Measuring the Complexity of Neural Networks with VC Entropy

Lingqi Zhang (lingqiz@sas.upenn.edu)

Introduction

Recent breakthroughs in machine learning, in particular deep neural network, are drastically changing the field of artificial intelligence [1]. However, despite rapid progress, our understanding of fundamental principles in training large models is still limited [2]. Bias-variance tradeoff is one of the most important guidelines in machine learning, which in its essence states that we need to choose the model class with the appropriate degree of complexity and proper inductive bias to ensure *both* good training and generalization performance [3].

In fact, previous research has shown that many popular empirical practices in deep learning is related to introducing limitations on model complexity: Simple norm-based regularization is proven to be effective even in large models [2]; In optimization, both gradient descent and early stopping put implicit constraints on the weights and learned matrices [4, 5] ; Similarly, dropout can prevent overfitting through explicitly representing model uncertainties [6, 7]. What is lacking, however, is a common framework to unify these effects.

The aim of our current project is to apply one of the formal definitions of model complexity we have learned in the class, namely Vapnik–Chervonenkis dimension [8], to neural network models. More precisely, we numerically approximate the growth function using the closely related notion of VC Entropy [9]. We found that the growth function can appropriately measure the differences in model complexity due to changes in network architecture, network depth versus width, and training techniques (i.e., dropout), but (in our simulations) failed to reveal the effect of regularization.

Methods

Formally, to compute VC dimension we need to calculate the growth function:

$$\pi_H(m) = \max_{S:|S|=m} \{|\pi_H(s)|\}$$

$$\text{where} \quad \pi_H(S) = \{< h(x_1), h(x_2), ..., h(x_m) >\} : h \in H.$$

For our simulation, we consider input x as 0 and 1 images in the MNIST database; H represents class of neural networks with a particular architecture for binary classification problem, and h are instances of that class given a set of network weights (parameters).

The $\max$ operation however, requires an exhaustive search procedure which is computationally intractable. So instead, we use the modified notion of VC Entropy [9], which replaces $\max$ with an expectation over input data x given its distribution $P(x)$:

$$E_H(m) = E_{S \sim P}[|\pi_H(S)|]$$

Concretely, for each set size m, we first randomly sample m number of input image x from our dataset of about $20,000$ images. Then for each set of images, we count the number of all possible labeling by H, using randomly sampled h (i.e., randomly sample the network weights) until convergence. In practice, this results in about $100,000$ samples for the largest set size simulated of 12 (since labeling size grows exponentially with set size, we are unable to simulate larger set due to limited computational resources). The whole procedure is then repeated 10 times to compute the average expected value.

Results

Input Dimension

To establish the basic validity of our approach, we first applied the procedure above to neural networks with different input dimension that are otherwise the same. Namely, we first performed a principal component analysis (PCA) on the images (which originally have a dimension of 784), before passing them on to the network. Intuitively, a lower dimensional input should limit the capacity of the network. Below we plot the growth function $E_H(m)$ given different input dimension (**Figure 1**). As expected, a lower dimensional input results in fewer average unique labeling induced by H, a fully-connected network with one hidden layer of 10 nodes.

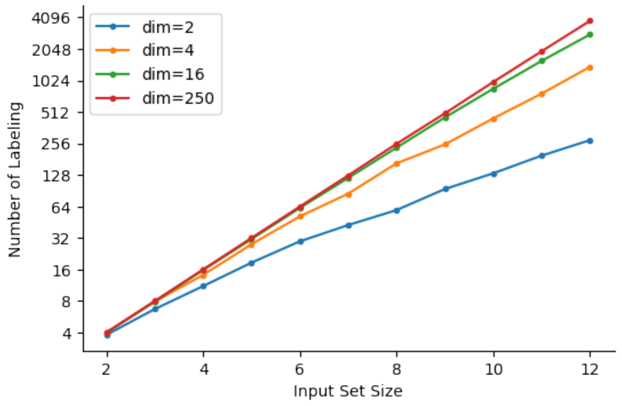

Figure 1: *Average number of unique labeling as a function of input set size, given different input dimension (line color). The x-axis is on a linear scale, while the y-axis is on a log scale.*

Network Architecture

We next investigate the effect of network architecture (i.e., depth). We created a series of networks with increasing number of hidden layers (with 25 nodes in each layer), and applied our procedure. We found that more hidden layers indeed correspond to network with higher complexity (**Figure 2**). Furthermore, we observed that adding the first non-linear hidden layer drastically increase the network's capacity, while more additional layers have a diminished return.

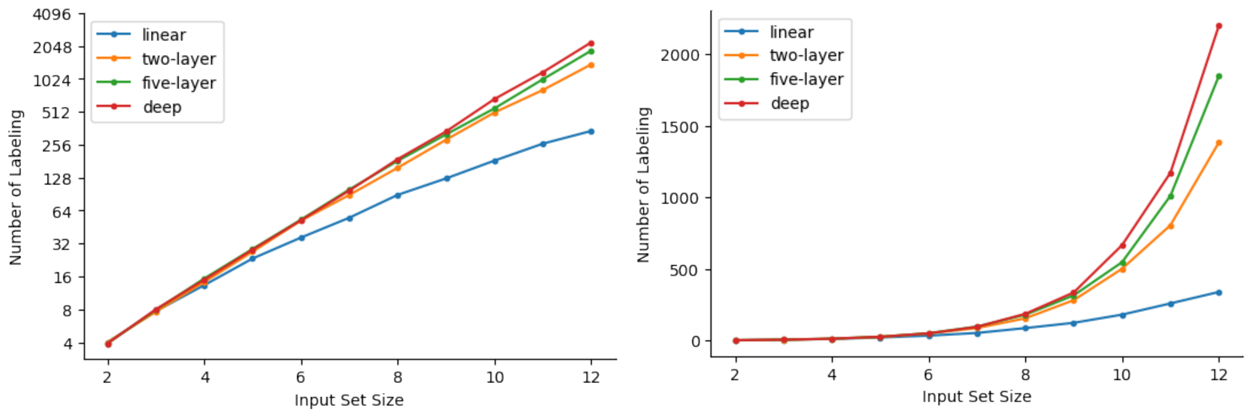

Figure 2: *Average number of unique labeling as a function of input set size, given different network architecture (line color). The y-axis is on a log scale on the left, and linear scale on the right.*

Depth vs. Width

It has been hypothesised that the success of deep neural networks may due to the fact that they are more "effective" in approximating complex functions for real-world problems, in comparison to a shallower, wide network [10, 11]. Although this is still controversial [12, 13], here we are interested in empirically measure the effect of depth versus width on the growth function of the network. Concretely, we compared two networks with matching number of total nodes. One network has one hidden layer that grows only in width as the number of nodes increases, whereas the other network grows only in depth. We computed the number of average unique labeling for input set size of 12, and how it changes as a function of network size (**Figure 3**). Interestingly, we found virtually no effect of increase in width, while increase in depth seems to steadily expand the capacity of the network. Thus, at least in our current limited setting, we did find an advantage of network depth over network width.

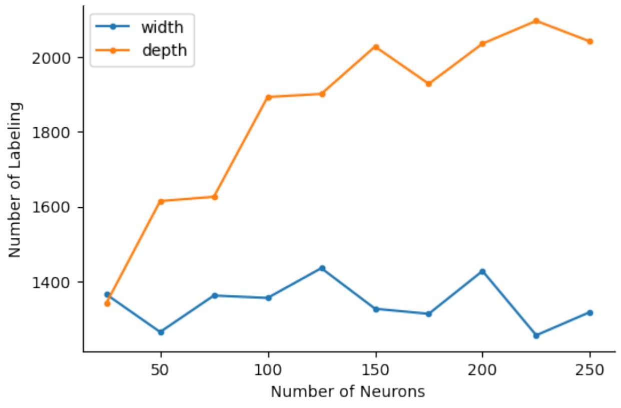

Figure 3: *Average number of unique labeling for a input set size of* 12, *as a function of network size. One network grows in width (blue), while the other network grows in depth (orange) with increasing number of neurons.*

Effect of Dropout

Dropout is a commonly used technique in training neural networks that often leads to better generalization [6], and has been used to introduce probabilistic representations to neural network [14, 15]. We compared two networks with otherwise equivalent architecture, but one with an additional dropout layer. We found that introducing dropout layer actually increases the capacity of the network (**Figure 4**). We verified that this is not due to the difference in the sampling procedure, since

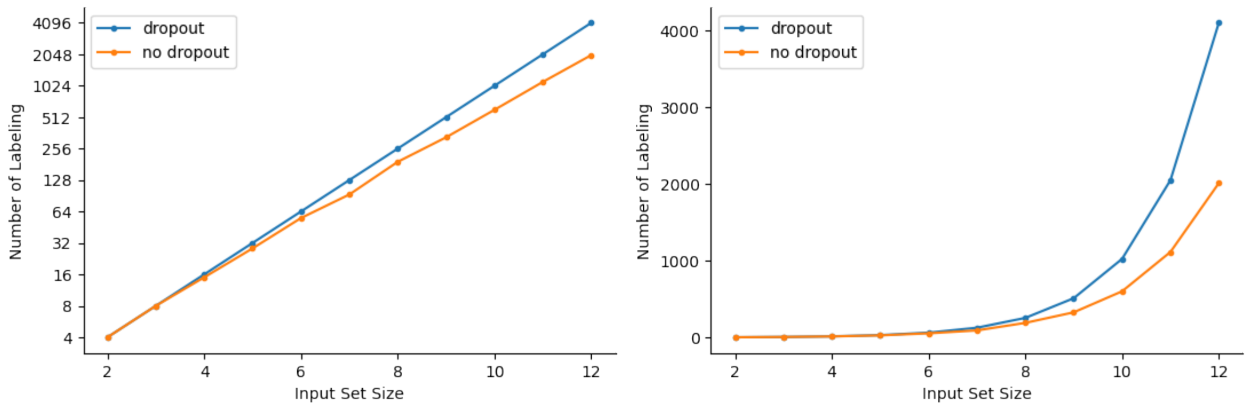

Figure 4: *Average number of unique labeling as a function of input set size, for networks with/without dropout. The y-axis is on a log scale on the left, and linear scale on the right.*

neither increase or decrease the number of samples by 10-fold change the results. Intuitively, dropout does explore a larger model space by randomly modifying the connections of the network on each trial. However, it is not clear how our results is related to the benefits of dropout in reducing model overfitting and improving generalization [6].

Effect of Regularization
Lastly, we aim to measure the effect of commonly used regularization techniques on model complexity. Regularization puts constraints on the weights of the network, such as limits on the vector norm or sparsity of the parameter vector. To simulate these constraints, we modified our procedure such that the weights are sampled from a bounded uniform distribution, or normal distribution with a fixed variance. We also simulated an additional heuristic that is popular in practice (i.e., Xavier initialization, see [16]). To our surprise, we did not find any effect of the weight distribution on the growth function of the neural networks (**Figure 5**).

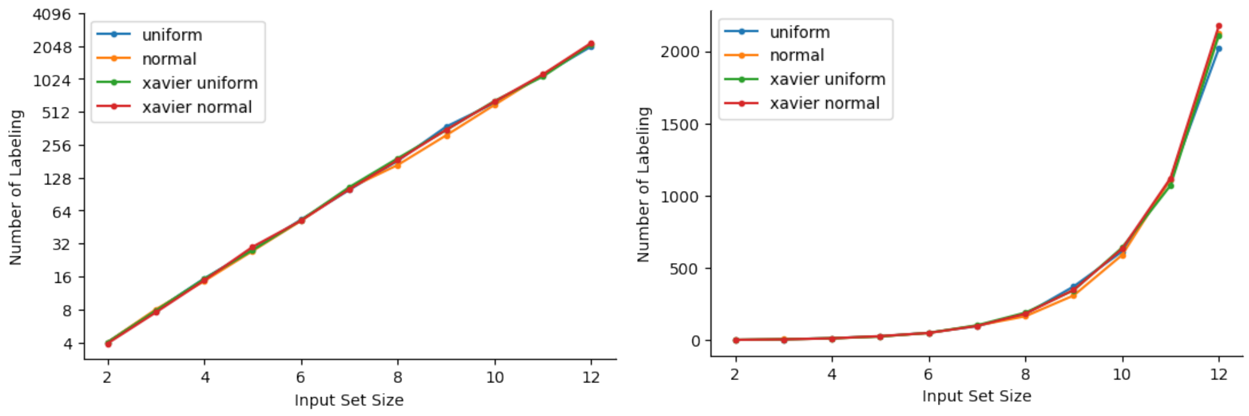

Figure 5: *Average number of labeling as a function of input set size, for networks with different constraints on the weights. The y-axis is on a log scale on the left, and linear scale on the right.*

To further understand our observation, we next focused on normally distributed weight initialization. The inverse variance of the distribution is equivalent to the L_2 norm of the weight vector [17], so a smaller variance should correspond to a stronger regularization. Thus, we computed the number of average unique labeling for input set size of 12, and how it changes as a function of the variance of the weight distribution (**Figure 6**). Rather than a gradual change in model complexity, we observed a bifurcation: Once the variance exceeds a certain threshold, the network capacity is similar; Otherwise, the variance is too small for the network to have any representational power.

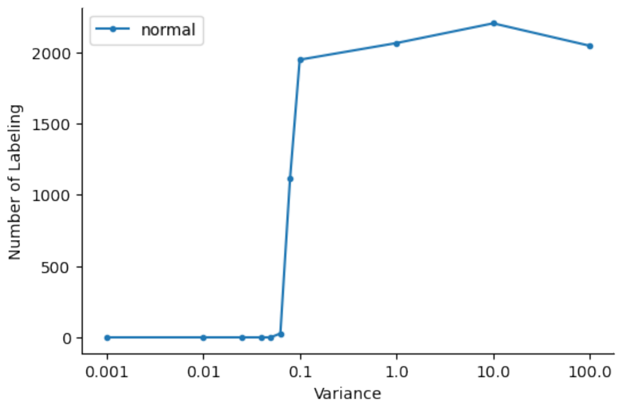

Figure 6: *Average number of unique labeling for set size* 12*, as a function of the variance of the Gaussian distributed initialization. Note that the x-axis is on a log (base 10) scale.*

Summary

We applied VC entropy and growth function, one of the formal definitions of model complexity, to neural network models with realistic input. We found that the growth function can reliably reveal the differences in model complexity due to changes in size and architecture of the network. Our preliminary results have also found an advantage of depth over width, and the effect of dropout, in increasing the capacity of neural networks.

The main drawback of the method is its computational complexity. Since growth function is exponential in input size, the algorithm runs in an exponential time of $O(2^m)$ in the best-case scenario. In addition, to make sure sufficient samples of h from model space H, a large number of samples is required. For this reason, in our current simulation, the largest set size we have tested is 12. This can potentially explain the fact that we did not find a meaningful effect of regularization, or weight distribution on model complexity. It is likely that we need to simulate a larger set size for us to be able to observer the effect of regularization.

References

1. LeCun, Y., Bengio, Y. & Hinton, G. Deep learning. *nature* **521,** 436–444 (2015).
2. Belkin, M., Hsu, D., Ma, S. & Mandal, S. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences* **116,** 15849–15854 (2019).
3. Kohavi, R., Wolpert, D. H., *et al. Bias plus variance decomposition for zero-one loss functions* in *ICML* **96** (1996), 275–83.
4. Arora, S., Cohen, N., Hu, W. & Luo, Y. Implicit regularization in deep matrix factorization. *arXiv preprint arXiv:1905.13655* (2019).
5. Prechelt, L. in *Neural Networks: Tricks of the trade* 55–69 (Springer, 1998).
6. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. & Salakhutdinov, R. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research* **15,** 1929–1958 (2014).
7. Gal, Y. & Ghahramani, Z. *Dropout as a bayesian approximation: Representing model uncertainty in deep learning* in *international conference on machine learning* (2016), 1050–1059.
8. Kearns, M. J., Vazirani, U. V. & Vazirani, U. *An introduction to computational learning theory* (MIT press, 1994).
9. Vapnik, V. N. An overview of statistical learning theory. *IEEE transactions on neural networks* **10,** 988–999 (1999).
10. Goodfellow, I., Bengio, Y. & Courville, A. *Deep Learning* (MIT Press, 2016).
11. Urban, G. *et al.* Do deep convolutional nets really need to be deep and convolutional? *arXiv preprint arXiv:1603.05691* (2016).
12. Ba, L. J. & Caruana, R. Do deep nets really need to be deep? *arXiv preprint arXiv:1312.6184* (2013).

13. Zagoruyko, S. & Komodakis, N. Wide residual networks. *arXiv preprint arXiv:1605.07146* (2016).

14. Gal, Y. & Ghahramani, Z. A theoretically grounded application of dropout in recurrent neural networks. *arXiv preprint arXiv:1512.05287* (2015).

15. Gal, Y., Hron, J. & Kendall, A. Concrete dropout. *arXiv preprint arXiv:1705.07832* (2017).

16. Glorot, X. & Bengio, Y. *Understanding the difficulty of training deep feedforward neural networks* in *Proceedings of the thirteenth international conference on artificial intelligence and statistics* (2010), 249–256.

17. Gelman, A. *et al. Bayesian data analysis* (CRC press, 2013).