Title

The capacity of feedforward neural networks

Authors

Baldi, Pierre

Vershynin, Roman

Publication Date

2019-08-01

DOI

10.1016/j.neunet.2019.04.009

THE CAPACITY OF FEEDFORWARD NEURAL NETWORKS

PIERRE BALDI AND ROMAN VERSHYNIN

Abstract. A long standing open problem in the theory of neural networks is the development of quantitative methods to estimate and compare the capabilities of different architectures. Here we define the capacity of an architecture by the binary logarithm of the number of functions it can compute, as the synaptic weights are varied. The capacity provides an upperbound on the number of bits that can be extracted from the training data and stored in the architecture during learning. We study the capacity of layered, fully-connected, architectures of linear threshold neurons with L layers of size $n_1, n_2, \ldots, n_L$ and show that in essence the capacity is given by a cubic polynomial in the layer sizes: $C(n_1, \ldots, n_L) = \sum_{k=1}^{L-1} \min(n_1, \ldots, n_k) n_k n_{k+1}$, where layers that are smaller than all previous layers act as bottlenecks. In proving the main result, we also develop new techniques (multiplexing, enrichment, and stacking) as well as new bounds on the capacity of finite sets. We use the main result to identify architectures with maximal or minimal capacity under a number of natural constraints. This leads to the notion of structural regularization for deep architectures. While in general, everything else being equal, shallow networks compute more functions than deep networks, the functions computed by deep networks are more regular and "interesting".

Keywords: neural networks; capacity; complexity; deep learning.

Contents

Date: March 29, 2019.

Work in part supported by DARPA grant D17AP00002 and NSF grant 1839429 to P. B., and U.S. Air Force grant FA9550-18-1-0031 to R. V.

1. Introduction

Since their early beginnings (e.g. [17, 21]), neural networks have come a significant way. Today they are at the center of myriads of successful applications, spanning the gamut from games all the way to biomedicine [22, 23, 4]. In spite of these successes, the problem of quantifying the power of a neural architecture, in terms of the space of functions it can implement as its synaptic weights are varied, has remained open. This quantification is fundamental to the science of neural networks. It is also important for applications in order to compare architectures, including the basic comparison between deep and shallow architectures, and to select the most efficient architectures. Furthermore, this quantification is essential for understanding the apparently unreasonable properties of deep learning and the well known paradox that deep learning architectures have a tendency to *not* overfit, even when the number of synaptic weights significantly exceeds the number of training examples [27]. To address these problems, in this work we introduce a notion of capacity for neural architectures and study how this capacity can be computed. We focus primarily on architectures that are feedforward, layered, and fully connected denoted by: $A(a_1, n_2, \ldots, n_L)$, where n_i is the number of neurons in layer i.

1.1. Functional capacity of neural networks.

Ideally, one would like to be able to describe the *functional capacity* of a neural network architecture, i.e. completely characterize the class of functions that it can compute as its synaptic weights are varied. In the purely linear case, such a program can easily be carried out. Indeed, let $p = \min(n_2, \ldots, n_{L-1})$. If $p \geq n_1$, then $A(n_1, \ldots, n_L)$ is simply the class of all linear functions from $\mathbb{R}^{n_1}$ to $\mathbb{R}^{n_L}$, i.e. it is equivalent to $A(n_1, n_L)$. If $p < n_1$, then there is a rank restriction and $A(n_1, \ldots, n_L)$ is the class of all linear functions from $\mathbb{R}^{n_1}$ to $\mathbb{R}^{n_L}$ with rank less or equal to p, i.e. it is equivalent to $A(n_1, p, n_L)$ [5]. In addition, if $n_L \leq p$ then the effect of the bottleneck layer is nullified and $A(n_1, p, n_L)$ is equivalent to $A(n_1, n_L)$, i.e. the effect of the bottleneck restriction is nullified. The exact same result is true in the case of unrestricted Boolean architectures (i.e. architectures with no restrictions on the Boolean functions being used), where the notion of rank is replaced by the fact that a Boolean layer of size p can only take 2^p distinct values.

Unfortunately, in the other and most relevant non-linear settings, such a program has proven to be difficult to carry out, except for some important, but limited, cases. Indeed, for a single threshold gate neuron, $A(n, 1)$ corresponds to the set of linearly separable functions. Variations of this model using sigmoidal or other non-linear transfer functions can be understood similarly. Furthermore, in the case of an $A(n_1, n_2)$ architecture, for a given input, the output of each neuron is independent of the weights, or the outputs, of the other neurons. Thus the functional capacity of $A(n_1, n_2)$ can be described in terms of n_2 independent $A(n_1, 1)$ components. When a single hidden layer is introduced, the main known results are those of universal approximation properties. In the Boolean case, using linear threshold gates, and noting that these gates can easily implement the standard AND, OR, and NOT Boolean operations, it is easy to see using conjunctive or disjunctive normal form that $A(n_1, 2^{n_1}, 1)$ can implement any Boolean function of n_1 variables, and thus $A(n_1, 2^{n_1}, m)$ can implement any Boolean map from $\{0,1\}^{n_1}$ to $\{0,1\}^m$. It is also known that in the case of Boolean unrestricted autoencoders, the corresponding architectures $A(n_1, n_2, n_1)$ implement clustering [3, 2]. In the continuous case, there are various universal approximation theorems [15, 13] showing, for instance, that continuous functions defined over compact sets can be approximated to arbitrary degrees of precision by architectures of the form $A(n_1, \infty, m)$, where we use "∞" to denote the fact that the hidden layer may be arbitrary large. Beyond these results, very little is known about the functional capacity of $A(n_1, \ldots, n_L)$.

1.2. **Cardinal capacity of neural networks.** In order to make progress on the capacity issue, here we define a simpler notion of capacity, the cardinal capacity. The *cardinal capacity* $C(A)$ of a finite class A of functions is simply the logarithms base two of the number of functions contained in A (Figure 1): $C(A) = \log_2 |A|$. The cardinal capacity can thus be viewed as the number of bits required to specify, or communicate, an element of A, in the worst case of a uniform distribution over A. We are particularly interested in computing the cardinal capacity of feedforward architectures $A(n_1, \ldots, n_L)$ of linear threshold functions. In continuous settings, the cardinal capacity can be defined in a similar way in a measure theoretic sense by taking the logarithm of the volume associated with A. In the rest of the paper, in the absence of any qualification, the term capacity is used to mean cardinal capacity.

While in general the notion of cardinal capacity is simpler and less informative than the notion of functional capacity, in the case of neural architectures the cardinal capacity has a very important interpretation. Namely, it provides an upperbound on the number of bits that can be stored in the architecture during learning. Indeed, the learning process can be viewed as a communication process over the learning channel [7], whereby information is transferred from the training data to the synaptic weights. Thus the learning process is a process for selecting and storing an element of A, which corresponds to $C(A)$ bits of information. Any increase in the precision of the synaptic weights that does not change the input-output function is not visible from the outside.

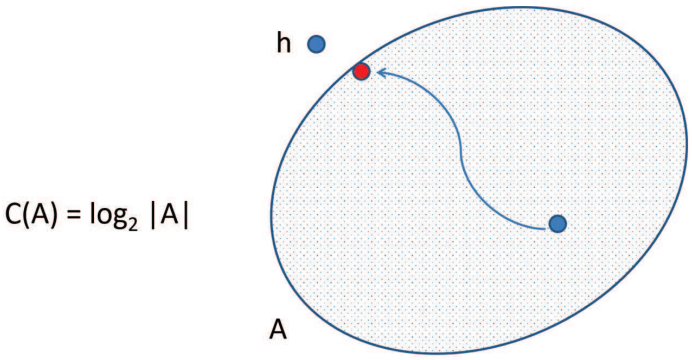

Figure 1. Learning framework where h is the function to be learnt and A is the available class of hypothesis or approximating functions. The cardinal capacity is the logarithm base two of the number, or volume, of the functions contained in A.

The bulk of this paper focuses on estimating the capacity of arbitrary feedforward, layered and fully-connected, architectures of any depth which are widely used in many applications. As a side note, the capacity of fully connected recurrent networks is studied in [6]. In the process, several techniques and theorems of self-standing interest are developed. In addition, the extremal properties of the capacity of such architectures is analyzed, contrasting the capacity of shallow versus deep architectures, and leading to the notion of structural regularization. Structural regularization provides a partial explanation for why deep neural networks have a tendency to avoid overfitting.

1.3. **Main result of the paper: the capacity formula.** The main result of this paper, Theorem 3.1, provides an estimate of the capacity of a general feedforward, layered, fully connected neural network of linear threshold gates. Suppose that such network has L layers with n_k neurons in layer k, where $k = 1$ corresponds to the input layer and n_L correspond

to the output layer. We show that, under some very mild assumptions on the sizes of the layers, the capacity $C(n_1, \ldots, n_L)$ of this network, defined as the binary logarithm of the total number of functions $f : \{0,1\}^{n_1} \to \{0,1\}^{n_L}$ it can compute, satisfies

$$C(n_1, \ldots, n_L) \asymp \sum_{k=1}^{L-1} \min(n_1, \ldots, n_k) n_k n_{k+1}. \tag{1.1}$$

Here the notation $a \asymp b$ means that there exists two positive absolute constants c_1, c_2 such that $c_1 b \leq a \leq c_2 b$. Actually, we will show that the upper bound in the capacity formula (1.1) holds with constant $c_2 = 1$. The absolute constant $c_1 \in (0,1)$ hidden in the lower bound may not depend on anything, in particular it is independent of the depth L of the network, or the widths n_k of the layers. The formula (1.1) thus shows that the capacity of such a network is essentially given by a cubic polynomial in the sizes of the layers, where the bottleneck layers play a special role.

1.4. **Capacity of sets.** In the process of proving the main capacity formula (1.1) we establish some other stand alone results of independent value. At the heart of our analysis are new lower bounds on the *capacity of sets*. We define the capacity $C(S)$ of a set $S \subset \mathbb{R}^n$ as the binary logarithm of the number of all the linear threshold functions $f : S \to \{0,1\}$. In other words, $C(S)$ measures the capacity of a single neuron when the inputs are restricted to S. Equivalently, $C(S)$ is the binary logarithm of the number of all possible ways S can be separated by affine hyperplanes. We prove that for any subset S of the Boolean cube $\{0,1\}^n$, the capacity satisfies

$$\frac{1}{16} \log_2^2 |S| \leq C(S) \leq 1 + n \log_2 \left(\frac{e|S|}{n} \right).$$

The upper bound was previously known, it holds for any subset $S \subset \mathbb{R}^n$, and it can be replaced by the simpler form $n \log_2 |S|$ when $n \geq 4$. The lower bound is a new contribution, and it improves over the previously known (and easy) lower bound of $1 + \log_2 |S|$, which is also true for any set $S \subset \mathbb{R}^n$, as soon as $|S| > 2^{17}$.

In the next section, we provide mathematical definitions of feedforward neural networks and their capacities and describe several known results. A reader familiar with neural network theory may glance through it and rapidly go to Section 3, which provides a description of the new results and provides a roadmap for the paper.

2. Neural architectures and their capacities

2.1. **Threshold functions and maps.** Throughout this paper, the n-dimensional *Boolean cube* is denoted by:

$$H^n = \{0,1\}^n.$$

The *Heaviside function* $h : \mathbb{R} \to \{0,1\}$ is defined by:

$$h(t) = \begin{cases} 1, & t \geq 0 \\ 0, & t < 0. \end{cases}$$

We have chosen the $\{0,1\}$ formalism for convenience. It can easily be replaced with the $\{-1,1\}$ formalism, using the Boolean cube $\{-1,1\}^n$ and replacing the Heaviside function h by the sign function.

Definition 2.1 (Threshold functions)**.** *Consider a set $S \subset \mathbb{R}^n$. A function $f : S \to \{0,1\}$ is called a* (linear) threshold function *on S if there exist $a \in \mathbb{R}^n$ and $\alpha \in \mathbb{R}$ such that f can be expressed as:*

$$f(x) = h\big(\langle a, x\rangle + \alpha\big), \quad x \in S.$$

The set of all threshold functions on S is denoted by $T(S, n, 1)$ and is often abbreviated to $T(S)$.

The notion of threshold functions generalizes naturally to the multivariate setting.

Definition 2.2 (Threshold maps)**.** *Consider a set $S \subset \mathbb{R}^n$. A function $f = (f_1, \ldots, f_m) : S \to H^m$ is called a* threshold map *if all components f_i are threshold functions on S. The set of all threshold maps is denoted $T(S, n, m)$; in the particular case where $S = H^n$, we abbreviate it to $T(n, m)$.*

2.2. **Neural architectures.** A neural architecture (or network) is represented by a weighted directed graph, where the nodes represent neurons and the weights represent synaptic connection strengths. Neurons have numerical states and operate by taking the weighted average of the corresponding parent states and applying a transfer function to this weighted average. This paper focuses on one of the most widely used class of architectures, namely *layered feedforward neural architectures*, where neurons are arranged into layers and connections run from one layer to the next. We will denote an architecture with L layers numbered from 1 to L, and n_k neurons in each layer k, by:

$$A(n_1, n_2, \ldots, n_L).$$

The architecture has n_1 input neurons and n_L output neurons. To simplify the analysis, we make two assumptions:

(a) full connectivity between the layers, i.e. we assume that each neuron in layer k is connected to every neuron in layer $k+1$, but not to any other neurons;
(b) the transfer function is the Heaviside threshold function.

These two assumptions are not absolutely essential, and we discuss how to relax them in the conclusion.

Under these assumptions, the input-output function computed by a layered feedforward neural architecture with a fixed set of weights is a composition of threshold maps. Indeed, the architecture $A(n_1, n_2, \ldots, n_L)$ computes an input-output function of the form

$$f = f_{L-1} \circ \cdots \circ f_2 \circ f_1 \tag{2.1}$$

where each $f_k : \mathbb{R}^{n_k} \to H^{n_{k+1}}$ is a threshold map. Generally, a network architecture $A(n_1, \ldots, n_L)$ is able to compute infinitely many functions $f : \mathbb{R}^{n_1} \to H^{n_L}$, as the synaptic weights and biases (threshold values) are varied. However, if the inputs are restricted to a given finite set $S \subset \mathbb{R}^{n_1}$, then the number functions $f : S \to H^{n_L}$ computable by the architecture becomes finite. We denote this class of functions by

$$T(S, n_1, \ldots, n_L).$$

In the most important case where $S = H^{n_1}$ is the Boolean cube, we drop the set S from the notation. Thus,

$$T(n_1, \ldots, n_L)$$

denotes the class of functions $f : H^{n_1} \to H^{n_L}$ computable by the architecture $A(n_1, \ldots, n_L)$; it consists of all functions that can be expressed as in (2.1) for some set of threshold maps $f_k \in T(n_k, n_{k+1})$ $(k = 1, \ldots, L-1)$.

2.3. **Definition of capacity.** The main question we address in this paper is: how many different functions can a given neural architecture compute? This leads us to the notion of capacity, which we define as follows:

Definition 2.3 (Capacity of a neural architecture)**.** *The capacity of a neural architecture $A(n_1, n_2, \ldots, n_L)$ is the binary logarithm of the number of different functions $f : H^{n_1} \to H^{n_L}$ it can compute, i.e.*

$$C(n_1, \ldots, n_L) = \log_2 |T(n_1, \ldots, n_L)|.$$

More generally, the capacity of a neural architecture on a given finite set $S \subset \mathbb{R}^{n_1}$ is

$$C(S, n_1, \ldots, n_L) = \log_2 |T(S, n_1, \ldots, n_L)|.$$

The capacity of an architecture can be interpreted as the number of bits required to specify a function computable by the architecture. Remarkably, this can also be viewed as an *upper bound on the number of bits that can stored in the architecture during learning*, or equivalently an upper bound on the number of bits that can be extracted from the training data. If the capacity of a network is C and the number of connections (weights) is W, then at most C/W bits can be stored on average per synaptic weight. This bound is independent, and more fundamental, than any hardware limitation on the precision of the synaptic weights. It can be viewed as a bound on the effective capacity of the deep learning channel [7]. The bound holds even if the weights have infinite precision and thus in principle contain an infinite amount of information. This is because in the current framework different sets of weights that implement the same overall input-output function are indistinguishable from the outside world.

Getting optimal bounds on the capacity can be non-trivial even for simple network architectures. Consider, for example, a *single-neuron network* $A(n, 1)$ with n inputs, which thus can implement any threshold function on $\mathbb{R}^n$. The capacity $T(S, n, 1)$ of this network on a given set of inputs $S \subset \mathbb{R}^n$ is the logarithm of the number of all distinct threshold functions that can be defined on S. We call this quantity the capacity of S.

Definition 2.4 (Capacity of a set)**.** *The capacity of a set $S \subset \mathbb{R}^n$ is the binary logarithm of the number of all threshold functions on S, i.e.*

$$C(S) = \log_2 |T(S)|.$$

Equivalently, $C(S)$ is the binary logarithm of the total number of ways the set S can be partitioned by affine hyperplanes in $\mathbb{R}^n$ (taking into account the binary assignment associated with each partition).

A significant part of this paper is devoted to studying the capacity of sets, in particular to deriving optimal estimates for $C(S)$ in terms of the cardinality of S.

2.4. **Basic properties of capacity.** Here we summarize a few elementary properties of the capacity of neural architectures.

Lemma 2.5 (Basic properties of capacity)**.**

1. *(Affine invariance) For any invertible affine transformation $F : \mathbb{R}^{n_1} \to \mathbb{R}^{n_1}$, we have:*
$$C\big(F(S), n_1, n_2, \ldots, n_L\big) = C(S, n_1, n_2, \ldots, n_L).$$
2. *(Monotonicity) If $n_k \leq m_k$ for all k, then:*
$$C(n_1, \ldots, n_L) \leq C(m_1, \ldots, m_L).$$

3. *(Sub-additivity) For any* $1 < k < L-1$, *we have:*
$$C(n_1, \dots, n_L) \leq C(n_1, \dots, n_k) + C(n_{k+1}, \dots, n_L).$$
4. *(Contractivity) Capacity may only increase if a layer is duplicated. For example:*
$$C(n, m, p) \leq C(n, m, m, m, p).$$
5. *For any set* $S \subset \mathbb{R}^n$*:*
$$C(S, n, m) = C(S)m.$$
6. *For any set* $S \subset \mathbb{R}^{n_1}$ *and a threhsold map* $f \in T(S, n_1, n_2)$, *we have:*
$$C\big(f(S), n_2, n_3, \dots, n_L\big) \leq C(S, n_1, n_2, \dots, n_L).$$

The proofs are elementary and left as an exercise.

2.5. **Known bounds on capacity of sets.** First, as a useful reminder, the following theorem about partitions of $\mathbb{R}^n$ by hyperplanes is well known (e.g. [25]) and straightforward to prove by recurrence.

Theorem 2.6. *The number* $K(m, n)$ *of connected regions created by* m *hyperplanes in* $\mathbb{R}^n$ *(passing through the origin) satisfies:*
$$K(m, n) \leq 2\sum_{k=0}^{n-1} \binom{m-1}{k}$$
and the number $L(m, n)$ *of connected regions created by* m *affine hyperplanes in* $\mathbb{R}^n$ *satisfies:*
$$L(m, n) \leq \sum_{k=0}^{n} \binom{m}{k}.$$
In both cases, equality is achieved if the hyperplanes are in general position.

One of the most basic questions addressed in this paper revolves around the best upper and lower bounds on the capacity $C(S)$ in terms of the cardinality of S. The following upper bound is known:

Lemma 2.7 (Capacity of sets: upper bound)**.** *The number of threshold functions on a given set* $S \subset \mathbb{R}^n$ *is bounded by*
$$2\sum_{k=0}^{n} \binom{|S|-1}{k}.$$
In particular, if $n \geq 4$ *then:*
$$C(S) \leq 1 + n \log_2\left(\frac{e|S|}{n}\right) \leq n \log_2 |S|.$$

Proof. The first part of the lemma is presented in [1, Section 4.2] and follows immediately from the first part of Theorem 2.6 by considering the number of regions into which $\mathbb{R}^n$ can be partitioned by the arrangement of $|S|$ hyperplanes of the form $x^\perp$, $x \in S$. The second part of the Lemma can then be deduced from the first using the elementary bound on the binomial sums:
$$\sum_{k=0}^{n} \binom{N}{k} \leq \left(\frac{eN}{n}\right)^n,$$
which is valid for all integers $1 \leq n \leq N$, see e.g. [24, Exercise 0.0.5]. The last part follows easily for $n \geq 4$. □

Remark 2.8 (Tightness). Lemma 2.7 gives the best possible upper bound on the capacity of a set $S \subset H^n$ in terms of the cardinality of S. In Section 9, we describe an enrichment method that for a given $k \leq n$ transforms the cube H^k into a subset $S \subset H^n$ of cardinality $|S| = 2^k$ for which:

$$C(S) \asymp nk = n \log_2 |S|.$$

This shows that the bound in Lemma 2.7 is optimal for almost any magnitude of the cardinality $|S|$.

Lemma 2.9 (Capacity of sets: lower bound)**.** *For any finite set $S \subset \mathbb{R}^n$, there exists at least $2|S|$ threshold functions on S. In particular:*

$$C(S) \geq \log_2 |S| + 1.$$

Proof. The proof is elementary and we only sketch it. The claim is easy to check for $n = 1$. For general n, choose a projection P in $\mathbb{R}^n$ onto some line and such that P is injective on S. Then $C(S) \geq C(P(S))$. By affine invariance, we can realize $P(S)$ as a subset of $\mathbb{R}$ without changing the capacity. Then, applying the statement for $n = 1$, we get $C(P(S)) \geq 2|S|$. Note that if $S = H^n$ the result can also be proved by noting that for any point of the hypercube there is a Boolean threshold function on S that is equal to 1 on that point, and equal to 0 everywhere else. Including all such functions and their negation yields the lower bound. □

Remark 2.10 (Tightness). The bound in Lemma 2.9 is generally tight: if the set S lies on some line in $\mathbb{R}^n$, the there are exactly $2|S|$ threshold functions on S.

Neverthelss, for many sets S the lower bound given in Lemma 2.9 is too weak and can be improved. Consider, for example, the entire Boolean cube $S = H^n$. Lemmas 2.7 and 2.9 give $\log n + 1 \leq C(H^n) \leq n^2$. As the following known result shows, the upper bound is tight, and the capacity of the Boolean cube is approximately n^2:

Theorem 2.11 (Capacity of the Boolean cube)**.** *For any $n > 1$, we have:*

$$\frac{n(n-1)}{2} \leq C(H^n) \leq n^2. \tag{2.2}$$

Moreover:

$$C(H^n) = n^2(1 + o(1)) \quad \text{as } n \to \infty. \tag{2.3}$$

The first, non-asymptotic, part of this theorem can be found in [1, Theorems 4.3, 4.5]; see also [12, 18]. It can also be derived from more general results in this paper: the upper bound on $C(H^n)$ follows from Lemmas 2.7 for $n \geq 4$, and the lower bound on $C(H^n)$ is derived in Example 6.9 below. The second, asymptotic, part of Theorem 2.11 was proved by Zuev [31]. A tighter estimate corresponding to:

$$C(H^n) = n^2 - n \log_2 n \pm O(n) \tag{2.4}$$

was obtained in [16].

Remark 2.12 (Extensions). Theorem 2.11 can be generalized to polynomial threshold functions [8] of degree d, i.e. functions of the form $f(x) = h(p(x))$ where p is a polynomial of degree d. The capacity $C_d(H^n)$, defined as the binary logarithm of the number of such functions on H^n, satisfies:

$$C_d(H^n) = \frac{n^{d+1}}{d!}(1 + o(1)) \quad \text{as } n \to \infty.$$

thus generalizing Zuev's result (2.3) which corresponds to $d = 1$. There exist further extensions of the capacity bounds for ReLU units, units with positive weights, and units with binary weights; they are described in [6].

Armed with these definitions and preliminary results, we are set up to study the capacity of arbitrary feedforward architectures.

2.6. **Asymptotic notation.** In the estimation of various quantities, we will use the notation $\asymp$ and $\lesssim$ for identities and inequalities that hold up to constant factors. To be precise, $a \asymp b$ means that there exists two positive absolute constants c_1 and c_2 such that:

$$c_1 b \le a \le c_2 b.$$

Similarly, $a \lesssim b$ means that there exists a positive absolute constant c such that:

$$a \le cb.$$

These notations are useful only when the quantities a and b vary as a function of certain parameters (e.g. layer sizes). Positive absolute constants, which we denote by $c_1, c_2, c, C, \dots$ may not depend on anything, in particular on the number L of layers or the number of nodes n_k in any layer k.

3. Overview of new results

3.1. **A capacity formula.** The main technical result of the paper is a two-sided bound on the capacity of fully-connected, layered, feedforward architectures $A(n_1, \dots, n_L)$ with threshold transfer functions.

Theorem 3.1 (Capacity formula)**.** *Consider a neural architecture $A(n_1, \dots, n_L)$ with $L \ge 2$ layers. Assume that the number of nodes in each layer satisfies $n_j > 18 \log_2(Ln_k)$ for any pair j, k such that $1 \le j < k \le L$. Then:*

$$C(n_1, \dots, n_L) \asymp \sum_{k=1}^{L-1} \min(n_1, \dots, n_k) n_k n_{k+1}.$$

The upper bound in Theorem 3.1 is not difficult; we derive it in Section 5 from Lemma 2.7 and the sub-additivity of the capacity. The lower bound is significantly more challenging and requires new tools, which we call *multiplexing, enrichment, and stacking*. Once these tools are developed, we use them to prove the lower bound in Section 10.2.

The upper bound in Theorem 3.1 actually holds with the optimal factor 1 if each non-output layer has at least four neurons (Proposition 5.2), and it does not require the assumption $n_j \gtrsim \log_2(Ln_k)$. This mild assumption is important in the lower bound though to prevents layer sizes from expanding too rapidly. Although this assumption has an almost optimal form (Section 10.3), it can be slightly weakened (Section 10.4).

For the special single-neuron case $A(n, 1)$, Theorem 3.1 gives

$$C(n, 1) \asymp n^2.$$

Since $C(n, 1) = C(H^n)$, this recovers the capacity estimate of the Boolean cube from Theorem 2.11 up to a constant factor. The proof of Theorem 2.11, however, does not offer any insights on how to compute the capacity of deeper networks.

The simplest new case of Theorem 3.1 is for networks $A(n, m, 1)$ with one hidden layer, where it states that $C(n, m, 1) \asymp n^2 m + \min(n, m)m \asymp n^2 m$. The constant factor implicit in this bound can be tightened for large n. Indeed, we will show in Corollary 8.4 that:

$$C(n, m, 1) = n^2 m(1 + o(1)) \tag{3.1}$$

if $n \to \infty$ and $\log_2 m = o(n)$. This extends Zuev's asymptotic result (Theorem 2.11).

An immediate and somewhat surprising consequence of Theorem 3.1 is that multiple output neurons can always be "channeled" through a single output neuron without a significant change in capacity of the network:

Corollary 3.2. *Under the assumptions of Theorem 3.1, we have:*

$$C(n_1, \dots, n_{L-1}, 1) \asymp C(n_1, \dots, n_{L-1}).$$

Proof. Comparing the capacity formulas for these two architectures, we see that all the terms in the two sums match except for the last (extra) term in $C(n_1, \dots, n_{L-1}, 1)$, which is $\min(n_1, \dots, n_{L-1})n_{L-1}$. However, this term is clearly bounded by $\min(n_1, \dots, n_{L-2})n_{L-2}n_{L-1}$, which is the last term in the capacity sum for $C(n_1, \dots, n_{L-1})$. Therefore, the capacity sums for the two architectures are within a factor of 2 from each other. □

Let us mention that the capacity formula in Theorem 3.1 obtained for inputs in H^{n_1} can be extended to inputs from other finite sets $C(S, n_1, n_2, \dots, n_L)$. In Propositions 5.2 and 10.5 we give upper and lower bounds on this variation of the capacity in terms of the cardinality of S.

3.2. **Networks achieving maximal capacity.** We can use the capacity formula in Theorem 3.1 to find networks that maximize the capacity subject to natural constraints. Here we find the most capable networks (a) with a given number of connections (weights) and (b) with a given number of nodes (neurons).

Let us start with (a). The number of connections, or synaptic weights, of the neural architecture $A(n_1, \dots, n_L)$ is

$$W = W(n_1, \dots, n_L) = \sum_{k=1}^{L-1} n_k n_{k+1}.$$

Fixing W makes sense because it is approximately the same as fixing the number of *parameters* P of the neural architecture. The difference between P and W are the biases of the neurons so that: $P = W + n_2 + \cdots + n_L$. Thus, we always have $W \leq P \leq 2W$. Furthermore, since the number of neurons is usually much smaller than the number of connections W, in most situations P approximately equals W. We have the following Corollary.

Corollary 3.3 (Optimal network with given number of connections)**.** *Under the conditions of Theorem 3.1, we have:*

$$C(n_1, \dots, n_L) \leq n_1 W.$$

Moreover, any network satisfying $n_1 \leq n_k$ for $k = 2, \dots, L-1$ approximately achieves maximal capacity:

$$C(n_1, \dots, n_L) \asymp n_1 W.$$

Proof. The first statement is known and follows from prior results on the growth function of general (not necessarily fully connected) networks [11, Corollary 3]. It also trivially follows from Theorem 3.1 and the fact that $\min(n_1, \dots, n_k) \leq n_1$. The second statement follows

from Theorem 3.1 and the fact that $\min(n_1, \dots, n_k) = n_1$ under the assumptions of the Corollary. □

Examples of standard architectures that satisfy the condition of the second statement of the Corollary include monotonically expansive feedforward networks satisfying $n_1 \leq n_2 \leq \cdots \leq n_{L-1}$) (the output layer can be expansive or contractive) satisfy the conditions of the Corollary. Likewise, expansive autoencoders networks satisfying $n_1 \leq n_2$ and $n_3 = n_1$ (in the case of a single hidden layer) also satisfy the condition of the Corollary. Finally any shallow network with a single hidden layer, where the hidden layer is larger than the input ($n_2 \geq n_1$), satisfies the condition of the Corollary and thus approximately achieves maximal capacity. In contrast, in many deep forward networks used in applications there exists layers that are smaller in size than the input layer and thus these networks do not achieve maximal capacity. On the positive side, this implies that such networks do not require W independent examples for their training.

Next, let us find the most capable network with a given number of nodes, or neurons. The constraint on the number of neurons is loosely inspired by biological situations where the number of neurons may stay approximately constant, but the number and pattern of connections among the neurons may vary.

It turns out that for a fixed number N of nodes

$$N = n_1 + \cdots + n_L$$

the most capable networks are shallow. To quickly see why, note that Theorem 3.1 yields:

$$C\Big(\frac{N}{L}, \dots, \frac{N}{L}\Big) \asymp \frac{N^3}{L^2}.$$

This shows that the capacity decreases if we rearrange the fixed set of nodes into more layers. Furthermore, we can identify the most capable neural architectures with given number of nodes:

Corollary 3.4 (Optimal network with given number of neurons: informal statement)**.** *Among all neural architectures $A(n_1, \dots, n_L)$ with a given number of nodes $N = n_1 + \cdots + n_L$, the architecture $A(2N/3, N/3)$ approximately maximizes capacity.*

Suppose that in addition to fixing N, we also fix the number of input neurons n_1. Then:

1. *If $n_1 < N/2$, the architecture $A(n_1, N/2, N/2 - n_1)$ approximately maximizes capacity.*
2. *If $n_1 \geq N/2$, the architecture $A(n_1, N - n_1)$ approximately maximizes capacity.*

This result is formally stated in Theorems 11.1 and 11.4. Due to the equivalence (3.2), similar results hold for architectures $A(n_1, \dots, n_L, 1)$ with a single output unit, as well for architectures with a fixed number of output units. Complementary minimization results (Theorem 11.9) show that, under fixed budgets of units or connections, the capacity is minimized by the deepest possible networks, those with a single unit in each hidden layer.

These optimization results go against the belief, held by some, that deep architectures are more powerful because they can compute more functions than shallow architectures. The contrary is actually true: everything else being equal, deep architectures tend to compute less functions, but the functions they compute are more " interesting", or have "better properties". This is related to the well-known regularizing effect of deep learning: deep architectures tend to avoid overfitting, even when the amount of training data is small compared to the number of parameters. While some of this regularizing effect can be attributed to learning methods based on stochastic gradient descent, our analysis shows that there is a strong structural component (Section 12).

3.3. **Capacity of sets.** The derivation of the capacity formula (Theorem 3.1) is based on new bounds on the capacity of finite sets. In Lemmas 2.9 and 2.7 we noted the upper and lower bounds

$$\log_2 |S| + 1 \leq C(S) \leq n \log_2 |S|, \tag{3.2}$$

which hold for any finite set S in $\mathbb{R}^n$. We mentioned that both bounds are generally best possible. Surprisingly, the lower bound can be significantly improved for subsets of the Boolean cube H^n. Indeed, the main result of Section 7 states the following:

Theorem 3.5 (Capacity of a set)**.** *The capacity of any set $S \subset H^n$ satisfies:*

$$C(S) > \frac{1}{16} \log_2^2 |S|.$$

This new bound is tight up to an absolute constant factor. Indeed, if $S = H^k$ is a Boolean cube canonically embedded in H^n, Theorem 3.5 gives $C(S) \gtrsim k^2$, which matches the upper bound $C(S) = C(H^k) \leq k^2$ in Theorem 2.11.

Unfortunately, even the new lower bound may be too weak for some applications. In particular, we need a stronger result to prove Theorem 3.1 even for three layers ($L = 3$). Thus one may wonder if in some sense *the capacity of S could be increased through some preprocessing of S*. Specifically, can we transform S into a set $F(S)$ whose capacity is significantly larger, ideally as large as the upper bound in (9.1) allows? Furthermore, in doing so, we would like to stay in the category of subsets of the Boolean cube and use only transformations F that a network of threshold units can compute. Specifically, we will require that the *enrichment map* F be a threshold map $F \in T(H^n, H^m)$. We address the enrichment problem in the particular case where $S = H^n$, leaving the general case for future investigations. The main result of Section 9 states the following.

Theorem 3.6 (Enrichment)**.** *Let n and m be positive integers satisfying $n \leq m \leq 2^{n/2}$. There exists an injective threshold map $F \in T(H^n, H^m)$ such that:*

$$C\big(F(H^n)\big) \asymp nm.$$

The enrichment map F transforms the cube $S = H^n$ into the set $S' := F(S) \subset H^m$. The enriched set S' has the same cardinality as S and almost the maximally possible capacity:

$$C(S') \asymp nm = m \log_2 |S'|,$$

which matches the upper bound in Lemma 2.7 in dimension m.

3.4. **Capacity of networks: new tools.** In addition to the new bounds on capacity of sets, our proof of Theorem 3.1 uses some other new tools, which may be helpful in other applications. Let us briefly explain our argument.

The upper bound in Theorem 3.1 can be quickly derived from the upper bound in Lemma 2.7 and the sub-additivity of capacity. A similar argument was used before to obtain upper bounds on the VC-dimension of neural networks, see e.g. [10].

The matching lower bound is considerably harder to prove. For networks with one hidden layer, the proof of (3.1) is based on a new method that is inspired by the idea of *multiplexing* in signal processing (Section 8). Recall that estimating the capacity $C(n, m, 1)$ of a two-layer network involves counting all functions $\phi \circ f$, where

$$f = (f_1, \dots, f_m) : H^n \to H^m$$

is a threshold map (i.e. a map whose all components f_i are threshold functions) and

$$\phi : H^m \to \{0, 1\}$$

is a threshold function. Due to Theorem 2.11, there are approximately $(2^{n^2})^m = 2^{n^2 m}$ different functions f. However, this does not yield any lower bound on the number of compositions $\phi \circ f$: it might happen that two different functions f, when composed with ϕ, produce the same function. The multiplexing method circumvents this issue by combining two signals: a selector signal, and a threshold map signal. It allows the network to compute any one of the m components f_i of f; the first $\log_2 m$ bits of the input vector $x \in H^n$ act as *selector* bits used to select which component f_i of the map should be in the output.

Next, the capacity of networks with two hidden layers $C(n, m, p, 1)$ is handled by combining multiplexing with *enrichment* (Section 9). A fixed enrichment map $F : H^n \to H^m$ whose existence is guaranteed by Theorem 3.6 is used to connect the first two layers of the network. The fact that the image of F has large capacity gives us plenty of different threshold maps $G : H^m \to H^p$ between the two hidden layers. Multiplexing is then used to preserve the multitude of functions in G when they are composed with an output function $\phi : H^p \to \{0, 1\}$.

Finally, to handle networks with arbitrarily many layers (Section 10), we *stack* three-layer networks in a particular way to ensure that: (1) they may perform computations independently; and (2) the number of nodes in each layer is at most n_k. Figure 4 illustrates the stacking method. Then Theorem 3.1 can be deduced from the capacity analysis of three-layer networks and their stacking.

3.5. **Paper roadmap.** In Section 4, we give a few basic examples of threshold functions and maps. In Section 5, we derive upper bounds on the capacity of networks, and in particular the upper bound in Theorem 3.1. The reader interested only in the proof of Theorem 3.1 may then skip to Section 8. In Sections 6, we develop combinatorial tools for the analysis of the capacity of sets. We use these tools in Section 7 to prove the main result on the capacity of subsets of the Boolean cube, Theorem 3.5. In Section 8, we develop the multiplexing technique and use it to estimate the capacity of networks with one hidden layer. In Section 9, we prove the Enrichment Theorem 3.6 and use it to handle networks with two hidden layers. In Section 10, we extend the resulst to arbitrary many layers, by stacking three-layer networks, and complete the proof of Theorem 3.1. In Section 11, we study networks with maximal or minimal capacity, and in particular prove a rigorous version of Corollary 3.4. Section 12, addresses the issue of structural regularization. Finally several open questions are discussed in the conclusion (Section 14).

4. Useful examples of threshold maps

In this section we give several examples of threshold functions and threshold maps. These examples will become useful in the proofs of the main results.

Throughout this paper, the symbol $\oplus$ denotes the direct sum. For two vectors $a \in \mathbb{R}^n$ and $b \in \mathbb{R}^m$, the direct sum $a \oplus b \in \mathbb{R}^{n+m}$ is obtained by concatenation of a and b. For two sets $A \subset \mathbb{R}^n$ and $B \subset \mathbb{R}^m$, the direct sum $A \oplus B \subset \mathbb{R}^{n+m}$ is defined as:

$$A \oplus B = \{a \oplus b : \ a \in A, \ b \in B\}.$$

A similar notation is used for the direct sum of a set and a vector, for example:

$$A \oplus b = A \oplus \{b\} = \{a \oplus b : \ a \in A\}.$$

4.1. **Examples of threshold functions.** It is well known and trivial to prove that the Boolean negation NOT is a threshold function on H^1, and the Boolean functions of n variables AND $(x_1 \wedge \cdots \wedge x_n)$, OR $(x_1 \vee \cdots \vee x_n)$, and their negations NAND and NOR, are all threshold

functions on H^n. Note that the AND operation $x_1 \wedge \cdots \wedge x_n$ amounts to checking whether all x_i are equal to 1. The value 1 is not special and can be replaced by any real number θ_i:

Lemma 4.1. *Consider the function on H^n that checks whether the argument equals a given vector $\theta \in \mathbb{R}^n$:*

$$f(x) = (x = \theta) = \begin{cases} 1 & \text{if } x = \theta \\ 0 & \text{if } x \neq \theta \end{cases}.$$

Then f is a Boolean threshold function, i.e. $f \in T(H^n)$.

Proof. We can assume without any loss of generality that $\theta = (\theta_1, \ldots, \theta_n) \in H^n$, for otherwise f is the zero function and trivially lies in $T(H^n)$. Let $m = \sum_{i=1}^n \theta_i$. Now, f can be expressed as:

$$f(x) = h\Big(2\langle \theta, x\rangle - \sum_{i=1}^n x_i - m + \frac{1}{2}\Big) \tag{4.1}$$

and therefore f is a threshold function. Indeed, if $x = \theta$ then $\langle \theta, x\rangle = m$ and the right hand side of (4.1) is equal to $h(1/2) = 1$. If $x \neq \theta$, we consider two cases: $\sum_i x_i > m$ and $\sum_i x_i \leq m$. It is easy to check that in each one of these cases, the argument of h in (4.1) is $-1/2$ or less, and thus $f = 0$. □

Lemma 4.1 can be generalized one step further. It is possible to combine two operations into one threshold function: check whether the argument equals θ, and compute a given Boolean threshold function f.

Lemma 4.2 (Adding a clause)**.** *Consider a Boolean threshold function $f \in T(H^n)$ and a vector $\theta \in \mathbb{R}^q$. Then the function*

$$g(x \oplus y) \coloneqq f(x) \wedge (y = \theta) = \begin{cases} f(x) & \text{if } y = \theta \\ 0 & \text{if } y \neq \theta \end{cases}$$

is a Boolean threshold function, i.e. $g \in T(H^{n+q})$.

Proof. We can assume without loss of generality that $\theta \in H^q$, for otherwise g is the zero function and trivially lies in $T(H^{n+q})$. Let $m = \sum_{i=1}^q \theta_i$. Express $f \in T(H^n)$ as:

$$f(x) = h\big(\langle a, x\rangle + \alpha\big)$$

for suitable $a \in \mathbb{R}^n$ and $\alpha \in \mathbb{R}$. Choose any suitable constants $K > 0$ and $b > 0$ such that $K(\langle a, x\rangle + \alpha - b)$ is in the $[-1/2, 0]$ interval for all the x that satisfy $f(x) = 1$, and in the $(-\infty, -1/2)$ interval for all the x that satisfy $f(x) = 0$. We claim that g can be expressed as

$$g(x \oplus y) = h\left(K\big(\langle a, x\rangle + \alpha - b\big) + 2\langle \theta, y\rangle - \sum_{i=1}^q y_i - q + \frac{1}{2}\right)$$

and therefore g is a threshold function. Indeed, we have seen in the proof of Lemma 4.1 that the quantity $2\langle \theta, y\rangle - \sum_{i=1}^q y_i - q + 1/2$ is either equal to $1/2$ when $y = \theta$, or at most $-1/2$ for all other values of y. It is then easy to check that when $y = \theta$, we have $g(x \oplus y) = f(x)$, and when $y \neq \theta$, we have $g(x \oplus y) = 0$. □

Lemma 4.2 easily generalize to functions computable by fedforward neural networks.

Lemma 4.3 (Adding a clause)**.** *Consider a function $f \in T(n_1, \ldots, n_L, 1)$ and a vector $\theta \in \mathbb{R}^q$. Then the function:*

$$g(x \oplus y) \coloneqq f(x) \wedge (y = \theta) = \begin{cases} f(x) & \text{if } y = \theta \\ 0 & \text{if } y \neq \theta \end{cases}$$

satisfies $g \in T(n_1 + q, \ldots, n_L + q, 1)$.

The proof is elementary: it suffices to use the identity map on the additional q coordinates up to the top layer.

4.2. **Examples of threshold maps.** Let us go over some basic examples of threshold maps. Obviously, these include the identity map on H^n and all threshold functions. The next lemma gives a more interesting example.

Lemma 4.4 (Exponential map)**.** *Fix an integer k and let $\{e_i\}$ denote the canonical vector basis in $\mathbb{R}^{2^k}$. Then any one-to-one map:*

$$f : H^k \to \{e_1, \ldots, e_{2^k}\}$$

is a Boolean threshold map, i.e. $f \in T(H^k, k, 2^k)$.

Proof. The components of the map $f = (f_1, \ldots, f_{2^k})$ trivially satisfy the following: $f_i(x)$ equals 1 if $f(x) = e_i$ and 0 otherwise. The last condition can be written as $x = f^{-1}(e_i)$. Then Lemma 4.1 implies that f_i is a threshold function, and hence f is a threshold map. □

A specific example of f in Lemma 4.4 is the *exponential map*, which interprets the input vector $x \in H^k$ as a binary representation of a number and returns the binary representation of 2^x. For example, if $k = 2$, then:

$$f(00) = 2^0 = (0000),\ f(01) = 2^1 = (0010),\ f(10) = 2^2 = (0100),\ f(11) = 2^3 = (1000).$$

5. Capacity of networks: upper bounds

In this section, we prove general upper bounds on the capacity of neural networks, from which the upper bound in Theorem 3.1 will follow as a special case. The results rely on the following key remark.

Remark 5.1. The capacity of a network is always upper bounded by the sum of the capacities of its neurons. However, in general this is a weak bound due to the restrictions in capacity posed by bottle-neck layers. To see this consider two consecutive layers k and $k+1$. In principle, a unit in layer $k+1$ could have capacity of the order of n_k^2 by Theorem 2.11. However, if there is a layer $i < k$ with $n_i < n_k$, for any setting of the weights, the units in layer k can only take at most 2^{n_i} values, rather than 2^{n_k}. By Lemma 2.7, this will reduce the capacity of a unit in layer $k+1$ to be at most of the order of $n_k n_i$ instead of n_k^2. The same effect is seen if the values of the input layer are restricted.

Proposition 5.2 (Capacity formula: upper bounds)**.** *For any $L \geq 2$ and $n_1, \ldots, n_{L-1} \geq 4$, $n_L \geq 1$, the following holds. Consider a finite set $S \subset \mathbb{R}^{n_1}$ and let $n = \log_2 |S|$. Then:*

$$C(S, n_1, n_2, \ldots, n_L) \leq n n_1 n_2 + \sum_{k=2}^{L-1} \min(n, n_2, \ldots, n_k) n_k n_{k+1}.$$

In particular, we have:

$$C(n_1, n_2, \ldots, n_L) \leq \sum_{k=1}^{L-1} \min(n_1, \ldots, n_k) n_k n_{k+1}. \tag{5.1}$$

Proof. The proof is by induction. First consider the case where $L = 2$. Using property 5 in Lemma 2.5, the capacity bound from Lemma 2.7, and the assumptions on S, we see that:

$$C(S, n_1, n_2) = C(S) n_2 \leq (n_1 \log_2 |S|) n_2 \leq n_1 n n_2, \tag{5.2}$$

which is the claimed bound. Assume the property is true for L layers. To prove it for $L+1$, just apply Remark 5.1 noting that the top layer contains n_{L+1} units, and the capacity of each unit is at most $n_L \cdot \min(n, n_2, \ldots, n_L)$. This completes the proof of the first inequality. The second inequality is obtained simply by letting $S = H^{n_1}$ (i.e. $n = n_1$). □

The assumption that each non-output layer should have at least four neurons can be removed from Proposition 5.2 at the cost of an absolute constant factor in the capacity formula.

Corollary 5.3 (Upper bound in Theorem 3.1)**.** *For any $L \geq 2$ and any $n_1, \ldots, n_L \geq 1$, we have*

$$C(n_1, n_2, \ldots, n_L) \lesssim \sum_{k=1}^{L-1} \min(n_1, \ldots, n_k) n_k n_{k+1}.$$

Proof. Apply Proposition 5.2 for the capacity $C(4n_1, \ldots, 4n_{L-1}, n_L)$ and note that $C(n_1, n_2, \ldots, n_L)$ can only be smaller. □

In summary, we have derived the general upper bound associated with Theorem 3.1, and shown that the upper bound holds with an absolute and optimal constant factor of 1 in the general case, where each non-output layer has at least four nodes.

Finally, let us note that the same capacity bound holds if we extend the network by adding a single output node.

Corollary 5.4 (Adding an output node)**.** *For any $L \geq 2$ and any $n_1, \ldots, n_L \geq 1$, we have*

$$C(n_1, n_2, \ldots, n_L, 1) \lesssim \sum_{k=1}^{L-1} \min(n_1, \ldots, n_k) n_k n_{k+1}.$$

The argument to prove this result is the same as the argument used to prove Corollary 3.2.

6. Capacity of product sets: slicing

Now that we have good upper bounds on the capacity of sets and neural networks, we turn to the lower bounds. In Lemma 2.9 we noted the elementary lower bound:

$$C(S) \geq \log_2 |S| + 1, \tag{6.1}$$

which is valid for any set $S \subset \mathbb{R}^n$. We observed in Remark 2.10 that this bound is in general tight. Nevertheless, it can often be improved if additional information about the set S is available. In Section 7, we will show that if S is a subset of the Boolean cube, then the lower bound in (6.1) can be significantly improved . In this section, we develop general combinatorial tools that will be needed to derive the improved lower bound.

Early lower bounds on $C(H^n)$, and in particular the lower bound in (2.2), were based on simple combinatorial considerations and induction [18, 1]. In this section, we extend these

combinatorial methods in order to be able to handle capacities $C(S)$ of arbitrary sets S. Although the methods can be applied to any subset $S \subset H^n$, the best results are obtained when S has a product structure, as explained below.

6.1. **Slicing.** The following theorem relates the capacity of a general set S to the capacities and cardinalities of the *slices* of S. A slice is obtained by fixing the values of certain coordinates. For example, the elements of S whose first four coordinates are 1011 form a slice of S. By monotonicity, the capacity of S is lower bounded by the capacity of any slice of S. This trivial bound can be boosted if, in addition, other slices have many points. Let us show this.

Theorem 6.1 (Slicing)**.** *Let $u_1, \dots, u_k \in \mathbb{R}^m$ be a linearly independent set of vectors, and let $V_1, \dots, V_k \subset \mathbb{R}^n$ be arbitrary finite sets. Consider the subset $S \subset \mathbb{R}^{m+n}$ whose fibers at u_i are V_i, i.e. let:*

$$S := \bigcup_{i=1}^{k} u_i \oplus V_i.$$

Then the number of threshold functions on S satisfies:

$$|T(S)| \geq |T(V_1)| \cdot \big(|V_2|+1\big) \cdot \big(|V_3|+1\big) \cdots \big(|V_k|+1\big).$$

The proof of Theorem 6.1 is based on a lifting trick. Given a vector $a \in \mathbb{R}^n$ and a set $V \subset \mathbb{R}^n$, let us denote by $T_a(V)$ the set of all functions $f \in T(V)$ that can be expressed as:

$$f(x) = f_{a,\alpha}(x) = h\big(\langle a, x\rangle + \alpha\big) \tag{6.2}$$

for some $\alpha \in \mathbb{R}$. Thus, the functions in $T_a(V)$ are obtained by "cloning", i.e. by fixing $a \in \mathbb{R}^n$ and varying a single parameter – the bias $\alpha \in \mathbb{R}$.

Lemma 6.2. *If a vector $a \in \mathbb{R}^n$ separates the points of a finite set $V \subset \mathbb{R}^n$, then:*

$$|T_a(V)| = |V| + 1, \tag{6.3}$$

i.e. every f has exactly $|V|+1$ different clones.

Proof. Let $V = \{x_1, \dots, x_N\}$, where the points x_i are ordered so that the sequence $t_i := -\langle a, x_i\rangle$ is increasing with i. Now increase α continuously from $-\infty$ to ∞. As α crosses a point t_i, the function $f_{a,\alpha}(x) = h\big(\langle a, x\rangle + \alpha\big)$ changes (since it changes its value on x_i from 0 to 1), and there are no other points α where $f_{a,\alpha}(x)$ changes. The N crossover points t_i partition $\mathbb{R}$ into $N+1$ intervals. Each interval corresponds to a different function $f_{a,\alpha}(x)$. Thus, the total number of such functions is $N+1$. □

The lifting trick described in the next lemma allows us to combine k given clones of f into a single threshold function on a larger domain.

Lemma 6.3 (Lifting)**.** *Let $u_1, \dots, u_k \in \mathbb{R}^m$ be a linearly independent set of vectors, and let $V_1, \dots, V_k \subset \mathbb{R}^n$ be arbitrary finite sets. Fix $a \in \mathbb{R}^n$ and consider any functions $f_i \in T_a(V_i)$, $i = 1, \dots, k$. Then we can find a function $F = F_{f_1,\dots,f_k} \in T(S)$ such that:*

$$F(u_i \oplus \cdot) = f_i, \quad i = 1, \dots, k.$$

Proof. By definition, the functions f_i can be expressed as:

$$f_i(x) = h\big(\langle a, x\rangle + \alpha_i\big).$$

Since the vectors $u_i \in \mathbb{R}^m$ are linearly independent, there exist $b \in \mathbb{R}^m$ such that:

$$\langle b, u_i\rangle = \alpha_i, \quad i = 1, \dots, k.$$

For any vectors $u \in \mathbb{R}^m$ and $x \in \mathbb{R}^n$, define:

$$F(u \oplus x) \coloneqq h\big(\langle b, u\rangle + \langle a, x\rangle\big).$$

Obviously, F is a threshold function on $\mathbb{R}^{m+n}$, and by restricting it to S we can say that $F \in T(S)$. Now:

$$F(u_i, x) = h\big(\langle b, u_i\rangle + \langle a, x\rangle\big) = h\big(\alpha_i + \langle a, x\rangle\big) = f_i(x).$$

The lemma is proved. □

Proof of Theorem 6.1. For each $f \in T(V_1)$, let us choose and fix a vector $a = a(f) \in \mathbb{R}^n$ so that (6.2) holds. Moreover, we can always choose a so that it separates the points of $V_2 \cup \cdots \cup V_k$, i.e. so that $\langle a, x\rangle \neq \langle a, y\rangle$ for any distinct pair of points $x, y \in V_2 \cup \cdots \cup V_k$. (This can be done by perturbing a slightly. Such perturbation does not change the function f but allows a to separate points.)

Consider the set of all k-tuples of functions $(f, f_2, f_3, \dots, f_k)$ where $f_i \in T_a(V_i)$ for each i. Each such tuple consists of a function $f \in T(V_1)$ and some $k-1$ "clones" f_i of f. Due to (6.3), each clone f_i in the tuple can be chosen in exactly $|V_i| + 1$ ways. Thus the number of all such tuples is:

$$|T(V_1)| \cdot \big(|V_2| + 1\big) \cdot \big(|V_3| + 1\big) \cdots \big(|V_k| + 1\big). \tag{6.4}$$

Lemma 6.3 implies that different tuples $(f, f_2, f_3, \dots, f_k)$ produce different liftings $F = F_{f,f_2,f_3,\dots,f_k} \in T(S)$. Indeed, one can uniquely recover the tuple from the fibers $F(u_i \oplus \cdot)$ of F.

Summarizing, $T(S)$ is lower bounded by the number of different liftings F, which in turn is lower bounded by the number of different tuples $(f, f_2, f_3, \dots, f_k)$, which finally is lower bounded by the expression in (6.4), completing the proof of Theorem 6.1. □

6.2. **Product sets.** Theorem 6.1 is especially useful when S is a product of sets.

Corollary 6.4 (Capacity of product sets)**.** *Let $U \subset \mathbb{R}^m$ and $V \subset \mathbb{R}^n$ be finite sets. If U is linearly independent, then:*

$$C(U \oplus V) \geq (|U| - 1) \log_2 |V| + C(V).$$

Proof. If $U = \{u_1, \dots, u_k\}$, we can write $U \oplus V = \bigcup_{i=1}^k u_i \oplus V$. Applying Theorem 6.1 for $V_i = V$, we get:

$$|T(U \oplus V)| \geq |T(V)| \cdot |V|^{|U|-1}.$$

Taking logarithms of both sides completes the proof. □

By induction, this bound extends to products of arbitrary many sets.

Corollary 6.5 (Capacity of product sets)**.** *Assume $S_p = U \oplus \cdots \oplus U \in \mathbb{R}^{pm}$ is the product of $p > 1$ copies of a linearly independent subset $U \subset \mathbb{R}^m$ with $|U| > 1$. Then:*

$$C(S_p) \geq \frac{1}{8} p^2 |U| \log_2 |U|.$$

Proof. Apply Corollary 6.4 for the sets U and $V = S_{p-1}$, whose cardinalities are $k \coloneqq |U|$ and $|V| = k^{p-1}$, and get:

$$C(S_p) = C(U \oplus S_{p-1}) \geq (p-1)(k-1) \log_2 k + C(S_{p-1}).$$

Apply Corollary 6.4 again for $S_{p-1} = U \oplus S_{p-2}$. Continuing in this way $p-1$ times, we obtain:

$$C(S_p) \geq \Big((p-1) + (p-2) + \cdots + 1\Big)(k-1)\log_2 k + C(U). \tag{6.5}$$

Now:

$$(p-1) + (p-2) + \cdots + 1 = \frac{p(p-1)}{2} \geq \frac{p^2}{4}$$

as $p \geq 2$, $k - 1 \geq k/2$ as $k \geq 2$, and $C(U) \geq 0$. Substituting this into (6.5) completes the proof. □

Remark 6.6 (Relaxing the linear independence assumption). In the main results of this section, we assumed that the set $U = \{u_1, \ldots, u_k\} \subset \mathbb{R}^m$ is linear independent. This could be relaxed by assuming only that the set:

$$U \oplus 1 = \{u_1 \oplus 1, \ldots, u_k \oplus 1\} \in \mathbb{R}^{m+1}$$

be linearly independent.

To see this, modify the argument in the Lifting Lemma 6.3 as follows. Since the vectors $u_i \oplus 1 \in \mathbb{R}^{m+1}$ are linearly independent, there exists a vector $b \oplus \beta \in \mathbb{R}^{m+1}$ such that $\langle b \oplus \beta, u_i \oplus 1\rangle = \langle b, u_i\rangle + \beta = \alpha_i$ for all $i = 1, \ldots, k$. Now define $F(u \oplus x) \coloneqq h\left(\langle b, u\rangle + \beta + \langle a, x\rangle\right)$.

6.3. **Totally separated sets.** In addition to product sets, Theorem 6.1 can easily be specialized to totally separated sets.

Definition 6.7. *Two subsets A and B of $\mathbb{R}^n$ are* totally separated *if they lie in two different parallel hyperplanes of $\mathbb{R}^n$.*

Lemma 6.8. *Let $S \subset \mathbb{R}^n$. If A and B are totally separated subsets of S then:*

$$|T(S)| \geq |T(A)| \cdot \big(|B| + 1\big).$$

Proof. By affine invariance, we may assume that $e_1 = (1, 0, \ldots, 0)$ is a normal vector to both hyperplanes in which A and B lie. Thus, we can express:

$$A = u_1 \oplus V_1, \quad B = u_2 \oplus V_2$$

for some distinct numbers $u_1, u_2 \in \mathbb{R}$ and sets $V_1, V_2 \subset \mathbb{R}^{n-1}$. Moreover, since $u_1 \neq u_2$, the vectors $u_1 \oplus 1 = (u_1, 1)$ and $u_2 \oplus 1 = (u_2, 1)$ are linearly independent in $\mathbb{R}^2$. Applying Theorem 6.1 together with Remark 6.6 yields:

$$|T(S)| \geq |T(A \cup B)| \geq |T(V_1)| \cdot \big(|V_2| + 1\big) = |T(A)| \cdot \big(|B| + 1\big).$$

The last step follows from the affine invariance of the capacity. □

Example 6.9 (Capacity of the Boolean cube). Let us apply Lemma 6.8 to the Boolean cube H^n. This cube splits naturally into two totally separated copies of H^{n-1} formed by opposite faces of H^n. Using Lemma 6.8 and taking logarithms of both sides, we get:

$$C(H^n) \geq C(H^{n-1}) + \log_2 |H^{n-1}| = C(H^{n-1}) + n - 1.$$

By induction, this gives:

$$C(H^n) \geq (n-1) + (n-2) + \cdots + 1 = \frac{n(n-1)}{2}.$$

This recovers the lower bound given in Theorem 2.11.

7. Capacity of general sets

At the beginning of Section 6 we stated that the simple lower bound:

$$C(S) \geq \log_2 |S| + 1,$$

which is valid for any finite set $S \subset \mathbb{R}^n$, can be significantly improved if we assume that S lies in the Boolean cube H^n. The following result (restating Theorem 7.1) gives such improvement.

Theorem 7.1 (Capacity of a set)**.** *The capacity of any set $S \subset H^n$ satisfies:*

$$C(S) > \frac{1}{16} \log_2^2 |S|.$$

Before we prove this result, let us note that this bound is generally tight for any magnitude of $|S|$, up to an absolute constant factor. Indeed, consider the cube $S = H^k$ as a subset of H^n. Theorem 7.1 gives:

$$C(S) = C(k) \asymp k^2 = \log_2^2 |S|,$$

which matches the bound in Theorem 2.11.

7.1. **A hierarchical decomposition.** To prove Theorem 7.1, we are going to construct a *hierarchical decomposition* of S into totally separated sets.[1] The next lemma defines the decomposition, and Lemma 6.8 will be used to keep track of the change in capacity at each step.

Lemma 7.2 (A totally separated partition)**.** *Any set $S \subset H^n$ that consists of more than one point can be partitioned into two non-empty totally separated subsets A and B.*

Proof. Choose a pair of distinct points $x, y \in S$. They must differ in at least one coordinate i, and without loss of generality we may assume that $x_i = 0$ and $y_i = 1$. Let the set A consist of all points of S whose i-th coordinate equals 0, and B consist of all points of S whose i-th coordinate equals 1. Then the sets A and B form a partition of S and they are non-empty since $x \in A$ and $y \in B$. □

We use the following procedure to decompose S into a tree of totally separated subsets. First, let:

$$U_0 := S.$$

If $|U_0| = 1$, stop. Otherwise Lemma 7.2 gives a partition:

$$U_0 = U_1 \sqcup V_1, \quad |U_1| \geq |V_1| \geq 1, \tag{7.1}$$

where U_1 and V_1 are totally separated sets. If $|U_1| = 1$, stop. Otherwise Lemma 7.2 gives a partition:

$$U_1 = U_2 \sqcup V_2, \quad |U_2| \geq |V_2| \geq 1, \tag{7.2}$$

where U_2 and V_2 are totally separated sets. Generally, after $i-1$ partitioning steps, we check if $|U_{i-1}| = 1$ and if so, we stop. Otherwise Lemma 7.2 gives a partition:

$$U_{i-1} = U_i \sqcup V_i, \quad |U_i| \geq |V_i| \geq 1, \tag{7.3}$$

where U_i and V_i are totally separated sets.

Since at each step the set U_i becomes strictly smaller, the iterative construction must terminate after a finite number K of steps, when we have:

$$|U_K| = 1.$$

[1] We introduced totally separated sets in Section 6.3.

Firgure 2 may help to visualize the decomposition process.

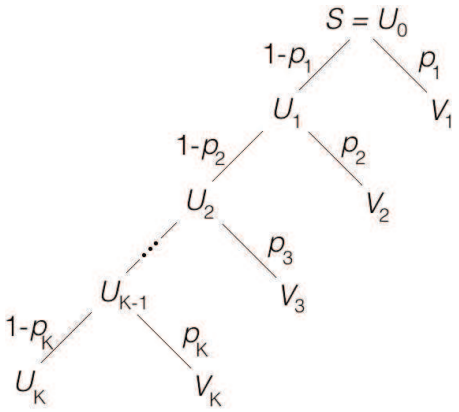

Figure 2. A hierarchical decomposition of a set $S \subset H^n$ into totally separated subsets.

There are two overlapping situations where a hierarchical decomposition of S automatically yields a good lower bound on the capacity of S: (1) when the tree is tall, i.e. K is large; and (2) when many "leaves" V_i are not too small. The following lemma quantifies this statement.

Lemma 7.3 (Hierarchical decomposition and capacity)**.** *In the hierarchical decomposition described above, one has:*

$$C(S) > K \qquad \textit{and} \qquad C(S) > \sum_{i=1}^{K} \log_2 |V_i|.$$

Proof. Applying Lemma 6.8 for decompositions (7.1) and (7.2), we get:

$$|T(S)| = |T(U_0)| \geq |T(U_1)| \cdot \big(|V_1|+1\big) \geq |T(U_2)| \cdot \big(|V_1|+1\big)\big(|V_2|+1\big).$$

Continuing in this way, after K steps we get:

$$|T(S)| \geq 2\big(|V_1|+1\big)\big(|V_2|+1\big)\cdots\big(|V_K|+1\big), \tag{7.4}$$

since at the last step $|U_K| = 1$ and thus $|T(U_K)| = 2$. To get the first conclusion of the lemma, note that $|V_i|+1 > 1$ and take the logarithm of both sides of (7.4). To get the second conclusion, note that $|V_i|+1 > |V_i|$ and finish similarly. □

7.2. **Proof of Theorem 7.1.** Let:

$$s := \log_2 |S|.$$

If $s \leq 16$, the conclusion of the theorem follows from the trivial capacity bound in Lemma 2.9:

$$C(S) > s \geq \frac{s^2}{16}.$$

Thus in the rest of the proof we can assume that:

$$s > 16. \tag{7.5}$$

Step 1. Stopping criterion. Consider the hierarchical decomposition of S constructed in Section 7.1. We will need only the initial portion of that tree decomposition, where the sets U_i are still large. Specifically, let k be the smallest integer such that:

$$|U_k| \le 2^{s/2}; \tag{7.6}$$

our argument will focus on the sets U_i and V_i for $i \le k$ only. Note that:

$$1 < k \le K.$$

The upper bound is trivial. To check the lower bound, recall that:

$$\begin{aligned} |U_1| &\ge \frac{1}{2}|U_0| \quad \text{(due to (7.1))} \\ &= \frac{1}{2}|S| = 2^{s-1} > 2^{s/2} \quad \text{(since } s > 16\text{)}. \end{aligned}$$

The definition of k then yields $k > 1$.

Step 2. Tall trees. If $k^2 \ge s^2/16$ then the conclusion of the theorem follows from the first bound in Lemma 7.3. Indeed, in this case we have:

$$C(S) > K \ge k \ge \frac{s^2}{16} = \frac{1}{16}\log_2^2 |S|.$$

Thus, in the rest of the proof we may assume that:

$$1 < k < \frac{s^2}{16}. \tag{7.7}$$

Step 3. Decomposition proportions. Recall that in the hierarchical decomposition (7.3), the set U_{i-1} is partitioned into two sets U_i and V_i. Let $1 - p_i$ and p_i denote the proportions of these sets (Figure 2), i.e.

$$|U_i| = (1 - p_i)|U_{i-1}| \quad \text{and} \quad |V_i| = p_i|U_{i-1}|. \tag{7.8}$$

The condition $|U_i| \ge |V_i| \ge 1$ in (7.3) implies that:

$$0 < p_i \le \frac{1}{2}.$$

By induction, we have:

$$|U_i| = |U_0|(1 - p_1)(1 - p_2)\cdots(1 - p_i).$$

Let us use this identity for $i = k$. By the stopping criterion (7.6), and since $|U_0| = |S| = 2^s$, we have:

$$2^{s/2} \ge 2^s(1 - p_1)(1 - p_2)\cdots(1 - p_k) \ge 2^s\, 2^{-2(p_1+\cdots p_k)}. \tag{7.9}$$

To get the last bound we used the numerical inequality $1 - x \ge 2^{-2x}$, which is valid for all $0 \le x \le 1/2$; we can apply it since $0 < p_i \le 1/2$ for all i. Rearranging the terms in the bound (7.9) gives:

$$p_1 + \cdots + p_k \ge \frac{s}{4}. \tag{7.10}$$

As a consequence, there must be many p_i that are not too small. Specifically, consider the subset of indices $I \subset \{1, \dots, k\}$ defined by:

$$I := \Big\{ i : \; p_i \ge \frac{s}{8k} \Big\}.$$

We claim that:

$$|I| \ge \frac{s}{4}. \tag{7.11}$$

Indeed, according to (7.10), we have:

$$\frac{s}{4} \le \sum_{i=1}^{k} p_i = \sum_{i \in I} p_i + \sum_{i \notin I} p_i.$$

There are $|I|$ terms p_i in the first sum, all of which are bounded by $1/2$. There are at most k terms in the second sum, all of which are bounded by $s/8k$ according to the definition of I. Therefore:

$$\frac{s}{4} \le |I| \cdot \frac{1}{2} + k \cdot \frac{s}{8k}.$$

Solving this inequality gives $|I| \ge s/4$, as claimed in (7.11).

Step 4. Short trees. We are going to use the second bound in Lemma 7.3. To apply it effectively, we will first show that all the sets V_i for $i \in I$ are not too small. So, fix an $i \in I$ and recall that by the definition of the proportions p_i, we have:

$$|V_i| = p_i |U_{i-1}|.$$

Since $i \in I$, one has: $p_i \ge 2/8k$. Furthermore considering that $i \le k$, together with the definition of the stopping time k, yields $|U_{i-1}| \ge 2^{s/2}$. Thus:

$$|V_i| \ge \frac{s}{8k} \cdot 2^{s/2} \ge \frac{2}{s} \cdot 2^{s/2} \ge 2^{s/4}. \tag{7.12}$$

In the second bound we use the assumption $k < s^2/16$ from (7.7), and in the last bound we use the assumption $s \ge 16$ from (7.5).

Now we are ready to apply the second bound in Lemma 7.3. It gives in particular:

$$C(S) > \sum_{i \in I} \log_2 |V_i|.$$

There are at least $s/4$ terms in this sum due to (7.11), each bounded below by $s/4$ according to (7.12). It follows that:

$$C(S) > \frac{s}{4} \cdot \frac{s}{4} = \frac{s^2}{16} = \frac{1}{16} \log_2^2 |S|.$$

completing the proof of Theorem 7.1. □

Although Theorem 7.1 gives a bound that is generally tight, for many subsets $S \subset H^n$ it can be improved even further. We address this phenomenon in Section 9, where we study the *enrichment* transformation as a way of increasing the capacity.

8. Networks with one hidden layer: multiplexing

Starting from this section, we focus on networks with at least one hidden layer. The ultimate goal is to prove the tight lower bound on their capacity stated in Theorem 3.1. But for now, we begin with a more basic question. For a given input set $S \subset \mathbb{R}^n$, can we relate the capacity of the network with one hidden layer $C(S, n, m, 1)$ to the capacity $C(S) = C(S, n, 1)$ of the set S? It is easy to derive a simple upper bound.

Proposition 8.1 (The effect of a hidden layer: upper bound)**.** *For any $n, m \ge 4$ and any finite set $S \subset \mathbb{R}^n$, we have:*

$$C(S, n, m, 1) \le 2C(S)m.$$

Proof. The argument is similar to the proof of Proposition 5.2. We need to count all functions of the form $\psi \circ \phi$ where $\phi \in T(S, n, m)$ and $\psi \in T(V, m, 1) = T(V)$, and where:

$$V := \operatorname{Im} \phi \subset H^m.$$

The cardinality of the image of ϕ is bounded by the cardinality of its domain, so:

$$|V| = |\operatorname{Im} \phi| \leq |S|.$$

There are $|T(S, n, m)|$ functions ϕ, and for each ϕ there are $|T(V)|$ functions ϕ. Thus the total number of compositions $\phi \circ f$ is:

$$|T(S, n, m, 1)| \leq |T(S, n, m)| \cdot \max_V |T(V)|,$$

where the maximum is taken over all subsets $V \subset H^m$ with cardinality at most $|S|$. Taking logarithms on both sides gives:

$$C(S, n, m, 1) \leq C(S, n, m) + \max_V C(V). \tag{8.1}$$

Property 5 of Lemma 2.5 gives:

$$C(S, n, m) = C(S)m.$$

Furthermore, using the capacity bound from Lemma 2.7, the assumption on $|V|$, and Lemma 2.9, wee see that:

$$C(V) \leq m \log_2 |V| \leq m \log_2 |S| \leq C(S)m.$$

Substituting these two bounds in (8.1) completes the proof. □

We can interpret Proposition 8.1 as a result that compares capacities of single-output and multiple-output networks. Indeed, due to part 5 of Lemma 2.5, the bound in Proposition 8.1 states that:

$$C(S, n, m, 1) \leq 2C(S, n, m).$$

What about the converse: can channeling the output through a single node substantially reduce the capacity of a network? In principle, it can. Indeed, $C(S, n, m, 1)$ is always bounded by $2^{|S|}$, the logarithm of the total number of binary functions on S, while $C(S, n, m) = mC(S)$ is always bounded below by $2m$ due to Lemma 2.9. Thus, whenever $|S| \ll \log_2 m$, we necessarily have:

$$C(S, n, m, 1) \ll C(S, n, m).$$

Nevertheless, we will now show how to prevent the collapse in capacity by modifying S a little – namely, by adding just $\log_2 m$ bits to the input.

Theorem 8.2 (The effect of a hidden layer: lower bound)**.** *Let $S \subset \mathbb{R}^n$ be a finite set. Let $m^- := \lceil \log_2 m \rceil$ and $S^+ := S \oplus H^{m^-}$. Then:*

$$C(S^+, n + m^-, m, 1) \geq C(S, n, m) = C(S)m.$$

The proof of this theorem is based on a *multiplexing* technique, which allows one to transmit m output functions through a single channel. To describe this technique, fix an arbitrary injective map:

$$\sigma : \{1, \dots, m\} \to H^{m^-},$$

where $m^- := \lceil \log_2 m \rceil$.

Lemma 8.3 (Multiplexing). *Let $S \subset \mathbb{R}^n$ be a finite set. Then, for any function $f = (f_1, \dots, f_m) \in T(S, n, m)$, we can construct a function $f^+ \in T(S^+, n + m^-, m, 1)$ such that:*

$$f^+(x \oplus x^-) = f_i(x) \tag{8.2}$$

if $x^- = \sigma(i)$ for some i.

Note that the injectivity of σ guarantees that there exists at most one i that satisfies (8.2).

Proof. Define:

$$f_i^+(x \oplus x^-) \coloneqq f_i(x) \wedge (x^- = \sigma(i)), \quad i = 1, \dots, m \tag{8.3}$$

and:

$$f^+ \coloneqq f_1^+ \vee \cdots \vee f_m^+.$$

This definition and the injectivity of σ ensure that (8.2) holds. Moreover, each f_i^+ is a threshold function according to Lemma 4.2, i.e. $f_i^+ \in T(S^+)$. Since the OR operation ($\vee$) is also a threshold function, it follows that: $f^+ \in T(S^+, n + m^-, m, 1)$. □

Proof of Theorem 8.2. The Multiplexing Lemma 8.3 implies that the transformation $f \mapsto f^+$ is an injective map from $T(S, n, m)$ into $T(S^+, n + m^-, m, 1)$. Indeed, (8.2) allows one to uniquely recover all the threshold functions f_i and thus $f = (f_1, \dots, f_m)$ from f^+. Thus:

$$|T(S^+, n + m^-, m, 1)| \geq |T(S, n, m)|.$$

Taking logarithms on both sides completes the proof. □

Specializing the result to $S = H^n$, yields a tight bound on the capacity of networks with a single hidden layer.

Corollary 8.4 (Capacity of networks with a single hidden layer). *If $n \geq 1.1\lceil \log_2 m \rceil$, then:*

$$C(n, m, 1) \asymp n^2 m. \tag{8.4}$$

Moreover, if $n \to \infty$ and $2 \leq \log_2 m \ll n$, then:

$$C(n, m, 1) = n^2 m(1 + o(1)). \tag{8.5}$$

Proof. The upper bound in (8.4) is a partial case of Corollary 5.4). To prove the asymptotic upper bound in (8.5), note that if $n, m \geq 4$, Proposition 5.2 gives

$$C(n, m, 1) \leq n^2 m + \min(n, m)m.$$

Furthermore, we have $\min(n, m)m \ll n^2 m$ if $n \to \infty$.

To obtain the lower bounds, apply Theorem 8.2 for $S = H^{n-m^-}$. It gives:

$$\begin{aligned} C(n, m, 1) &\geq C(n - m^-)m \\ &\gtrsim (n - m^-)^2 m \quad \text{(by Theorem 2.11)} \\ &\gtrsim n^2 m, \end{aligned}$$

where in the last step we used the assumption that $m^- = \lceil \log_2 m \rceil$. This proves the first part of the corollary. The second part follows from the same argument and the assumption that $m^- \ll n$. □

Figure 3 illustrates the multiplexing technique of Lemma 8.3. The additional m^- input bits form the vector x^- act as *selector* bits. These bits are used to select any one of the m functions $f_1, \dots, f_m$ to be the final output of the network. Since $m^- \ll m$, the selector is very small and usually does not interfere with the capacity count.

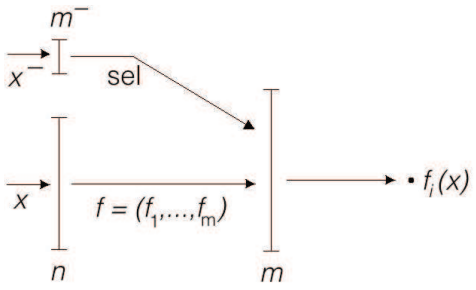

Figure 3. Multiplexing allows one to transmit any one of the m given functions f_i through a single output channel. The selector bits x^- are used to select which function to transmit.

9. Networks with two hidden layers: enrichment

9.1. **Enrichment.** A key recurrent question in this paper is: what is the relation between the capacity and cardinality of a general set $S \subset H^n$? Lemma 2.7 and Theorem 7.1 established upper and lower bounds that are generally best possible:

$$\log_2^2 |S| \lesssim C(S) \leq n \log_2 |S|. \tag{9.1}$$

The lower bound, however, is sometimes too weak for practical applications, particularly for the forthcoming analysis of networks with two hidden layers. One may wonder if the capacity of S can be increased by first preprocessing S. In particular, can we transform S into a set $F(S)$ whose capacity is significantly larger, ideally as large as the upper bound in (9.1) allows? In doing so, we would like to stay in the category of subsets of the Boolean cube and use only transformations F that a neural network can compute. Thus, we require the *enrichment map* F to be a threshold map $F \in T(n, m)$, i.e. a map from H^n to H^m whose all m components are threshold functions. We address the enrichment problem in the particular case where $S = H^n$, leaving the general case of $S \subset H^n$ for future work.

Theorem 9.1 (Enrichment)**.** *Let n and m be positive integers satisfying $n \leq m \leq 2^{n/2}$. There exists an injective linear threhsold map $F \in T(n, m)$ such that:*

$$C\big(F(H^n)\big) \asymp nm.$$

Let us make two remarks before proving this result. First, the map F transforms the cube $S = H^n$ into an "enriched" version $S' := F(S) \subset H^m$. The enriched set S' has the same cardinality as S and almost the largest possible capacity:

$$C(S') \asymp nm = m \log_2 |S'|,$$

which matches the upper bound in (9.1) in dimension m. Second, note also that an upper bound associated with Theorem 9.1 holds for any map $F : H^n \to H^m$. This follows straight from Lemma 2.7. Indeed, $S' = F(H^n)$ is a subset of H^m and has cardinality 2^n, so:

$$C(S') \leq m \log_2 |S'| = mn.$$

The non-trivial part in Theorem 9.1 is the lower bound. Our construction of F will be based on sparsity considerations.

9.2. **Construction of the enrichment map.** Let k be a positive integer and e_i be the canonical basis vectors of $\mathbb{R}^{2^k}$. Fix any one-to-one map:

$$f : H^k \to \{e_1, \dots, e_{2^k}\}.$$

According to Lemma 4.4, $f \in T(k, 2^k)$. Define the enrichment map $F : H^n \to H^m$ by applying f to each block of k successive coordinates of x. For F to be well defined, the length k of the blocks must satisfy the equation:

$$\frac{n}{k} = \frac{m}{2^k}, \tag{9.2}$$

as both sides of the equation determine the number of blocks. Assume for now that this equation has an integer solution $k \in [2, n/2]$, and let us prove the theorem in this 'balanced' case. The general case will be considered in Sections 9.4–9.5.

For this, we partition a vector $x \in H^n$ into n/k vectors $x_i \in H^k$, each containing a block of successive coordinates of length k:

$$x = \bar{x}_1 \oplus \cdots \oplus \bar{x}_{n/k},$$

and define:

$$F(x) \coloneqq f(\bar{x}_1) \oplus \cdots \oplus f(\bar{x}_{n/k}).$$

Since f is a Boolean threshold map, F is a threshold map too, i.e. $F \in T(n, m)$ as required.

9.3. **Proof of Theorem 9.1 in the balanced case.** By construction, the image of F consists of $p = n/k$ copies of the image of f:

$$F(H^n) = U \oplus \cdots \oplus U, \quad \text{where} \quad U \coloneqq f(H^k).$$

Next, recall that the image of f is the set of canonical vectors of $\mathbb{R}^{2^k}$, i.e.

$$U = \{e_1, \ldots, e_{2^k}\}.$$

Let us apply Corollary 6.5. Since U is linear independent, $|U| = 2^k \geq 2$, and $p = n/k \geq 2$ by the assumptions on k, the corollary can be applied. This application gives:

$$\begin{aligned} C(F(H^n) &\geq \frac{1}{8} p^2 |U| \log_2 |U| = \frac{1}{8} \Big(\frac{n}{k}\Big)^2 2^k k \\ &= \frac{1}{8} n^2 \cdot \frac{2^k}{k} = \frac{1}{8} n^2 \cdot \frac{m}{n} \quad \text{(by (9.2))} \\ &= \frac{1}{8} nm. \end{aligned}$$

This proves Theorem 9.1 in the special balanced case, where the equation (9.2) has an integer solution $k \in [2, n/2]$. Note that the argument so far did not use the assumption $4n \leq m \leq 2^{n/2}$ of the theorem; this assumption is used next in order to address the general (unbalanced) case. □

9.4. **Balancing.** The following lemma shows how to adjust n and m so that the equation (9.2) has an integer solution k.

Lemma 9.2. *Let $n \geq 4$ and m be positive integers such that $4n \leq m \leq 2^{n/2}$. Then there exist integers $n_0 \in [n/2, n]$, $m_0 \in [m/8, m/2]$, and $k \in [2, n/2]$ such that:*

$$\frac{n_0}{k} = \frac{m_0}{2^k}. \tag{9.3}$$

Proof. We claim that:

$$\frac{n}{x} = \frac{m/2}{2^x} \quad \text{for some } x \in \Big[2, \frac{n}{2}\Big]. \tag{9.4}$$

To show this, consider the function $r : \mathbb{R}^+ \to \mathbb{R}^+$ defined by:

$$r(x) = \frac{2^x}{x}.$$

It is easy to check that r increases to infinity on the interval $[2,\infty)$. Since $r(2) = 2 \le m/(2n)$ by assumption, the intermediate value theorem guarantees the existence of a point $x \ge 2$ where $r(x) = m/(2n)$. Equivalently, the equation (9.4) has a solution $x \in [2,\infty)$. To give an upper bound on x, note that by the assumptions on n and m we have:

$$r(x) = \frac{m}{2n} \le \frac{2^{n/2}}{n/2} = r(n/2).$$

Since r is an increasing function on the interval $[2,\infty)$ and both x and $n/2$ lie in this interval, it follows that $x \le n/2$. This verifies our claim.

Now define:

$$k := \lceil x \rceil, \quad n_0 := \left\lceil \frac{n}{k} \right\rceil k, \quad m_0 := \left\lceil \frac{n}{k} \right\rceil 2^k.$$

Then the identity (9.3) obviously holds. Next, we must check the ranges for k, n_0 and m_0.

By the definition of k, we have $k = \lceil x \rceil \ge 2$ since $x \ge 2$ and $k = \lceil x \rceil \le x \le n/2$. Thus $k \in [2, n/2]$, as required.

By the definition of n_0, we have: $n_0 \le (n/k)k = n$ and:

$$n_0 \ge \Big(\frac{n}{k} - 1\Big)k = n - k \ge \frac{n}{2},$$

where the last bound holds since $k = \lceil x \rceil \le x \le n/2$. Thus $n_0 \in [n/2, n]$, as required.

Finally, by the definition of m_0, we have:

$$m_0 \le \frac{n}{k} 2^k \le \frac{n}{x} 2^x = \frac{m}{2} \tag{9.5}$$

where the middle bound holds since r increases on the interval $[2,\infty)$, both k and x lie in that interval, and $k = \lceil x \rceil \le x$. The last bound in (9.5) follows from (9.4).

As for the lower bound on m_0, the definition of m_0 yields:

$$\begin{aligned} m_0 &\ge \Big(\frac{n}{k} - 1\Big)2^k \\ &\ge \Big(\frac{n}{x} - 1\Big)2^{x-1} \quad \text{(since } k = \lceil x \rceil \text{ satisfies } x - 1 \le k \le x\text{)} \\ &\ge \frac{n}{2x} \cdot \frac{1}{2} 2^x \quad \text{(since } n/x \ge 2\text{, which follows from (9.4))} \\ &= \frac{m}{8} \quad \text{(by the identity (9.4) that determines } x\text{).} \end{aligned}$$

Thus, in short, $m_0 \in [m/8, m/2]$ and the proof of the lemma is complete. □

9.5. **Proof of Theorem 9.1 in full generality.** Without any loss of generality, we can assume that:

$$n \ge 4 \quad \text{and} \quad 4n \le m \le 2^{n/2}.$$

Indeed, for $n < 4$ the conclusion of the theorem is trivially true by adjusting the implicit absolute constant factors. In the range $n \le m \le 4n$, we can use the identity embedding F and get the conclusion from Theorem 7.1 or Theorem 2.11.

This allows us to apply Lemma 9.2. Let n_0, m_0 and k be the numbers from the conclusion of that lemma. Then there exist a map $F' \in T(n_0, m_0)$ such that:

$$C\big(F'(H^{n_0})\big) \gtrsim n_0 m_0 \tag{9.6}$$

This follows from the balanced case of the theorem we proved in Sections 9.2–9.3, by replacing n and m with n_0 and m_0 in that argument.

Now extend $F' \in T(n_0, m_0)$ to a map $F \in T(n, m)$ using the identity function. More formally, partition each vector $x \in H^n$ as:

$$x = \bar{x}' \oplus \bar{x}'', \quad \text{where } \bar{x}' \in H^{n_0} \text{ and } \bar{x}'' \in H^{n-n_0},$$

and define $F(x) \in H^m$ by:

$$F(x) \coloneqq F'(\bar{x}') \oplus \bar{x}'' \oplus 0.$$

Here $0 = (0, \dots, 0)$ is a padding vector of zeros, which we add in order to make $F(x)$ consist of exactly m coordinates.

We must check that $F(x)$ is well defined. The vector $F'(\bar{x}')$ consists of m_0 coordinates and $\bar{x}''$ consists of $n - n_0$ coordinates. In order for the concatenation of these two vectors to fit in H^m, we must have: $m_0 + (n - n_0) \leq m$. This is indeed the case since $m_0 \leq m/8$ and $n - n_0 \leq n \leq m/4$ by Lemma 9.2.

Since both F' and the identity map are injective threshold maps, the map F is an injective threshold map too. By construction, the projection of $F(H^n)$ onto the first m_0 coordinates equals $F'(H^{n_0})$. Therefore:

$$C\big(F(H^n)\big) \geq C\big(F'(H^{n_0})\big) \gtrsim n_0 m_0 \gtrsim nm,$$

where we used (9.6) and Lemma 9.2. This completes the proof of Theorem 9.1. □

9.6. **Capacity of networks with two hidden layers.** As an application of the Enrichment Theorem 9.1, we can estimate the capacity of networks with two hidden layers.

Theorem 9.3 (Two hidden layers)**.** *If $n \geq 3\lceil \log_2 m \rceil$, $m \geq 3\lceil \log_2 p \rceil$, and $n \geq 3\lceil \log_2 p \rceil$, then:*

$$C(n, m, p, 1) \asymp n^2 m + \min(n, m) mp.$$

Proof. The upper bound follows as a special case of Corollary 5.4. To prove the lower bound, let us first only assume that:

$$n \geq 2\lceil \log_2 m \rceil, \quad m \geq 2\lceil \log_2 p \rceil, \quad n \geq 2\lceil \log_2 p \rceil. \tag{9.7}$$

Then we can obtain the $n^2 m$ term by comparing the $A(n, m, p, 1)$ network with the $A(n, m, 1)$ network. Indeed, we have:

$$\begin{aligned} C(n, m, p, 1) &\geq C(n, m, 1, 1) \quad \text{(by monotonicity)} \\ &\geq C(n, m, 1) \quad \text{(by contractivity, see part 4 of Lemma 2.5)} \\ &\gtrsim n^2 m \quad \text{(by Corollary 8.4).} \end{aligned}$$

Next, we consider two cases: $n \geq m$ and $n < m$.

Case 1: $n \geq m$. In this regime, we can compare the two-hidden-layers network with the single-hidden-layer network $A(m, p, 1)$. Just like above, using monotonicity, contractivity, and Corollary 8.4, we get:

$$C(n, m, p, 1) \geq C(m, m, p, 1) \geq C(m, p, 1) \gtrsim m^2 p.$$

Case 2: $n < m$. In this regime, we use both enrichment and multiplexing. The first assumption in (9.7) yields $m \le 2^{n/2}$, which allows us to use Theorem 9.1. Fix an enrichment map $F \in T(n,m)$ whose existence is guaranteed by Theorem 9.1. Applying part 6 of Lemma 2.5, for the map $(F \oplus \mathrm{id})(x \oplus x^-) = F(x) \oplus x^-$ that belongs to the class $T(n+p^-, m+p^-)$ and for $S = H^{n+p^-}$, we obtain:

$$\begin{aligned} C(n+p^-, m+p^-, p, 1) &\ge C\big(F(H^n) \oplus H^{p^-}, m+p^-, p, 1\big) \\ &\ge C\big(F(H^n)\big)p \quad \text{(by Theorem 8.2)} \\ &\gtrsim nmp \quad \text{(by Theorem 9.1)}. \end{aligned}$$

Putting everything together. In summary, we showed that $C(n+p^-, m+p^-, p)$ is always bounded below by $n^2 m$, and is also bounded below by $m^2 p$ if $m < n$, and by nmp if $m \ge n$. This means that:

$$C(n+p^-, m+p^-, p, 1) \gtrsim n^2 m + \min(n,m)mp.$$

The last two assumptions in (9.7) state that: $p^- = \lceil \log_2 p \rceil \le n/2$ and $p^- \le m/2$. Thus monotonicity gives:

$$C(\lfloor 3n/2 \rfloor, \lfloor 3m/2 \rfloor, p, 1) \ge C(n+p^-, m+p^-, p, 1) \gtrsim n^2 m + \min(n,m)mp.$$

Recall that we proved this result under the assumptions (9.7), which are weaker than those in the statement of the theorem. Applying this result for $\lfloor 2n/3 \rfloor$ instead of n, and for $\lfloor 2m/3 \rfloor$ instead of m, completes the proof. □

Theorem 9.4 (Two hidden layers, multiple-outputs)**.** *If* $n \ge 2\lceil \log_2 m \rceil$ *and* $m \ge 2\lceil \log_2 p \rceil$, *then:*

$$C(n,m,p) \asymp n^2 m + \min(n,m)mp.$$

Proof. The upper bound is a partial case of Corollary 5.3. For the lower bound, we can essentially repeat the proof of Theorem 9.3 except for the multiplexing in the last step, which is not needed in this case. Instead, we can just use part 6 of Lemma 2.5 followed by the enrichment Theorem 9.1 and get:

$$C(n,m,p) \ge C\big(F(H^n), m, p\big) \gtrsim nmp.$$

The proof is complete. □

10. Networks with arbitrarily many layers: stacking

Now we extend the capacity lower bounds to feedforward networks with arbitrarily many layers, thus completing the proof of the main result (Theorem 3.1). Denote:

$$\bar{n}_k := \min(n_1, \dots, n_k).$$

Let us handle networks with three hidden layers first.

Lemma 10.1 (Three hidden layers)**.** *Let* $n_j \ge 3\lceil \log_2 n_k \rceil$ *for all* $1 \le j < k \le 4$. *Then:*

$$C(n_1, n_2, n_3, n_4, 1) \asymp n_1^2 n_2 + \bar{n}_2 n_2 n_3 + \bar{n}_3 n_3 n_4.$$

Proof. The upper bound is a special case of Corollary 5.4. As for the lower bound, monotonicity, contractivity (Lemma 2.5), and Theorem 9.3 yield:

$$C(n_1, n_2, n_3, n_4, 1) \ge C(n_1, n_2, n_3, 1) \gtrsim n_1^2 n_2 + \bar{n}_2 n_2 n_3$$

and also:

$$C(n_1, n_2, n_3, n_4, 1) \ge C(\bar{n}_2, \bar{n}_2, n_3, n_4, 1) \ge C(\bar{n}_2, n_3, n_4, 1) \gtrsim \bar{n}_3 n_3 n_4.$$

Combining the two lower bounds, we conclude that:

$$C(n_1, n_2, n_3, n_4, 1) \gtrsim \max\left(n_1^2 n_2 + \bar{n}_2 n_2 n_3, \bar{n}_3 n_3 n_4\right) \geq \frac{1}{2}\left(n_1^2 n_2 + \bar{n}_2 n_2 n_3 + \bar{n}_3 n_3 n_4\right).$$

The proof is complete. □

10.1. **Stacking.** In principle, networks with arbitrarily many layers can be handled by a similar argument. However, instead of producing the sum over the layers claimed by Theorem 3.1, this argument will only produce the maximum over the layers. The maximum can be replaced with the sum by paying a factor of $1/L$, which is weaker than the constant factor claimed in Theorem 3.1. Thus, to overcome this limitation, we develop a *stacking* technique and prove the following.

Lemma 10.2 (Four and more hidden layers)**.** *Assume that $L \geq 5$ and $n_j \geq 3\lceil \log_2(Ln_k)\rceil$ for all $1 \leq j < k \leq L$. Then:*

$$C(6n_1, \ldots, 6n_L, 1) \gtrsim \sum_{k=1}^{L-1} \bar{n}_k n_k n_{k+1}.$$

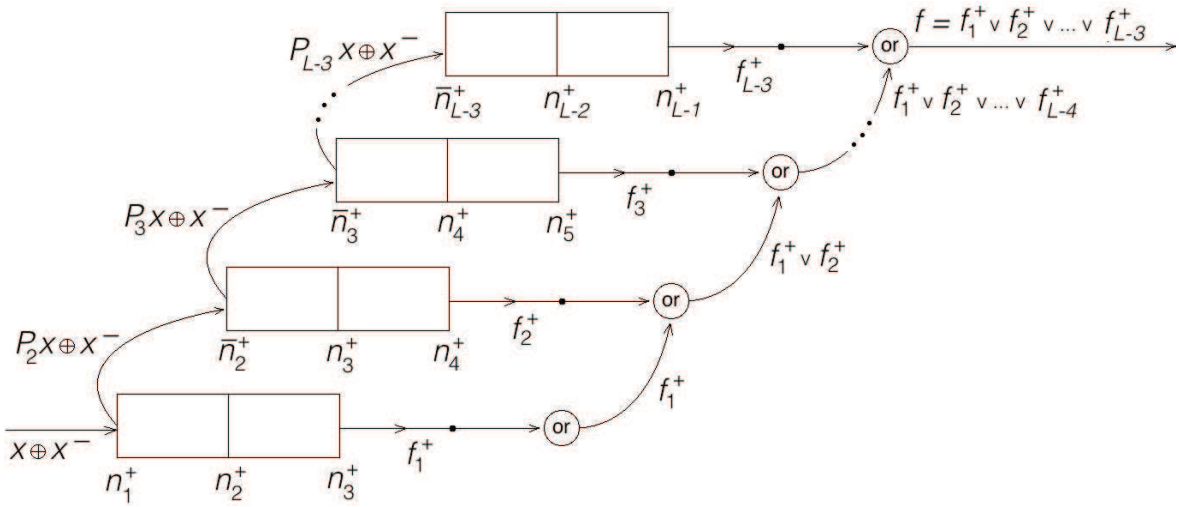

Figure 4. A network with almost largest possible capacity can be constructed by stacking three-layer networks.

Proof. To prove the lemma, we will compare the network $A(6n_1, \ldots, 6n_L, 1)$ with a smaller network, which we construct by "stacking" $L-3$ three-layer modules, and doing multiplexing in each one of them.

Step 1. Construction of the network. Fix an arbitrary injective map

$$\eta : \{1, \ldots, L-3\} \to H^{L^-} \quad \text{where} \quad L^- := \lceil \log_2(L-3)\rceil. \tag{10.1}$$

Consider arbitrary functions

$$f_k \in T(\bar{n}_k, n_{k+1}, n_{k+2}, 1), \quad k = 1, \ldots, L-3,$$

Lemma 4.2 states that the function

$$f_k^+(x \oplus x^-) := f_k(x) \wedge \left(x^- = \eta(k)\right), \quad x \in H^{\bar{n}_k}, \quad x^- \in H^{L^-}, \tag{10.2}$$

belongs to $T(\bar{n}_k^+, n_{k+1}^+, n_{k+2}^+, 1)$, where we let:

$$n_k^+ := n_k + L^- \quad \text{and} \quad \bar{n}_k^+ := \bar{n}_k + L^-. \tag{10.3}$$

Now connect three-layer modules $A(\bar{n}_k^+, n_{k+1}^+, n_{k+2}^+, 1)$, $k = 1, \dots, L-3$, as shown in Figure 4. In that figure, P_k denotes the coordinate projection onto $\mathbb{R}^{\bar{n}_k}$ that retains the first $\bar{n}_k$ coordinates of a vector. Given an input $x \oplus x^-$, the first module computes $f_1^+(x \oplus x^-)$ in layer 4, and it passes $P_2 x \oplus x^-$ to layer 2 as the input to the second module. The second module computes $f_2^+(P_2 x \oplus x^-)$ in layer 5, then takes the 'or' with the output of the second module in layer 6, thereby computing $f_1^+(x \oplus x^-) \vee f_2^+(P_2 x \oplus x^-)$; it also passes $P_3 x \oplus x^-$ to layer 3 as the input to the third module, etc. Continuing in this way, we see that the network ultimately computes and outputs the function:

$$f(x \oplus x^-) := f_1^+(x \oplus x^-) \vee f_2^+(P_2 x \oplus x^-) \vee \cdots \vee f_{L-3}^+(P_{L-3} x \oplus x^-). \tag{10.4}$$

Step 2. Estimating the capacity of the network using the capacities of modules. Now that we described the architecture, let us estimate how many Boolean functions f the architecture can compute. Let us denote the set of all such computable functions f^+ by T. By definition of the functions f_k^+ in (10.2) and f in (10.4), we have:

$$f(x \oplus x^-) = f_k(P_k x) \tag{10.5}$$

if $x^- = \eta(k)$ for some[2] k. This implies that the map $(f_1, \dots, f_{L-3}) \mapsto f^+$ is an injective transformation from $\prod_{k=1}^{L-3} T(\bar{n}_k, n_{k+1}, n_{k+2}, 1)$ to T. Indeed, Equation (10.5) allows one to uniquely recover all f_k and thus $f = (f_1, \dots, f_{L-3})$ from f^+.) Therefore:

$$|T| \geq \prod_{k=1}^{L-3} |T(\bar{n}_k, n_{k+1}, n_{k+2}, 1)|.$$

The right hand side can be estimated using the tight bounds on the capacity of three-layer networks from Theorem 9.3. Note that the conditions on n_k guarantee that the assumptions of Theorem 9.3 are satisfied. We obtain:

$$\begin{aligned}
\log_2 |T| &\geq \sum_{k=1}^{L-3} C(\bar{n}_k, n_{k+1}, n_{k+2}, 1) \\
&\gtrsim (n_1^2 n_2 + \bar{n}_2 n_2 n_3) + \bar{n}_3 n_3 n_4 + \bar{n}_4 n_4 n_5 + \cdots + \bar{n}_{L-2} n_{L-2} n_{L-1} \\
&= \sum_{k=1}^{L-2} \bar{n}_k n_k n_{k+1}.
\end{aligned} \tag{10.6}$$

Step 3. Counting nodes. As is evident from Figure 4, the overall architecture has L layers of units (not counting the output). The number of nodes in the k-th layer of units, $k = 1, \dots, L-1$, is bounded by:

$$\begin{aligned}
2 + 2n_k^+ + \bar{n}_k^+ &\leq 2 + 3n_k^+ \\
&\leq 2 + 3\big(n_k + \lceil \log_2 L \rceil\big) \quad \text{(by definition of } n_k^+ \text{ in (10.3) and } L^- \text{ in (10.1))} \\
&\leq 6n_k \quad \text{(by the assumptions on } n_k\text{)}.
\end{aligned}$$

Hence, by monotonicity:

$$\log_2 |T| \leq C(6n_1, \dots, 6n_L, 1).$$

[2]The injectivity of η guarantees that there exists at most one k that satisfies (10.5).

Combining this with the lower bound (10.6), we conclude that:

$$C(6n_1,\dots,6n_L,1)\gtrsim\sum_{k=1}^{L-2}\bar{n}_k n_k n_{k+1}. \tag{10.7}$$

Step 4. Adding one term to the sum. To complete the proof, we just need to add one last term to this sum. We can get it by comparison with a three-layer network $A(\bar{n}_{L-2},n_{L-1},n_L,1)$. Indeed, monotonicity, contractivity (Lemma 2.5), and Theorem 9.3 give:

$$C(6n_1,\dots,6n_L,1)\geq C(\bar{n}_{L-2},\dots,\bar{n}_{L-2},n_{L-1},n_L,1)\geq C(\bar{n}_{L-2},n_{L-1},n_L,1)\gtrsim\bar{n}_{L-1}n_{L-1}n_L.$$

Combining this with (10.7), we conclude that:

$$C(6n_1,\dots,6n_L,1)\gtrsim\max\left(\sum_{k=1}^{L-2}\bar{n}_k n_k n_{k+1},\bar{n}_{L-1}n_{L-1}n_L\right)\geq\frac{1}{2}\sum_{k=1}^{L-1}\bar{n}_k n_k n_{k+1}.$$

This completes the proof of the Lemma. □

10.2. **The lower bound in Theorem 3.1.** Now we prove a partial case of Theorem 3.1 for networks with a single output node:

Theorem 10.3. *Under the conditions of Theorem 3.1, we have:*

$$C(n_1,\dots,n_L,1)\asymp\sum_{k=1}^{L-1}\bar{n}_k n_k n_{k+1}.$$

Proof. We already proved the upper bound on the capacity in Corollary 5.4. The lower bound follows from Corollary 8.4 for a single hidden layer, Theorem 9.3 for two hidden layers, Lemma 10.1 for three hidden layers, and Lemma 10.2 for four and more layers applied[3] using $\lfloor n_k/6\rfloor$ instead of n_k. □

Finally, we are ready to complete the proof of the main result:

Proof of Theorem 3.1. The upper bound was already proven in Proposition 5.2. It remains to prove the lower bound. For $L=2$, the result follows from Theorem 2.11, which gives:

$$C(n_1,n_2)=C(n_1)n_2\gtrsim n_1^2 n_2.$$

Now let $L\geq 3$. Monotonicity and Theorem 10.3 yield:

$$C(n_1,\dots,n_L)\geq C(n_1,\dots,n_{L-1},1)\gtrsim\sum_{k=1}^{L-2}\bar{n}_k n_k n_{k+1}.$$

To complete the proof, we need to add just one last term to this sum. We can get it by comparison with a three-layer network $A(\bar{n}_{L-2},n_{L-1},n_L)$. Indeed, monotonicity, contractivity (Lemma 2.5), and Theorem 9.4 give:

$$C(n_1,\dots,n_L)\geq C(\bar{n}_{L-2},\dots,\bar{n}_{L-2},n_{L-1},n_L)\geq C(\bar{n}_{L-2},n_{L-1},n_L)\gtrsim\bar{n}_{L-1}n_{L-1}n_L.$$

Combining this with (10.7), we conclude that:

$$C(n_1,\dots,n_L)\gtrsim\max\left(\sum_{k=1}^{L-2}\bar{n}_k n_k n_{k+1},\bar{n}_{L-1}n_{L-1}n_L\right)\geq\frac{1}{2}\sum_{k=1}^{L-1}\bar{n}_k n_k n_{k+1}.$$

[3]Precisely, the assumptions of Theorem 3.1 yield: $n_j\geq 18\log_2(Ln_k)\geq 18\lceil\log_2(Ln_k/6)\rceil$. Dividing both sides by 6 and taking the integer part, we get: $\lfloor n_j/6\rfloor\geq 3\lceil\log_2(Ln_k/6)\rceil\geq 3\lceil\log_2(L\lfloor n_k/6\rfloor)\rceil$. This means that Lemma 10.2 can indeed be applied using $\lfloor n_k/6\rfloor$ instead of n_k.

This completes the proof. □

10.3. **Why are rapidly expanding networks excluded?** We stated Theorem 3.1 under the assumption that the network is not expanding too rapidly, as quantified by requiring that:

$$n_j \geq 18 \log_2(Ln_k) \quad \text{for all } j \leq k. \tag{10.8}$$

It is worth noting that this requirement is almost optimal. To see this, note first that the number of all Boolean functions on H^{n_1} is $2^{2^{n_1}}$. This yields the trivial upper bound:

$$C(n_1, \dots, n_L, 1) \leq 2^{n_1}.$$

Combining it with the lower bound given by Theorem 3.1 (and Corollary 3.2), we get

$$2^{n_1} \gtrsim \sum_{k=1}^{L-1} \min(n_1, \dots, n_k) n_k n_{k+1} \geq \sum_{k=2}^{L} n_k.$$

Thus, in order for Theorem 3.1 to hold, we must have:

$$n_1 \gtrsim \log_2 \left(\sum_{k=2}^{L} n_k \right).$$

In particular, if all n_k for $k \geq 2$ are of the same order (e.g. equal to each other), we must have:

$$n_1 \gtrsim \log_2(Ln_k).$$

This shows that the condition (10.8) can not be removed and that it has an almost optimal form.

10.4. **Relaxing the assumption on the number of nodes.** Although the assumption $n_j \gtrsim \log_2(Ln_k)$ in Theorem 3.1 is almost optimal, it can still be slightly improved in order to accommodate small top layers. Specifically, with a little more work, it can be relaxed to:

$$n_j \gtrsim \log_2 \big((L - k + 1) n_k \big) \quad \text{for all } j \leq k.$$

This relaxed condition can be useful since it allows for very small top layers.

The idea behind the relaxed condition is that in the proof of Lemma 10.2, it is not necessary to transmit all bits of x^- to the top layer. Indeed, choose $\eta(k)$ to be the binary representation of the number $L - 3 - k$. Thus, the first bit of $\eta(k)$ is 0 if k it is in the uppet half of the layers, the first two bits are 00 if k is in the upper quarter, the first three bits are 000 if k is in the upper eighth, etc. Now, we can drop the first bit of x^- when we pass it between the modules in the upper half of the layers (i.e. for $k \geq (L-3)/2$); instead of verifying the clause $\eta(k) = x^-$, we verify the equivalent clause $Q_1\eta(k) = Q_1 x^-$, where Q_1 is the coordinate projection that drops the first bit. Similarly, we can drop the second bit of x^- in the upper quarter of the layers, etc. Thus, the length of the portion of x^- passed to the k-th layer is approximately $\log_2(L - k + 1)$ instead of the full length, i.e. $L^- = \log_2(L)$. The rest of the proof is unchanged.

10.5. **Restricted capacity.** In this section we extend Theorem 3.1 to the case where the input to the network are not all possible binary vectors, but rather lie in a subset $S \subset H^{n_1}$. We introduced this restricted version of capacity in Section 2.3 and denoted it by:

$$C(S, n_1, n_2, \dots, n_L).$$

We proved an upper bound on $C(S, n_1, n_2, \dots, n_L)$ in Proposition 5.2. Now we will complement it with a lower bound. The notion of VC-dimension (see e.g. [24, Section 8.3]) allows us to reduce the problem of restricted capacity to the case of unrestricted capacity.

Lemma 10.4 (Restricted vs. unrestricted capacity)**.** *Consider a subset $S \subset H^{n_1}$. Then, for any number of layers $L \geq 2$ and any number of nodes $n_2, \dots, n_L$ in each layer, we have:*

$$C(S, n_1, n_2, \dots, n_L) \geq C(H^d, n_1, n_2, \dots, n_L) = C(d, n_2, \dots, n_L),$$

where d is the VC-dimension of S.

Proof. By the definition of VC-dimension, there exists a subset of indices $I \subset \{1, \dots, n_1\}$ of cardinality $|I| = d$ that is shattered by S. This means that:

$$P_I S = H^I$$

where $P_I : H^{n_1} \to H^I$ is the coordinate projection that retains the coordinates in I and drops the coordinates outside of I. By excluding the input nodes outside I, one immediately obtains:

$$C(S, n_1, n_2, \dots, n_L) \geq C(P_I S, n_1, n_2, \dots, n_L) = C(H^d, n_2, \dots, n_L, 1) = C(d, n_2, \dots, n_L).$$

The proof is complete. □

Combining this bound with the Sauer-Shelah Lemma, we obtain the following:

Proposition 10.5 (Restricted capacity: a lower bound)**.** *Consider a subset $S \subset H^{n_1}$ such that $|S| \leq 2^n$. Then, for any number of layers $L \geq 2$ and any number of nodes $n_2, \dots, n_L$ in each layer, there exists an integer d such that:*

$$d \gtrsim \frac{n}{\log_2(en_1/n)}$$

and:

$$C(S, n_1, n_2, \dots, n_L) \geq C(H^d, n_1, n_2, \dots, n_L) = C(d, n_1, \dots, n_L).$$

Proof. The Sauer-Shelah Lemma (see e.g. [24, Section 8.3.3]) gives the upper bound:

$$|S| \leq \sum_{k=0}^{d} \binom{n_1}{k} \leq \left(\frac{en_1}{d}\right)^d,$$

where d is the VC-dimension of S. On the other hand, we have the lower bound $|S| \leq 2^n$. Combining the two bounds and taking logarithms, we get:

$$n \leq d \log_2 \left(\frac{en_1}{d}\right).$$

An elementary computation then yields:

$$d \gtrsim \frac{n}{\log(en_1/n)}.$$

An application of Lemma 10.4 completes the proof. □

Combining Proposition 10.5 with the capacity formula for $C(d, n_2, \dots, n_L)$ given by Theorem 3.1, we can obtain a general lower bound on the restricted capacity in terms of the cardinality of S.

Remark 10.6 (Tightness). The bound in Proposition 10.5 is generally best possible up to a logarithmic factor. Indeed, if $S = H^d$ and $d = n_1$ then:

$$C(S, n_1, n_2, \dots, n_L) = C(d, n_2, \dots, n_L).$$

11. Extremal capacity

The capacity formula in Theorem 3.1 is particularly useful when one wants to maximize the capacity of a network under some natural constraints. For example, if we *fix the number of parameters* of a network, which is essentially the same as fixing the number of edges, Corollary 3.3 states that any monotonically expansive network approximately maximizes capacity. In this section, we consider what happens if instead we *fix the number of nodes* and, possibly, also the number of nodes in the input layer.

We will use the symbols $\approx$ for identities that hold up to an $1 + o(1)$ factor, that is $a_n \approx b_n$ means that $a_n = (1 + o(1))b_n$ as $n \to \infty$. As before, we continue to use the symbols $\asymp$ and $\lesssim$ for identities and inequalities that hold up to an absolute constant factor.

11.1. **Fixing the number of nodes.** It turns out that a network with a given number of nodes that asymptotically maximizes capacity is *shallow*. Specifically, the optimal network has just one hidden layer, which is half the size of the input layer:

Theorem 11.1. *Let $L \geq 2$ and $n_1, \dots, n_{L-1} \geq 4$, $n_L \geq 1$. Let $N := n_1 + \cdots + n_L$ denote the total number of nodes. Then:*

$$C(n_1, \dots, n_L) \leq \frac{4}{9} N^3 \approx C\Big(\frac{2N}{3}, \frac{N}{3}\Big)$$

as $N \to \infty$.

We shall first prove a version of Theorem 11.1 for the *estimated* capacity:

$$\widehat{C}(n_1, \dots, n_L) := \sum_{k=1}^{L-1} \min(n_1, \dots, n_k) n_k n_{k+1}, \tag{11.1}$$

and then replace the estimated capacity. The following lemma yields a general recipe to increase the (estimated) capacity of any network, by moving all nodes from layer 3 and up into the input layer.

Lemma 11.2 (Move nodes out of upper layers to increase capacity)**.** *Let $L \geq 3$. Then:*

$$\widehat{C}(n_1, \dots, n_L) \leq \widehat{C}\Big(n_1 + \sum_{k=3}^{L} n_k, n_2\Big). \tag{11.2}$$

Proof. Let us fist handle the case $L = 3$, where we have to show that:

$$\widehat{C}(n, m, p) \leq \widehat{C}(n + p, m). \tag{11.3}$$

The definition of the estimated capacity (11.1) yields:

$$\widehat{C}(n, m, p) = n^2 m + \min(n, m) mp \leq (n^2 + np)m.$$

In the last step we used that $\min(n, m) \leq n$. On the other hand, the same definition yields:

$$\widehat{C}(n + p, m) = (n + p)^2 m \geq (n^2 + 2np)m.$$

Hence (11.3) is evident.

Next, let $L \geq 4$. Combining the definition of the estimated capacity (11.1) with the fact that $\min(n_1, n_2) \leq n_1$, $\min(n_1, \dots, n_k) \leq n_2$, we obtain:

$$\widehat{C}(n_1, \dots, n_L) \leq n_1^2 n_2 + n_1 n_2 n_3 + \sum_{k=3}^{L-1} n_2 n_k n_{k+1} = \Big(n_1^2 + n_1 n_3 + \sum_{k=3}^{L-1} n_k n_{k+1}\Big) n_2.$$

On the other hand, using the same definition and expanding the square, we get:

$$\widehat{C}\Big(n_1 + \sum_{k=3}^{L} n_k, n_2\Big) = \Big(n_1 + \sum_{k=3}^{L} n_k\Big)^2 n_2 \geq \Big(n_1^2 + 2n_1 n_3 + 2\sum_{k=3}^{L-1} n_k n_{k+1}\Big) n_2.$$

Hence (11.2) is evident. □

Armed with the recipe given in Lemma 11.2, we can easily maximize the (estimated) capacity over all networks with two layers and a given number of nodes.

Lemma 11.3 (The most capable network with two layers)**.** *Let $N = n + m$. Then:*

$$\widehat{C}(n, m) = n^2 m \leq \widehat{C}\Big(\frac{2N}{3}, \frac{N}{3}\Big) = \frac{4}{9} N^3.$$

Proof. The maximum of $n^2 m = n^2(N - n)$ is attained for $n = 2N/3$. □

Combining Lemmas 11.2 and 11.3, we obtain a version of Theorem 11.1 for the estimated capacity:

$$\widehat{C}(n_1, \dots, n_L) \leq \widehat{C}\Big(n_1 + \sum_{k=3}^{L} n_k, n_2\Big) \leq \widehat{C}\Big(\frac{2N}{3}, \frac{N}{3}\Big) = \frac{4}{9} N^3. \tag{11.4}$$

Proof of Theorem 11.1. Because the capacity is bounded by the estimated capacity (Proposition 5.2), and using (11.4), we get:

$$C(n_1, \dots, n_L) \leq \widehat{C}(n_1, \dots, n_L) \leq \widehat{C}\Big(\frac{2N}{3}, \frac{N}{3}\Big) = \frac{4}{9} N^3.$$

Furthermore, Theorem 2.11 implies that:

$$C\Big(\frac{2N}{3}, \frac{N}{3}\Big) = C\big(H^{2N/3}\big) \frac{N}{3} \approx \Big(\frac{2N}{3}\Big)^2 \frac{N}{3} = \frac{4}{9} N^3$$

as $N \to \infty$. This completes the proof. □

11.2. **Fixing both the total number of nodes and the size of the input layer.** In many applications, the input layer is fixed and can not be optimized. In such situations, it makes sense to maximize capacity of networks with a given total number of nodes N, as well as a given number of nodes n_1 in the input layer. While here we focus on the case where the total number of neurons and the size of the input layer are fixed, similar results are obtained also for the case where in addition the size of the output layer is fixed.

It turns out that a network that maximizes capacity under these constraint is again shallow. If $n_1 \leq N/2$, the optimal network has two hidden layers, the first having n_1 more nodes than the second. If $n_1 \geq N/2$, such architecture is impossible; the optimal network has just one hidden layer. The following theorem makes this precise.

Theorem 11.4. *Let $L \geq 2$. Assume that the total number of nodes is $N \coloneqq n_1 + \cdots + n_L$. Then, the following holds if $n_1 \to \infty$ and $\log N \ll n_1$.*

1. If $n_1 \leq N/2$ then:

$$C(n_1, \ldots, n_L) \leq \frac{n_1 N^2}{4} \asymp C\Big(n_1, \frac{N}{2}, \frac{N}{2} - n_1\Big).$$

2. If $n_1 \geq N/2$ then:

$$C(n_1, \ldots, n_L) \leq n_1^2(n_2 + \cdots + n_L) \approx C(n_1, n_2 + \cdots + n_L).$$

As in the previous section, we first prove a version of Theorem 11.4 for the estimated capacity $\widehat{C}(n_1, \ldots, n_L, 1)$ defined in (11.1). The following elementary fact will be helpful in our analysis.

Lemma 11.5. *For any $L \geq 2$, and any positive real numbers $x_1, \ldots, x_L$, we have:*

$$\Big(\sum_{k=1}^{L} x_k\Big)^2 \geq 4 \sum_{k=1}^{L-1} x_k x_{k+1}.$$

Proof. Consider the difference:

$$\Big(\sum_{k=1}^{L} x_k\Big)^2 - \Big(\sum_{k=1}^{L} (-1)^k x_k\Big)^2 = 2 \sum_{i,j=1}^{L} \Big(1 - (-1)^i (-1)^j\Big) x_i x_j = 4 \sum_{i,j \in \mathcal{O}} x_i x_j \tag{11.5}$$

where $\mathcal{O} \subset \{1, \ldots, L\}^2$ is the set of pairs (i, j) such that either i is even and j is odd, or i is odd and j is even. In particular, $\mathcal{O}$ contains all pairs of the form $(k, k+1)$. Since all the terms $x_i x_j$ of the sum are positive, this yields:

$$\sum_{i,j \in \mathcal{O}} x_i x_j \geq \sum_{k=1}^{L-1} x_k x_{k+1}.$$

Combining this with (11.5), we conclude that:

$$\Big(\sum_{k=1}^{L} x_k\Big)^2 - \Big(\sum_{k=1}^{L} (-1)^k x_k\Big)^2 \geq 4 \sum_{k=1}^{L-1} x_k x_{k+1}.$$

This yields the conclusion of the lemma. □

We are ready to prove the "estimated" version the first part of Theorem 11.4.

Lemma 11.6 (Small input layer)**.** *If $n_1 \leq N/2$, then:*

$$\widehat{C}(n_1, \ldots, n_L) \leq \frac{n_1 N^2}{4} = \widehat{C}\Big(n_1, \frac{N}{2}, \frac{N}{2} - n_1\Big).$$

Proof. On one hand, the definition (11.1) of the estimated capacity yields:

$$\widehat{C}\Big(n_1, \frac{N}{2}, \frac{N}{2} - n_1\Big) = n_1^2 \cdot \frac{N}{2} + \min\Big(n_1, \frac{N}{2}\Big) \frac{N}{2} \Big(\frac{N}{2} - n_1\Big) = \frac{n_1 N^2}{4},$$

where in the last step we used the assumption $n_1 \leq N/2$ and simplified the expression. On the other hand, definition (11.1) gives:

$$\begin{aligned}\widehat{C}(n_1,\dots,n_L,1) &\leq n_1 \sum_{k=1}^{L-1} n_k n_{k+1} \quad (\text{since } \min(n_1,\dots,n_k) \leq n_1)\\ &\leq \frac{1}{4} n_1 (n_1+\cdots+n_L)^2 \quad (\text{using Lemma 11.5})\\ &= \frac{n_1 N^2}{4}.\end{aligned}$$

Comparing the two bounds completes the proof. □

Lemma 11.7 (Large input layer)**.** *If $n_1 \geq N/2$, then:*

$$\widehat{C}(n_1,\dots,n_L) \leq n_1^2(n_2+\cdots+n_L) = \widehat{C}(n_1, n_2+\cdots+n_L).$$

Proof. The assumption that $n_1 \geq N/2 = (n_1+\cdots+n_L)/2$ implies that $n_1 \geq n_2+\cdots+n_L$, and in particular we have $n_1 \geq n_k$ for all $k \geq 1$. Therefore, by the definition of the estimated capacity (11.1), we have:

$$\begin{aligned}\widehat{C}(n_1,\dots,n_L) &\leq n_1^2 \sum_{k=1}^{L-1} n_{k+1} \quad (\text{since } \min(n_1,\dots,n_k) \leq n_1 \text{ and } n_k \leq n_1)\\ &= n_1^2(n_2+\cdots+n_L).\end{aligned}$$

On the other hand, by the definition of the estimated capacity (11.1), we also have:

$$\widehat{C}(n_1, n_2+\cdots+n_L) = n_1^2(n_2+\cdots+n_L).$$

This completes the proof. □

Proof of Theorem 11.4. Consider the case $n_1 \leq N/2$ first. Because the capacity is bounded by the estimated capacity (Proposition 5.2), Lemma 11.6 gives:

$$C(n_1,\dots,n_L) \leq \widehat{C}(n_1,\dots,n_L) \leq \frac{n_1 N^2}{4} = \widehat{C}\Big(n_1, \frac{N}{2}, \frac{N}{2}-n_1\Big).$$

Furthermore, by Theorem 9.4, the estimated capacity is equivalent to the actual capacity, i.e.

$$C\Big(n_1, \frac{N}{2}, \frac{N}{2}-n_1\Big) \asymp \widehat{C}\Big(n_1, \frac{N}{2}, \frac{N}{2}-n_1\Big).$$

This yields the first part of the conclusion.

We can argue similarly in the case $n_1 \geq N/2$. Indeed, using Lemma 11.7, we obtain:

$$C(n_1,\dots,n_L) \leq \widehat{C}(n_1,\dots,n_L) \leq n_1^2(n_2+\cdots+n_L) = \widehat{C}(n_1, n_2+\cdots+n_L).$$

Finally, Theorem 2.11 yields:

$$C(n_1, n_2+\cdots+n_L) = C(H^{n_1})(n_2+\cdots+n_L) \approx n_1^2(n_2+\cdots+n_L) = \widehat{C}(n_1, n_2+\cdots+n_L).$$

This completes the proof of the theorem. □

Remark 11.8 (Optimal single-output newtorks). One can state similar results for single-output architectures $A(n_1, ..., n_L, 1)$, because their capacities are equivalent to the capacities of $A(n_1,\dots,n_L)$ (Corollary 3.2). We skip the details.

11.3. **Minimizing capacity.** In the theorems above we have maximized the capacity. It is also possible to minimize the capacity and here too, everything else being equal, we find that capacity tends to be minimized by deep architectures. For example, we have the theorem:

Theorem 11.9. *Consider the set of architectures of the form $A(n_1, \ldots, n_L, 1)$ with $L \geq 2$. Assume that n_1 is fixed, and that either the number of connections W or the number of nodes N is fixed. In either case, the capacity is minimized by the deepest possible architecture with $n_2 = n_3 = \cdots = n_L = 1$.*

Proof. By definition, we must have at least one unit in each hidden layer, and each layer must be fully connected to the following layer. By Theorem 2.11, the first hidden layer contributes at least $n_1^2(1 + o(1))$ to the capacity and this number is minimized by having a single unit in the first hidden layer. If we stack layers of size 1 above this layer, the capacity remains unchanged and thus is minimized. Note that in this case the number of layers L is dictated by the value of W or N. Thus, the minimal capacity is attained by the architecture $A(n_1, 1, \ldots, 1)$. □

12. Structural regularization

Some have attributed the power of deep networks to the ability of being able to compute more functions. The results of the previous section, summarized in Corollary 3.4, show that this cannot be the case as the opposite is true: everything else being equal, capacity tends to be maximized by *shallow* networks. However the functions computable by shallow and deep networks are different. For example, R. Eldan and O. Shamir [14] found that a three-layer network with moderate-sized hidden layers is able to compute certain functions that a two-layer network is unable to compute, unless its hidden layer has exponential size. Thus, the emerging picture is that *deeper networks with the same number of nodes compute fewer but more sophisticated functions*. This lead to the notion of *structural regularization*.

It has often been noted that deep networks have a tendency to avoid overfitting, even when the size of the training set is small compared to the number of parameters W ([27] and references therein). Some of this affect has been attributed to the regularizing properties of the main learning algorithm–stochastic gradient descent, and its inherent tendency to converge towards critical points with relatively broad basins of attraction (e.g. [28] and references therein). However, the results presented here show that there is a major regularization associated with deep architectures that is purely structural and independent of the learning algorithm: compared to shallow networks, deep networks compute fewer functions, but these functions tend to be "smoother and more sophisticated". The functions we see in practice are a tiny fraction of the universe of all possible functions, but they are the most interesting ones. And deep networks are able to "focus" on them. To see this more formally we can look at the behavior of various architectures on real-valued inputs. The situation is very different in the one-dimensional case, versus all other higher dimensional cases, as shown in the following results. In the one-dimensional case, the behavior of the architecture depends exclusively on the size of the first hidden layer and adding hidden layers does not increase the space of functions that can be implemented.

Proposition 12.1. *The set $T(1, n_2, \ldots, n_L, 1)$ consists of all piecewise-constant functions $f : \mathbb{R} \to \{0, 1\}$ with at most n_2 points of discontinuity. In particular, this class is determined entirely by n_2 alone.*

Proof. The first hidden layer, through the n_2 biases, creates n_2 potential points of discontinuity. Since there is a single output, every function $f \in T(1, n_2, \ldots, n_L, 1)$ must be constant,

and equal to 0 or 1 on each of the corresponding $n_2 + 1$ regions. It is possible to select the units in the hidden layer such that the leftmost region is coded by the vector $(0, 0, \ldots, 0)$ in the hidden layer, the second leftmost regions is coded by the vector $(1, 0, \ldots, 0)$, the third leftmost region is coded by the vector $(1, 1, \ldots, 0)$, and so forth until the rightmost region which is coded by $(1, 1, \ldots 1)$. The corresponding matrix, augmented with the vector $1, 1, \ldots 1)$ to account for the bias has full rank $n_2 + 1$. Therefore, by selecting the proper weights and biases, any value 0 or 1 can be assigned by the architecture to each one of the regions. □

Proposition 12.2. *The set of functions $T(n, m, 1)$ is characterized first by a splitting of $\mathbb{R}^n$ into at most $L(m, n) = \sum_{k=0}^{n} \binom{m}{k}$ regions, each one of which produces a constant binary vector in the hidden layer, and then the assignment of a 0 or 1 output to each region which can be achieved in at most:*

$$2 \sum_{k=0}^{m} \binom{L(m,n) - 1}{k} \leq m \log_2 L(m, n)$$

different ways (the last inequality assumes $m \geq 4$).

Proof. The proof is easily obtained by using Theorem 2.6 to obtain the number $L(m, n)$ of regions, noting that each region is mapped into a fixed vector in the hidden layer, and them applying Lemma 2.7 with $|S| = L(m, n)$. □

As an example, consider the class of $A(2, m, 1)$ architectures. The hidden layer gives rise to m affine lines that partition the input space $\mathbb{R}^2$ into $(m^2 + m + 2)/2 \approx m^2/2$ regions. The number of possible binary assignments to these regions scales like $2^{m^2/2}$. While in principle the output unit could have capacity m^2 and thus be able to handle all these assignments, in reality is capacity is reduced because only $m^2/2$ vectors, out of all possible 2^m vectors, are seen in the hidden layer. Thus the capacity of the output unit is considerably reduced to be at most: $m \log_2(m^2/)2$, using the standard upperbound on the capacity of sets.

The same approach can be applied to deep architectures.

Proposition 12.3. *The set of functions $T(n_1, n_2, \ldots, n_L, 1)$ is characterized first by a splitting of $\mathbb{R}^{n_1}$ into $L(n_2, n_1) = \sum_{k=0}^{n_1} \binom{n_2}{k}$ regions. Each one of these regions is mapped to a fixed binary vector in the first hidden layer, creating a set $S \subset \mathbb{R}^{n_2}$. The capacity of the number of functions that can be computed by the upper part of the architecture is given by $C(S, n_2, n_3, \ldots, n_L, 1)$ and can be bounded using the results in Sections 5 and 10.5.*

In short, the emerging intuitive picture is that the first hidden layer determines the number of regions into which the input space is fractured. The overall function is constant in each one of these regions, irrsepective of its depth. The larger the first hidden layer is, the greater the number of such regions. A network with a single, non-exponential hidden layer, has limited power in terms of assigning values to these regions. A deep network with the same number of parameters and hence a smaller first hidden layer will fracture the input in less regions and thus its output will have fewer regions of discontinuity. On the other hand the deep network will be able to compute more complex assignments to these regions.

13. Polynomial threshold functions

In search of more accurate models for biological neurons, or more powerful computational models, one may replace the linear activation with a polynomial activation of degree d in the input variables, usually using a lower degree polynomial. Recently, we were able to develop a theory for the capacity $C_d(n, 1)$ of a single polynomial threshold gate [8] with n

inputs, generalizing Zuev's result (Theorem 2.11) for all $d \geq 1$ and showing that $C_d(n,1) = [n^{d+1}/d!](1+o(1))$. The set and network capacity results presented here should be extended to feedforward networks of polynomial threshold functions. We present a first step in that direction beyond what is already in [8]. First we have the following theorem which generalizes Lemma 2.7.

Theorem 13.1 (Polynomial set capacity)**.** *Consider a finite subset $S \subset \mathbb{R}^n$, where $n > 1$. Then, for any degree $1 < d \leq n$, we have:*

$$C_d(S) \leq \log_2 \left(2 \sum_{k=0}^{M(n,d)-1} \binom{|S|-1}{k}\right) \leq (M(n,d)-1)\log_2 |S| \leq \left(\frac{2en}{d}\right)^d \log_2 |S|.$$

where:

$$M(n,d) = \sum_{k=0}^{d} \binom{n+k-1}{k} \leq \sum_{k=0}^{d} \binom{2n}{k} \leq \left(\frac{2en}{d}\right)^d.$$

Proof. First, it is easy to see that the number of coefficients of a polynomial of degree d in n variables $x_1, \ldots, x_n$ is given by $M(n,d)$, including the constant term (bias). A vector $x \in \mathbb{R}^n$ can be canonically and injectively mapped into a vector $f(x) \in \mathbb{R}^{M(n,d)-1}$ whose components are the various monomials. Using this mapping, we can represent any polynomial $p(x)$ of degree d over $\mathbb{R}^n$ as a linear affine function over $f(x)$. And vice versa, any linear affine function over $f(x)$ is a polynomial of degree d over x. For example, if $d = 2$, the vector $x = (x_1, x_2) \in \mathbb{R}^2$ is canonically mapped to the vector $f(x) = (x_1, x_2, x_1x_2, x_1^2, x_2^2) \in \mathbb{R}^5$. Any polynomial $p(x) = a_0 + a_1x_1 + a_2x_2 + a_{12}x_1x_2 + a_{11}x_1^2 + a_{22}x_2^2$ over $\mathbb{R}^2$ is clearly an affine function of $f(x)$, and vice versa. Therefore:

$$C_d(S) = C_1(f(S)) = C(f(S)).$$

We complete the proof by applying Lemma 2.7 for the set $f(S) \subset \mathbb{R}^{M(n,d)-1}$, noting that $f(S)$ has the same cardinality as S since f is injective. Note that when $n \geq 2$ and $d \geq 2$, $M(n,d) \geq 6$ which is required for the application of the second part of Lemma 2.7. □

Note that if we apply Theorem 13.1 to $S = H^n$, we get:

$$C_d(H^n) \leq n\left(\frac{2en}{d}\right)^d,$$

which is somewhat weaker asymptotically than the result in [8] giving:

$$C_d(H^n) = C_d(n,1) = \frac{n^{d+1}}{d!}(1+o(1)).$$

Note also that the general lower bound: $1 + \log_2 |S| \leq C_d(S)$, and its improved version when S is a subset of the Boolean cube: $\log_2^2 |S|/16 \leq C_d(S)$ are trivially satisfied.

Using Theorem 13.1, we can now prove the first result for fully-connected feedforward networks of polynomial threshold gates of degree d with a single hidden layer:

Theorem 13.2 (Polynomial capacity of a single-hidden-layer network)**.** *Consider a feedforward, fully-connected, feedforward network $A(n,m,1)$ of polynomial threshold gates of fixed degree d. If $n \to \infty$ and $\log_2 m = o(n)$ then:*

$$m\frac{n^{d+1}}{d!}(1+o(1)) \leq C_d(n,m,1) \leq m\frac{n^{d+1}}{d!} + \min\left[\frac{m^{d+1}}{d!}, n\left(\frac{2em}{d}\right)^d\right].$$

Proof. The proof follows somewhat what happens in the case $d = 1$, using the result in [8] for single units. The capacity of the hidden layer alone is given by:

$$m\frac{n^{d+1}}{d!}(1 + o(1))$$

and it is easy to check that the multiplexing technique can equally be applied to this case. This immediately yields the lower bound:

$$m\frac{n^{d+1}}{d!}(1 + o(1)) \leq C_d(n, m, 1).$$

For the upper bound, the total capacity is bounded by the sum of the capacity of all the units. The capacity of each unit in the hidden layers is bounded by $n^{d+1}/d!$. The capacity of the output unit is bounded by $m^{d+1}/d!$. For the output unit, using Theorem 13.1, its capacity is also bounded by:

$$n\Big(\frac{2em}{d}\Big)^d$$

if $m > n$, and by:

$$m\Big(\frac{2nm}{d}\Big)^d$$

if $m \leq n$. This completes the proof. □

Note that in the case where $m^d/n^{d+1} = o(1)$ we get:

$$C_d(n, m, 1) = m\frac{n^{d+1}}{d!}(1 + o(1)).$$

The work presented here naturally leads to several open research questions. We briefly mention a few.

14. Open questions

14.1. **Polynomial threshold functions.** The initial results given above on polynomial threshold functions need to be extended in several directions. We leave the tightening of the capacity result in Theorem 13.2 and its extensions to networks with multiple hidden layers of polynomial threshold functions of degree d for future work. The same is also true for any extensions of the lower bound on set capacity in Theorem 7.1 to polynomial threshold functions of degree d.

14.2. **Asymptotic tightness.** Theorem 3.1 presents a capacity formula that is accurate within an absolute constant factor. Is this formula asymptotically tight? In other words, is it true that:

$$C(n_1, \ldots, n_L) = (1 + o(1)) \sum_{k=1}^{L-1} \min(n_1, \ldots, n_k) n_k n_{k+1}$$

as the number of nodes n_k for some (or all) layers k increases to infinity?

The upper bound in Theorem 3.1 is indeed tight (Proposition 5.2), but we only know that lower bound is asymptotically tight for networks with a single hidden layer (Corollary 8.4). And even beyond that, can the $1 + (o(1)$ term be further improved, in the same way that for single neurons the result in [16] refines the capacity estimate in [30]?

14.3. **Restricted capacity.** In Sections 5 and 10.5 we estimated the restricted capacity $C(S, n_1, n_2, \ldots, n_L, 1)$ for a general input set $S \subset H^n$. Our lower bound (Proposition 10.5) is tight within a logarithmic factor. Can this factor be removed?

To do so, one may try to follow the proof of the lower bound in Theorem 3.1 and use Theorem 7.1 instead of the estimate $C(H^n) \asymp n^2$ in this argument. But this reasoning meets an obstacle: we do not know a version of the Enrichment Theorem 9.1 for a general set S.

Thus, an important related problem is to generalize the Enrichment Theorem 9.1 for a general set $S \subset H^n$. Is it true that there exists an injective map $F : S \to H^m$ all of whose components are threshold functions and such that:

$$C(S) \asymp m \log |S|?$$

14.4. **Other transfer functions.** This paper focused exclusively on networks with the threshold (Heaviside) transfer function $h = \mathbf{1}_{\{t \geq 0\}}$. One may wonder if our results hold true for other transfer functions, such as the ReLU function $\max(0, t)$, or the sigmoidal transfer functions (e.g. logistic $1/(1 + e^{-t})$ or $\tanh t$) that are commonly used in neural networks. Suppose some general transfer function $f(t)$ is applied at each hidden layer. As long as the inputs are from a finite set and the threshold function is applied at the output nodes, the set of functions that can be implemented by the architecture remains finite and the same definition of capacity can be applied without the need for any adjustments. How many functions can the network compute?

Our preliminary analysis, left for further investigations, suggests that the capacity might not depend too much on the shape of the transfer function f, provided that f is a piecewise-constant function that consists of more than one piece, but less than an exponential number of pieces. Thus, the capacity formula in Theorem 3.1 might be universal for the class of networks with piecewise-constant transfer functions.

Beyond piecewise-constant, we can consider piecewise-linear transfer functions, such as ReLU functions. We have recently shown that the capacity of a single unit is increased by a ReLU transfer function, but only marginally as it remains equal to $n^2(1 + o(1))$ [6]. We conjecture that, in essence, the same remains true for multilayer networks with ReLU transfer functions in the hidden layers. In light of the known results ([10, 9]) that show that the VC-dimension of ReLU neural networks may grow *super-linearly* with the depth L, it is interesting to find out if the capacity of the networks (e.g. with equal sizes of layers) grows super-linearly as well.

The definition of capacity used here relies here on a class of functions or hypotheses T that is finite. However, the definition could clearly be extended to networks that can compute an infinite number of functions, for instance using real-valued inputs and continuous transfer functions everywhere, including in the output layer. For this purpose, one could still define capacity by $C = \log_2 |T|$ but defining $|T|$ in a broader measure theoretic sense (e.g. volume).

14.5. **Other connectivity models.** Finally, it is possible to consider several other connectivity models, either by having more sparse connections or by constraining the values of the connections. A first example is to consider synaptic weights that are constrained to belong to a finite set, for instance $\{-1, 1\}$ (binary synapses). It is easy to see that the capacity of a binary synapse linear threshold neuron is exactly equal to the number of inputs n [6], but the extension to multiple layers has not been studied. Likewise, it is possible to consider the case where all the incoming, or outgoing, synaptic weights must have the same sign (e.g. purely excitatory or purely inhibitory neurons). We have shown that, for a single linear threshold unit, if all the incoming weights are positive the capacity of the unit is marginally decreased

but remains in the same class and equal to $n^2(1 + o(1))$. Again the extension to multiple layers has not been studied. Finally, there is the case of local connectivity, where the indegree or outdegree of a neuron may be restricted. How can the theory be extended to some of those cases? Consider a typical convolutional neural network for computer vision applications. In this case, a convolutional layer may comprise an array of n neurons, where each neuron has an identical set of $m \times m$ incoming weights, the so-called weight-sharing approach. Because the weights are tied, the entire layer can implement only one set of weights, and thus only one function. If the neurons are modeled as linear threshold gates, the capacity of the entire layer can be estimated, and is equal to $m^4(1 + (o(1))$. Thus, in short, under the right assumptions the methods presented here can readily be applied to convolutional neural networks. All the previous examples, assumed feedfoward patterns of connections. Extensions to recurrent networks, other than the fully-connected case, have not been investigated.

15. Conclusion

In the 1940s, McCulloch and Pitts [17] and others introduced a simple neuronal model, whereby a neuron processes information by first computing an activation A and then an output $O = f(A)$. The activation A is typically a weighted linear, or polynomial, function of the inputs. The transfer function f is typically a non-linear function, such as a threshold function, ReLU (rectified linear unit) function or more generally a piecewise linear function, or sigmoidal function (e.g. logistic, tanh).

Networks of McCulloch and Pitts model neurons are important for at least four fundamental reasons. First, although far simpler than biological neurons in their processing details, these simplified neural models have proven over and over to be useful to better understand biological networks (e.g. [29, 20, 26]). Second, these models are also used to guide the development of new, power efficient, neuromorphic chips [19]. Third, these neural network models are widely used today in all kinds of AI/deep learning applications with impressive results, often matching or exceeding human capabilities in specific tasks across the gamut of applications, from games to biomedicine (e.g.[22, 23, 4]). And finally, from a foundational standpoint, they are the dominant, and perhaps simplest, available analytical model for studying the neural style of storing information.

Indeed, in the standard description of McCulloch and Pitts neurons given above, the emphasis is placed on the processing aspect of these models, the input-output relationship. However, equal or even more emphasis should be given to the storage aspect of this neural model. Information about the world, e.g. in the form of "training sets", is stored in a distributed "holographic" way in the synaptic weights (i.e. the coefficients of the activation function), through a learning process. This is significant because to achieve intelligent behavior, information processing systems must be able to learn and store information. To store information, there are two completely different approaches: (1) the Turing tape model, where information is stored at well organized, discrete, addresses of a physical substrate–this is the style used in all living systems at the cellular level (DNA), and in all our digital devices, from cell phones to supercomputers; (2) the neural model, where information is stored "holographically" in neural networks across large numbers of synapses–this is the style believed to be used by the brain, and simulated in our neural network–deep learning–technology. While the Turing style is relatively well understood, the neural style is not.

In this paper, we set out to study the most fundamental property of the neural style of storage, namely how many bits can be stored in a given neural architecture. To address this question, we first had to introduce the notion of cardinal capacity, the logarithm base two of

the number of different functions a given architecture can compute. Remarkably, for neural architectures, the cardinal capacity is equal to the total number of bits that can be stored in a given architecture, or the number of bits that can be "communicated" from the outside world to the architecture by the learning process. We then estimated the capacity of feedforward neural architectures of arbitrary depth, under a relatively mild set of assumptions on the connectivity and the transfer functions of these architectures. The capacity is typically a cubic polynomial in the sizes of the layers. For fully connected, feedforward, architectures it is essentially given by: $C(n_1, \ldots, n_L) \approx \sum_{k=1}^{L-1} \min(n_1, \ldots, n_k) n_k n_{k+1}$. As a side note, the capacity of fully connected recurrent networks can also be estimated [6], essentially by unfolding them in time and computing the capacity of the underlying feedforward network. In addition, we have improved the bounds on the capacity of sets, analyzed the extremal properties of the capacity and the structural regularization effects of deep architectures, and began to extend the theory of capacity to polynomial threshold functions. In addition, we have briefly surveyed several open questions in this area.

Finally, although this falls beyond the scope of this paper, the capacity is a fundamental quantity that can be related to other measures of complexity and generalization including the VC dimension, the growth function, the Rademacher and Gaussian complexity, the metric entropy, and the minimum description length (MDL). For example, if the function h to be learnt as MDL D, and the neural architecture being used as capacity $C < D$ then it is easy to see that: (1) h cannot be learnt without errors; and (2) the number E of errors made by the best approximating function implementable by the architecture must satisfy $E > (D-C)/N$. Another example of connection is the connection to the VC dimension that was used in the bounds in Section 10.5. These connections will be described more systematically elsewhere.

Appendix: Examples

In this appendix, we apply the main result to a few basic architectures, assuming the layers are large so that the asymptotic regimes can be applied.

First, consider a deep architecture $A(n_1, \ldots, n_L)$ which is expansive, where expansive is defined by the property: $n_1 \leq n_2 \leq \ldots \leq n_{L-1}$. Then, using the main result:

$$C(A) \approx n_1 \sum_{k=1}^{L-1} n_k n_{k+1} \approx n_1 W.$$

Second, consider a deep architecture $A(n_1, \ldots, n_L)$ which is compressive, where compressive is defined by the property: $n_1 \geq n_2 \geq \ldots \geq n_{L-1}$. Then, using the main result:

$$C(A) \approx \sum_{k=1}^{L-1} n_k^2 n_{k+1}.$$

Third, we consider autoencoder architectures $A(n, m, n)$ with a single hidden layer (Figure 5). Clearly $W = 2nm$. For the capacity, there are two cases depending on whether the autoencoder is expansive or compressive. In the compressive case ($m < n$), $C(n, m, n) \approx mn^2 + m^2 n = mn(n + m) \approx mn^2 = nW/2$. If we let $m = n^{1-\nu}$ for $0 < \nu < 1$, then:

$$C(n, m, n) \approx n^{3-\nu}.$$

In the expansive case ($m > n$), $C(n, m, n) \approx mn^2 + mn^2 = 2mn^2 = nW$. If we let $m = n^{1+\nu}$ for $0 < \nu$, then:

$$C(n, m, n) \approx 2n^{3+\nu}.$$

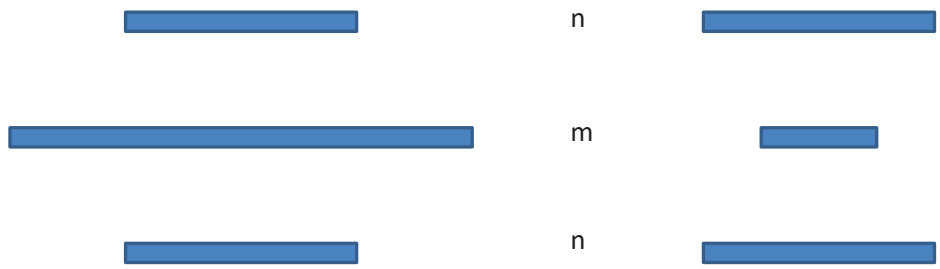

Figure 5. Expansive (left) and compressive (right) autoencoder architectures $A(n, m, n)$.

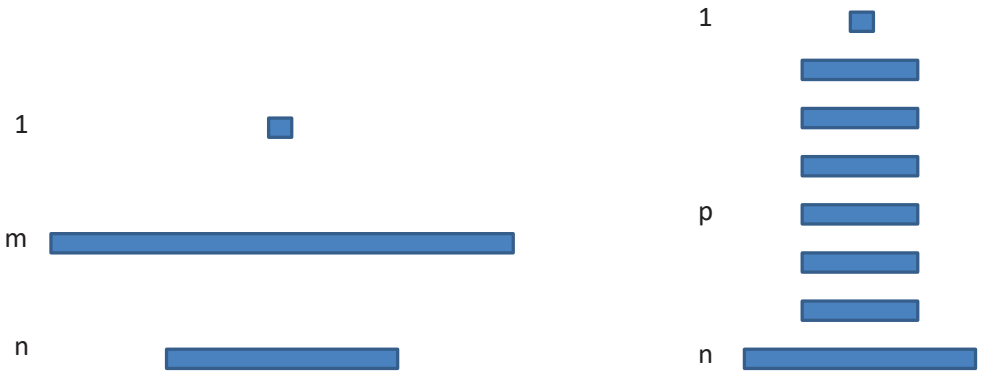

Figure 6. Shallow (left) $A(n, m, 1)$ architecture and deep (right)$A(n, p, p, ..., p, 1)$ architecture for classification.

Finally, we contrast a shallow $A(n, m, 1)$ and a deep $A(n, p, \ldots, p, 1)$ classification architectures (Figure 6). For the shallow architecture, we have:

$$W \approx nm \quad \text{and} \quad C \approx mn^2.$$

For the deep architecture, we have:

$$W \approx np + Lp^2 \quad \text{and} \quad C \approx pn^2 + Lp^3$$

if $p \leq n$, and:

$$W \approx np + Lp^2 \quad \text{and} \quad C \approx pn^2 + Lnp^2$$

if $p > n$. Here L is a parameter that represents the depth–the entire architecture has $L + 2$ layers, not counting the single-unit output layer. Consider, for instance, the expansive case where $m \geq n$ and $p \geq n$. Then both architectures satisfy: $C \approx nW$ and will have roughly the same capacity when they have roughly the same number of parameters. If we let $m = n^{1+\alpha}$ ($\alpha \geq 0$), $p = n^{1+\beta}$ ($\beta \geq 0$), and $L = n^{\gamma}$ ($\gamma \geq 0$) then for the architectures to have approximately the same number of parameters (and thus approximately the same capacity), one must have: $2\beta + \gamma = \alpha$. The other cases can be analyzed similarly.

References

[1] Martin Anthony. *Discrete mathematics of neural networks: selected topics*, volume 8. Siam, 2001.

[2] P. Baldi. Autoencoders, Unsupervised Learning, and Deep Architectures. *Journal of Machine Learning Research. Proceedings of 2011 ICML Workshop on Unsupervised and Transfer Learning*, 27:37–50, 2012.

[3] P. Baldi. Boolean autoencoders and hypercube clustering complexity. *Designs, Codes, and Cryptography*, 65(3):383–403, 2012.

[4] P. Baldi. Deep learning in biomedical data science. *Annual Review of Biomedical Data Science*, 1:181–205, 2018.

[5] P. Baldi and A. F. Atiya. Oscillations and synchronizations in neural networks: an exploration of the labeling hypothesis. *International Journal of Neural Systems*, 1(2):103–124, 1989.

[6] P. Baldi and R. Vershynin. On neuronal capacity. In *NIPS 2018*. Accepted for oral presentation.

[7] Pierre Baldi and Peter Sadowski. A theory of local learning, the learning channel, and the optimality of backpropagation. *Neural Networks*, 83:61–74, 2016.

[8] Pierre Baldi and Roman Vershynin. Boolean polynomial threshold functions and random tensors. *arXiv preprint arXiv:1803.10868*, 2018.

[9] Peter L Bartlett, Nick Harvey, Chris Liaw, and Abbas Mehrabian. Nearly-tight vc-dimension and pseudodimension bounds for piecewise linear neural networks. *arXiv preprint arXiv:1703.02930*, 2017.

[10] Peter L Bartlett, Vitaly Maiorov, and Ron Meir. Almost linear vc dimension bounds for piecewise polynomial networks. In *Advances in Neural Information Processing Systems*, pages 190–196, 1999.

[11] Eric B Baum and David Haussler. What size net gives valid generalization? In *Advances in neural information processing systems*, pages 81–90, 1989.

[12] Thomas M Cover. Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE Transactions on Electronic Computers*, (3):326–334, 1965.

[13] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4):303–314, 1989.

[14] Ronen Eldan and Ohad Shamir. The power of depth for feedforward neural networks. In *Conference on Learning Theory*, pages 907–940, 2016.

[15] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[16] Jeff Kahn, János Komlós, and Endre Szemerédi. On the probability that a random±1-matrix is singular. *Journal of the American Mathematical Society*, 8(1):223–240, 1995.

[17] W. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 7:115–133, 1943.

[18] Saburo Muroga. Lower bounds of the number of threshold functions and a maximum weight. *IEEE Transactions on Electronic Computers*, (2):136–148, 1965.

[19] Emre O. Neftci, Somnath Paul, Charles Augustine, and Georgios Detorakis. Event-Driven Random Back-Propagation: Enabling Neuromorphic Deep Learning Machines. *Frontiers in Neuroscience*, 11, 2017.

[20] Bruno A Olshausen and David J Field. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381(6583):607, 1996.

[21] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[22] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.
[23] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.
[24] R. Vershynin. *High-dimensional probability. An introduction with applications in data science*. Cambridge University Press, 2018.
[25] RO Winder. Partitions of n-space by hyperplanes. *SIAM Journal on Applied Mathematics*, 14(4):811–818, 1966.
[26] Daniel LK Yamins and James J DiCarlo. Using goal-driven deep learning models to understand sensory cortex. *Nature neuroscience*, 19(3):356–365, 2016.
[27] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 2016.
[28] Zhanxing Zhu, Jingfeng Wu, Bing Yu, Lei Wu, and Jinwen Ma. The anisotropic noise in stochastic gradient descent: Its behavior of escaping from minima and regularization effects. *arXiv preprint arXiv:1803.00195*, 2018.
[29] David Zipser and Richard A Andersen. A back-propagation programmed network that simulates response properties of a subset of posterior parietal neurons. *Nature*, 331(6158):679–684, 1988.
[30] Yu A Zuev. Asymptotics of the logarithm of the number of threshold functions of the algebra of logic. *Soviet Mathematics Doklady*, 39(3):512–513, 1989.
[31] Yu A Zuev. Combinatorial-probability and geometric methods in threshold logic. *Diskretnaya Matematika*, 3(2):47–57, 1991.

Department of Computer Science, University of California, Irvine
E-mail address: pfbaldi@uci.edu

Department of Mathematics, University of California, Irvine
E-mail address: rvershyn@uci.edu