Simplicity bias in the parameter-function map of deep neural networks

Anonymous Authors[1]

Abstract

The idea that neural networks may exhibit a bias towards simplicity has a long history (1; 2; 3; 4). *Simplicity bias* (5) provides a way to quantify this intuition. It predicts, for a broad class of input-output maps which can describe many systems in science and engineering, that simple outputs are exponentially more likely to occur upon uniform random sampling of inputs than complex outputs are. This simplicity bias behaviour has been observed for systems ranging from the RNA sequence to secondary structure map, to systems of coupled differential equations, to models of plant growth. Deep neural networks can be viewed as a mapping from the space of parameters (the weights) to the space of functions (how inputs get transformed to outputs by the network). We show that this parameter-function map obeys the necessary conditions for simplicity bias, and numerically show that it is hugely biased towards functions with low descriptional complexity. We also demonstrate a Zipf like power-law probability-rank relation. A bias towards simplicity may help explain why neural nets generalize so well.

1. Introduction to simplicity bias

In a recent paper (5), an inequality inspired by the coding theorem from algorithmic information theory (AIT) (6), and applicable to computable input-output maps was derived using the following simple procedure. Consider a map $f : I \to O$ between N_I inputs and N_O outputs. The size of the inputs space is parameterized as n, e.g. if the inputs are binary strings, then $N_I = 2^n$. Assuming f and n are given, implement the following simple procedure: first enumerate all 2^n inputs and map them to outputs using f. Then order the outputs by how frequently they appear. Using a Shannon-Fano code, one can then describe x with a code of length $-\log_2 P(x) + O(1)$, which therefore upper bounds the Kolmogorov complexity, giving the relation $P(x) \leq 2^{-K(x|f,n)+O(1)}$. The $\mathcal{O}(1)$ terms are independent of x (but hard to estimate). Similar bounds can be found in standard works (6). As pointed out in (5), if the maps are simple, that is *condition 1*: $K(f) + K(n) \ll K(x) + \mathcal{O}(1)$ holds, then because $K(x) \leq K(x|f,n) + K(f) + K(n) + \mathcal{O}(1)$, and $K(x|f,n) \leq K(x) + \mathcal{O}(1)$, it follows that $K(x|f,n) \approx K(x) + \mathcal{O}(1)$. The problem remains that Kolmogorov complexity is fundamentally uncomputable (6), and that the $\mathcal{O}(1)$ terms are hard to estimate. However, in reference (5) a more pragmatic approach was taken to argue that a bound on the probability $P(x)$ that x obtains upon random sampling of inputs can be approximated as

$$P(x) \leq 2^{-a\tilde{K}(x)-b}, \tag{1}$$

where $\tilde{K}(x)$ is a suitable approximation to the Kolmogorov complexity of x. Here a and b are constants that are independent of x and which can often be determined from some basic information about the map. These constants pick up multiplicative and additive factors in the approximation to $K(x)$ and to the $\mathcal{O}(1)$ terms.

In addition to the simplicity of the the input-output map f (condition (1)), the map also needs to obey conditions (2) *Redundancy*: that the number of inputs N_I is much larger than the number of outputs N_O, as otherwise $P(x)$ can't vary much; 3) *Large systems* where $N_O \gg 0$, so that finite size effects don't play a dominant role; 4) *Non-linear*: If the map f is linear it won't show bias and 5) *Well-behaved*: The map should not have a significant fraction of pseudorandom outputs because it is hard to find good approximations $\tilde{K}(x)$. For example many random-number generators produce outputs that appear complex, but in fact have low $K(x)$ because they are generated by a relatively simple algorithms with short descriptions.

Some of the steps above may seem rather rough to AIT purists. For example: Can a reasonable approximation to $K(x)$ be found? What about $\mathcal{O}(1)$ terms? And, how do you know condition 5) is fulfilled? Notwithstanding these important questions, in reference (5) the simplicity bias bound (1) was tested empirically for a wide range of different maps, ranging from a sequence to RNA secondary

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

structure map, to a set of coupled differential equations, to L-systems (a model for plant morphology and computer graphics) to a stochastic financial model. In each case the bound works remarkably well: High probability outputs have low complexity, and high complexity outputs have low probability (but not necessarily vice versa). A simple matrix map that allows *condition 1* to be directly tested also demonstrates that when the map becomes sufficiently complex, simplicity bias phenomena disappear.

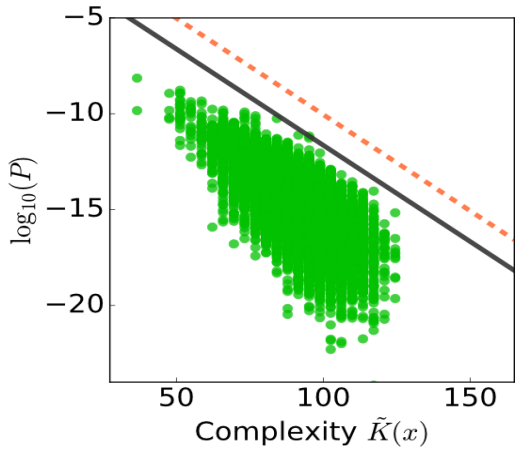


Figure 1. Probability that an RNA secondary structure x obtains upon random sampling of length $L = 80$ sequences versus a Lempel-Ziv measure of the complexity of the structure. The black solid line is the simplicity-bias bound (1), while the dashed line denotes the bound with the parameter b set to zero.

In Fig. 1 we illustrate an iconic input-output map for RNA, a linear biopolymer that can fold into well-defined sructures due to specific bonding between the four different types of nucleotides ACUG from which its sequences are formed. While the full three-dimensional structure is difficult to predict, the secondary structure, which records which nucleotide binds to which nucleotide, can be efficiently and accurately calculated. This mapping from sequences to secondary structures fulfills the conditions above. Most importantly, the map, which uses the laws of physics to determine the lowest free-energy structure for a given sequence, is independent of the length of the sequences, and so fulfills the simplicity condition (1). The structures (the outputs x) can be written in terms of a ternary string, and so simple compression algorithms can be used to estimate their complexity. In Fig. 1, we observe, as expected, that the probability $P(x)$ that a particular secondary structure x is found upon random sampling of sequences is bounded by Eq. (1) as predicted. Similar robust simplicity bies behaviour to that seen in this figure was observed for the other maps.

Similar scaling (5) was also observed for this map with a series of other approximations to $K(x)$, suggesting that the precise choice of complexity measure was not critical, as long as it captures some basic essential features.

In summary then, the simplicity bias bound (1) works robustly well for a wide range of different maps. The predictions are strong: the probability that an output obtains upon random sampling of inputs should drop (at least) exponentially with linear increases in the descriptional complexity of the output. Nevertheless, it is important to note that while the 5 conditions above are sufficient for the bound (1) to hold, they are not sufficient to guarantee that the map will be biased (and therefore simplicity biased). One can easily construct maps that obey them, but do not show bias. Understanding the conditions resulting in biased maps is very much an open area of investigation.

The question we will address here is: Can deep learning be re-cast into the language of input-output maps, and if so, do these maps also exhibit the very general phenomenon of simplicity bias?

2. The parameter-function map

Definition 1. *(Parameter-function map) For a parametrized supervised learning model, let the input space be $\mathcal{X}$ and the output space be $\mathcal{Y}$. The space of functions that the model can express is then $\mathcal{F} \subseteq \mathcal{Y}^{|\mathcal{X}|}$. If the model has p real valued parameters, taking values within a set $\Theta \subseteq \mathbb{R}^p$, the parameter-function map, $\mathcal{M}$, is defined as:*

$$\begin{aligned} \mathcal{M} : \Theta &\to \mathcal{F} \\ \theta &\mapsto f_\theta \end{aligned}$$

where f_θ is the function implemented by the model with choice of parameter vector θ.

It is not hard to see that the map above obeys *condition 1*: The shortest description of the map grows slowly with the logarithm of the size of the space of functions (which determines the typical $K(x)$). Conditions 2-4 are also clearly met. Condition 5 is more complex and requires empirical testing. But given that simplicity bias was observed for such a wide range of maps, our expectation is that it will hold robustly for neural networks also.

3. Simplicity bias on CIFAR10

In order to explore the properties of the parameter-function map, we consider *random neural networks*. We put a probability distribution over the space of parameters Θ, and are interested in the distribution over functions induced by this distribution via the parameter-function map of a given neural network. We consider Gaussian and uniform distributions over parameters. In the following, when we say "probability of a function", we imply we

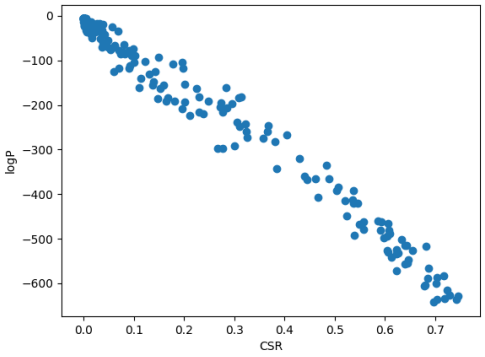
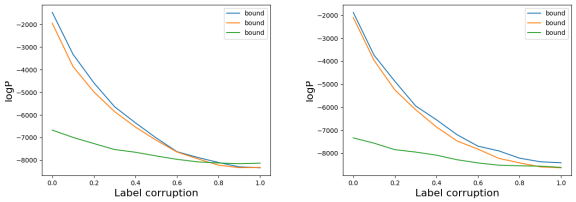


Figure 2. Probability (using GP approximation) versus critical sample ratio (CSR) of labelings of 1000 random CIFAR10 inputs, produced by 250 random samples of parameters. The network is a 4 layer CNN.



(a) (b)

Figure 3. log probability of a labelling on 10k random images on three different datasets, as the fraction of label corruption increases. The probability is computed using the Gaussian process approximation for (a) a CNN with 4 layers and no pooling. (b) a 1 hidden layer FC network.

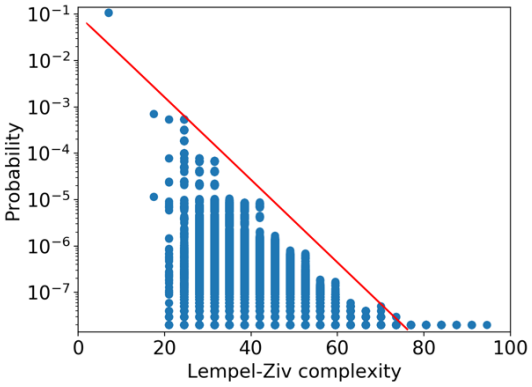


Figure 4. Probability versus Lempel-Ziv complexity. Probabilities are estimated from a sample of 10^8 parameters. The red line is the simplicity bias bound of Eq.(1)

are using a Gaussian or uniform i.i.d. distribution over parameters unless otherwise specified.

For a convolutional network with 4 layers and no pooling[1], we have used the Gaussian process approximation(7; 8) to estimate the probability of different labellings[2] on a random sample of 1000 images from CIFAR10. We used the critical sample ratio (CSR) (4) as a measure of the complexity of the functions[3]. Fig. 2, depicts the strong correlation between the log probability and CSR consistent with Eq. (1).

In Fig. 3, we also show the values of the log probability for a CNN and a FC network, on a larger sample of 10k images from CIFAR10, MNIST, and fashion-MNIST. This illustrates the large range of orders of magnitude for the probability of different labellings, as well as the negative correlation with increasing label corruption. While label corruption is not a direct measure of descriptional complexity, it is almost certainly the case that increasing label corruption generally corresponds to an increase in measures of Kolmogorov complexity.

4. Simplicity bias in a model system

Direct measurements of simplicity bias are extremely computationally expensive, and so far we have not been able to obtain directly sampled results for datasets larger than CIFAR10.

To further isolate the phenomenon of simplicity bias in the parameter-function map, and study the effect of using different complexity measures, we look at a series of MLPs with 7 Boolean input neurons, 1 Boolean output neuron, and a number of layers of 40 hidden neurons each. We use ReLU activation in all hidden layers. For each such architecture, we sample the parameters i.i.d. according to a Gaussian or uniform distribution. We then count how often individual Boolean functions are obtained, and use the normalized empirical frequencies as estimate of their probability. In Fig. 4, we plot the probability (in log scale) versus the Lempel-Ziv complexity, which is defined in Appendix C.1. In Appendix C.3, we show similar results for other complexity measures, demonstrating that the simplicity bias is robust to the choice of complexity measure, as long as it not too simple (like entropy). The simplicity bias observed in Fig. 4 very closely resembles that seen for the other maps in (5).

In Fig. 4 in Appendix C.3, we show the same data as in Fig. 4, but as a histogram, highlighting that most of the probability mass, when sampling parameters, is close to the upper bound. In addition, in Fig. 6 in Appendix D, we show the probability versus complexity plots for fully-

[1] See Appendix A for more details on the architecture

[2] See Appendix A.1 for details on the Gaussian process approximation

[3] See Appendix C for more details on complexity measures

connected networks with increasing number of layers, showing that the correlation is similar in all cases, although for deeper networks, the bias is a bit stronger.

5. Zipf's law conjecture

In our experiments in the previous two sections, we observe that the range of probabilities of functions spans many orders of magnitude. Here we show that for the experiment in Section 4, when we plot the probabilities versus the rank of the function (when ranked by probability), we find that the probabilities asymptotically follow Zipf's law:

$$\text{Prob}(\text{rank}) = \frac{1}{(\ln N_O)\text{rank}}, \tag{2}$$

where N_O is the total number of functions.

This was also observed in (9) for networks of Boolean functions, and in (10) for more general circuits. We formalize our findings for neural networks in the following conjecture:

Conjecture 1 (Zipf's law beahviour) *The probability-rank relation of functions for sufficiently over-parametrized neural networks follows Zipf's law (Equation 2)*

By "sufficiently over-parametrized" we mean that the networks are in the regime where the Gaussian process approximation to the distribution over function is a good approximation (which appears to be the case for many cases in practice (11; 8; 12)). We now explain the experiments we conducted to test this conjecture.

In Fig. 5, we show the probabilities of the functions versus their rank (when ranked by probability), for different choices of parameter distribution. We observe that for all the parameter distributions the probability-rank plot appears to approach Zipf's law.

To determine the parameter N_O in Zipf's law, we checked that this architecture can produce almost all Boolean functions of 7 inputs, by finding that it could fit all the functions in a sample of 1000 random Boolean functions. We thus used $N_O = 2^{2^7}$ in the Zipf's law curve in Fig. 5, finding excellent agreement with no free parameters. These results imply in particular that the distribution is extremely biased, with probabilities spanning a huge range of orders of magnitudes. Note that the mean $P(x)$ over all functions is $1/N_O \approx 3 \times 10^{-39}$, so that in our plots we are only showing a tiny fraction of functions for which $P(x)$ is orders of magnitude larger than the mean.

We can't reliably obtain a probability-rank plot for the experiments we did on CIFAR10, because our sampling of the space of functions is much sparser in this case. However, the probabilities still span a huge range of orders of magnitude, as seen in Figs. 2, 3. In fact, the range of orders of magnitude appears to scale with m, the size of the input space. This is consistent with a probability-rank following Zipf's law.

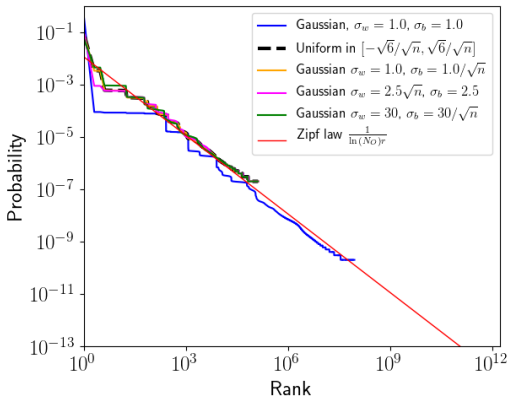


Figure 5. Probability versus rank of each of the functions (ranked by probability) from a sample of 10^{10} (blue) or 10^7 (others) parameters. The labels are different parameter distributions. σ_w^2/n and σ_b^2 are weight and bias variances, respectively, where n is the number of hidden neurons

6. Generalization

The fact that neural networks are biased towards simple solutions has been conjectured to be the main reason they generalize (4; 3; 12). Here we have explored a form of simplicity bias encoded in the parameter-function map. It turns out that this bias is enough to guarantee good generalization, as was shown in (12) via a PAC-Bayesian formalism. In Fig. 1 in Appendix B, we show the PAC-Bayesian bounds versus the true generalization error they obtained, finding that the bounds are not only nonvacuous but follow the trends of the true error closely.

Simplicity bias also provides a lens through which to naturally understand various phenomena observed in deep networks, for example, that the number of parameters does not strongly affect generalization.

7. Conclusion

We have provided evidence that neural networks exhibit simplicity bias. The fact that the phenomena observed are remarkably similar to those of a wide range of maps from science and engineering (5) suggests that this behaviour is general, and will hold for many neural network architectures. It would be interesting to test this claim for larger systems, which will require new sampling techniques, and to derive analytic arguments for a bias towards simplicity, as done in (13).

References

[1] Geoffrey E. Hinton and Drew van Camp. Keeping the neural networks simple by minimizing the description length of the weights. In *Proceedings of the Sixth Annual Conference on Computational Learning Theory*, COLT '93, pages 5–13, New York, NY, USA, 1993. ACM.

[2] Jürgen Schmidhuber. Discovering neural nets with low kolmogorov complexity and high generalization capability. *Neural Networks*, 10(5):857–873, 1997.

[3] Lei Wu, Zhanxing Zhu, et al. Towards understanding generalization of deep learning: Perspective of loss landscapes. *arXiv preprint arXiv:1706.10239*, 2017.

[4] David Krueger, Nicolas Ballas, Stanislaw Jastrzebski, Devansh Arpit, Maxinder S. Kanwal, Tegan Maharaj, Emmanuel Bengio, Asja Fischer, Aaron Courville, Simon Lacoste-Julien, and Yoshua Bengio. A closer look at memorization in deep networks. Proceedings of the 34th International Conference on Machine Learning (ICML'17), 2017.

[5] Kamaludin Dingle, Chico Q Camargo, and Ard A Louis. Input–output maps are strongly biased towards simple outputs. *Nature communications*, 9(1):761, 2018.

[6] Ming Li and Paul Vitányi. *An introduction to Kolmogorov complexity and its applications*. Springer Science & Business Media, 2013.

[7] Adrià Garriga-Alonso, Laurence Aitchison, and Carl Edward Rasmussen. Deep convolutional networks as shallow Gaussian processes. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019.

[8] Roman Novak, Lechao Xiao, Jaehoon Lee, Yasaman Bahri, Daniel A Abolafia, Jeffrey Pennington, and Jascha Sohl-Dickstein. Bayesian convolutional neural networks with many channels are gaussian processes. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019.

[9] Christian Van den Broeck and Ryoichi Kawai. Learning in feedforward boolean networks. *Physical Review A*, 42(10):6210, 1990.

[10] Karthik Raman and Andreas Wagner. The evolvability of programmable hardware. *Journal of the Royal Society Interface*, 8(55):269–281, 2010.

[11] Alexander G de G Matthews, Mark Rowland, Jiri Hron, Richard E Turner, and Zoubin Ghahramani. Gaussian process behaviour in wide deep neural networks. *arXiv preprint arXiv:1804.11271*, 2018.

[12] Guillermo Valle-Perez, Chico Q Camargo, and Ard A Louis. Deep learning generalizes because the parameter-function map is biased towards simple functions. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019.

[13] Giacomo De Palma, Bobak Toussi Kiani, and Seth Lloyd. Deep neural networks are biased towards simple functions. *arXiv preprint arXiv:1812.10156*, 2018.

A. Architecture details

In the main experiments of the paper we used two classes of architectures. Here we describe them in more detail.

- Fully connected networks (FCs), with varying number of layers. The size of the hidden layers was the same as the input dimension, and the nonlinearity was ReLU. The last layer was a single Softmax neuron. We used default Keras settings for initialization (Glorot uniform).
- Convolutional neural networks (CNNs), with varying number of layers. The number of filters was 200, and the nonlinearity was ReLU. The last layer was a fully connected single Softmax neuron. The filter sizes alternated between $(2, 2)$ and $(5, 5)$, and the padding between SAME and VALID, the strides were 1 (same default settings as in the code for (1)). We used default Keras settings for initialization (Glorot uniform).

A.1. Gaussian process approximation to the prior over functions

In recent work ((2; 3; 1; 4)), it was shown that infinitely-wide neural networks (including convolutional and residual networks) are equivalent to Gaussian processes. This means that if the parameters are distributed i.i.d. (for instance with a Gaussian with diagonal covariance), then the (real-valued) outputs of the neural network, corresponding to any finite set of inputs, are jointly distributed with a Gaussian distribution. More precisely, assume the i.i.d. distribution over parameters is $\tilde{P}$ with zero mean, then for a set of n inputs $(x_1, ..., x_n)$,

$$P_{\theta\sim\tilde{P}}\left(f_\theta(x_1) = \tilde{y}_1, ..., f_\theta(x_n) = \tilde{y}_n\right) \propto \exp\left(-\frac{1}{2}\tilde{\mathbf{y}}^T\mathbf{K}^{-1}\tilde{\mathbf{y}}\right), \tag{1}$$

where $\tilde{\mathbf{y}} = (\tilde{y}_1, ..., \tilde{y}_n)$. The entries of the covariance matrix $\mathbf{K}$ are given by the kernel function k as $K_{ij} = k(x_i, x_j)$. The kernel function depends on the choice of architecture, and properties of $\tilde{P}$, in particular the weight variance σ_w^2/n (where n is the size of the input to the layer) and the bias variance σ_b^2. The kernel for fully connected ReLU networks has a well known analytical form known as the arccosine kernel ((5)), while for convolutional and residual networks it can be efficiently computed[1].

The main quantity in the PAC-Bayes theorem, $P(U)$, is precisely the probability of a given set of output labels for the set of instances in the training set, also known as *marginal likelihood*, a connection explored in recent work ((6; 7)). For binary classification, these labels are binary, and are related to the real-valued outputs of the network via a nonlinear function such as a step functionwhich we denote σ. Then, for a training set $U = \{(x_1, y_1), ..., (x_m, y_m)\}$, $P(U) = P_{\theta\sim\tilde{P}}\left(\sigma(f_\theta(x_1)) = y_1, ..., \sigma(f_\theta(x_m)) = y_m\right)$.

This distribution no longer has a Gaussian form because of the output nonlinearity σ. We will discuss approximations to deal with this in the following. There is also the more fundamental issue of neural networks not being infinitely-wide in practice. However, the Gaussian process limit has been found to be a good approximation for nets with reasonable widths (3; 4; 8).

In order to calculate $P(U)$ using the GPs, we use the expectation-propagation (EP) approximation, implemented in (9), which is more accurate than the Laplacian approximation (see (10) for a description and comparison of the algorithms). In (8), the authors compare the two approximations and find that EP appears to give better approximations.

B. PAC-Bayesian generalization bounds

C. Other complexity measures

One of the key steps to practical application of the simplicity bias framework of Dingle et al. in (11) is the identification of a suitable complexity measure $\tilde{K}(x)$ which mimics aspects of the (uncomputable) Kolmogorov complexity $K(x)$ for the problem being studied. It was shown for the maps in (11) that several different complexity measures all generated the same qualitative simplicity bias behaviour:

$$P(x) \leq 2^{-(a\tilde{K}(x)+b)} \tag{2}$$

[1] We use the code from (1) to compute the kernel for convolutional networks

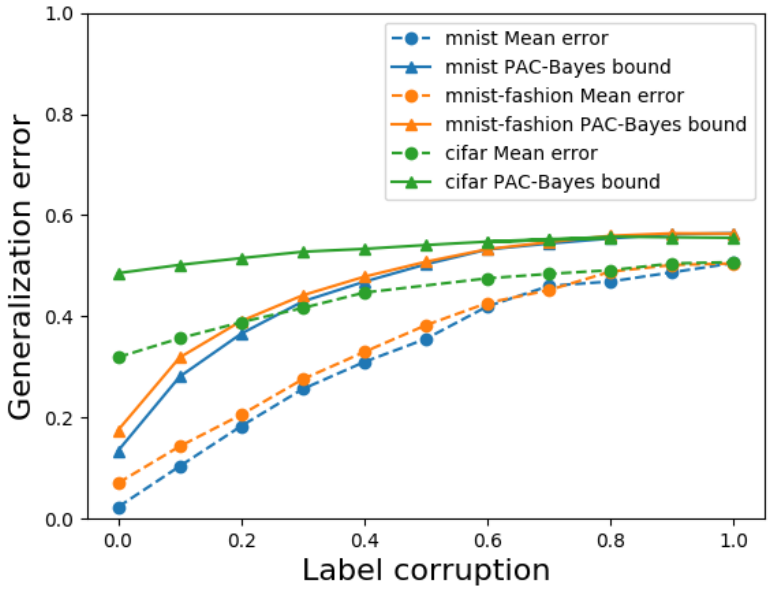


Figure 1. PAC-Bayes bound and generalization error versus training set size for three different datasets, for a CNN with 4 layers and no pooling, for three datasets and a training set of size 10000. Training set error is 0 in all experiments. (reproduced with permission from (8))

but with different values of a and b depending on the complexity measure and of course depending on the map, but independent of output x. Showing that the same qualitative results obtain for different complexity measures is sign of robustness for simplicity bias.

Below we list a number of different descriptional complexity measures which we used, to extend the experiments in Section 4 in the main text.

C.1. Complexity measures

Lempel-Ziv complexity (*LZ complexity* for short). The Boolean functions studied in the main text can be written as binary strings, which makes it possible to use measures of complexity based on finding regularities in binary strings. One of the best is Lempel-Ziv complexity, based on the Lempel-Ziv compression algorithm. It has many nice properties, like asymptotic optimality, and being asymptotically equal to the Kolmogorov complexity for an ergodic source. We use the variation of Lempel-Ziv complexity from (11) which is based on the 1976 Lempel Ziv algorithm ((12)):

$$K_{LZ}(x) = \begin{cases} \log_2(n), & x = 0^n \text{ or } 1^n \\ \log_2(n)[N_w(x_1 ... x_n) + N_w(x_n ... x_1)]/2, & \text{otherwise} \end{cases} \tag{3}$$

where n is the length of the binary string, and $N_w(x_1 ... x_n)$ is the number of words in the Lempel-Ziv "dictionary" when it compresses output x. The symmetrization makes the measure more fine-grained, and the $\log_2(n)$ factor as well as the value for the simplest strings ensures that they scale as expected for Kolmogorov complexity. This complexity measure is the primary one used in the main text.

We note that the binary string representation depends on the order in which inputs are listed to construct it, which is not a feature of the function itself. This may affect the LZ complexity, although for low-complexity input orderings (we use numerical ordering of the binary inputs), it has a negligible effect, so that $K(x)$ will be very close to the Kolmogorov complexity of the function.

Entropy. A fundamental, though weak, measure of complexity is the entropy. For a given binary string this is defined as $S = -\frac{n_0}{N}\log_2\frac{n_0}{N} - \frac{n_1}{N}\log_2\frac{n_1}{N}$, where n_0 is the number of zeros in the string, and n_1 is the number of ones, and

$N = n_0 + n_1$. This measure is close to 1 when the number of ones and zeros is similar, and is close to 0 when the string is mostly ones, or mostly zeros. Entropy and $K_{LZ}(x)$ are compared in Fig. 2, and in more detail in supplementary note 7 (and supplementary information figure 1) of reference (11). They correlate, in the sense that low entropy $S(x)$ means low $K_{LZ}(x)$, but it is also possible to have Large entropy but low $K_{LZ}(x)$, for example for a string such as 10101010....

Boolean expression complexity. Boolean functions can be compressed by finding simpler ways to represent them. We used the standard SciPy implementation of the Quine-McCluskey algorithm to minimize the Boolean function into a small sum of products form, and then defined the number of operations in the resulting Boolean expression as a *Boolean complexity* measure.

Generalization complexity. L. Franco et al. have introduced a complexity measure for Boolean functions, designed to capture how difficult the function is to learn and generalize ((13)), which was used to empirically find that simple functions generalize better in a neural network ((14)). The measure consists of a sum of terms, each measuring the average over all inputs fraction of neighbours which change the output. The first term considers neighbours at Hamming distance of 1, the second at Hamming distance of 2 and so on. The first term is also known (up to a normalization constant) as average sensitivity ((15)). The terms in the series have also been called "generalized robustness" in the evolutionary theory literature ((16)). Here we use the first two terms, so the measure is:

$$C(f) = C_1(f) + C_2(f),$$
$$C_1(f) = \frac{1}{2^n n} \sum_{x \in X} \sum_{y \in \text{Nei}_1(x)} |f(x) - f(y)|,$$
$$C_1(f) = \frac{2}{2^n n(n-1)} \sum_{x \in X} \sum_{y \in \text{Nei}_2(x)} |f(x) - f(y)|,$$

where $\text{Nei}_i(x)$ is all neighbours of x at Hamming distance i.

Critical sample ratio. A measure of the complexity of a function was introduced in (17) to explore the dependence of generalization with complexity. In general, it is defined with respect to a sample of inputs as the fraction of those samples which are *critical samples*, defined to be an input such that there is another input within a ball of radius r, producing a different output (for discrete outputs). Here, we define it as the fraction of all inputs, that have another input at Hamming distance 1, producing a different output.

C.2. Correlation between complexities

In Fig. 2, we compare the different complexity measures against one another. We also plot the frequency of each complexity; generally more functions are found with higher complexity.

C.3. Probability-complexity plots

In Fig. 3 we show how the probability versus complexity plots look for other complexity measures. The behaviour is similar to that seen for the LZ complexity measure in Fig 1(b) of the main text. In Fig. 5 we show probability versus LZ complexity plots for other choices of parameter distributions.

In Fig. 4, we show a histogram of the functions in the log probability - complexity plane. The histogram counts are weighted by the probability of the function, so that the total weight is proportional to the probability of obtaining a function in a particular bin, when sampling the parameters.

D. Effect of number of layers on simplicity bias

In Figure 6 we show the effect of the number of layers on the bias (for feedforward neural networks with 40 neurons per layer). The left figures show the probability of individual functions versus the complexity. The right figure shows the histogram of complexities, weighted by the probability by which the function appeared in the sample of parameters.

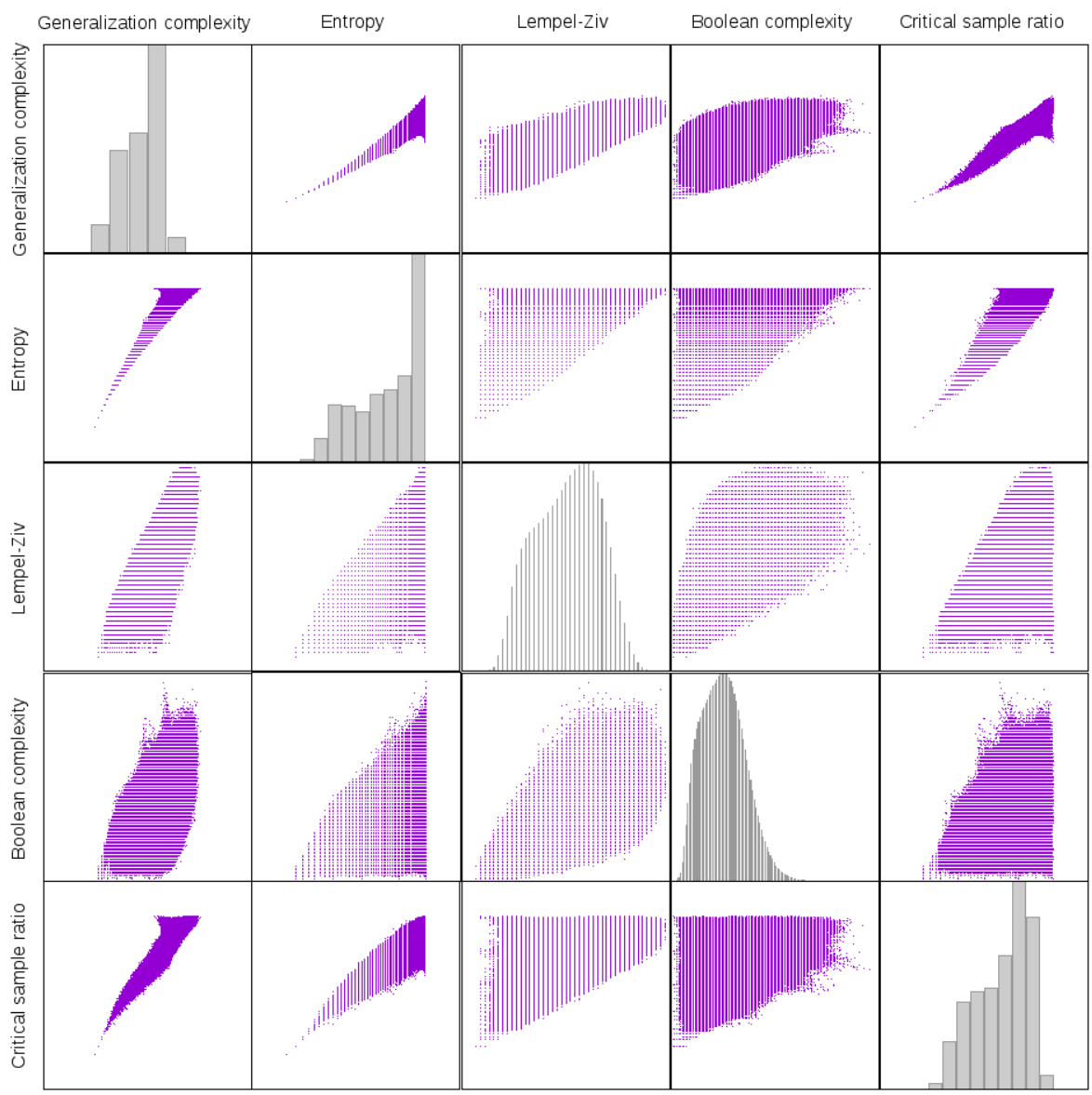
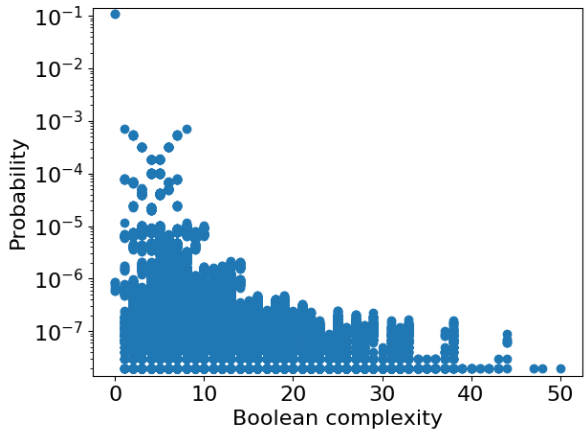
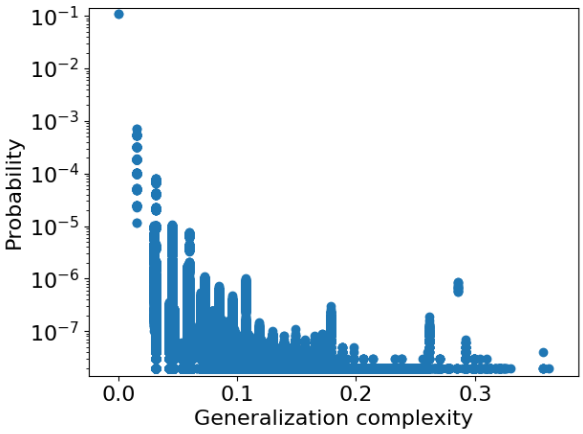


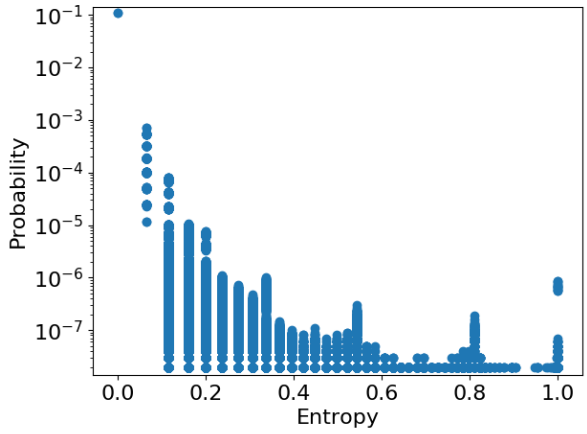
Figure 2. Scatter matrix showing the correlation between the different complexity measures used in this paper On the diagonal, a histogram (in grey) of frequency versus complexity is depicted. The functions are from the sample of 10^8 parameters for the $(7, 40, 40, 1)$ network.



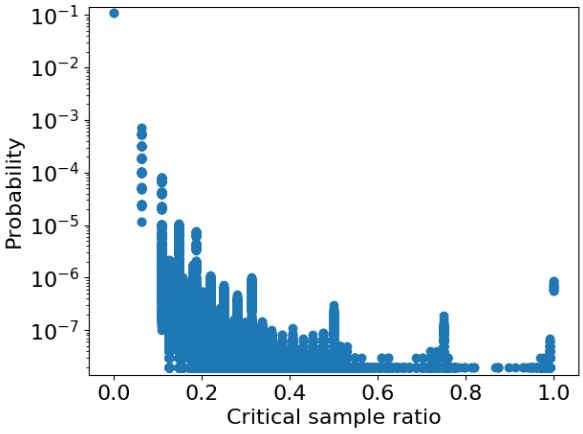
(a) Probability versus Boolean complexity



(b) Probability versus generalization complexity



(c) Probability versus entropy



(d) Probability versus critical sample ratio

Figure 3. Probability versus different measures of complexity (see main text for Lempel-Ziv), estimated from a sample of 10^8 parameters, for a network of shape $(7, 40, 40, 1)$. Points with a frequency of 10^{-8} are removed for clarity because these suffer from finite-size effects (see Appendix E). The measures of complexity are described in Appendix C.

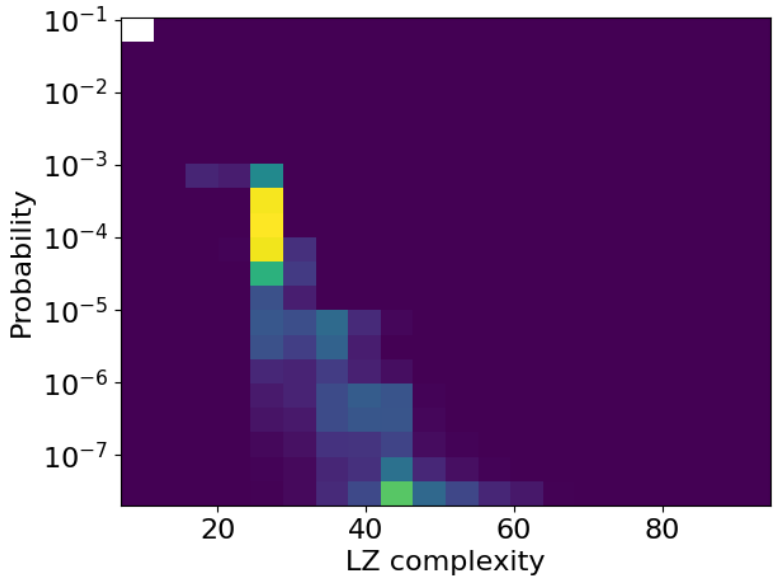


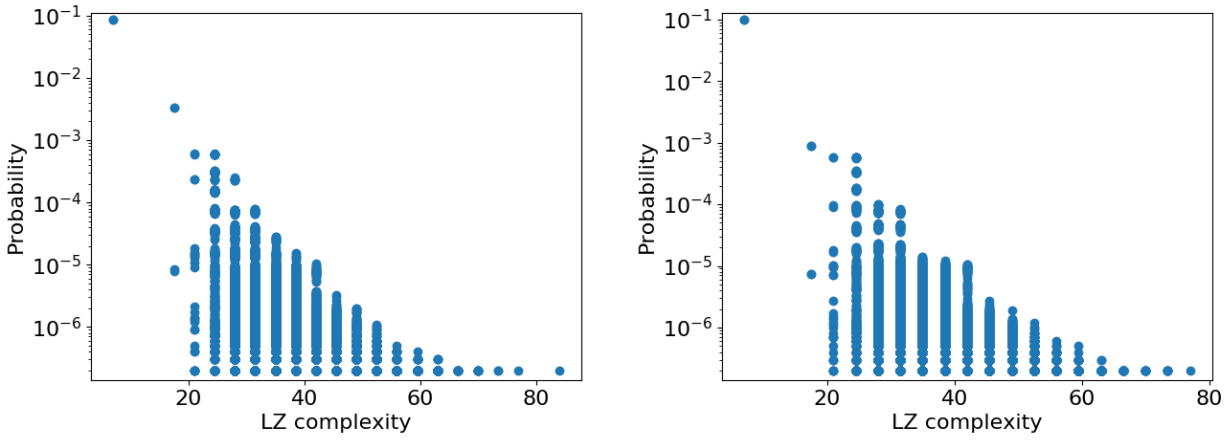
Figure 4. Histogram of functions in the probability versus Lempel-Ziv complexity plane, weighted according to their probability. Probabilities are estimated from a sample of 10^8 parameters, for a network of shape $(7, 40, 40, 1)$

The histograms therefore show the distribution over complexities when randomly sampling parameters[2] We can see that between the 0 layer perceptron and the 2 layer network there is an increased number of higher complexity functions. This is most likely because of the increasing expressivity of the network. For 2 layers and above, the expressivity does not significantly change, and instead, we observe a shift of the distribution towards lower complexity.

E. Finite-size effects for sampling probability

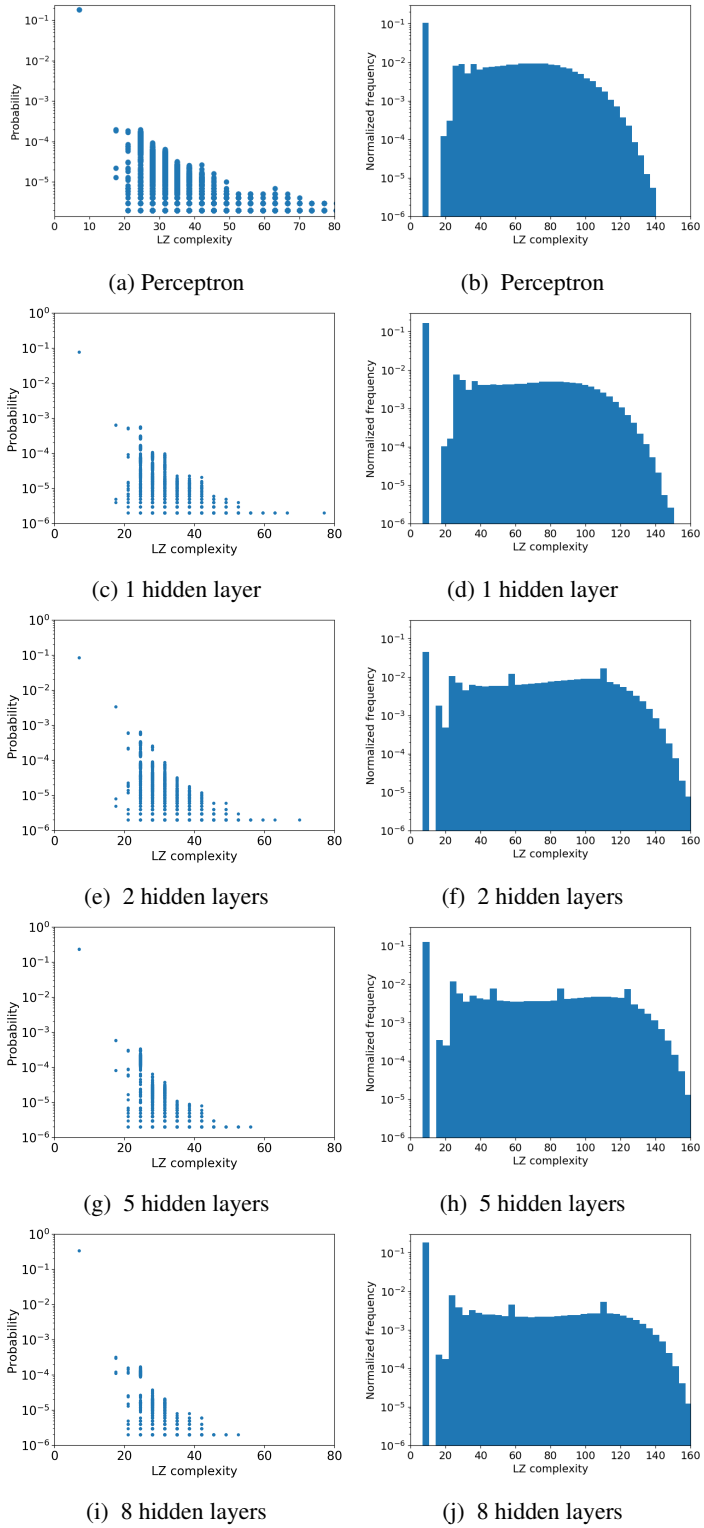
Since for a sample of size N the minimum estimated probability is $1/N$, many of the low-probability samples that arise just once may in fact have a much lower probability than suggested. See Figure 7), for an illustration of how this finite-size sampling effect manifests with changing sample size N. For this reason, these points are typically removed from plots.

[2]using a Gaussian with $1/\sqrt{n}$ variance in this case, n being number of inputs to neuron



(a) (b)

Figure 5. Probability versus LZ complexity for network of shape $(7, 40, 40, 1)$ and varying sampling distributions. Samples are of size 10^7. (a) Weights are sampled from a Gaussian with variance $1/\sqrt{n}$ where n is the input dimension of each layer. (b) Weights are sampled from a Gaussian with variance 2.5



(a) Perceptron (b) Perceptron

(c) 1 hidden layer (d) 1 hidden layer

(e) 2 hidden layers (f) 2 hidden layers

(g) 5 hidden layers (h) 5 hidden layers

(i) 8 hidden layers (j) 8 hidden layers

Figure 6. Probability versus LZ complexity for networks with different number of layers. Samples are of size 10^6 (a) & (b) A perceptron with 7 input neurons (complexity is capped at 80 in (a) to aid comparison with the other figures). (c) & (d) A network with 1 hidden layer of 40 neurons (e) & (f) A network with 2 hidden layer of 40 neurons (g) & (h) A network with 5 hidden layers of 40 neurons each. (i) & (j) A network with 8 hidden layers of 40 neurons each

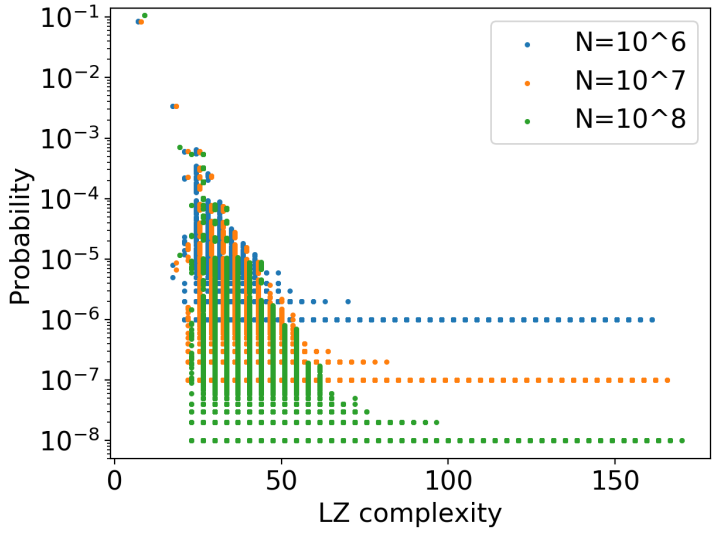


Figure 7. Probability (calculated from frequency) versus Lempel-Ziv complexity for a neural network of shape $(7, 40, 40, 1)$, and sample sizes $N = 10^6, 10^7, 10^8$. The lowest frequency functions for a given sample size can be seen to suffer from finite-size effects, causing them to have a higher frequency than their true probability.

References

[1] Adrià Garriga-Alonso, Laurence Aitchison, and Carl Edward Rasmussen. Deep convolutional networks as shallow Gaussian processes. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019.

[2] Jaehoon Lee, Yasaman Bahri, Roman Novak, Samuel S Schoenholz, Jeffrey Pennington, and Jascha Sohl-Dickstein. Deep neural networks as gaussian processes. *arXiv preprint arXiv:1711.00165*, 2017.

[3] Alexander G de G Matthews, Mark Rowland, Jiri Hron, Richard E Turner, and Zoubin Ghahramani. Gaussian process behaviour in wide deep neural networks. *arXiv preprint arXiv:1804.11271*, 2018.

[4] Roman Novak, Lechao Xiao, Jaehoon Lee, Yasaman Bahri, Daniel A Abolafia, Jeffrey Pennington, and Jascha Sohl-Dickstein. Bayesian convolutional neural networks with many channels are gaussian processes. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019.

[5] Youngmin Cho and Lawrence K Saul. Kernel methods for deep learning. In *Advances in neural information processing systems*, pages 342–350, 2009.

[6] Samuel L. Smith and Quoc V. Le. A bayesian perspective on generalization and stochastic gradient descent. *CoRR*, abs/1710.06451, 2017.

[7] Pascal Germain, Francis Bach, Alexandre Lacoste, and Simon Lacoste-Julien. Pac-bayesian theory meets bayesian inference. In *Advances in Neural Information Processing Systems*, pages 1884–1892, 2016.

[8] Guillermo Valle-Perez, Chico Q Camargo, and Ard A Louis. Deep learning generalizes because the parameter-function map is biased towards simple functions. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019.

[9] GPy. GPy: A gaussian process framework in python. `http://github.com/SheffieldML/GPy`, since 2012.

[10] Carl Edward Rasmussen. Gaussian processes in machine learning. In *Advanced lectures on machine learning*, pages 63–71. Springer, 2004.

[11] Kamaludin Dingle, Chico Q Camargo, and Ard A Louis. Input–output maps are strongly biased towards simple outputs. *Nature communications*, 9(1):761, 2018.

[12] Abraham Lempel and Jacob Ziv. On the complexity of finite sequences. *IEEE Transactions on information theory*, 22(1):75–81, 1976.

[13] Leonardo Franco and Martin Anthony. On a generalization complexity measure for boolean functions. In *Neural Networks, 2004. Proceedings. 2004 IEEE International Joint Conference on*, volume 2, pages 973–978. IEEE, 2004.

[14] Leonardo Franco. Generalization ability of boolean functions implemented in feedforward neural networks. *Neurocomputing*, 70(1):351–361, 2006.

[15] Ehud Friedgut. Boolean functions with low average sensitivity depend on few coordinates. *Combinatorica*, 18(1):27–35, 1998.

[16] Sam F Greenbury, Steffen Schaper, Sebastian E Ahnert, and Ard A Louis. Genetic correlations greatly increase mutational robustness and can both reduce and enhance evolvability. *PLoS computational biology*, 12(3):e1004773, 2016.

[17] David Krueger, Nicolas Ballas, Stanislaw Jastrzebski, Devansh Arpit, Maxinder S. Kanwal, Tegan Maharaj, Emmanuel Bengio, Asja Fischer, Aaron Courville, Simon Lacoste-Julien, and Yoshua Bengio. A closer look at memorization in deep networks. Proceedings of the 34th International Conference on Machine Learning (ICML'17), 2017.