
Abstract

This paper proposes an empirical black-box testing framework that uses satisfiable Boolean formulae to measure the capacity of a fully connected neural network with respect to classification tasks. We specifically focus on the case of perfectly solvable binary classification tasks with Boolean features, which allows us to reduce such problems to instances of solvable 3-SAT problems. By varying the complexity of the formulae, our empirical testing framework can be used to measure the capacity of a neural network. This empirically measured capacity surface can potentially aid in comparing different neural networks and choosing appropriate networks for binary classification tasks. We propose such an empirical methodology as an alternative to purely theoretical metrics in order to derive a computable measure that can evaluate the true learning capacity of a neural network model for binary classification tasks.

1 INTRODUCTION

Model selection is an important problem in machine learning. It is of great interest to choose an appropriate model for a given task such that it performs with acceptable accuracy while minimizing both sample complexity and runtime complexity [Zhang et al., 2018]. There is rich literature on such a tradeoff for machine learning models like support vector machines [Cortes and Vapnik, 1995], linear regression, and other types of classifiers [Holden and Rayner, 1995], but there is still no widely accepted standard for model selection in feed-forward neural networks (FNN). Model parameters are chosen largely using human intuition and knowledge of the domain [Krizhevsky et al., 2012]. While this methodology has proven to be quite successful in tasks like image recognition, and machine translation [Cho et al., 2014], we still think it is of interest to develop further intuition on the exact selection of hyperparameters that yield a minimal-complexity FNN with sufficient generalization error.

At a high level, in order to select a sufficient fully connected network (FNN) for a given task, we must compare it with other FNNs. There are two primary factors that we care about when choosing a model. We want to balance model performance with model complexity. These factors will be the basis for our framework which takes as input a family of models, and a particular supervised learning problem, and returns the best model for the task. Developing these metrics for every type of supervised learning problem is intractable, so we focus on a particular case of supervised learning that yields itself to easier analysis. We focus primarily on binary classification tasks. Furthermore, we assume that the features are binarized, rather than in $\mathbb{R}$ and that they are all marginally independent. Finally, in this paper, we assume that there exists a separating hyperplane that is consistent with all of the training data.

We use this setup to propose an empirical black-box testing methodology to capture a realistic estimate of a model’s ability to learn a given binary classification task. This empirical benchmark should generalize across the space of neural networks, and be able to inform the architecture of a model, given a specific task. Our selection of the problem space yields functions that are equivalent to Boolean satisfiability (SAT) problems. By varying the complexity of the Boolean formulae along with providing sufficient data to the neural network for each formula, we can empirically derive a learnability capacity surface that captures the neural network performance across the space of satisfiable Boolean formulae. Empirically measuring the capacity of neural networks allows us to compare different networks, or pick an appropriate network for a new binary classification task with Boolean features.

1.1 Related work

There have been many attempts to better understand how to pick the most optimal network for a given task. The first of these approaches involves com-
puting some sort of capacity metric for a given neural network [N. Vapnik and Ya. Chervonenkis, 1971, Collins et al., 2017]. However, these methods focus on model complexity, rather than considering the complexity of the data.

More recently, there has been work on compressibility [Zhou et al., 2018], and hyperparameter optimization [Snoek et al., 2012, Liu et al., 2017]. However, these approaches consider the model, the optimization procedure, and the data jointly, making it difficult to disentangle the effects of each one on the other.

There has also been some work on using algebraic topology to measure the complexity of a given dataset [Guss and Salakhutdinov, 2018]. Our method is similar in that we consider the complexity of the dataset and the capacity of a classifier separately, but we formulate our measure of complexity using SAT problems.

The field of Boolean logic synthesis [Devadas et al., 1994] studies the general problem of transforming an abstract form of desired circuit behavior into a design implementation in terms of logic gates. Existing hardware specification languages like VHDL [IEEE, 1988] and Verilog [Nielsen et al., 2016] provide Boolean minimization software that reduces the size of the circuit for a given truth table representation. While this body of work can in principle be viewed similar to the problem of Boolean functional learning, a fundamental difference in our case is that the size of the training data is very small compared to the state space, unlike logic synthesis works which require a cleaner specification of the logic over the entire space. In essence, the function that the neural network is aiming to learn for a binary classification task is only partially specified for a small portion of the state space.

### 1.2 Main contributions

First, we formalize task complexity with respect to the family of supervised learning tasks we consider. This complexity is a model-free way of determining how “hard” a problem is to solve. We draw on literature from formal logic to theoretically estimate the hardness of learning a Boolean satisfiability problem via the probability of satisfiability. We then use this complexity to formalize model capacity. We define a capacity surface as a measure of a fully connected neural network’s across a family of classification tasks with varying complexity. We empirically test this structure for comparing classifiers on a range of tasks using a black-box testing framework. We also show that such a framework is useful in identifying the appropriate classifier for a given task. The next section shows how we derive our framework using properties of boolean satisfiability problems.

### 2 DEFINING TASKS OF VARYING COMPLEXITY

In this paper, we aim to design a methodology to empirically measure the effectiveness of any given fully connected network (FNN) for binary classification tasks. Using this methodology, given a supervised learning binary classification task, we aim to compare the effectiveness of different FNNs and determine the “best” FNN for the task, while minimizing computational complexity. Our methodology creates binary classification tasks of varying complexity and aims to determine the learning capacity of a given FNN as a function of the task complexity.

Our methodology makes two simplifying assumptions. First, we restrict our methodology to the set of supervised learning tasks where the input features are constrained to be binary valued and pair-wise independent. Second, similar to some other measures of complexity [Holden and Rayner, 1995], we focus on the specific subspace of classification tasks where the problem is perfectly satisfiable. In other words, there exists a solution to each task which is consistent with every training sample.

Let \( h \) denote a neural network which takes as input, a \( d \)-dimensional binary vector \( X = (x_1, \ldots x_d) \) whose entries are marginally independent. Let \( y \) denote the binary classification output of the neural network \( h(X) \). Equations (1) - (4) outline these parameters.

\[
\begin{align*}
y &= h(X) \\
y &\in \{0,1\} \\
X &\in \{0,1\}^d \\
\forall j, k \in [1,d], x_j &\perp \perp x_k \text{ if } j \neq k
\end{align*}
\]

Consider input training data \( D = \{(X_1,y_1)\ldots(X_m,y_m)\} \) for a binary classification task of size \( m \) where \( X_1, \ldots X_m \) are the \( d \)-dimensional input vectors and \( (y_1, \ldots y_m) \) denote their corresponding binary classification values. The goal of training a neural network \( h \) is to use the input training data to learn a separating hyper-plane that effectively separates the two classes in a binary classification task. If we ignore the model and simply observe the structure of the dataset, we observe an equivalence to a truth table which is partially specified for a small portion of the input vector space. This problem closely resembles a Boolean satisfiability (SAT) prob-
lem [Cook, 1971]. If the goal is to learn a hyper-plane that works effectively for the given input data $D$, then one can easily construct a simple Boolean satisfiable formula that exactly matches the partial truth table specified in $D$ by considering the logic corresponding to the class 1 values in the truth table and setting every unspecified value in the truth table to the class 0. Clearly, such a formula will perform poorly over any new test data.

In the ideal scenario, assume that the input training data $D$ was generated from a Boolean formula $f$ on $d$ binary variables where $f$ is not known to the neural network $h$. The ideal learning goal for the neural network is: given any future test input generated by the formula $f$, can the neural network accurately predict the output class corresponding to the input. While $f$ is not guaranteed to be unique for a given training data $D$, we are specifically interested in cases where the training data is generated by a simple formula $f$ that is relatively easy to compute.

To estimate the learning capacity of a fully connected neural network for binary classification tasks, we outline a methodology for generating easily computable formulae of increasing complexity. Given a specific formula, the goal is to leverage the formula to generate sufficient training and testing data and measure the efficacy of a neural network to learn the formula. Given several formulae of the same complexity, our goal is to define a composite metric that captures the relationship between the complexity of the formula and the learning efficacy of the fully connected neural network.

### 2.1 Random k-SAT functions

To generate formulae of varying complexity, we use random $k$-SAT functions. We define the “hardness” of a SAT problem $f(x_1, x_2, \ldots)$ as the likelihood of guessing a satisfying assignment to its variables $x_1, x_2, \ldots$. For example, the logical OR of each of the variables should be a very simple problem to solve as all but 1 assignment to its variables is satisfying. We formalize this notion by terming this, the probability of guessing a correct solution. Now, given a SAT formula, we can gauge how difficult it will be to learn the formula given its satisfying and unsatisfying assignments. The most straightforward way to computing the likelihood of guessing a correct solution is enumerating the input space and counting the percentage of positive outputs. However, this may prove intractable for large $d$ so we turn to random k-SAT theory [Coja-Oghlan, 2010] to help us estimate this likelihood.

We begin with a result from [Cook, 1971] that shows that every SAT problem can be written as a k-SAT problem, as long as $k \geq 3$. Therefore, given a truth table, it is always possible to generate a k-SAT formula if there exists a solution. A random k-SAT function is a k-SAT function where the variable literals in each clause are randomly sampled. A k-SAT function is parameterized by the number of clauses $(m)$, the number of variables $(n)$, and the number of variables per clause $(k)$. Using parameterization, we can show that there is a way to estimate the likelihood of guessing a correct solution, or the probability of satisfiability.

Without loss of generality, given the universal representation of 3-SAT, we set $k = 3$ in this paper for our analysis. A 3-SAT clause is a disjunction of 3 Boolean variables or their negations [Emmanuel and Andrea, 2014]. Let $C(n)$ be the set of all possible clauses in a 3-SAT problem. There are $(\binom{3}{3})$ ways to choose 3 variables, and $2^3$ ways to negate the variables in each clause. Therefore, there are $2^3(\binom{3}{3})$ possible clauses on $n$ Boolean variables. We can also refactor $m = \alpha n$, where $\alpha$ is the ratio of the number of clauses to the number of variables. Let $f(n, \alpha)$ indicate the conjunction of every element in $D(n)$ where $D(n)$ is created by selecting every element in $C(n)$ with probability $\frac{\alpha n}{|C(n)|}$. In expectation, the number of clauses in $f(n, \alpha)$ is $\alpha n$. We can now come up with a complexity measure for our $f$s using the probability of satisfiability.

The probability of satisfiability ($\rho$) of solving a 3-SAT instance depends on the ratio of satisfying assignments in $f$ to the total number of assignments. It has been shown that varying $\alpha$ affects $\rho$ [Emmanuel and Andrea, 2014]. We show here a rough upper bound for $\rho$. A simple and crude estimation of the probability of satisfiability is given by:

$$\left(1 - \frac{1}{2^3}\right)^{\alpha n}$$

We use this crude approximation to determine a family of random k-SAT formulae of variable complexity that is parameterized easily by $n$ and $\alpha$.

### 2.2 Generating satisfiable 3-SAT functions

Now that we have a formal notion of complexity with respect to a k-SAT function, it is trivial to compute the complexity of a 3-SAT function, given its parameters $n$ and $\alpha$. We can now focus on how to create tasks of varying complexity for our FNNs. We want to generate training examples such that the logic is approximated by the neural network. In order to generate only satisfiable functions, we can use the
satisfiability threshold to inform our generation procedure. We begin by first determining the threshold for a given \( f(n, \alpha) \). Let \( Z_n \) be the number of satisfying assignments in \( f(n, \alpha) \). Then:

\[
E[Z_n] \leq 2^n \left(1 - \frac{1}{8}\right)^{\alpha n} \tag{6}
\]

\[
E[Z_n] = 1 \tag{7}
\]

\[
\alpha \leq -\log_2 \frac{2}{\log_2 8} \tag{8}
\]

In order to generate functions with significantly different probabilities of satisfiability, we use \( \alpha \) as that never exceed this value. We also lower bound \( \alpha \), as very low values for \( \alpha \) cause the probability of satisfiability for all \( f(n, \alpha) \) to tend to 1.

In any supervised learning procedure, we split the dataset into a training and testing portion to first optimize the parameters of our model, then test its generalization ability on the unseen testing portion. In this experiment, we split our “truth table” training data (generated using a randomly sampled 3-SAT function) into training and testing sets. If a formula is learned using only the training dataset, but generalizes well to samples not included in this set, we say that it has learned the formula.

Furthermore, the network of minimal computational complexity that is able to generalize well to a left-out portion of the truth table for a 3-SAT formula can be viewed as the simplest FNN that is able to effectively learn a boolean function. We can now perform model selection on our family of FNNs to identify the minimum computational complexity FNN for the task.

3 CAPACITY SURFACES

Now that we have a complexity ordering on the space of our supervised learning tasks, we will detail the basis of comparison between different FNNs. We first define the capacity of a particular neural network as its ability to generalize to the test set, i.e. achieve sufficient accuracy or loss on the test set. The test set is a randomly left-out portion of the training set. More intuitively, if a network is able to achieve low test error on a complex task, its capacity is higher than those that cannot achieve as low an error rate. Given a task, it is not difficult to compare the difference between two classifiers. However, if we want to characterize the performance of our classifiers in a more general way, we need a different measure for comparing them. This measure must detail the complexity of functions a particular FNN can and cannot learn sufficiently. We therefore define a capacity surface \( CS(h) \) which represents the performance of classifier \( h \) across a broad range of tasks. In this case, since we are considering the family of 3-SAT functions, we can further specify \( CS(h) \) as \( CS(h, F) \) where \( F \) is the space of all 3-SAT functions ordered by complexity. It is natural for \( CS \) to take on the same ordering.

The capacity surfaces we consider are defined on the parameter space of \( f(n, \alpha) \in F \). Each point on this surface represents the performance of \( h \) for a given \( f(n, \alpha) \). We further define this surface by formalizing our definition of performance. Let \( M = \langle m_1, m_2, \ldots \rangle \) be the space of performance metrics where each \( m_i \in M \) is a measure of loss or accuracy for a given FNN and a given task. For example, in this paper, we use \( F_1 \) score as one of our metrics. We then define a capacity vector \( CV \) as:

\[
CV(h, f(n, \alpha)) = \langle m_1(h), m_2(h), \cdots \rangle \tag{9}
\]

Where \( m_i(h) \) represents the performance of \( h \) on the test set with respect to metric \( m_i \). We can then formally define a capacity surface \( CS \) as an ordered set of capacity vectors:

\[
CS(h) = \{ CV(h, f(n, \alpha)) : f(n, \alpha) \in F \} \tag{10}
\]

In order to identify the formula of maximum complexity a given classifier \( h \) can sufficiently learn, we can simply take a maximum over the capacity surface as shown in (11). This is the largest value for \( n \) and \( \alpha \) such that the values for the evaluation metrics are above some threshold \( t \). Note that \( t \) is a vector so we can potentially have separate thresholds for each metric.

\[
n^*_h, \alpha^*_h = \arg \max_{n, \alpha : CS(h) \geq t} CV(h, f(n, \alpha)) \tag{11}
\]

4 BLACK-BOX TESTING FRAMEWORK

Given our method of generating tasks using 3-SAT formulas, we can build an empirical testing framework for a variety of FNNs. We begin by first generating a capacity surface for a particular FNN. We then generate these for a variety of FNNs, and illustrate how to compare the surfaces across a variety of networks. Finally, we use these capacity surfaces to perform a simple model selection procedure to identify the FNN with the minimal number of parameters but sufficient generalization performance.
4.1 Empirical capacity surfaces

A capacity surface (CS) for a classifier $h$ consists of numerous capacity vectors (CVs). We begin by empirically generating each CV. For each CV, we require a set of performance metrics $M$. In this paper, we consider the $F_1$ score (12), negative log-likelihood (NLL) (13), and the empirical generalization error (15) on a hold-out test set. We calculate the empirical generalization error by subtracting the training error from the test error. Given our definition of CS with respect to performance metrics, we must only use metrics which take on higher values to indicate better performance. For this reason, we consider the inverses of both the NLL and the generalization error.

$$F_1(h) = \frac{2Y^TH(X)}{\sum_{i=1}^{m} Y^{(i)} + \sum_{i=1}^{m} h(X^{(i)})}$$  \hspace{1cm} (12)

$$\text{NLL}(h) = -\frac{1}{m} \sum_{i=1}^{m} Y^{(i)} \log h(Y^{(i)}) + (1 - Y^{(i)}) \log(1 - h(Y^{(i)}))$$  \hspace{1cm} (13)

$$E_{ge}(h) = E_{te}(h) - E_{tr}(h)$$  \hspace{1cm} (15)

Using this procedure for generating each CV, we repeat our steps over a range of 3-SAT functions $f(n, \alpha)$ to generate a CS for a given classifier $h$.

4.2 Comparing classifiers

Now that we have outlined a methodology for generating a capacity surface for a single classifier, we generate this surface for a variety of classifiers. As mentioned earlier, the goal is to identify the classifier that not only performs adequately on the classification task, but is of minimal computational complexity. This requires some ordering of the classifiers. In our case, we simply look at the hyperparameters of our FNNs. More specifically, we look at the number of hidden units and hidden layers. To order all the results, we create a $d$ dimensional tensor $\mathcal{R}$, where $d = |M| \times 2$ where $M$ is the set of performance metrics being used. An intuitive way to think of this is a 2D grid, where we have the number of hidden units on the $x$ axis and the number of hidden layers on the $y$ axis. Each classifier $h_i$ is a point on this grid with coordinates $(x_i, y_i)$. Each point is itself a capacity surface $CS_{n, \alpha}$. Now that we have an ordering on each of the dimensions in $\mathcal{R}$, it becomes trivial to solve our constraint optimization of finding the least complicated of the acceptable classifiers.

4.3 Model selection

Given a new task, we now want to find an ideal model to solve the task. We can use our performance tensor $\mathcal{R}$ to first identify the point on each capacity surface that represents that new task. We then filter out models with insufficient performance with respect to each metric in $M$, and return the model with minimal complexity. In our case, we are solving this objective, given a new task $f(n, \alpha)$, and a classifier $h$ with $p_1$ hidden units and $p_2$ hidden layers:

$$\min_{u, l} u \ast l$$  \hspace{1cm} (16)

Where $l$ is the number of hidden layers, and $u$ is the number of hidden units. We use this specific setup as $u \ast l$ is the total number of parameters of a given FNN $h$. This means that the generalization error we use to compute our capacity surfaces can be directly compared to VC dimension. This is because the tightest existing generalization error bound for FNNs using VC dimension is $O(u \ast l)$, the number of parameters [Harvey et al., 2017].

5 EMPIRICAL EXPERIMENTS

For this set of experiments, we look at fully connected networks with ReLU activations [Nair and Hinton, 2010]. We first select a family of 3-SAT functions and generate the required training data for an FNN to learn them. We then generate capacity vectors for each 3-SAT function and for each classifier $h_i$.

5.1 Generating training data

As mentioned earlier, the training data for each FNN is simply a subset of the truth table generated by each 3-SAT function. We look at 3-SAT functions $f(n, \alpha)$ which have a probability of satisfiability within a specific range (17) and the number of variables restricted to a range that was computationally reasonable for enumerating the entire truth table if needed.

$$\rho \in [0.2, 0.8]$$  \hspace{1cm} (17)

$$n \in \{2^i : i \leq 1 \leq 10\}$$  \hspace{1cm} (18)

Given 3-SAT that met these constraints on $n$ and $\alpha$, we employ a SAT solver [Srensson and Een, 2002] to enumerate the satisfying solutions. The resulting variable assignments $X^{(i)}$ are positive examples for the learning problem. We then randomly sample the space
\{0,1\}^n to generate variable assignments. Each sample \(X^{(i)} \in \{0,1\}^n\) is evaluated under \(f(n, \alpha)\) and kept if \(f\) evaluates to FALSE.

We generate \(O(n) << 2^n\) samples to prevent memorization by a classifier \(h\). We also ensure a roughly equal distribution of positive and negative examples for learning \(f(n, \alpha)\) while sampling for unsatisfying variable assignments.

5.2 Fully connected networks

For our family of FNNs, we look specifically at neural networks that have the same number of hidden units per hidden layer. Since we are more focused on characterizing the capacity of a given architecture than optimization techniques, we fix our optimization procedure to gradient descent [Ruder, 2016] with Nesterov momentum [Botev et al., 2017]. We also tune hyper-parameters like learning rate and the momentum parameter as necessary to minimize testing error when generating each capacity vector.

6 RESULTS

We first observe the usefulness of 3-SAT as a benchmark in the setting of architecture selection. We observe the performance of a given classifier \(h\) on one setting of the test-bed. The labels \((i,j)\) in Figure 1 refer to a network of with hidden layers \(h_l = i\) and hidden units \(h_u = j\) per layer.

We notice that for a given \(\alpha n\), we see a consistent decrease in the binary cross-entropy loss with increasing \(n\). This loss also starts higher for networks with fewer parameters. Another way to interpret this graph is in Figure 1 where we see an increase in \(\alpha\) leads to increasing losses, but at a faster rate for smaller networks than bigger ones.

For visualization purposes, we compare four 1-layer networks with the number of hidden units \(h_u \in \{2, 4, 8, 16\}\) (Figures 3, 4). The labels \((i, j)\) refer to a network with hidden layers \(h_l = i\) and hidden units \(h_u = j\) per layer. In this setting, we fix \(f(n, \alpha)\) with \(n = 1024\), and \(\alpha = 0.5\).

For visualization purposes, we compare four 1-layer networks with the number of hidden units \(h_u \in \{2, 4, 8, 16\}\) (Figures 3, 4). The labels \((i, j)\) refer to a network with hidden layers \(h_l = i\) and hidden units \(h_u = j\) per layer. In this setting, we fix \(f(n, \alpha)\) with \(n = 1024\), and \(\alpha = 0.5\).

We notice that using our test-bed, there is a very discernible difference between each of the four networks. In Figures 3 and 4, as we increase the number of hidden units, the distribution of log-losses shifts more to the left. In Figure 4, we notice a pattern consistent with the previous observation. As the number of hidden units increases, the \(F_1\) scores are more concentrated around 1. This is a good diagnostic result which quantifies the differences in performance between simple and more complicated models.
6.1 Classifier comparison using capacity surfaces

We now expand our analysis to multiple networks compared on the entire testing framework. To generate statistically significant results, we sampled the space of random 3-SAT functions 100 times for every setting of \( f(n, \alpha) \) and averaged their capacity surfaces. We noticed that as the values of \( n \) and \( \alpha \) increase, the log-loss increases as well. Additionally, the \( F_1 \) score increases with capacity. This is consistent with our hypothesis regarding the complexity parameters. However, contrary to the VC-dimension hypothesis for neural networks, as the number of parameters increases, the empirical generalization error actually decreases. On an individual level, we notice that increasing the number of hidden units decreases the overall error (as shown in the color scales). Figure 5 is a visualization of part of the \( CS(h) \) for each classifier \( h \).

![Figure 5: Visualization of part of the CS(h) for each classifier](image)

We notice here that as the number of parameters in the network increase, we see both \( n^* \) and \( \alpha^* \) increase. We can now label a classifier \( h^{(h_l, h_u)} \) by its optimal \( SC \) values. Finally, we also observe the generalization gap distribution for a fixed \( f(n, \alpha) \) for each of the classifiers in Figure 6. We notice that \( (h_l, h_u) = (2, 16) \) is the optimal network due to the model performing almost identically on the training and test set. We also see that the test error is actually lower than the training error in the case of larger models.

![Figure 6: Generalization error distribution across various h](image)

We outline three key takeaways. First, our choice of 3-SAT functions clearly highlights the performance difference of different types of neural networks on various classification tasks of varying complexity. Second, we observe that the capacity surfaces provide a richer comparison between classifiers than any one metric as they are defined over a large space of functions. Finally, the surfaces also serve as an easy way to filter out models with insufficient performance, and can aid in identifying the neural network of minimum computational complexity for a given task.

7 CONCLUSION

This paper seeks to formalize the model selection procedure for fully connected networks. We chose a specific family of supervised learning tasks that are closely related to Boolean satisfiability in order to come up with a model-free measure of task complexity. We then define model capacity with respect to this measure of task complexity. We draw on literature from formal logic to theoretically estimate the hardness of learning a Boolean satisfiability problem via the probability of satisfiability. We then define a capacity surface as a measure of a fully connected neural network’s across a family of classification tasks with varying complexity.

Using these notions, we create a testing framework on which we can compare different networks. This empirical methodology captures the capacity of various FNNs with respect to their ability to learn these classification problems. We validate the fact that a larger number of parameters correlates with a larger capacity, but also show that this doesn’t necessarily mean a weaker bound on generalization error.

Finally, we demonstrate that this framework can be used to identify the most appropriate neural network model that balances the tradeoff between performance and computational complexity.

References


Figure 5: $F_1$ across various $f(n, \alpha)$


