Deep-layered machines have a built-in Occam's razor, vs 1.4

Thomas Fink

London Institute for Mathematical Sciences, Royal Institution, 21 Albermarle St, London W1S 4BS, UK

Input-output maps are prevalent throughout science and technology. They are empirically observed to be biased towards simple outputs, but we don't understand why. To address this puzzle, we study the archetypal input-output map: a deep-layered machine in which every node is a Boolean function of all the nodes below it. We give a mathematical theory for the distribution of outputs, and we confirm our predictions through extensive computer experiments. As the network depth increases, the distribution becomes exponentially biased towards simple outputs. This suggests that deep-layered machines and other learning methodologies may be inherently biased towards simplicity in the models that they generate.

Introduction

This paper presents a unified understanding of two seemingly different scientific puzzles. The first is the observed tendency of input-output maps to be biased towards simple outputs. The second is the success of deep-layered machines and other learning frameworks at producing parsimonious solutions. Our overall approach is to introduce a predictive theory for the output of deep-layered machines, show that this output is biased towards simplicity, and, by regarding learning frameworks as input-output maps, argue that they have a built-in Occam's razor.

Input-output maps are prevalent in biology, physics, mathematics and technology [1, 2]. The inputs can be thought of as instructions, and the outputs can be thought of as functions. Input-output maps tend to be many-to-one because a lot of different instructions produce the same function—there's more than one way to skin a cat.

One example of an input-output map is RNA folding, in which nucleotide sequences (input) fold to RNA secondary structures (output). Another is protein folding, in which sequences of amino acids fold to 3D molecular shapes [3]. In logic circuits [4] and Boolean networks [5, 6], local logics (input) generate global dynamics (output). In 2D models of self-assembly, polyominoes (input) combine to form finite or periodic shapes (output). A similar process occurs in 3D when proteins self-assemble into protein complexes. In neural networks, synapse weights (input) determine the overall function of the arguments (output).

If we pick a random input, we might well expect a random output. After all, there's no *a priori* reason to expect one output over another. But in fact most input-output maps are empirically observed to be exponentially biased towards simple outputs [1, 2]. They are simple in the broad sense of possessing low Kolmogorov complexity—they have short description lengths.

Learning can be thought of as an input-output map with a constraint on the output. Consider, for example, protein design. The target is a protein conformation constrained to have a particular active site. The task is to find a sequence that folds to one of the many conformations that have the active site. Similar reasoning applies to an associative memory. The target is a classifier that maps, say, cat-like images to cats and dog-like images to dogs; how other images get mapped is incidental. The task is to find a set of weights that yields one of the many valid cat and dog classifiers. As we shall see, just as Occam's razor prescribes the simplest explanation that fits the facts, deep-layered machines are biased towards the simplest outputs that meet the constraints.

In this paper we study the archetypal input-output map: a deep-layered machine in which each node is a Boolean function, or logic, of all of the nodes below it (see Fig. 1). In particular, we do four things, which correspond to the next four sections. One, we give a mathematical theory of the distribution of outputs given a random choice of inputs. Two, we confirm our theory by doing extensive computer experiments for networks of different sizes. Three, we show that the output distribution becomes exponentially biased towards simple functions as the network depth increases. Four, we conjecture that this bias is part of a more general phenomenon in which the repeated application of irreversible rules gives rise to a bias towards simple outputs. If so, it suggests that a broad range of learning frameworks are biased towards simplicity in the models that they generate.

A puzzle

For a conceptual understanding of the simplicity bias in deep-layered machines, consider a puzzle about a group of friends. Each person has one of two moods: happy or sad. Your own mood depends on the moods of Alice and Bob. For instance, you might be happy only if Alice and Bob are happy. Or you might ignore Alice and copy Bob. There are 16 such dependencies, each of which occurs with the same chance: around 6%.

Now imagine that the mood of Alice depends, in turn, on Carol and Dan, and the same is true of Bob. Again, there are 16 dependencies for Alice and 16 for Bob. So ultimately your mood is governed by the moods of Carol and Dan. There are 16^3 ways of configuring this puzzle (the inputs), but only 16 ways in which you can depend on Carol and Dan (the outputs). The $k = 2$ architecture in Fig. 1 shows the friendship network, and Fig. 2 shows the complete input-output map.

You might think your dependence on Carol and Dan is uniformly distributed, since the local dependencies are assigned uniformly. But in fact you are biased towards simple dependencies. For instance, the chance of ignoring Carol and Dan and always being happy is 17%. The chance of being happy if either person is (happy $^3/_4$ of the time) is 5%. The chance of copying one and ignoring the other (happy $^1/_2$ of the time) is 4%.

The favored dependencies are simple in that they are happy most of the time or sad most of the time. Were we to extend the game such that the mood of Carol and Dan each depends on Eve and Frank, the bias would be stronger still.

Distribution of outputs

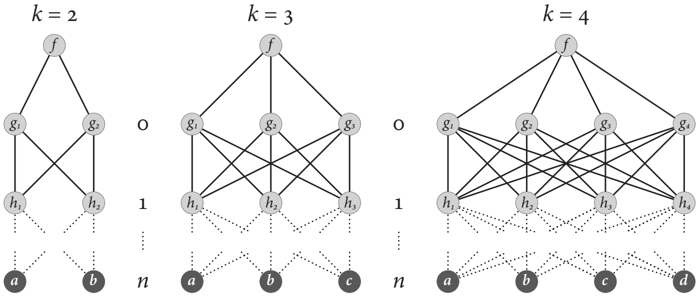

FIG. 1: **Deep-layered machines.** In a network of k arguments (a, $b, \ldots$), each logic depends on all k of the arguments below it, each of which depends on the k arguments below it, and so on, down to n levels. Our goal is to determine the distribution of $f(a, b, \ldots)$ (the output) given a random assignment of logics to f; $g_1, g_2, \ldots$; and so on (the input).

In this section we work out the probability of the output of a deep-layered machine given a random input, that is, a random assignment of logics to the light gray nodes in Fig. 1.

Distribution for small k

In a deep-layered machine, each Boolean function, or logic for short, depends on the k arguments in the layer below it. There are 2^{2^k} logics of k arguments. Thus the number of inputs grows as $\left(2^{2^k}\right)^{nk+1}$: the number of logics per node to the power of the number of nodes in the network. The number of outputs is much smaller: just 2^{2^k}.

For $k = 1$, there are four logics: true, false, a and $\overline{a}$ (not a). There are 4^{n+1} inputs but only four outputs. We can write down the probabilities of the outputs explicitly: the probability of a and $\overline{a}$ are both $1/2^{n+2}$, and the probability of true and false are both ${}^1\!/\!{}_2 - 1/2^{n+2}$.

For $k = 2$, there are 16 logics, shown in Table I Top. In a network of depth $n = 1$, which is the puzzle described above, the output is $f(g_1(a, b), g_2(a, b))$ (see Methods for examples of logic composition). There are 16^3 inputs (ways of assigning logics to f, g_1 and g_2), but only 16 outputs. A visual representation of this is shown in Fig. 2. The probabilities of different output functions are shown in Table I Top for various values of n.

For $k = 3$, there are 256 logics, the truth tables of which are given in Table I. The probabilities of different outputs are shown in Table I Bottom.

Distribution for general k

We want to know the probability of any given output function for general k. What we find is that the probability depends only on the Hamming weight of the output function, that is, the number of 1s in its truth table, which ranges from 0 to 2^k. For this reason, we don't need to keep track of 2^{2^k} probabilities, but rather just $2^k + 1$. We call this vector of probabilities $\mathbf{x}(n)$. In Table I, $\mathbf{x}(n)$ is given by the columns on the right: for $k = 2$, $\mathbf{x}(0) = ({}^1\!/\!{}_{16}, {}^1\!/\!{}_{16}, {}^1\!/\!{}_{16}, {}^1\!/\!{}_{16}, {}^1\!/\!{}_{16})$, and so on.

Notice how there are $\binom{2^k}{w}$ output functions with a given value of w. For example, for $k = 2$, there are $1, 4, 6, 4$ and 1 functions for $w = 0, \ldots, 4$. So the probability that an output function has Hamming weight w is given by the vector

$$\mathbf{z}_i = \binom{2^k}{w}\mathbf{x}_i. \tag{1}$$

Thus for $k = 2$, $\mathbf{z}(0) = ({}^1\!/\!{}_{16}, {}^4\!/\!{}_{16}, {}^6\!/\!{}_{16}, {}^4\!/\!{}_{16}, {}^1\!/\!{}_{16})$. Throughout this paper we work with the more natural $\mathbf{z}$, since its components sum to one. But we are ultimately interested in $\mathbf{x}$, which gives the probabilities of specific outputs. Eq. (1) is how we translate between them.

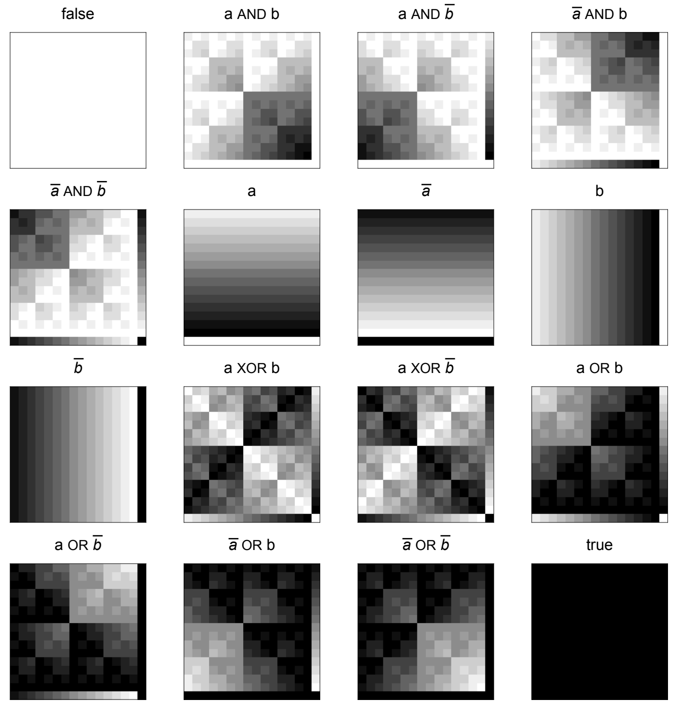

FIG. 2: **Inputs-output map for two arguments.** For $k = 2$ arguments and network depth $n = 1$, there are 16^3 inputs but only 16 outputs. The outputs are indicated by the gray level, from white to black, which correspond to false to true (in the order given in Table I Top). The inputs are the assignments of logics to f, g_1 and g_2 in $f(g_1(a, b), g_2(a, b))$. Each panel is a different choice of f, and within each panel are the 16×16 choices of g_1 and g_2.

Logic functions		*Hamming*	*Probability of function*			
of $k=2$ arguments		*weight w*	$n=0$	$n=1$	$n=2$	$n=3$
false	0000	0	$\frac{1}{16}$	$\frac{680}{16^3}$	$\frac{261056}{16^5}$	$\frac{83663360}{16^7}$
ab	1000	1	$\frac{1}{16}$	$\frac{216}{16^3}$	$\frac{42048}{16^5}$	$\frac{8087040}{16^7}$
$a\overline{b}$	0100	1				
$\overline{a}b$	0010	1				
$\overline{a}\overline{b}$	0001	1				
a	1100	2	$\frac{1}{16}$	$\frac{168}{16^3}$	$\frac{31680}{16^5}$	$\frac{6068736}{16^7}$
$\overline{a}$	0011	2				
b	1010	2				
$\overline{b}$	0101	2				
$a\oplus b$	0110	2				
$a\oplus\overline{b}$	1001	2				
$a+b$	1110	3	$\frac{1}{16}$	$\frac{216}{16^3}$	$\frac{42048}{16^5}$	$\frac{8087040}{16^7}$
$a+\overline{b}$	1101	3				
$\overline{a}+b$	1011	3				
$\overline{a}+\overline{b}$	0111	3				
true	1111	4	$\frac{1}{16}$	$\frac{680}{16^3}$	$\frac{261056}{16^5}$	$\frac{83663360}{16^7}$

Logic functions	*Hamming*	*Prob. of function*	
of $k = 3$ arguments	*weight w*	$n=0$	$n=1$
00000000	0	$\frac{1}{256}$	$\frac{136761984}{256^4}$
00000001, 00000010, …	1	$\frac{1}{256}$	$\frac{40611200}{256^4}$
00000011, 00000101, …	2	$\frac{1}{256}$	$\frac{19714688}{256^4}$
00000111, 00001011, …	3	$\frac{1}{256}$	$\frac{13086080}{256^4}$
00001111, 00010111, …	4	$\frac{1}{256}$	$\frac{11457152}{256^4}$
00011111, 00101111, …	5	$\frac{1}{256}$	$\frac{13086080}{256^4}$
00111111, 01011111, …	6	$\frac{1}{256}$	$\frac{19714688}{256^4}$
01111111, 10111111, …	7	$\frac{1}{256}$	$\frac{40611200}{256^4}$
11111111	8	$\frac{1}{256}$	$\frac{136761984}{256^4}$

TABLE I: **Distribution of output functions. Top.** For $k = 2$ arguments, there are 16 logic functions, which can also be expressed by their binary truth tables. In our notation, $\overline{a}$ means NOT a, ab means a AND b, $a\oplus b$ means a XOR b (exclusive or), and $a+b$ means a OR b. For network depth $n = 0$, 1, 2 and 3, we show the probability of producing each of the output functions. The probability depends only on the Hamming weight w of the function, that is, the number of 1s in the truth table. **Bottom.** For $k = 3$ arguments, there are 256 logics, which we express by their binary truth tables. They are grouped by their Hamming weight w. For network depth $n = 0$ and 1, we show the probability of each of the output functions in the Hamming weight group.

There exists a $2^k + 1$ by $2^k + 1$ transition matrix $\mathbf{A}$ such that

$$\mathbf{z}(n) = \mathbf{A}^n \mathbf{z}(0). \tag{2}$$

The elements of the matrix $\mathbf{A}$ satisfy

$$\mathbf{A}_{i,j} = \frac{1}{\ell^\ell}\binom{\ell}{j} i^j (\ell - i)^{\ell - j},$$

where for convenience we set $\ell = 2^k$, and we take $0^0 = 1$, a common convention in combinatorics. For example, for $k = 1$,

$$\mathbf{A} = \frac{1}{2^2}\begin{pmatrix} \binom{2}{0} & & \\ & \binom{2}{1} & \\ & & \binom{2}{2} \end{pmatrix}\begin{pmatrix} 0^0 2^2 & 1^0 1^2 & 2^0 0^2 \\ 0^1 2^1 & 1^1 1^1 & 2^1 0^1 \\ 0^2 2^0 & 1^2 1^0 & 2^2 0^0 \end{pmatrix}.$$

For $k = 2$,

$$\mathbf{A} = \frac{1}{4^4}\begin{pmatrix} \binom{4}{0} & & & & \\ & \binom{4}{1} & & & \\ & & \binom{4}{2} & & \\ & & & \binom{4}{3} & \\ & & & & \binom{4}{4} \end{pmatrix}\begin{pmatrix} 0^0 4^4 & 1^0 3^4 & 2^0 2^4 & 3^0 1^4 & 4^0 0^4 \\ 0^1 4^3 & 1^1 3^3 & 2^1 2^3 & 3^1 1^3 & 4^1 0^3 \\ 0^2 4^2 & 1^2 3^2 & 2^2 2^2 & 3^2 1^2 & 4^2 0^2 \\ 0^3 4^1 & 1^3 3^1 & 2^3 2^1 & 3^3 1^1 & 4^3 0^1 \\ 0^4 4^0 & 1^4 3^0 & 2^4 2^0 & 3^4 1^0 & 4^4 0^0 \end{pmatrix}.$$

Properties of the transition matrix

The matrix $\mathbf{A}$ has $2^k + 1$ eigenvalues and eigenvectors; see the Methods for the $k = 2$ example. The first two eigenvalues are $\lambda_1 = \lambda_2 = 1$, corresponding to the eigenvectors $(1, 0, \ldots, 0)$ and $(0, \ldots, 0, 1)$. As we confirm in Methods, the jth eigenvalue is

$$\lambda_j = \frac{(\ell)_{j-1}}{\ell^{j-1}}, \tag{3}$$

where $\ell = 2^k$ and $(\ell)_j = \ell(\ell - 1)\ldots(\ell - j + 1)$ is the falling factorial. Thus in the limit of large depth n, the output function is true and false each with probability $1/2$, with the probabilities of all other outputs vanishing.

However, the situation is more interesting than the large-n limit suggests. Notice how the first two eigenvectors tell us nothing about the bulk of the outputs, that is, all of the $2^\ell - 2$ functions that are not true and false. For large depth n, the shape of $\mathbf{z}(n)$ for the bulk is rather given given by the third eigenvector $\mathbf{v_3}$ of $\mathbf{A}$. We don't know how to write it down explicitly, but we can show that it is approximately flat apart from the endpoints. In particular, the ratio of the smallest and largest components of $\mathbf{v_3}$ is at least $(1 - e)/e = 0.632$ and at most 1. As we shall see, this flatness is key to our main result, namely, that the distribution of output functions is exponentially biased towards simple functions.

Comparison with computer experiments

To confirm our theoretical predictions, we conducted extensive computer experiments for various values of the number of arguments k and network depth n. In all cases, our computer experiments match our theory. The computational cost of enumerating all possible inputs is formidable: it grows as $\left(2^{2^k}\right)^{nk+1}$. Therefore our experiments include complete enumeration of the inputs when possible, and sampling from the ensemble of inputs otherwise.

For $k = 2$ arguments (Fig. 1A), there are $16^3, 16^5, 16^7$ and 16^9 input configurations for network depths $n = 1, 2, 3$ and 4. We enumerated all of these inputs and, for each, determined the network's output function. Since the probability of an output is the same for outputs with the same Hamming weight w, we plot the probability $\mathbf{z}(n)$ of obtaining a given w in Fig. 3A (points). This exactly matches our theoretical predictions given by eq. (2). The solid line indicates the probabilities of the bulk of the outputs and the dotted line connects to the outputs false ($w = 0$) and true ($w = 4$). As n increases, the likelihood of true and false approach $1/2$ and the likelihoods of the remaining outputs fall, bearing in mind that the likelihoods sum to one.

For $k = 3$ arguments (Fig. 1B), there are 256^4, 256^7 and

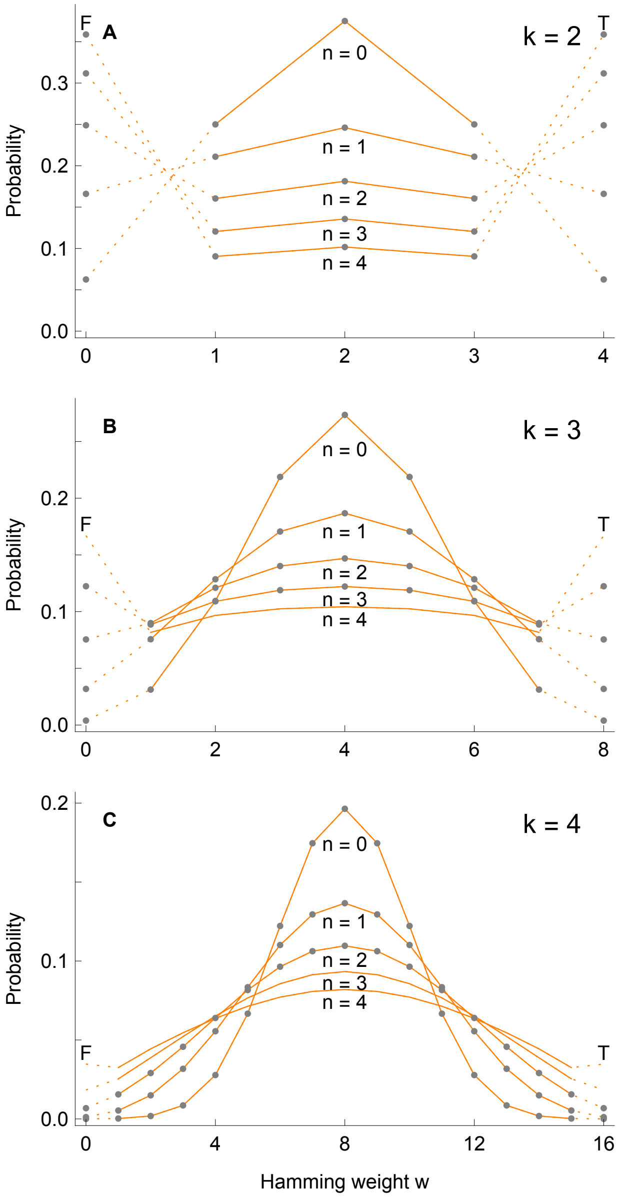

FIG. 3: **Computer experiments confirm our theory.** We compare our theoretical predictions of $\mathbf{z}(n)$ given by eq. (2) (lines) with computer experiments (points), for various values of the number of arguments k and the network depth n. The vertical axis shows the probability that the network produces an output function with a given Hamming weight w (the number of 1s in its truth table), since outputs with the same w have the same probability. In all cases, our experiments agree with our theory exactly or, when sampling, to within statistical significance. **A** For $k = 2$, we enumerated all of the input configurations up to network depth $n = 4$. As n increases, the distribution of the output function flattens out and falls. But for false and true ($w = 0$ and $w = 4$), the probabilities approach one half. **B** For $k = 3$, we show exact results for $n = 0$ and 1, and sample the inputs for $n = 2$ and 3. We show our $n = 4$ theory for comparison. **C** For $k = 4$, we show exact results for $n = 0$, and sample the inputs for $n = 1$ and 2. We show our $n = 3$ and 4 theory for comparison.

256^{10} input configurations for network depths $n = 1, 2$ and 3. For $n = 1$, we were able to enumerate all of the inputs. For $n = 2$ and 3, this is computationally infeasible, so instead we sampled the inputs. We randomly assigned one of the 256 logics to each of the nodes and then determined the network output, repeating this two million times. These are plotted in Fig. 3B (points). We calculated errors bars, but these are negligible compared to the point size in the plots. Our theory predicts the computer experiments exactly for $n = 1$ and to within statistical significance for $n = 2$ and 3. We also show our $n = 4$ predictions for comparison.

For $k = 4$ arguments (Fig. 1C), there are 65536^5 and 65536^9 inputs for network depths $n = 1$ and 2. These are too many to enumerate, so we took two million samples for $n = 1$ and the same for $n = 2$. These are plotted in Fig. 3C. Once again, our theory predicts the experiments to within statistical significance. We also show our $n = 3$ and $n = 4$ predictions for comparison.

Bias towards simplicity

To our surprise, we find that the distribution of the output function is biased towards simplicity in an easily quantifiable way. This effect gets stronger as the network depth n increases.

At network depth $n = 0$, the distribution of the 2^{2^k} logic functions is uniform: the probability of each is $1/2^{2^k}$. But at $n = 1$, where we first start to combine logics, some outputs become more likely than others. The probability of a given output depends solely on its Hamming weight w; all functions with a given Hamming weight have the same probability. For $k = 2$ and $k = 3$, these probabilities are given in Table I.

Information content of a function

The simplicity of a logic function can be measured by its information content, where simpler functions contain less information.

A logic can be represented by its binary truth table of 2^k bits. This just specifies the value of the function for all possible combinations of its k arguments, where 0 is false and 1 is true. For example, if the function is abc (a AND b AND c), the truth table is 10000000 (adopting the Mathematica convention for ordering), since the function is true only when all three of its inputs are true. If the function is false, the truth table is 00000000.

A logic function of k arguments with Hamming weight w can be uniquely indicated by first specifying its Hamming weight, of which there $2^k + 1$ possibilities, then by specifying which of the functions with the given Hamming weight it is, of which there are $\binom{2^k}{w}$ possibilities. Thus the information content of a logic function f is at most

$$I(f) = \log_2 \binom{2^k}{w} + \log_2(2^k + 1). \qquad (4)$$

For example, for the function abc described above, which has $w = 1$, $I = \log_2 \binom{8}{1} + \log_2 9 = 6.2$ bits. For the function false, which has $w = 0$, $I = \log_2 \binom{8}{0} + \log_2 9 = 3.2$ bits.

Probability versus information content

We show the bias towards simplicity in Fig. 4 for $k = 4$, 6 and 8 arguments, and various values of the network depth n. For $n = 0$, the distribution of outputs is flat—all output functions are equally likely. As n increases, the distribution changes in two different ways. First, the bulk of the outputs (everything but true and false) becomes exponentially biased towards simple outputs. Second, the probability of obtaining true and false each approaches 1/2 and the distribution of the bulk vanishes. The first effect governs the shape of the bulk distribution, whereas the second shrinks the bulk distribution along the vertical axis.

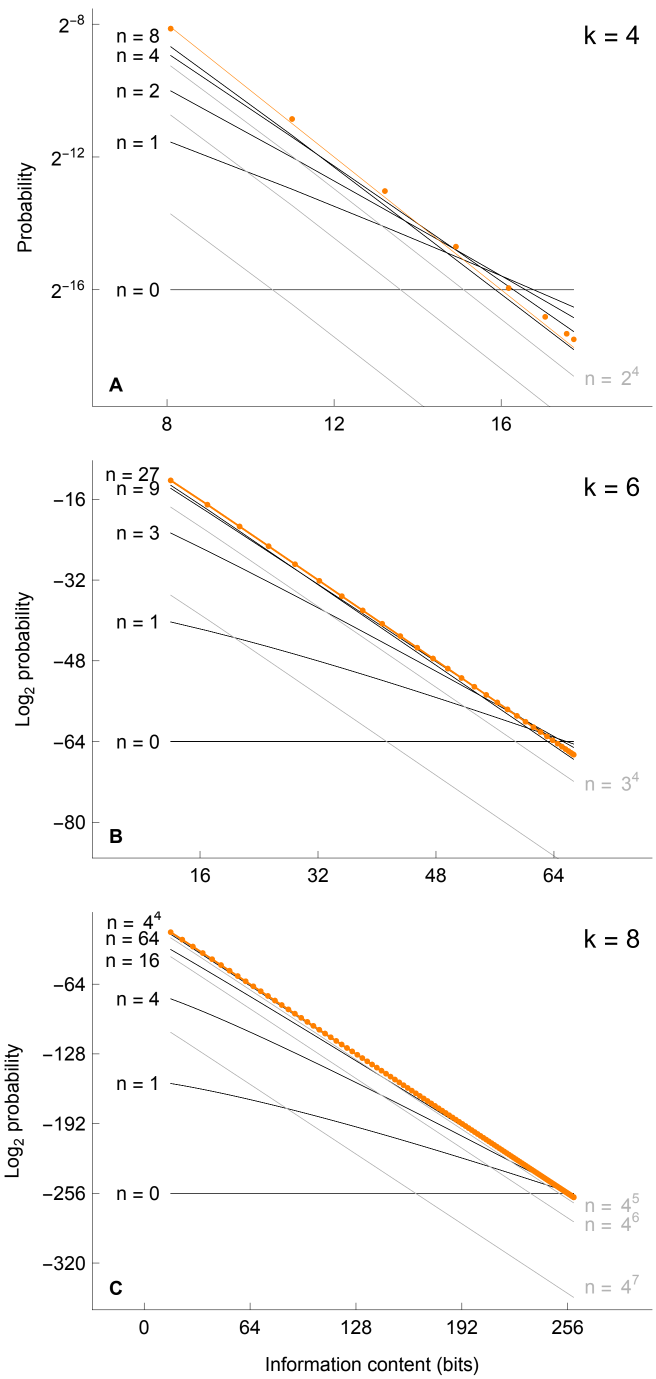

FIG. 4: **Probability of an output function versus its information content.** As the network depth n increases, the logarithm of the probability distribution of the output rotates clockwise from horizontal to a nearly straight line with slope −1. On a slower time scale, it also falls as the outputs true and false (not shown here) dominate. The black curves show the distribution rising and the gray curves show it falling. The points (orange online) are the third eigenvector of the transition matrix $\mathbf{A}$, which governs the shape of the bulk, translated from $\mathbf{z}$ to $\mathbf{x}$ via eq. (1); the slope of the orange line which they approach is −1. **A** For $k = 4$ arguments, we show the distribution for $n = 0$ and powers of 2. **B** For $k = 6$ arguments, we show it for $n = 0$ and powers of 3. **C** For $k = 8$, we show it for $n = 0$ and powers of 4.

For clarity the rising curves are black and the falling curves are gray.

Notice how, at its highest level, the distribution approaches the third eigenvector of the transition matrix $\mathbf{A}$, translated from $\mathbf{z}$ to $\mathbf{x}$ via eq. (1). This is indicated in Fig. 4 by the points (orange online). As n increases, the distribution appears to get closer and closer to it before falling away. If we take $\mathbf{z}(n)$, for large n, to be strictly rather than approximately flat, then all of the components are $1/(2^k+1)$, and the probability $P(I)$ is $1/\binom{2^k}{w} \cdot 1/(2^k+1)$, and by eq. (4) $\log_2 P(I) = -I$. In Fig. 4 this is the orange line which the orange points approach.

Discussion

In this paper we give the exact solution to the output of a deep-layered machine with randomly chosen Boolean functions, or logics, at each node. This is to our knowledge the first exact solution to a non-trivial input-output map. As well as ordinary digital computing, our deep-layered machine encompasses discretized neural networks in which the logics are threshold Boolean functions.

Critical network depth

Crucially, the bias of the bulk towards simplicity happens faster than true and false take over—the distribution flattens out before it shrinks. We have verified this computationally, as shown in Fig. 3 from the $\mathbf{z}$ perspective and in Fig. 4 from the $\mathbf{x}$ perspective. The reason is that the spectral gap $\lambda_1 - \lambda_3 = \lambda_2 - \lambda_3 = 1/2^k$, which governs the equilibration of true and false, is smaller than the spectral gap $\lambda_3 - \lambda_4 = (1 - 1/2^k)\, 2/2^k$, which governs the equilibration of the bulk. As k increases, the latter gap approaches twice the former gap.

Intriguingly, these two time scales—one for the endpoints and one for the bulk—implies the existence of a critical network depth n_{crit} beyond which the simplicity bias breaks down and the network becomes dominated by true and false. In Fig. 4, this is the depth at which the distribution goes from black (rising) to gray (falling). Since the equilibration time is proportional to the inverse of the spectral gap, n_{crit} grows as 2^k. A separate argument, given in the Methods, also suggests that true and false start to dominate around network depth 2^k.

As an aside, there is an intuitive proof that true and false must dominate eventually. At each level in the network, there is a finite probability that all of the k logics are true, independent of the level n. When this happens, the output function f must be true, regardless of what happens farther down the network. Thus the probability of true and false each asymptotically approach $^1/_2$ (though faster than this lower bound argument suggests). Mozeika *et al.* [11] observed a similar phenomena when they considered deep-layered machines with rectified linear unit functions: "random deep ReLU networks compute only *constant* Boolean functions in the infinite depth limit".

Generalizations and deep learning

In the architecture we considered, every logic is a function of all k of the arguments below it. It is regular in the sense that it contains all possible loops (multiple paths to the same point), and this regularity is key to its solvability. In separate work, not yet published, we considered the opposite extreme: no loops, but rather a uniform branching structure. For example, instead of $f(g_1(a,b), g_2(a,b))$ ($k = 2$ and $n = 1$), we studied $f(g_1(a,b), g_2(c,d))$ (branching degree 2 and $n = 1$). For this branching architecture we also observed an exponential bias towards simplicity.

We conjecture that the bias towards simplicity described in this paper is just one instance of a more general phenomenon: the repeated application of irreversible local rules generates a global bias towards simplicity. In other words, simplicity bias in input-output maps is not the exception but the rule.

Methods

Representing and composing logics

In our notation, $\overline{a}$ means NOT a, ab means a AND b, $a \oplus b$ means a XOR b (exclusive or), and $a + b$ means a OR b. The order of operations is AND takes precedence over XOR, which takes precedence over OR.

The composition of logics works just like the composition of ordinary functions. When composing logics by hand, it's convenient to write them in disjunctive normal form, which consists of a disjunction of conjunctions. In other words, we write them as ORs of ANDs, or sums of products.

Let's work out a couple of examples for $k = 2$ arguments and network depth $n = 1$. The output of the network is

$$f(g_1(a,b), g_2(a,b)). \tag{5}$$

Set $g_1 = a$ OR b and $g_2 = \overline{a}$ OR $\overline{b}$, that is,

$$g_1 = a + b, \qquad g_2 = \overline{a} + \overline{b}.$$

If we set $f = g_1$ AND g_2, then

$$f = g_1 g_2 = (a+b)(\overline{a} + \overline{b}) = a\overline{b} + \overline{a}b = a \oplus b,$$

that is, $f = a$ XOR b. But if we set $f = g_1$ OR g_2, then

$$f = g_1 + g_2 = a + b + \overline{a} + \overline{b} = \text{true}.$$

To take this to the next level ($n = 2$), in eq. (5) we would replace a with $h_1(a,b)$ and b with $h_2(a,b)$.

Example of the eigenvalues and eigenvectors

For $k = 2$, the transition matrix $\mathbf{A}$ has five eigenvalues λ and eigenvectors $\mathbf{v}$:

$$\begin{array}{llccccc}
\lambda_1 = 1 & \mathbf{v}_1 = & (0 & 0 & 0 & 0 & 1) \\
\lambda_2 = 1 & \mathbf{v}_2 = & (1 & 0 & 0 & 0 & 0) \\
\lambda_3 = {}^3/_4 & \mathbf{v}_3 = & (-1 & {}^{16}/_{25} & {}^{18}/_{25} & {}^{16}/_{25} & -1) \\
\lambda_4 = {}^3/_8 & \mathbf{v}_4 = & (-1 & 2 & 0 & -2 & 1) \\
\lambda_5 = {}^3/_{32} & \mathbf{v}_5 = & (1 & -4 & 6 & -4 & 1).
\end{array}$$

Then we can write

$$\mathbf{z}(n) = \mathbf{A}^n \mathbf{z}(0) = c_1 \mathbf{v_1} + c_2 \mathbf{v_2} + c_3 \lambda_3^n \mathbf{v_3} + c_4 \lambda_4^n \mathbf{v_4} + c_5 \lambda_5^n \mathbf{v_5}.$$

The projection of the initial condition $\mathbf{z}(0)$ onto the eigenvectors is $\mathbf{V}^{-1}\mathbf{z}(0)$, where $\mathbf{V}$ is the matrix with the eigenvectors as its columns. For $\mathbf{z}(0) = ({}^1/_{16}, {}^4/_{16}, {}^6/_{16}, {}^4/_{16}, {}^1/_{16})$, this gives $(c_1, c_2, \ldots) = ({}^1/_2, {}^1/_2, {}^{25}/_{56}, 0, {}^1/_{112})$. Then the probability distribution of the output having Hamming weight w is

$$\mathbf{z}(n) = {}^1/_2 \mathbf{v_1} + {}^1/_2 \mathbf{v_2} + {}^{25}/_{56} ({}^3/_4)^n \mathbf{v_3} + {}^1/_{112} ({}^3/_{32})^n \mathbf{v_5}.$$

Leading eigenvector for the bulk is flat

The third eigenvector of the transition matrix $\mathbf{A}$ is the principal eigenvector for the bulk of the outputs (everything but true and false). We are unable to work it out explicitly, but we can show that, apart from the endpoints, it is approximately flat. Let $\mathbf{B}$ be the $2^k - 1$ by $2^k - 1$ matrix that is the interior of $\mathbf{A}$, that is, everything but the outer edge. The principal eigenvector of $\mathbf{B}$ is the interior of the third eigenvector of $\mathbf{A}$.

We know, in general, that the principal eigenvector is at least as flat as the column sums of the matrix that it satisfies. For our matrix $\mathbf{B}$, with $\ell = 2^k$, the column sums are

$$\sum_{j=1}^{\ell-1} \mathbf{B}_{ij} = \frac{1}{\ell^\ell} \sum_{j=1}^{\ell-1} \binom{l}{j} i^j (\ell - i)^{\ell-j}.$$

If we extend the bounds in the sum to 0 and ℓ, by the binomial theorem the sum is just 1. So we know that

$$\sum_{j=1}^{\ell-1} \mathbf{B}_{ij} = 1 - \left(\frac{i}{\ell}\right)^\ell - \left(\frac{\ell - i}{\ell}\right)^\ell. \tag{6}$$

This is minimized when $i = 1$ and $i = \ell - 1$, and maximized when $i = \ell/2$. For even modest values of k, $\ell = 2^k$ is large, and the minimum and maximum values of the sum tend to $(1 - e)/e$ and 1. Thus in the limit of large ℓ,

$$\sum_{j=1}^{\ell-1} \mathbf{B}_{ij} \in \left[\frac{1-e}{e}, 1\right], \tag{7}$$

where $(1 - e)/e$ is 0.632. For example, for $k = 3$, the minimum and maximum column sums are 0.656 and 0.992. So the ratio of the smallest and largest components of the interior of the third eigenvector of $\mathbf{A}$ is at least $(1 - e)/e$ and at most 1. (Cf. the third eigenvector for $k = 2$ above.)

Critical network depth

The first two eigenvalues govern the leading behavior of the endpoints, and the third eigenvalue governs the leading behavior of everything else (the bulk). Let the initial state be $\left(\binom{2^k}{0}, \binom{2^k}{1}, \ldots, \binom{2^k}{2^k}\right)$. The first two terms in the projection onto the eigenvectors are both $1/2$, and call the third c_3. Keeping just the first three terms,

$$\mathbf{z}(n) \simeq 1/2\mathbf{v_1} + 1/2\mathbf{v_2} + c_3((1 - 1/2^k)^n \mathbf{v_3}.$$

From above, we know the interior of $\mathbf{v_3}$ is approximately flat. If we take it to be strictly flat, then

$$\mathbf{z}(n) = (1/2, 0, \ldots, 0, 1/2) + c_3(1 - 1/2^k)^n(-1, \tfrac{2}{\ell-1}, \ldots, \tfrac{2}{\ell-1}, -1).$$

True and false start to dominate when the probability of the endpoints equals the probability of the interior, that is,

$$2\big(1/2 - c_3(1 - 1/2^k)^{n_{\text{crit}}}\big) = 1/2,$$

which gives

$$n_{\text{crit}} = 2^k \ln(4c_3),$$

where $c_3 \sim 1$.

Comparing the trace and the sum of the eigenvalues

In general the sum of the eigenvalues of a matrix is equal to its trace. We show that this is the case for the transition matrix $\mathbf{A}$ as a confirmation of the form of the eigenvalues in eq. (3). The trace is

$$\text{tr}(\mathbf{A}) = \frac{1}{\ell^\ell} \sum_{j=0}^{\ell} \binom{\ell}{j} j^j (\ell - j)^{\ell-j},$$

where we take $0^0 = 1$, a common convention in combinatorics. Then, by Abel's binomial theorem,

$$\begin{aligned} \text{tr}(\mathbf{A}) &= \frac{1}{\ell^\ell} \sum_{j=0}^{\ell} \frac{\ell!}{j!} \ell^j \\ &= \frac{1}{\ell^\ell} \sum_{j=0}^{\ell} \binom{\ell}{j} (\ell - j)!\, \ell^j. \end{aligned}$$

Since $\binom{l}{j}$ is symmetric, we can replace $(\ell - j)!\, \ell^j$ with $j!\, \ell^{\ell-j}$, so

$$\begin{aligned} \text{tr}(\mathbf{A}) &= \sum_{j=0}^{\ell} \binom{\ell}{j} \frac{j!}{\ell^j} \\ &= \sum_{j=0}^{\ell} \frac{(\ell)_j}{\ell^j} \\ &= \sum_{j=0}^{\ell} \lambda_j. \end{aligned}$$

Acknowledgements: The author acknowledges Evgeny Sobko for helpful discussions.

Competing interests: The author declares that he has no competing interests.

[1] K. Dingle, C. Q. Camargo, A. A. Louis, Input-output maps are strongly biased towards simple outputs, Nat Commun **9**, 761 (2018).

[2] I. G. Johnston et al., Symmetry and simplicity spontaneously emerge from the algorithmic nature of evolution, P Natl Acad Sci USA **119**, e2113883119 (2022).

[3] J. L. England, E. I. Shakhnovich, Structural determinant of protein designability, Phys Rev Lett **90**, 218101 (2003).

[4] S. E. Ahnert, T. M. A. Fink, Form and function in gene regulatory networks, J Roy Soc Interface **13**, 20160179 (2016).

[5] T. Fink, F. Sheldon. Number of cycles in the critical Kauffman model is exponential, Phys Rev Lett, **131**, 267402 (2023).

[6] F. Sheldon, T. Fink Insights from number theory into the critical Kauffman model with connectivity one, J Phys A, **57**, 275003 (2024).

[7] T. M. A. Fink and R. Hannam, Biological logics are restricted, arxiv.org/abs/2109.12551.

[8] T. M. A. Fink, On the number of biologically permitted logics, Nat Rev Genet

[9] N. J. A. Sloane, editor, The On-Line Encyclopedia of Integer Sequences, published electronically at https://oeis.org, 2021.

[10] K. Raman, A. Wagner, The evolvability of programmable hardware, J Roy Soc Interface **8**, 269 (2011).

[11] A. Mozeika, B. Li, D. Saad, The space of functions computed by deep layered machines, Phys Rev Lett **125**, 168301 (2020).