds for network support vectors. In deep learning, the notion of memorizing a given individual sample is less clear, but we often describe a DNN with wildly divergent test and train accuracies as having "memorized the dataset". For example, DNNs will often achieve zero training error even when there is no relationship between inputs $\mathcal{X}$ and outputs $\mathcal{Y}$.

If we randomize each label of $S^{(m)}$ prior to training so that the training data is sampled from a product of marginal distributions instead, $S^{(m)} \sim \mathcal{D}_{\mathcal{X}} \times \mathcal{D}_{\mathcal{Y}}$, we observe experimentally that $s/m \approx 0.6$ (Figure 9). This can be understood as follows: although the labels are independent of the inputs, there are natural clusters in the input that the model can use to fit these random labels in the training data. Each sample has a label consistent with at least half of the training set, since half of the training data have the correct label. Thus, the DNN is learning a pattern corresponding to the true labeling (or its reverse) and *building in exceptions* for the rest of the data by adding them as support vectors. Note that learning this labeling on MNIST requires $0.1m$ support vectors (from before). The addition of $0.5m$ training samples that violate the first learned labeling results in the observed $0.6m$ total.

In a conventional SVM setting, models with fewer support vectors are thought of as more parsimonious. Furthermore, the fraction of training samples that are support vectors can be concretely linked to generalization bounds through sample compression techniques, as in [19]. An important observation is that the SVM solution can be reconstructed from the subset of support vectors $S^{(s)} \subset S^{(m)}$, so bounding $s = |S^{(s)}|$ controls the number of training samples the model can memorize. Similarly, in Section IV, we construct analogous bounds for deep neural networks.

In experiments, we find that the fraction $s/m$ increases logarithmically as we increase the width $\Omega$ (Figure 3) but *decreases* significantly as we increase the depth $d$ (Figure 4). The decrease in $s$, when studied experimentally, appears to be superlinear so that $ns$ decreases with depth since the number of neurons is linearly proportional to depth. While these experimental relationships are interesting, these are preliminary in nature, and much more in-depth study is required to make concrete claims on the relationships between parameters of the network.

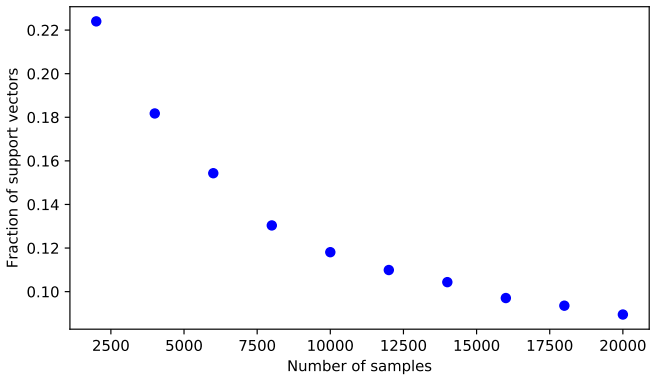


Fig. 2: Support vector fraction of data $s/m$ vs Number of samples m: Increasing the size of the training set decreases asymptotically the fraction $s/m$ of support vectors.

## IV. SAMPLE COMPRESSION BOUNDS

In this section, we present a relationship between the number of network support vectors and a bound of the test error of deep neural networks satisfying assumptions as outlined in Section III-C. Just as a SVM max-margin classifier is determined entirely by its cast of support vectors, *only finitely many* neural networks satisfying the max-margin assumption (Assumption 2) correspond to a given set of at most $s$ network support vectors. This is presented as the following theorem (proof given in Appendix VI-D):

**Theorem 1.** *Let $\mathcal{N}$ refer to a leaky-ReLU neural network with $d$ hidden layers each consisting of width $\Omega$ neurons so that we have $n = d\Omega$ neurons total. Let the weights $w$ be deterministic functions of $S^{(m)}$, which is a set of $m$ i.i.d. data samples from $\mathcal{D}$. Let $s < m$ be a fixed integer which does not depend on $S^{(m)}$. Supposing that:*

1) *Assumption 1 (Zero training error): $\mathcal{N}_w^{sign}(x) = y$ $\forall (x, y) \in S^{(m)}$,*
2) *Assumption 2 (Max-margin): $\Lambda(w)$ is some positively scaled version of the max-margin classifier for $\{(\phi(x, w), y) : (x, y) \in S^{(m)}\}$, and*
3) *(At most $s$ support vectors): $\Lambda(w) = \sum_{k=1}^{m} \alpha_k y^k \phi(x^k, w)$ for some set of coefficients $\alpha_k$, at most $s$ of which are nonzero.*

*then we have, $\forall \delta \in (0, 1]$*

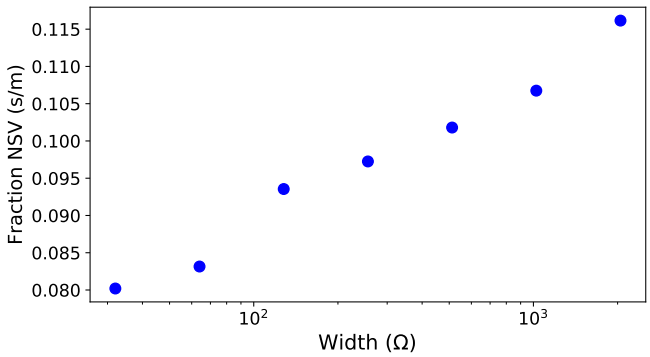


Fig. 3: Fraction Network Support Vectors $(s/m)$ vs width $\Omega$: ReLU networks of varying width $\Omega$ are trained to classify MNIST images. Each width-dependent trained set of network weights, $w$, is used to define an embedding $\phi(\cdot, w)$. The number of support vectors, $s$, corresponding to the maximum margin classification of $(\phi(x^j, w), y^j)_{j=1}^m$ is measured ($m$ is constant). Each point represents an average of three runs. The results indicate that $s$ grows proportionally to $\log(\Omega)$.

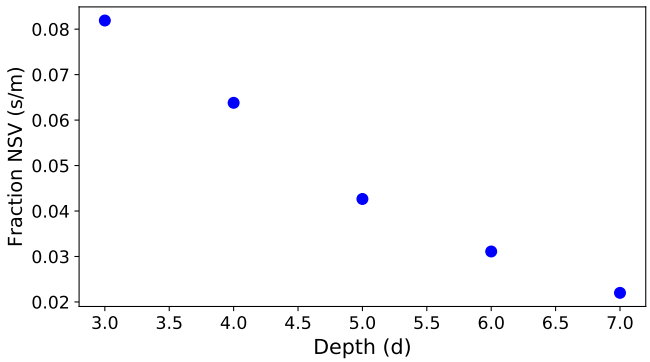


Fig. 4: Fraction Network Support Vectors $(s/m)$ vs depth $d$: The depth, i.e., the number of hidden layers, is varied, resulting in a depth-dependent embedding of the training data, $(\phi(x^j, w), y^j)_{j=1}^m$, where $w$ is the set of weights obtained from training a DNN with $d$ layers to classify data in $S^{(m)}$. The number of support vectors $s$ decreases superlinearly with depth in our setup. (For eg., doubling the number of layers from 3 to 6 more than halves $s$). Each point represents an average of three runs. It is interesting to note that $s$ decreases with depth $d$ in these cases. Given Theorem 1, which bounds the test error by $\mathcal{O}(ns/m)$ assuming Assumption 2, we observe a decrease in the generalization bound with depth, since $n$ increases linearly with $d$, while $s$ decreases superlinearly with depth (for $3 \leq d \leq 7$).

$$\Pr_{S^{(m)}\sim\mathcal{D}^m}\left[R_{\mathcal{D}}(\mathcal{N}_w^{sign}) \leq \mathcal{F}(m,d,\Omega,s,\delta)\right] \geq 1-\delta$$

*where*

$$\mathcal{F}(m,n,s,\delta) = \frac{n+ns+s+s\ln\left(\frac{m}{s}\right)+\ln\left(\frac{1}{\delta}\right)}{m-s} \approx \frac{ns+\ln\left(\frac{1}{\delta}\right)}{m} \quad (4)$$

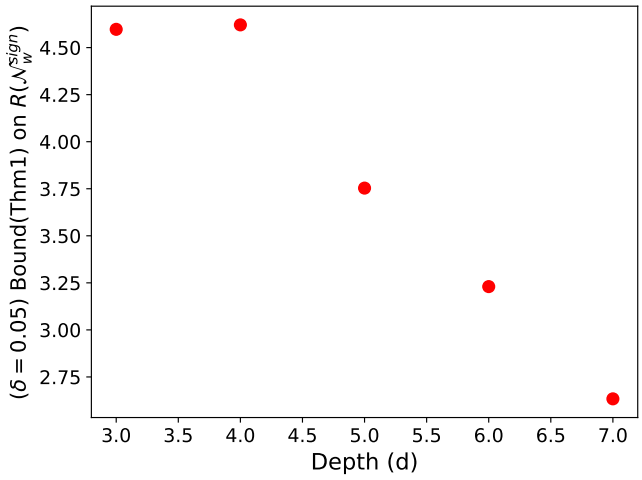


Fig. 5: Numerical Value of the Risk Upper Bound, $\mathcal{F}(m,n,s,\delta)$, in Theorem 1 as Depth $d$ Varied: The outcomes and values of the experiments in the previous section (see Appendix VI-A for details) were used for the generalization bound in Theorem 1 *as if* the assumptions apply.

**Remark:** Note that we do not claim that our bounds are tight, only that they are depth-dependent and therefore are qualitatively similar to phenomena observed in practice. Although a comparison with other bounds would be interesting, in practice any of these bounds can be made arbitrarily large or small by for different architectures and settings. Given that theoretical bounds are still in their early stages, any comparison would to too fungible to prove to be fair and useful to the reader.

Sample compression bounds, as in Theorem 1, are based on the premise that each learned classifier is specified by some small enough subset of the training data. For example, a SVM model can always be identified by its set of $s$ support vectors. On the contrary, if $K > s$ training samples are "memorized" during learning, then the SVM model cannot be specified by $s < K$ samples. Suppose, *a priori*, that the SVM model has at most $s$ support vectors, then there are some $m-s$ training samples on which the learned model has minimal dependence. Thus, the risk on those $m-s$ samples should approximate the true risk. This intuitively explains why specifying a DNN by means of a subset of the training data is related to generalization.

A more general approach allows subsets of training samples to specify a sufficiently small set of $N$ models containing the learned model. The bound produced by this generalization is related to the previous $N=1$ bound by an additive factor of $\ln(N)/m$. Note that, for any fixed $T \in (\mathcal{X}\times\mathcal{Y})^s$, at most $2^{s+ns+n}$ different DNN classifiers, $\{x \mapsto \mathcal{N}_w^{sign}(x) : w \in \mathcal{W}\}$, can simultaneously have weights $w$ that satisfy the maximum margin Assumption 2 for some set of network support vectors contained in $T$.

Hence, we present a two step procedure as follows:

1) Use the max-margin assumption in combination with the given network support vectors, $S^{(s)}$, to identify possible $\bar{w}$. Unlike the SVM setting, the embedding map $\phi(\cdot,w)$ is not fixed, so multiple vectors $\bar{w}$ may be realizable as max-margin classifiers corresponding to different embedding maps. Note that there are only finitely many ways to embed any given set of support vectors.
2) Given $\bar{w}/\|\bar{w}\|$, bounding the number of corresponding $w$ does not result in finite bounds as $|\Lambda^{(-1)}(\bar{w})| = \infty$. However, each feature space classifier $\bar{w}$ can only correspond to $2^n$ *network classifiers* (which return values in $\mathcal{Y}$), despite corresponding to infinitely many sets of *network weights*. For this, we use Theorem 2 as follows: (Proof in Appendix VI-B).

**Theorem 2.** *For $P \subset \mathcal{F}$, define $\mathcal{N}^{sign}(\cdot,\Lambda^{-1}(P)) \triangleq \{\mathcal{N}^{sign}(\cdot,w) : w \in \mathcal{W}, \Lambda(w) \in P\}$. For $\bar{w} \in \mathcal{F}$, define $\mathbb{R}^+\bar{w} \triangleq \{\alpha\bar{w} : \alpha > 0\}$.*

*Then*

$$|\mathcal{N}^{sign}(\cdot,\Lambda^{-1}(\mathbb{R}^+\bar{w}))| \leq 2^n \quad (5)$$

*where $n = d\Omega$ is the number of neurons in the network.*

There are two main ideas underlying Theorem 2. Note that $\Lambda(w)$ only describes products of weights, which creates ambiguity in the scale of individual weight parameters. For example, replacing entries of $w$, $(A^{(l+1)}, A^{(l)})$, with $(\alpha A^{(l+1)}, \alpha^{-1}A^{(l)})$, does not change $\Lambda(w)$ for any choice of $\alpha > 0$. This implies $|\Lambda^{(-1)}(\bar{w})| = \infty$. However, the nonlinearity $\rho$ commutes with positive diagonal matrices, and class predictions are obtained as the sign of the network outputs, $sign \circ \mathcal{N}$. Theorem 2 implies that replacing $w$ with $\mathcal{N}_w^{sign}$ eliminates scale information that causes ambiguity in $w$ given $\Lambda(w)$ alone. In other words, the set $\mathcal{N}^{sign}(\cdot,\Lambda^{-1}(\mathbb{R}^+\bar{w}))$ can potentially be finite as its elements cannot be indexed by a continuously-valued positive-scale parameter.

Given only $\Lambda(w)$, the second type of ambiguity in the weights $w$ is that of sign parity. Overally, $\bar{w} = \Lambda(w)$ forms a system of equations (one per path) involving products of the variables $A_{i,j}^{(l)}$ that cannot be solved without additional information. If the sign of each network weight was known, we could determine the network weights by solving a system of linear equations $\ln(|\bar{w}|) = \ln(|\Lambda(w)|)$ in the variables $\ln(|A_{i,j}^{(l)}|)$. This provides a bound of $2^P \approx 2^{d\Omega^2}$ over the number of possibilities of $sign(w)$, where $P$ is the number of parameters. However, this would translate to a bound governed by the ratio of the number of parameters to samples. Such a bound is slightly unexciting in the context of deep learning, where often $P >> m$. Another idea contained in Theorem 2 is that one can replace the number of parameters with the number of neurons. The knowledge of $sign(\bar{w})$ can be used to reduce the bound to $2^n = 2^{d\Omega}$, where $n$ is the total number of neurons. In fact, it is an interesting intermediate result

that given $\bar{w}$, $w$ is determined entirely by the sign of just $n$ weights in a particular geometric configuration (see Figure 10). (Interestingly, the *sign of the weights*, which featured prominently in earlier experiments (Figure 1), reappears as relevant theoretical quantity). Consequently, we arrive at an improved bound governed by $n/m$.

The bound on the true risk, $R_{\mathcal{D}}(\mathcal{N}_w^{sign})$, depends on bounding the log of the number of classifiers consistent with any given training set. To summarize which steps in our bound over classifiers feature most prominently in our bound on $R_{\mathcal{D}}(\mathcal{N}_w^{sign})$, we tabulate the results from previous discussion in Table I. As each step in our argument has an additive effect on the bound, we speak of the "contribution of each step" to the bound on $R_{\mathcal{D}}(\mathcal{N}_w^{sign})$ separately.

TABLE I: Additive Effect on Sample Complexity

| **STEP** | **#WAYS** | (m-s)$\mathcal{F}(m,n,s,\delta)$ |
|---|---|---|
| $S^{(m)} \rightarrow$NSVs | $2^s\binom{m}{s}$ | $s\ln\left(\frac{m}{s}\right)+O(s)$ |
| NSVs$\rightarrow \Lambda(\mathcal{W})$ | $2^{ns}$ | $ns$ |
| $\Lambda(\mathcal{W}) \rightarrow \mathcal{W} \rightarrow \mathcal{Y}^{\mathcal{X}}$ | $2^n$ | $n$ |

### A. On Improvements and Further Research

A significant reduction in the generalization bound of Theorem 1 to well below $\mathcal{O}(ns/m)$ may be possible in practical settings. Specifically, the largest term in the numerator of the bound, $ns$, arises due to a bound over path activations on NSVs that allows each sample, $x$, to choose its embedding $\bar{\sigma}_w(x)$ independently. Experimentation, however, suggests that this bound is pessimistic under practical circumstances, and training samples are instead embedded in a co-dependent manner.

To understand the dispersion of $\{\bar{\sigma}(x,w) : (x,) \in S^{(m)}\}$, we train a ReLU network with depth $d = 3$ and width $\Omega = 10$ for $50,000$ iterations on MNIST. As an output, we measure the number of unique patterns of path activation in the network, $|\{\bar{\sigma}(x,w) : (x,) \in S^{(m)}\}|$, over either training or test data as the number of training samples $m$ varied (Table 6). For emphasis, we count $\bar{\sigma}(x^i,w) \neq \bar{\sigma}(x^j,w)$ as distinct patterns if even a single neuron, say $i_l$ in layer $l$, behaves differently on $x^i$ and $x^j$, i.e., $\sigma^{(l)}(x,w)_{i_l} \neq \sigma^{(l)}(x,w)_{i_l}$.

Based on previous experiments (see Figure 2), a reasonable guess for the number of support vectors is $s = |S^{(s)}| \approx 0.1m$. If, in practice, for each $j \in [m]$, the embedding of the $j^{\text{th}}$ NSV, $\bar{\sigma}(x^j,w)$, was unconstrained by that of the others, $\{\bar{\sigma}(x,w) : x \in S^{(s)} - x^j\}$, then with high likelihood we would expect to see around $0.1m$ unique path activations counted among support vectors. Although we do not measure this directly, we measure a realtively pessimistic upper bound instead by counting the number of unique path activations over *the entire training set*. We observe that $|\{\bar{\sigma}(x,w) : x \in S^{(s)}\}| \leq |\{\bar{\sigma}(x,w) : x \in S^{(m)}\}| \approx 0.01m$ (Table 6). The number of unique test embeddings of the $10k$ test samples are also relatively few (second row). This suggests that the embeddings, $x \mapsto \phi(x,w) = \bar{\sigma}(x,w) \otimes x$, over training and test data are actually tightly coordinated, which may help further bound the number of possible embeddings of a given set of support vectors.

Fig. 6: Unique Sets of Active Paths Over Inputs

| $m = \lvert S^{(m)}\rvert$ | 100 | 500 | 5000 | 20000 | 50000 |
|---|---|---|---|---|---|
| $\lvert\{\bar{\sigma}(x_{train})\}\rvert$ | 49 | 75 | 210 | 282 | 711 |
| $\lvert\{\bar{\sigma}(x_{test})\}\rvert$ | 75 | 153 | 240 | 265 | 468 |

Future research: We recognize considerable further experimentation is needed, particularly one would like to know "under what circumstances does Assumption 2 hold?". We point out that to even suspect that this is an interesting question to ask requires the experimental and theoretical contributions of this paper–sometimes finding the right question is difficult in and of itself. These contributions are themselves starting points: The existence of a relationship between the number of support vectors and the architecture parameters is intriguing but warrants further exploration. And, the theoretical generalization bounds we present that depend on the number of support vectors are notable for being the only sample-compression based bounds for neural networks, but by no means do they represent the most sharpened bounds possible. Our future goal is to develop improved bounds by continuing this line of thought in the future.

## V. CONCLUSION

In this paper, we motivate and develop the study of Leaky-ReLU type deep neural networks as SVM models with embedding maps locally independent of the weights. Towards this end, we make an idealized assumption, that the neural network results in a "max-margin" classifier. We provide an example of an experimental observation involving the configuration of the signs of the weights that is difficult to reconcile without the lens of this max-margin assumption.

Exploring the implications of this assumption, we demonstrate the experimental behavior and theoretical relevance of resulting "network support vectors", and draw parallels between conventional support vectors and NSVs. Subsequently, we develop a generalization bound for deep neural networks that are depth-dependent in Theorem 1. The conceptual shift underlying the concrete ideas in the paper is to *parameterize* the neural network not by the weights, but as the solution to one of a small number of optimization problems.

## REFERENCES

[1] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The Bulletin of Mathematical Biophysics*, vol. 5, no. 4, pp. 115–133, dec 1943. [Online]. Available: http://link.springer.com/10.1007/BF02478259

[2] R. Novak, Y. Bahri, D. A. Abolafia, J. Pennington, and J. Sohl-Dickstein, "Sensitivtiy and Generalization in Neural Networks: An Empirical Study," *ArXiv e-prints*, 2018. [Online]. Available: https://arxiv.org/pdf/1802.08760.pdf

[3] B. Neyshabur, Z. Li, S. Bhojanapalli, Y. Lecun, and N. Srebro, "Towards Understanding the Role of Over-Parametrization in Generalization of Neural Networks," *CoRR*, 2018. [Online]. Available: https://arxiv.org/pdf/1805.12076.pdf

[4] N. Golowich, A. Rakhlin, and O. Shamir, "Size-Independent Sample Complexity of Neural Networks," *Proceedings of the 31st Conference On Learning Theory*, dec 2018. [Online]. Available: http://arxiv.org/abs/1712.06541

[5] B. Neyshabur, R. Tomioka, and N. Srebro, "Norm-Based Capacity Control in Neural Networks," *Proceeding of the 28th Conference on Learning Theory (COLT)*, vol. 40, pp. 1–26, 2015. [Online]. Available: http://proceedings.mlr.press/v40/Neyshabur15.pdf

[6] P. L. Bartlett, D. J. Foster, and M. J. Telgarsky, "Spectrally-normalized margin bounds for neural networks," *NIPS*, pp. 6241–6250, 2017. [Online]. Available: http://papers.nips.cc/paper/7204-spectrally-normalized-margin-bounds-for-neural-networks

[7] J. Sokolic, R. Giryes, G. Sapiro, and M. R. D. Rodrigues, "Robust Large Margin Deep Neural Networks," *IEEE Transactions on Signal Processing*, vol. 65, no. 16, pp. 4265–4280, aug 2017. [Online]. Available: http://ieeexplore.ieee.org/document/7934087/

[8] B. Neyshabur, S. Bhojanapalli, D. McAllester, and N. Srebro, "A PAC-Bayesian Approach to Spectrally-Normalized Margin Bounds for Neural Networks," *arXiv*, jul 2017. [Online]. Available: http://arxiv.org/abs/1707.09564

[9] D. A. McAllester, "Some PAC-Bayesian Theorems," *Machine Learning*, vol. 37, no. 3, pp. 355–363, 1999. [Online]. Available: http://link.springer.com/10.1023/A:1007618624809

[10] D. McAllester, "A PAC-Bayesian Tutorial with A Dropout Bound," *arXiv e-prints*, jul 2013. [Online]. Available: http://arxiv.org/abs/1307.2118

[11] J. Langford and R. Caruana, "(Not) Bounding the True Error," in *Advances in Neural Information Processing Systems 14*, T. G. Dietterich, S. Becker, and Z. Ghahramani, Eds. MIT Press, 2002, pp. 809–816. [Online]. Available: http://papers.nips.cc/paper/1968-not-bounding-the-true-error.pdfhttps://papers.nips.cc/paper/1968-not-bounding-the-true-error

[12] G. K. Dziugaite and D. M. Roy, "Computing Nonvacuous Generalization Bounds for Deep (Stochastic) Neural Networks with Many More Parameters than Training Data," *UAI*, mar 2017. [Online]. Available: http://arxiv.org/abs/1703.11008

[13] B. Neyshabur, S. Bhojanapalli, D. Mcallester, and N. Srebro, "Exploring Generalization in Deep Learning," *NIPS*, 2017. [Online]. Available: http://papers.nips.cc/paper/7176-exploring-generalization-in-deep-learning.pdf

[14] S. Hochreiter and J. Schmidhuber, "Flat Minima," *Neural Computation*, vol. 9, no. 1, pp. 1–42, jan 1997. [Online]. Available: http://www.mitpressjournals.org/doi/10.1162/neco.1997.9.1.1

[15] K. Kawaguchi, L. P. Kaelbling, and Y. Bengio, "Generalization in Deep Learning," *arXiv preprint*, 2017. [Online]. Available: https://arxiv.org/pdf/1710.05468.pdf

[16] B. Neyshabur, R. Tomioka, and N. Srebro, "In Search of the Real Inductive Bias: On the Role of Implicit Regularization in Deep Learning," *CoRR*, dec 2014. [Online]. Available: http://arxiv.org/abs/1412.6614

[17] S. Arora, R. Ge, B. Neyshabur, and Y. Zhang, "Stronger generalization bounds for deep nets via a compression approach," *arXiv pre-print*, feb 2018. [Online]. Available: http://arxiv.org/abs/1802.05296

[18] W. Zhou, V. Veitch, M. Austern, R. P. Adams, and P. Orbanz, "Compressibility and Generalization in Large-Scale Deep Learning," *Arxiv*, 2018. [Online]. Available: https://arxiv.org/pdf/1804.05862.pdf

[19] N. Littlestone and M. K. Warmuth, "Relating Data Compression and Learnability," Univeristy of California Santa Cruz, Tech. Rep., 1986. [Online]. Available: https://users.soe.ucsc.edu/{~}manfred/pubs/T1.pdf

[20] S. Floyd and M. Warmuth, "Sample compression, learnability, and the Vapnik-Chervonenkis dimension," *Machine Learning*, vol. 21, no. 3, pp. 269–304, dec 1995. [Online]. Available: http://link.springer.com/10.1007/BF00993593

[21] F. Laviolette and M. Marchand, "PAC-Bayes Risk Bounds for Stochastic Averages and Majority Votes of Sample-Compressed Classifiers," *Journal of Machine Learning Research*, vol. 8, no. Jul, pp. 1461–1487, 2007. [Online]. Available: http://www.jmlr.org/papers/v8/laviolette07a.html

[22] P. Germain, A. Lacoste, F. Laviolette, M. Marchand, and S. Shanian, "A PAC-Bayes Sample Compression Approach to Kernel Methods," *Proceedings of the 28th International Conference on Machine Learning*, 2011. [Online]. Available: http://www.icml-2011.org/papers/218{_}icmlpaper.pdf

[23] D. Soudry, E. Hoffer, M. S. Nacson, S. Gunasekar, and N. Srebro, "The Implicit Bias of Gradient Descent on Separable Data (revised)," *ArXiv e-prints*, oct 2018. [Online]. Available: http://arxiv.org/abs/1710.10345

[24] R. Shwartz-Ziv and N. Tishby, "Opening the Black Box of Deep Neural Networks via Information," *CoRR*, mar 2017. [Online]. Available: http://arxiv.org/abs/1703.00810

[25] D. Soudry and Y. Carmon, "No bad local minima: Data independent training error guarantees for multilayer neural networks," *ArXiv e-prints*, 2016. [Online]. Available: https://arxiv.org/pdf/1605.08361.pdf

[26] B. Schölkopf, R. Herbrich, and A. J. Smola, "A Generalized Representer Theorem," in *COLT*. Springer, Berlin, Heidelberg, 2001, pp. 416–426. [Online]. Available: http://link.springer.com/10.1007/3-540-44581-1{_}27

## VI. Appendix

### A. Experiment Details

Concerning the experiment described in Section III-D, the data and learned decision boundary are shown in Figure 8. A version of the weights shown graphically in Figure 1 are reproduced below with annotated values (Figure 7). We use a truncated normal weight initialization centered around $0$ with $0.025$ standard deviation. We train with gradient descent for $15000$ iterations with a learning rate of $0.005$. Our nonlinearity, LeakyReLU, has slopes $\beta = 0.1$ and $\gamma = 1.0$.

The primary finding from the experiment, $\bar{w} > 0$, happens reliably as long as the weight initialization and learning rate are suitably small. Just as we are not claiming Assumption 2 always holds, we are also not claiming that $\bar{w} > 0$ always holds exactly under all related circumstances. For example, if the weight initialization is too large, it is possible to have some few very small weights with signs that do not agree with $\bar{w} > 0$, though the entries of $\bar{w}$ with largest magnitude will all have the same positive sign. Optically, it seems like the gradient can become small too quickly to overcome a large initialization of a given weight with the "wrong" sign. Though, a complete analysis of this phenomenon is not part of the scope of this work.

For Figures 2, 3, and 4, the setup is slightly different. We train a fully connected neural network with nonlinearity $\rho(x) = ReLU(x)$ ($ReLU(x) = \max\{0, x\}$) using SGD with momentum parameter $0.05$, learning rate $0.01$, and batch size $100$. We train on "flattened" MNIST images ($f = 28 \times 28$) with labels grouped into the binary classes $\{0, 1, 2, 3, 4\}$ and $\{5, 6, 7, 8, 9\}$. Unless explicitly varied in the figure, we use a fixed, random subset of $m = 20000$ training samples and an architecture consisting of $d = 3$ hidden layers of uniform width, $\Omega = 16$. All experiments displayed actually achieved $0$ training error. The reason we use only $2/5^{ths}$ of the training data is because achieving *exactly* $0$ training error with every architecture considered is necessary to compare the number of support vectors and difficult to do with the entire training set.

Once we train the network to learn $w$, we experimentally determine the set of network support vectors by running a SVM classifier on the embedded data defined by the *fixed* feature map $x \mapsto \phi(x, w)$. To match the constraints.svm.SVC function in the scikit-learn library, we use hinge loss and regularization constant $C = 1e^{-5}$. We argue though that when the training error is identically $0$ and the data is linearly separable, the SVC model with hinge loss will return the maximum margin classifier independently of the value of $C$. This is because for any $C$, the weights are eventually near the optimum where none of the constraints are active. This agrees with what we see experimentally when we varied $C$ (not shown).

The data points in Figures 3 and 4 representing the number of support vectors vs width and depth are all averages of 3

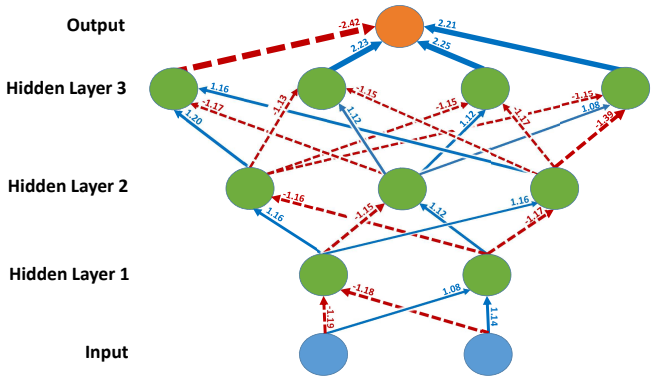


Fig. 7: Network weights after training on quadrant separated data (data shown in Figure 8). Negative [positive] weights are represented by red dotted [blue solid] arrows [respectively]. Thicker arrows correspond to weights of larger magnitude. The finding is that each path from any input feature to the output contains an even number of red arrows (negative weights). This coordination of weight signs across layers is a striking feature of training that is implied by Assumption 2, but is not readily explained otherwise. (This is a more detailed version of Figure 1)

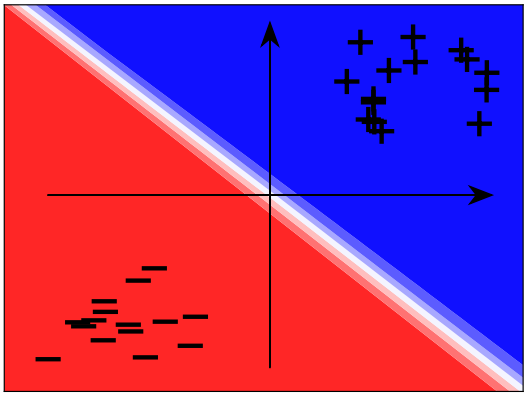

Fig. 8: Learned decision boundary and training data corresponding to the learned weights in figures 17. "Black plus [minus] signs correspond to locations of positively [negatively] labeled training data. Blue [red] regions correspond to positive [negative] evaluations by the network.

trials. One tricky experimental detail is that neural network models have to be trained for a very long time, sometimes upwards of 100 epochs, in order to get *exactly* 0 training error needed to guarantee linear separability. This is especially true for the larger width and larger depth runs.

When we randomize the labels, as in Figure 9, we are determining *every* sample label by a fair coin flip once before training starts, then fixing that label during training.

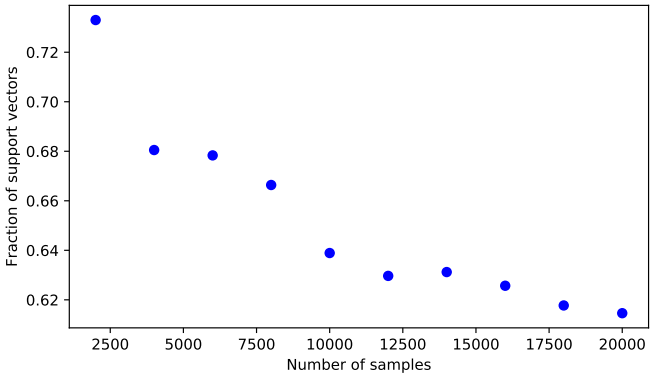


Fig. 9: Fraction Network Support Vectors $(s/m)$ vs $m$ under Randomized Labels: Once before training, the label of each training datum is replaced by a sample drawn uniformly from $\mathcal{Y}$. Compared to the setting with true labels (Figure 2), the data appear shifted up vertically by $0.5$.

### B. *Theorem 2: The Skeleton and NN Recovery*

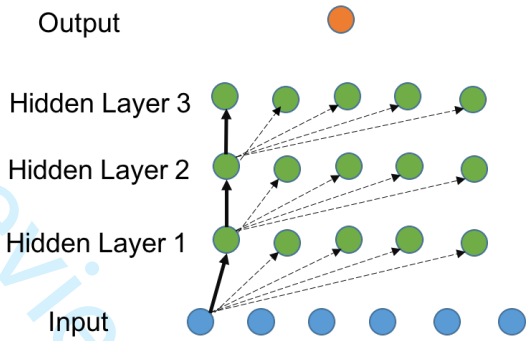


Fig. 10: An illustration of one possible collection of edges corresponding to a skeleton (the key ingredient in the proof of Theorem 2). A "skeleton" is a collection of $n$ edges, $Skel$, with corresponding network weights, $SkelW$, containing for each neuron one path from some input feature to that neuron. For each $\bar{w}$, $SkelW$ is in bijection with $\Lambda^{-1}(\bar{w})$. We may imagine the solid black lines to be the "spine" and the dotted lines to be the "ribs", though there are valid configurations that are less anatomic.

**Theorem 2.** *For $P \subset \mathcal{F}$, define $\mathcal{N}^{sign}(\cdot, \Lambda^{-1}(P)) \triangleq \{\mathcal{N}^{sign}(\cdot, w) : w \in \mathcal{W}, \Lambda(w) \in P\}$. For $\bar{w} \in \mathcal{F}$, define $\mathbb{R}^+\bar{w} \triangleq \{\alpha\bar{w} : \alpha > 0\}$.*

*Then*

$$|\mathcal{N}^{sign}(\cdot, \Lambda^{-1}(\mathbb{R}^+\bar{w}))| \leq 2^n \tag{5}$$

*where $n = d\Omega$ is the number of neurons in the network.*

*Proof.* Suppose we are given a positive multiple of $\bar{w}$. We may assume without loss of generality that every neuron $\eta$ belongs to at least some path, $p(\eta)$, with $\bar{w}_{p(\eta)} \neq 0$. If some neuron is not a member of any such path, then it makes no contribution to

the function $x \mapsto \mathcal{N}(x,w) = \sum_{p \in Paths} \bar{w}_p \phi(x,w)_p$. Thus we may drop all such neurons without affecting which functions $\mathcal{N}_w^{sign}$ are feasible given $\bar{w}/\|\bar{w}\|$. If by removing neurons in this manner we run out of neurons in a single hidden layer, then the bound is trivially true since $\mathcal{N}_w$ must be the zero function.

Thus, for all $1 \leq l \leq d$, every neuron $i_1$ in layer 1 has at least some corresponding index $s_0(i_1)$ in layer 0 such that $A^{(0)}_{i_1,s_0(i_1)} \neq 0$. Because LeakyReLU commutes with positive diagonal matrices, we can rescale column $i_1$ of $A^{(1)}$ by $|A^{(0)}_{i_1,s_0(i_1)}|$ and row $s_0(i_1)$ of $A^{(0)}$ by $|A^{(0)}_{i_1,s_0(i_1)}|^{-1}$.

We continue renormalizing this way until every row of each of the weight matrices $A^{(0)}$ through $A^{(d-1)}$ has at least one weight in $\{-1,1\}$. For each neuron not corresponding to the input or output, fix a particular choice of indices in the previous layer, $Skel \triangleq \{(l,i_l,s_{l-1})\}_{l,i_l}$, so that the corresponding weights $SkelW \triangleq \{A^{(l-1)}_{i_l,s_{l-1}}\}_{l,i_l}$ are all in $\{-1,1\}$. Call these indices, $Skel$, a "skeleton" of the network, and the corresponding weights $SkelW \subset \{-1,1\}^n$ "skeleton weights". A path $p = (i_d,\ldots,i_0)$ will be said to be "in the skeleton" if $\forall l\ A^{(l)}_{i_{l+1},i_l}$ is a skeleton weight. We have shown that as long as every neuron is along some path $p$ with $\bar{w}_p \neq 0$, then the network has a skeleton. (Figure 10 illustrates one such configuration of weights, but is not explicitly used in this proof).

We will show that given $\alpha\bar{w}$ and a skeleton $Skel$, every choice of skeleton weights determines a different set of weights $w = (\alpha A^{(d)}, A^{(d-1)}, \ldots, A^{(0)})$ with $\Lambda(w) = \alpha\bar{w}$. Thus we will show that the set of weights compatible with $\bar{w}$ are in bijection with the set of $2^n$ possible skeleton weights, up to rescaling of $A^{(d)}$. Since scaling $A^{(d)}$ by $\alpha > 0$ doesn't change the sign of the network output, we will have at most $2^n$ distinct possible classification functions compatible with some scaling of $\bar{w}$.

Fix a choice of skeleton $Skel$ and skeleton weights $SkelW$. Then for every layer $l$, for every neuron $i_l$ in layer $l$, there is a path $(j_{l-1}, j_{l-2}, \ldots, j_0)$ from the input to that neuron which stays in the skeleton and for which the product of weights is $\pm 1 \neq 0$. We introduce the notation $\bar{b}$ by using $\bar{b}_{j_{l-1},j_{l-2},\ldots,j_0}$ to mean "the product of weights along a particular path within $Skel$ from the input to a particular neuron":

$$\bar{b}_{i_l,j_{l-1},j_{l-2},\ldots,j_0} \triangleq A^{(l-1)}_{i_l,j_{l-1}} A^{(l-2)}_{j_{l-1},j_{l-2}} \cdots A^{(0)}_{j_1,j_0}. \tag{6}$$

First we show that all of the projection weights, $A^{(d)}$, are determined up to scale by $\alpha$. For neuron $i_d$ in layer $d$, get a path $j_{d-1}, j_{d-2}, \ldots, j_0$ from the input to neuron $i_d$ within the skeleton so that $\bar{b}_{j_{d-1},j_{d-2},\ldots,j_0} \neq 0$. Then simply solve

$$\frac{\alpha\bar{w}_{i_d,j_{d-1},j_{d-2},\ldots,j_0}}{\bar{b}_{i_d,j_{d-1},j_{d-2},\ldots,j_0}} = \frac{\alpha A^{(d)}_{i_d} A^{(d-1)}_{i_d,j_{d-1}} \cdots A^{(0)}_{j_1,j_0}}{A^{(d-1)}_{i_d,j_{d-1}} \cdots A^{(0)}_{j_1,j_0}} = \alpha A^{(d)}_{i_d}. \tag{7}$$

We next show how to find any weight. let $l < d, i_{l+1}, i_l$ be arbitrary. From neuron $i_{l+1}$ in layer $l+1$ and neuron $i_l$ in layer $l$ get paths $(i_{l+1}, j_l, j_{l-1}, \ldots, j_0)$ and $(i_l, k_{l-1}, \ldots, k_0)$ within $Skel$ with $\bar{b}_{i_{l+1},j_l,j_{l-1},\ldots,j_0}$ and $\bar{b}_{i_l,k_{l-1},\ldots,k_0}$ nonzero. Furthermore, we are guaranteed some indices $e_d, e_{d-1}, \ldots, e_{l+2}$ such that $\bar{w}_{e_d,e_{d-1},\ldots,e_{l+2},i_{l+1},j_l,j_{l-1},\ldots,j_0} \neq 0$ since every neuron is connected to the output through at least some path with nonzero weights. Then we simply solve

$$\frac{\alpha\bar{w}_{e_d,e_{d-1},\ldots,i_{l+1},i_l,k_{l-1},\ldots,k_0}}{\bar{b}_{i_l,k_{l-1},\ldots,k_0}} \frac{\bar{b}_{i_{l+1},j_l,j_{l-1},\ldots,j_0}}{\alpha\bar{w}_{e_d,e_{d-1},\ldots,i_{l+1},j_l,j_{l-1},\ldots,j_0}} \tag{8}$$

$$= \frac{\alpha A^{(d)}_{e_d} A^{(d-1)}_{e_d,e_{d-1}} \cdots A^{(l+1)}_{e_{l+2},i_{l+1}} A^{(l)}_{i_{l+1},i_l}}{\alpha A^{(d)}_{e_d} A^{(d-1)}_{e_d,e_{d-1}} \cdots A^{(l+1)}_{e_{l+2},i_{l+1}}} \tag{9}$$

$$= A^{(l)}_{i_{l+1},i_l}. \tag{10}$$

□

### C. PAC-Bayes Background

In this section, we review the sample compression version of PAC-Bayes bounds, which we will invoke to prove Theorem 1. We are largely following [21].

In the PAC-Bayes framework (without sample compression), one typically works with a distribution over classifiers that is updated after seeing the training set $S^{(m)}$. A prior $P$ over $\mathcal{H} \subset \mathcal{Y}^{\mathcal{X}}$ is declared before training, without reference to the specific samples in $S^{(m)}$. Then consider another distribution over classifiers, $Q$, called a "posterior" to reflect that it is allowed to depend on $S^{(m)}$. Each posterior $Q$ defines a Gibbs classifier $G_Q$ that makes predictions stochastically by sampling classifiers according to $Q$. Similarly, we define the true risk $R(G_Q)$ and empirical risk $R_S(G_Q)$ of the Gibbs classifier $G_Q$ as

$$R_{\mathcal{D}}(G_Q) = \mathop{\mathbb{E}}_{h\sim Q}[R_{\mathcal{D}}(h)] \quad R_{S^{(m)}}(G_Q) = \mathop{\mathbb{E}}_{h\sim Q}[R_{S^{(m)}}(h)]$$

PAC-Bayes gives a very elegant characterization of the relationship between the true and empirical risks of Gibbs classifiers. Let $KL(Q||P)$ be the Kullback-Leibler divergence between distributions $Q$ and $P$. For scalars $q, p$, define $kl(q||p)$ to be the Kullback-Leibler divergence between Bernoulli $q$ and $p$ distributions. We have the following classical uniform bound over posteriors $Q$ $\forall \delta \in (0,1]$ (Theorem 1 in [21])

$$\Pr_{S^{(m)}\sim\mathcal{D}^m}[\forall Q \quad \Phi(Q,P,m,\delta)] \geq 1-\delta$$

where $\Phi(Q,P,m,\delta)$ is the event

$$kl(R_{S^{(m)}}(G_Q)||R_{\mathcal{D}}(G_Q)) \leq \frac{KL(Q||P) + \ln\frac{m+1}{\delta}}{m} \tag{11}$$

To relate this to classical bounds for finite hypothesis sets, notice that if $P = Unif(\mathcal{H})$ and $Q$ is a delta distribution on $\xi_0 \in \mathcal{H}$, then the PAC-Bayes bound is governed by the ratio $KL(Q||P) = \ln|\mathcal{H}|$ to $m$. When $\mathcal{H}$ is not finite, one can still get bounds for stochastic neural network classifiers as in [12], or one can convert these bounds into bounds for deterministic classifiers by considering the risk of the classifier which outputs the majority vote over $Q$ (see [21] section 3 for example). We take neither of these approaches, but just mention them for the reader's interest.

Thus far we have discussed "data-independent" priors. We now turn to [21] to discuss priors $P_{S^{(m)}}$ over hypotheses that

depend on the training set through a "reconstruction function", $\mathcal{R}$, mapping subsets of the training data and some "side-information" to a hypothesis.

The idea is to describe classifiers in terms of a subset of training samples, called a "compression sequence", and an element from some auxiliary set, called a "message". For the moment, consider any arbitrary sequence[2] , $T \subset (\mathcal{X} \times \mathcal{Y})^m$. Given $T$, define a set of "allowable messages" $\mathcal{M}(T)$ so that we have a "reconstruction function", $R : T \times \mathcal{M}(T) \mapsto \mathcal{Y}^{\mathcal{X}}$, which sends arbitrary sets of samples $T$, and optionally some side information in $\mathcal{M}(T)$, to a classifier mapping $\mathcal{X}$ to $\mathcal{Y}$.

Let $I$ be the set of subsets of $[m]$. Considering now our training set $S^{(m)}$, for $\mathbf{i} \in I$ define $S^{(m)}_{\mathbf{i}}$ to be the subset of training points at indices $\mathbf{i}$. We now introduce a single (data-dependent) set for the support of our prior and posterior. Define

$$\mathcal{M}_{S^{(m)}} \triangleq \bigcup_{\mathbf{i} \in I} \mathcal{M}(S^{(m)}_{\mathbf{i}}). \tag{12}$$

In the sample compression setting, we sample hypotheses in $\mathcal{Y}^{\mathcal{X}}$ by sampling $(\mathbf{i}, z)$ from $I \times \mathcal{M}_{S^{(m)}}$ according to either our ($S^{(m)}$-dependent) prior $P_{S^{(m)}}(\mathbf{i}, z)$ or our posterior $Q(\mathbf{i}, z)$ and passing $(\mathbf{i}, z)$ to our reconstruction function $R$ to obtain the hypothesis $R(\mathbf{i}, z) : \mathcal{X} \mapsto \mathcal{Y}$.

For the results to follow, we require our prior and posterior to factorize accordingly:

$$P_{S^{(m)}}(\mathbf{i}, z) = P_I(\mathbf{i}) P_{\mathcal{M}(S^{(m)}_{\mathbf{i}})}(z)$$
$$Q(\mathbf{i}, z) = Q_I(\mathbf{i}) Q_{\mathcal{M}(S^{(m)}_{\mathbf{i}})}(z)$$

That is, though the prior does depend on the training set, the marginal prior $P_I$ over subsets $\mathbf{i} \in I$ does *not*. Also, conditioned on $\mathbf{i} \in I$, the prior on messages $z \in \mathcal{M}(S^{(m)}_{idx})$ only depends on those training samples, $S^{(m)}_{\mathbf{i}} \subset S^{(m)}$, indexed by $\mathbf{i}$ and *not* the whole training set. The same factorization is likewise required of $Q$. In fact, we will assume throughout that any distribution on $I \times \mathcal{M}_{S^{(m)}}$ has this factorization.

Given training set $S^{(m)}$ and posterior $Q = Q_I Q_{\mathcal{M}(S^{(m)})}$ (possibly depending on $S^{(m)}$) the Gibbs classifier $G_Q$ classifies new $x$ by sampling $\mathbf{i} \sim Q_I(\mathbf{i})$, $z \sim Q_{\mathcal{M}(S^{(m)}_{\mathbf{i}})}(z)$, setting $\xi = R(\mathbf{i}, z)$, and returning the label $\xi(x)$.

In analogy with the data independent setting, the goal is again to claim that the empirical Gibbs risk $R_{S^{(m)}}(G_Q)$ is close to the true Gibbs risk $R_{\mathcal{D}}(G_Q)$ when $KL(Q||P)$ is small compared to the number of samples $m$. This is the content of Theorem 3 in [21], which, though more general than we require, we cite verbatim for reference. For example, the theorem uses notation $\bar{Q}$ and $d_{\bar{Q}}$, which will simplify to $\bar{Q} = Q$ and $d_{\bar{Q}} = s$ in our more specialized setting where $P, Q$ have nonzero weight only for $|\mathbf{i}| = s$. A specialized version to follow:

**Theorem 3.** *(Theorem 3 in [21])*
*For any $\delta \in (0, 1]$, for any reconstruction function mapping compression sequences and messages to classifiers, for any $S^{(m)} \in (\mathcal{X} \times \mathcal{Y})^m$ and for any prior $P_{S^{(m)}}$ on $I \times \mathcal{M}_{S^{(m)}}$, we have*

$$\Pr_{S^{(m)} \sim \mathcal{D}^m} [\forall Q \quad \Phi(Q, P, m, \delta)] \geq 1 - \delta$$

*where $\Phi(Q, P, m, \delta)$ is the event*

$$kl(R_{S^{(m)}}(G_Q)||R_{\mathcal{D}}(G_Q)) \leq \frac{KL(\bar{Q}||P) + \ln \frac{m+1}{\delta}}{m - d_{\bar{Q}}}$$

In the special case we consider where $P, Q$ have nonzero weight only for $|\mathbf{i}| = s$, we have the reduction $\bar{Q} = Q$ and $d_{\bar{Q}} = s$. More specialized still, we consider [21] Theorem 9, which specializes to the case where $G_Q$ achieves zero training error. It is *slightly* tighter than simply plugging in 0 for $R_{S^{(m)}}(G_Q)$ by an additive factor of $\ln(m+1)/m$. Notice in the following that the form of the bound arises because $kl(0||R) = -\ln(1 - R)$:

**Theorem 4.** *(Special case of Theorem 9 in [21])*
*Fix $s \leq m$. Let $I_s \subset I$ be the set of $s$-sized subsets of indices $[m]$. For any $\delta \in (0, 1]$, for any reconstruction function mapping compression sequences and messages to classifiers, for any fixed prior $P_T$ that defines for every arbitrary sequence $T \in \mathcal{X} \times \mathcal{Y}^m$ a distribution on $I_s \times \mathcal{M}_{S^{(m)}}$ , we have*

$$\Pr_{S^{(m)} \sim \mathcal{D}^m} [\forall \{Q : R_{S^{(m)}}(G_Q) = 0\} \quad \Phi(Q, P, m, \delta, s)] \geq 1 - \delta$$

*where $\Phi(Q, P, m, \delta, s)$ is the event*

$$R_{\mathcal{D}}(G_Q) \leq 1 - exp\left[ -\frac{KL(Q||P_{S^{(m)}}) + \ln\left(\frac{1}{\delta}\right)}{m - s} \right] \tag{13}$$

Notice though that

$$0 \leq -\ln(1 - R_{\mathcal{D}}(\mathcal{N}_w^{sign})) - R_{\mathcal{D}}(\mathcal{N}_w^{sign}) \leq \epsilon(R_{\mathcal{D}}(\mathcal{N}_w^{sign})) \tag{14}$$

, where $\epsilon(R_{\mathcal{D}}(\mathcal{N}_w^{sign})) \leq 0.03$ for the reasonable operating range, $R_{\mathcal{D}}(\mathcal{N}_w^{sign}) \leq 0.2$. Therefore, as a matter of taste, in place of Equation 13 in the above theorem we can claim the (very slightly) weaker but notationally more compact bound:

$$R_{\mathcal{D}}(G_Q) \leq \frac{KL(Q||P_{S^{(m)}}) + \ln\left(\frac{1}{\delta}\right)}{m - s} \tag{15}$$

In fact as long as for in the range of $R_{\mathcal{D}}(\mathcal{N}_w^{sign})$, we would Now we are ready to prove our main theorem.

### *D. Theorem 1:A Neural Network Sample Compression Bound*

We restate and prove Theorem 1 from Section IV.

**Theorem 1.** *Let $\mathcal{N}$ refer to a leaky-ReLU neural network with $d$ hidden layers each consisting of width $\Omega$ neurons so that we have $n = d\Omega$ neurons total. Let the weights $w$ be deterministic functions of $S^{(m)}$, which is a set of $m$ i.i.d. data samples from $\mathcal{D}$. Let $s < m$ be a fixed integer which does not depend on $S^{(m)}$. Supposing that:*

1) *Assumption 1 (Zero training error): $\mathcal{N}_w^{sign}(x) = y$ $\forall (x, y) \in S^{(m)}$,*
2) *Assumption 2 (Max-margin): $\Lambda(w)$ is some positively scaled version of the max-margin classifier for $\{(\phi(x, w), y) : (x, y) \in S^{(m)}\}$, and*

[2]The terminology "sequence" is used here to highlight situations which apply to *any* set of inputs, not just probable ones.

*3) (At most $s$ support vectors): $\Lambda(w) = \sum_{k=1}^{m} \alpha_k y^k \phi(x^k, w)$ for some set of coefficients $\alpha_k$, at most $s$ of which are nonzero.*

*then we have,* $\forall \delta \in (0, 1]$

$$\Pr_{S^{(m)} \sim \mathcal{D}^m} \left[ R_{\mathcal{D}}(\mathcal{N}_w^{sign}) \leq \mathcal{F}(m, d, \Omega, s, \delta) \right] \geq 1 - \delta$$

*where*

$$\mathcal{F}(m, n, s, \delta) = \frac{n + ns + s + s \ln\left(\frac{m}{s}\right) + \ln\left(\frac{1}{\delta}\right)}{m - s} \approx \frac{ns + \ln\left(\frac{1}{\delta}\right)}{m} \tag{4}$$

*Proof.* We start by defining without reference to a training set: our reconstruction function, our base space, and a fixed prior $P_T$ for every possible sequence $T \subset (\mathcal{X} \times \mathcal{Y})^m$.

Let $T \in (\mathcal{X} \times \mathcal{Y})^m$ be arbitrary. Let $I^{(s)}$ be the set of subsets of $s$ elements from $[m]$. Let $\mathcal{M}^\sigma(T)$ be the set of tuples of neuron states for inputs $T$ that are achievable with at least some network weights: $\mathcal{M}^\sigma(T) \triangleq \{(\bar{\sigma}(x, v))_{(x,y) \in T} : v \in \mathcal{W}\}$.

For future convenience, define a "max-margin conditional", $C_{MM}(T_{\mathbf{i}}, \Sigma)$, to be "True" iff there exist a nonempty set of weights $\mathcal{W}(T_{\mathbf{i}}, \Sigma) \subset \mathcal{W}$ such that $\forall v \in \mathcal{W}(T_{\mathbf{i}}, \Sigma)$: (1) $\Sigma = (\bar{\sigma}(x, v))_{(x,y) \in T}$ and (2) $\Lambda(v)$ is the max-margin classifier for $\{(\phi(x, v), y)\}_{(x,y) \in T}$. Put $\kappa(T_{\mathbf{i}}, \Sigma) = |\{\mathcal{N}_v^{sign} : v \in \mathcal{W}(T_{\mathbf{i}}, \Sigma)\}|$ to be the number of neural network classifiers obtained from some model parameter in $\mathcal{W}(T_{\mathbf{i}}, \Sigma)$. Note that $\kappa(T_{\mathbf{i}}, \Sigma) \leq 2^n$ by Theorem 2.

For $\Sigma \in \mathcal{M}^\sigma(T)$, if $C_{MM}(T_{\mathbf{i}}, \Sigma)$ is True, put $\mathcal{M}^\pi(T, \Sigma) = [\kappa(T_{\mathbf{i}}, \Sigma)]$ and put $\mathcal{M}^\pi(T, \Sigma) = [1]$ otherwise.

Let our prior $P_T(\mathbf{i}, \Sigma, j)$ have support on $I^{(s)} \times \mathcal{M}_T^\sigma \times \mathcal{M}_T^\pi$, where the component spaces are defined as

$$\mathcal{M}_T^\sigma \triangleq \bigcup_{\mathbf{i} \in I^{(s)}} \mathcal{M}^\sigma(T_{\mathbf{i}}).$$

$$\mathcal{M}_T^\pi \triangleq \bigcup_{\mathbf{i} \in I^{(s)} \Sigma \in \mathcal{M}^\sigma(T_{\mathbf{i}})} \mathcal{M}^\pi(T_{\mathbf{i}}, \Sigma)$$

where the prior $P_T$ has the factorization $P_T(\mathbf{i}, \Sigma, j) = P^I(\mathbf{i}) P^\sigma_{T_{\mathbf{i}}}(\Sigma) P^\pi_{(T_{\mathbf{i}}, \Sigma)}(j)$, where $P^I$ is not allowed to depend on the sequence $T$, and each factor distribution is uniform on the corresponding set of allowable messages:

$$\begin{aligned} P^I &= Uniform(I^{(s)}) \\ P^\sigma &= Uniform(\mathcal{M}^\sigma(T_{\mathbf{i}})) \\ P^\pi &= Uniform(\mathcal{M}^\pi(T_{\mathbf{i}}, \Sigma)) \end{aligned} \tag{16}$$

Our reconstruction function maps each $(\mathbf{i}, \Sigma, j) \in I^{(s)} \times \mathcal{M}_T^\sigma \times \mathcal{M}_T^\pi$ to a classifier as follows: if $C_{MM}(T_{\mathbf{i}}, \Sigma)$ is True, $\mathcal{R}(T_{\mathbf{i}}, \Sigma, j)$ returns the $j^{\text{th}}$ classifier, in $\{\mathcal{N}_w^{sign} : w \in \mathcal{W}(T_{\mathbf{i}}, \Sigma)\}$ (any total ordering on network classifiers $\{\mathcal{N}_v^{sign} : v \in \mathcal{W}\}$, can be used to clarify the meaning of $j^{\text{th}}$). Else if $C_{MM}(T_{\mathbf{i}}, \Sigma)$ is False, $\mathcal{R}(T_{\mathbf{i}}, \Sigma, j)$ returns a "dummy" classifier. To make a concrete choice, return the constant classifying function: $\mathcal{R}(T_{\mathbf{i}}, \Sigma, j) = (x \mapsto +1)$ if $C_{MM}(T_{\mathbf{i}}, \Sigma)$ False.

Only now, let $S^{(m)}$ be a training set sampled from $\mathcal{D}^m$. Consider now only the "posterior" distributions $Q$ on $I^{(s)} \times \mathcal{M}^\sigma_{S^{(m)}} \times \mathcal{M}^\pi_{S^{(m)}}$ that satisfy $\operatorname{supp} Q \subset \operatorname{supp} P_{S^{(m)}}$ and factorize according to $Q(\mathbf{i}, \Sigma, j) = Q^I(\mathbf{i}) Q^\sigma_{S^{(m)}_{\mathbf{i}}}(\Sigma) Q^\pi_{(S^{(m)}_{\mathbf{i}}, \Sigma)}(j)$. Note, in contrast with the prior, here each of $Q^I, Q^\sigma, Q^\pi$ are allowed to depend on the samples $S^{(m)}$. Let $G_Q$ be the Gibbs classifier which classifies $x$ stochastically by sampling $(\mathbf{i}, \Sigma, j) \sim Q(\mathbf{i}, \Sigma, j)$ and returning $\mathcal{R}(\mathbf{i}, \Sigma, j)(x)$.

Then, from Theorem 4 and Equation 15, we know that $\forall \delta \in (0, 1]$,

$$\Pr_{S^{(m)} \sim \mathcal{D}^m} [\forall \{Q : R_{S^{(m)}}(G_Q) = 0\} \Phi(Q, P, m, \delta, s)] \geq 1 - \delta$$

where $\Phi(Q, P, m, \delta, s)$ is the event

$$R_{\mathcal{D}}(G_Q) \leq \frac{KL(Q || P_{S^{(m)}}) + \ln\left(\frac{1}{\delta}\right)}{m - s} \tag{17}$$

Consider the weights $w$ classifier $\mathcal{N}_w^{sign}$ we obtain from training the neural network $\mathcal{N}$ on $S^{(m)}$. Since the above bound is uniform over all $Q$, if we can find a posterior $Q_{\mathcal{N}}$ such that $G_{Q_{\mathcal{N}}} = \mathcal{N}_w^{sign}$, then we can use Equation 17 to bound the true risk $R_{\mathcal{D}}(\mathcal{N}_w^{sign})$ of our neural network. Else if we cannot, then Equation 17 does not comment on the risk $R_{\mathcal{D}}(\mathcal{N}_w^{sign})$. However, we will show that whenever the three assumptions of the theorem hold, we can find such a posterior, and the bound will hold.

Well, by Assumption 2, we know that $\Lambda(w)$ is the unique max-margin classifier for $(\phi(x^j, w), y^j)_{j=1}^m$. But, since we have also assumed at most $s$ network support vectors, we know that $\Lambda(w)$ is *also* the unique max-margin classifier for some subset of support vectors $S^{(s)} \subset S^{(m)}$. Since $|S^{(s)}| \leq s$, we can get $\mathbf{i}_{\mathcal{N}} \in I^{(s)}$ such that $S^{(s)} \subset S^{(m)}_{\mathbf{i}_{\mathcal{N}}}$. Furthermore, there is at least one value $\Sigma_{\mathcal{N}} \in \mathcal{M}^\sigma_{S^{(m)}}$, namely $\Sigma_{\mathcal{N}} \triangleq (\bar{\sigma}(x, w))_{(x,y) \in S^{(m)}_{\mathbf{i}_{\mathcal{N}}}}$, for which $C_{MM}(S^{(m)}_{\mathbf{i}_{\mathcal{N}}}, \Sigma_{\mathcal{N}})$ is True and $\mathcal{W}(S^{(m)}_{\mathbf{i}_{\mathcal{N}}}, \Sigma_{\mathcal{N}}) \ni w$ is nonempty. Hence, for some $j_{\mathcal{N}} \in \mathcal{M}^\pi_{S^{(m)}}$, $\mathcal{R}(\mathbf{i}_{\mathcal{N}}, \Sigma_{\mathcal{N}}, j_{\mathcal{N}}) = \mathcal{N}_w^{sign}$ as desired.

Let $Q^I$ be a distribution on $I^{(s)}$ which samples the index set $\mathbf{i}_{\mathcal{N}}$ with probability 1. Let $Q^\sigma$ be a distribution on $\mathcal{M}^\sigma_{S^{(m)}}$ which is uniform over the set of activations consistent with $\mathcal{N}_w^{sign}$:

$$Sym(w, S^{(m)}_{\mathbf{i}_{\mathcal{N}}}) \triangleq \{\Sigma : \exists v \in \mathcal{W}(S^{(m)}_{\mathbf{i}_{\mathcal{N}}}, \Sigma) \text{ with } \mathcal{N}_v^{sign} = \mathcal{N}_w^{sign}\} \tag{18}$$

$$Q^\sigma = Uniform(Sym(w, S^{(m)}_{\mathbf{i}_{\mathcal{N}}})). \tag{19}$$

For example, within-layer neuron permutations yield different $\Sigma$ but the same classifier. At last, for each $\Sigma \sim Q^\sigma$, let $Q^\pi_{(S^{(m)}_{\mathbf{i}}, \Sigma)}$ be a distribution on $\mathcal{M}^\pi_{S^{(m)}}$ placing all of its mass on the (unique) index $j_\Sigma$ such that $\mathcal{R}(\mathbf{i}_{\mathcal{N}}, \Sigma, j_\Sigma) = \mathcal{N}_w^{sign}$ as functions. Therefore, $Q_{\mathcal{N}} \triangleq Q^I Q^\sigma Q^\pi$ is a posterior distribution returning $\mathcal{N}_w^{sign}$ with probability one. Thus the Gibbs classifier $G_{Q_{\mathcal{N}}}$ is a deterministic classifier and is equal to $\mathcal{N}_w^{sign}$.

There, we may claim Equation 17 holds for posterior $Q_{\mathcal{N}}$ with probability at least $1 - \delta$.

To conclude our theorem, we simply expand and upper bound $KL(Q_{\mathcal{N}}, P_{S^{(m)}})$:

$$
\begin{aligned}
&KL(Q_{\mathcal{N}}||P_{S^{(m)}}) = \\
&= \mathop{\mathbb{E}}_{\mathbf{i}\sim Q^I} \mathop{\mathbb{E}}_{\Sigma\sim Q^\sigma} \mathop{\mathbb{E}}_{j\sim Q^\pi} \ln\left(\frac{Q^I(\mathbf{i})Q^\sigma(\Sigma)Q^\pi(j)}{P^I(\mathbf{i})P^\sigma(\Sigma)P^\pi(j)}\right) \\
&= \mathop{\mathbb{E}}_{\mathbf{i}\sim Q^I} \ln\left(\frac{Q^I(\mathbf{i})}{P^I(\mathbf{i})}\right) + \mathop{\mathbb{E}}_{\mathbf{i}\sim Q^I} \mathop{\mathbb{E}}_{\Sigma\sim Q^\sigma} \ln\left(\frac{Q^\sigma(\Sigma)}{P^\sigma(\Sigma)}\right) \\
&\quad + \mathop{\mathbb{E}}_{\mathbf{i}\sim Q^I} \mathop{\mathbb{E}}_{\Sigma\sim Q^\sigma} \mathop{\mathbb{E}}_{j\sim Q^\pi} \ln\left(\frac{Q^\pi(j)}{P^\pi(j)}\right) \\
&= \ln\left(\binom{m}{s}\right) + \mathop{\mathbb{E}}_{\mathbf{i}\sim Q^I} \mathop{\mathbb{E}}_{\Sigma\sim Q^\sigma} \ln\left(\frac{|\mathcal{M}^\sigma(S^{(m)}_{\mathbf{i}_{\mathcal{N}}})|}{|Sym(w, S^{(m)}_{\mathbf{i}_{\mathcal{N}}})|}\right) \\
&+ \mathop{\mathbb{E}}_{\mathbf{i}\sim Q^I} \mathop{\mathbb{E}}_{\Sigma\sim Q^\sigma} \mathop{\mathbb{E}}_{j\sim Q^\pi} \ln\left(|\mathcal{M}^\pi(S^{(m)}_{\mathbf{i}_{\mathcal{N}}}, \Sigma)|\right). \qquad (20)
\end{aligned}
$$

To conclude the proof, we crudely upper bound $\forall \mathbf{i}$ $|\mathcal{M}^\sigma(S^{(m)}_{\mathbf{i}})| \leq 2^{ns}$, which follows because at each of $s$ samples $(x, y) \in S^{(m)}_{\mathbf{i}_{\mathcal{N}}}$ and at each neuron, $(l, i_l)$, of $n$ possible neurons, $\sigma^{(l)}(x, w)_{i_l}$ can take one of two values. We also have $|\mathcal{M}^\sigma(T_{\mathbf{i}})| \leq 2^n$ by Theorem 2. Clearly, $|Sym(w, S^{(m)}_{\mathbf{i}_{\mathcal{N}}})| \geq 1$. Combining this with Equation 20, we have

$$
KL(Q_{\mathcal{N}}||P_{S^{(m)}}) \leq \ln\left(\binom{m}{s}\right) + \ln(2^{ns}2^n)
$$

where we simply drop $\ln(2) < 1$, and approximate $\binom{m}{s} \leq (\frac{me}{s})^s$ to arrive at

$$
KL(Q_{\mathcal{N}}||P_{S^{(m)}}) \leq s\ln\left(\frac{m}{s}\right) + s + ns + n
$$

Substituting the above into Equation 17 finishes the proof. □