Why AI works: the iteration of simple rules in a fully-connected architecture imposes Occam's razor (vs 1.2)

Thomas Fink
London Institute for Mathematical Sciences, Royal Institution, 22 Albemarle St, London W1S 4BS, UK
(Dated: January 8, 2026)

We show that the repeated application of logics in a globally connected architecture gives rise to an exponential bias towards simple output functions. It suggests an explanation for why neural networks and other learning methodologies are biased towards simplicity in the models that they generate.

Introduction

Artificial intelligence is the most recent chapter in the long arc of automation. Mechanical automation has progressed for millennia and electronic automation for over a century. The novelty of machine learning is that, instead of using trial and error to find a program that generates a given output, it seeks to automatically reverse engineer the program from the output. There is no unique solution to this problem, because in general a vast number of programs produce the same output.

What is striking about neural networks and other learning methodologies is that the reverse-engineered program that generates the desired output tends to be simple. In this sense, there is a built-in Occam's razor lurking in AI. Just as Occam's razor prescribes the simplest explanation that fits the facts, there is an inclination for deep layered machines to generate the simplest program that generates the output.

In this paper we do two things. First, we give a mathematical formula for the distribution of the global function for k variables and depth n (see Fig. 1). This is confirmed by computer experiments that we did in prior work [13]. Second, we show that as the network depth n increases, the network function becomes exponentially biased towards simple functions.

Input-output maps. A broad class of physical, biological and mathematical input-output maps have been empirically observed to be biased towards simple outputs. This was first observed in biological systems, in which some phenotypes are generated by a vast number of genotypes, whereas others are much less designable in this sense. One of the best-studied example is RNA folding, in which RNA nucleotide sequences (programs) map to RNA secondary structures (functions). This system is exactly solvable because the space of conformations can be combinatorially enumerated by sequences of dots and brackets, indicating hairpins and loops. The designability of proteins has also attracted a lot of attention [1]. In small genetic circuits, many assingments of update rules lead to the same dynamical behavior. Similar effects have been found in simple Boolean networks and the self-assembly of polyominoes [4]. In all cases, the most designable outputs are also the simplest and most symmetric.

One of the earliest, albeit highly abstracted, studies of input-output maps is due to Solomonoff. He considered the probability that a random program fed into some universal Turing machine generates a given output. A classic result is that a string of arbitrary length can be compressed by q bits with probability $1/2^q$. Thus a random program is exponentially more likely to generate simple (in the sense of compressible) outputs.

Composition of logics. There are 2^{2^k} logic functions of k variables. For $n = 2$, the 16 logics are given in Fig. 2. 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. In the fully connected architecture (see Fig. 1), each logic is a function of the k variables in the layer below it. The goal of this paper is to understand the distribution of logics at depth n when we compose the logic functions.

To get a sense of how these logics are composed, consider the case of $k = 2$ variables in a network of depth $n = 1$. We want to know the distribution of outputs of $f(g_1(a,b), g_2(a,b))$. There are 16^3 ways of assigning logics to f, g_1 and g_2, but only 16 possible outputs. For example, if $f = g_1$ AND g_2, $g_1 = a$ OR b,

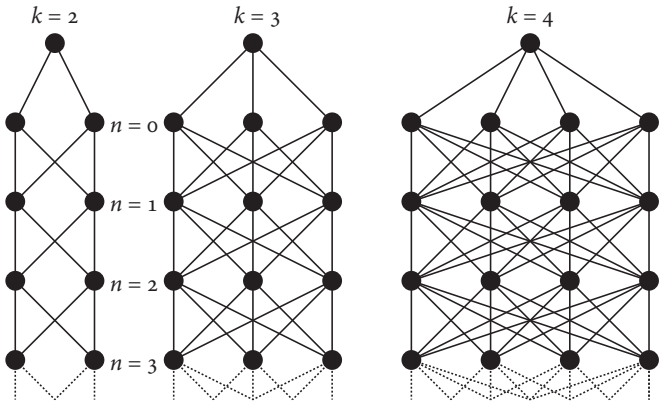

FIG. 1: **Fully connected architectures.** In an architecture of k variables, each logic depends on all k of the variables below it, each of which in turn depends on the k variables below it, and so on, down to n levels. This paper is concerned with the distribution of logics at a given depth n. What we find is an exponential bias towards simple logics (those with high or low Hamming weight w), which grows with depth n.

and $g_2 = \overline{a}$ OR $\overline{b}$, then $f = a$ XOR b. But if instead we set $f = g_1$ OR g_2, then $f =$ true. Running through all 4,096 assignments, we find that some functions are more designable than others, as shown by the numerators in the $n = 1$ column of Fig. 2. The spread becomes more pronounced for network depth $n = 2$, and so on.

In general there are $\left(2^{2^k}\right)^{nk+1}$ programs which produce 2^{2^k} functions. The degeneracy breaks into $2^k + 1$ classes, which depend only on the Hamming weight of the logic, that is, the number of 1s in its truth table, which can range from 0 to 2^k. For example, for $k = 2$, there are 5 classes, shown in Fig. 2.

Transition matrix and its properties

Transition matrix. Let $\mathbf{x}(n)$ be the distribution of the logic functions at depth n. For convenience let $\ell = 2^k$. There exists an $\ell - 1$ by $\ell - 1$ transition matrix $\mathbf{B}(k)$ such that

$$\mathbf{x}(n) = \mathbf{B}^n(k)\mathbf{x}(0). \tag{1}$$

Logic function		*Hamming weight* w	*Probability of function* $n=0$	$n=1$	$n=2$	$n=3$
false	0000	0	$1\cdot\frac{1}{16}$	$1\cdot\frac{680}{16^3}$	$1\cdot\frac{261056}{16^5}$	$1\cdot\frac{83663360}{16^7}$
ab	1000	1	$4\cdot\frac{1}{16}$	$4\cdot\frac{216}{16^3}$	$4\cdot\frac{42048}{16^5}$	$4\cdot\frac{8087040}{16^7}$
$a\overline{b}$	0100	1				
$\overline{a}b$	0010	1				
$\overline{a}\overline{b}$	0001	1				
a	1100	2	$6\cdot\frac{1}{16}$	$6\cdot\frac{168}{16^3}$	$6\cdot\frac{31680}{16^5}$	$6\cdot\frac{6068736}{16^7}$
$\overline{a}$	0011	2				
b	1010	2				
$\overline{b}$	0101	2				
$a\oplus b$	0110	2				
$a\oplus\overline{b}$	1001	2				
$a+b$	1110	3	$4\cdot\frac{1}{16}$	$4\cdot\frac{216}{16^3}$	$4\cdot\frac{42048}{16^5}$	$4\cdot\frac{8087040}{16^7}$
$a+\overline{b}$	1101	3				
$\overline{a}+b$	1011	3				
$\overline{a}+\overline{b}$	0111	3				
true	1111	4	$1\cdot\frac{1}{16}$	$1\cdot\frac{680}{16^3}$	$1\cdot\frac{261056}{16^5}$	$1\cdot\frac{83663360}{16^7}$

FIG. 2: **Distribution of logics for two variables.** For $k = 2$ variables, there are 16 logic functions, which can also be expressed by their binary truth tables. For $n = 0$, 1, 2 and 3 layers, we show the probability of producing the functions. Thus for $n = 0$ the probability of producing ab, $a\overline{b}$, $\overline{a}b$ and $\overline{a}\overline{b}$ is $1/16$ for each, and $4 \cdot 1/16$ for any of them. The probability depends only on the Hamming weight w of the function, that is, the number of 1s in the truth table. For large n, the probability of true and false each approach a half, with the other probabilities going to zero. But this happens slowly compared to the time it takes to equilibrate to the principle eigenvector of the matrix $\mathbf{B}$ (see Fig. 6).

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

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

For example, for $k = 2$,

$$\mathbf{B}(2) = \frac{1}{4^4}\begin{pmatrix}\binom{4}{1} & & \\ & \binom{4}{2} & \\ & & \binom{4}{3}\end{pmatrix}\begin{pmatrix}1^1 3^3 & 2^1 2^3 & 3^1 1^3 \\ 1^2 3^2 & 2^2 2^2 & 3^2 1^2 \\ 1^3 3^1 & 2^3 2^1 & 3^3 1^1\end{pmatrix}.$$

For $k = 3$,

$$\mathbf{B}(3) = \frac{1}{8^8}\begin{pmatrix}\binom{8}{1} & & & & & & & \\ & \binom{8}{2} & & & & & & \\ & & \binom{8}{3} & & & & & \\ & & & \binom{8}{4} & & & & \\ & & & & \binom{8}{5} & & & \\ & & & & & \binom{8}{6} & & \\ & & & & & & \binom{8}{7} & \end{pmatrix}\begin{pmatrix}1^1 7^7 & 2^1 6^7 & 3^1 5^7 & 4^1 4^7 & 5^1 3^7 & 6^1 2^7 & 7^1 1^7 \\ 1^2 7^6 & 2^2 6^6 & 3^2 5^6 & 4^2 4^6 & 5^2 3^6 & 6^2 2^6 & 7^2 1^6 \\ 1^3 7^5 & 2^3 6^5 & 3^3 5^5 & 4^3 4^5 & 5^3 3^5 & 6^3 2^5 & 7^3 1^5 \\ 1^4 7^4 & 2^4 6^4 & 3^4 5^4 & 4^4 4^4 & 5^4 3^4 & 6^4 2^4 & 7^4 1^4 \\ 1^5 7^3 & 2^5 6^3 & 3^5 5^3 & 4^5 4^3 & 5^5 3^3 & 6^5 2^3 & 7^5 1^3 \\ 1^6 7^2 & 2^6 6^2 & 3^6 5^2 & 4^6 4^2 & 5^6 3^2 & 6^6 2^2 & 7^6 1^2 \\ 1^7 7^1 & 2^7 6^1 & 3^7 5^1 & 4^7 4^1 & 5^7 3^1 & 6^7 2^1 & 7^7 1^1\end{pmatrix}$$

Eigenvalues. The transition matrix B has $\ell - 1$ eigenvalues, where recall $\ell = 2^k$. The principal eigenvalue is $\lambda_1 = (\ell - 1)/\ell$ and and the jth eigenvalue is

$$\lambda_j = \prod_{i=1}^{j} \frac{\ell - i}{\ell}.$$

Logic function	w	*Prob. of funct.* $n=0$	$n=1$
00000000	0	$1\cdot\frac{1}{256}$	$1\cdot\frac{136761984}{256^4}$
00000001, 00000010, ...	1	$8\cdot\frac{1}{256}$	$8\cdot\frac{40611200}{256^4}$
00000011, 00000101, ...	2	$28\cdot\frac{1}{256}$	$28\cdot\frac{19714688}{256^4}$
00000111, 00001011, ...	3	$56\cdot\frac{1}{256}$	$56\cdot\frac{13086080}{256^4}$
00001111, 00001111, ...	4	$70\cdot\frac{1}{256}$	$70\cdot\frac{11457152}{256^4}$
00011111, 00101111, ...	5	$56\cdot\frac{1}{256}$	$56\cdot\frac{13086080}{256^4}$
00111111, 01011111, ...	6	$28\cdot\frac{1}{256}$	$28\cdot\frac{19714688}{256^4}$
01111111, 10111111, ...	7	$8\cdot\frac{1}{256}$	$8\cdot\frac{40611200}{256^4}$
11111111	8	$1\cdot\frac{1}{256}$	$1\cdot\frac{136761984}{256^4}$

FIG. 3: **Distribution of logics for three variables.** For $k = 3$, there are 256 logics, which we express by their binary truth tables. As in Fig. 2, they are grouped by their Hamming weight w. For $n = 0$ and $n = 1$ layers, we show the probability of producing each of the functions in the Hamming weight group.

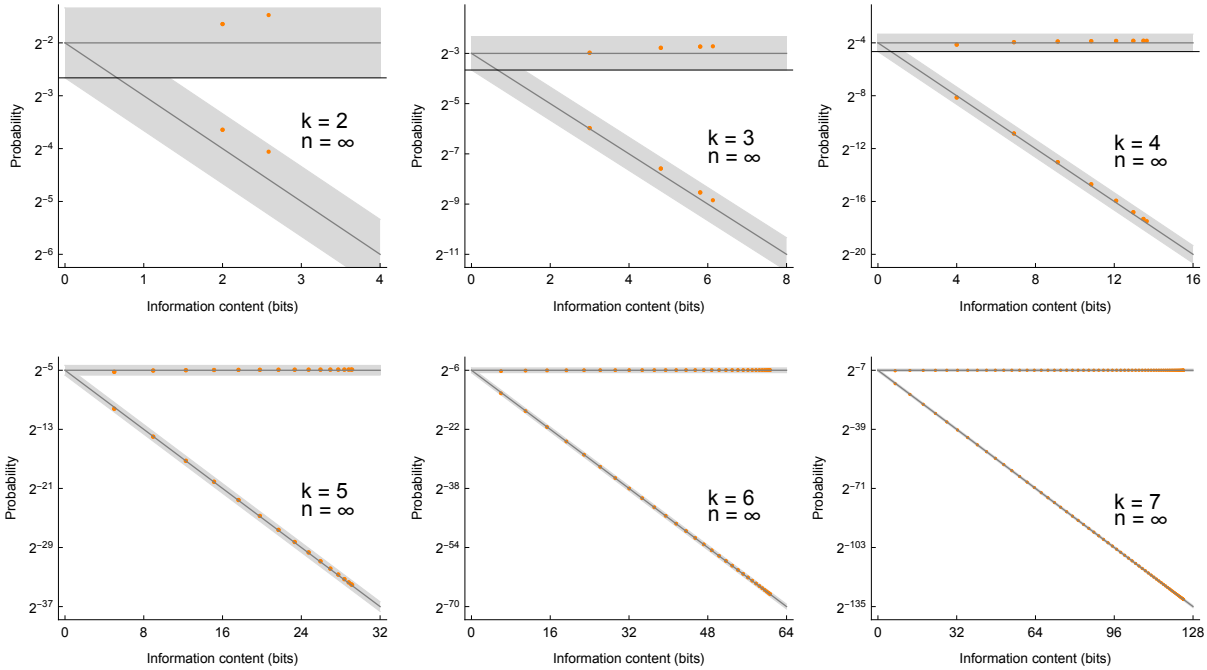

FIG. 4: **Likelihood of producing a function with a given Kolmogorov complexity.** The top straight curves show the probability that the output function can be compressed by k bits. The gray region indicates the effective error bars, that is, the bounds on the principal eigenvector given by eq. (3).

As a reality check, the sum of the eigenvalues is

$$\begin{aligned}\sum_{j=1}^{\ell-1}\lambda_j &= \sum_{j=1}^{\ell-1}\prod_{i=1}^{j}\frac{\ell-i}{\ell}\\ &= -2+\frac{\ell!}{\ell^\ell}\sum_{j=0}^{\ell}\frac{\ell^j}{j!}\\ &= \frac{\ell!}{\ell^\ell}\sum_{j=1}^{\ell-1}\frac{j^j}{j!}\frac{(\ell-j)^{\ell-j}}{(\ell-j)!}\\ &= \frac{1}{\ell^\ell}\sum_{j=1}^{\ell-1}\binom{\ell}{j}j^j(\ell-j)^{\ell-j},\end{aligned}$$

which is precisely the trace of $\mathbf{B}$, as expected.

Bounding the principal eigenvector. The principal eigenvector of $\mathbf{B}$ gives the distribution of the logics (apart from true and false) in the limit of large network depth n. We don't know how to write it down explicitly, but we can show that it is approximately flat.

In particular, we know that the principal eigenvector is at least as flat as the column sums of the matrix that it satisfies. The column sums of $\mathbf{B}$ 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

$$\sum_{j=1}^{\ell-1}\mathbf{B}_{ij} = 1-\left(\frac{i}{\ell}\right)^\ell-\left(\frac{\ell-i}{\ell}\right)^\ell. \tag{2}$$

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,

$$\sum_{j=1}^{\ell-1}\mathbf{B}_{ij} \in \left[\frac{1-e}{e},1\right], \tag{3}$$

where $(1-e)/e$ is 0.632. So the ratio of the smallest and largest components of the principal eigenvector of $\mathbf{B}$ is at least $(1-e)/e$ and at most 1. For example, for $k=3$, the minimum and maximum column sums are 0.656 and 0.992. As we shall see, the flatness of the principal eigenvector of $\mathbf{B}$ is key to our main result, namely, that the probability that the network function has Kolmogorov complexity q is constant.

Probability of true and false. The probability of generating any non-constant function (not true or false) at depth $n=1$ is the sum over i of the columns sums given

in (2) times the initial condition, that is,

$$P(\text{not T or F}) = \sum_{i=1}^{\ell-1}\left(1-\left(\frac{i}{\ell}\right)^{\ell}-\left(\frac{\ell-i}{\ell}\right)^{\ell}\right)\binom{\ell}{i}\Big/2^{\ell}$$
$$= \sum_{i=0}^{\ell}\left(1-2\left(\frac{i}{\ell}\right)^{\ell}\right)\binom{\ell}{i}\Big/2^{\ell}$$
$$= 1-\frac{2}{\ell^{\ell}2^{\ell}}\sum_{i=0}^{\ell} i^{\ell}\binom{l}{i}.$$

So at $n=1$,

$$P(\text{true}) = P(\text{false}) = \frac{1}{\ell^{\ell}}\frac{1}{2^{\ell}}\sum_{i=0}^{\ell} i^{\ell}\binom{\ell}{i}, \qquad (4)$$

where recall $\ell = 2^k$. Asymptotically, this is

$$P(\text{true}) \sim \frac{1}{\sqrt{1+c}}\frac{1}{(2ec)^{l}} = \frac{0.884}{1.514^{2^k}},$$

where c is the value of the Lambert W function at $1/e$.

Comparison with computer experiments

The computational cost of enumerating all possible logic assignments on a network of width k and depth n is formidable: it grows as 2^{2^k}, which is the number of assignments per node, to the power $nk+1$, which is the number of nodes in the network. In previous work [13], we conducted extensive computer experiments for various values of k and n. In all cases, our computer experiments agree with our theoretical predictions.

For $k=2$ variables (Fig. 1A), there are $16^3, 16^5, 16^7$ and 16^9 programs for network depths $n=1,2,3$ and 4. We enumerated all of these configurations and, for each, determined the network's global function. We plot the probability of obtaining a given function in Fig. 5A (points), where we group together functions with the same Hamming weight w. This exactly matches our theoretical predictions. The solid line indicates the probabilities of the non-constant logic functions (eq. (1)) and the dotted line true ($w=4$) and false ($w=0$) (eq. (4)) The probabilities correspond to the values in Fig. 2. As n increases, the likelihood of true and false approach $1/2$, causing the likelihood of the non-constant functions to fall.

For $k=3$ variables (Fig. 1B), there are 256^4, 256^7 and 256^{10} programs for network depths $n=1,2$ and 3. For $n=1$, we were able to enumerate all of the configurations. For $n=2$ and 3, this is computationally infeasible, so instead we sampled the configurations. We randomly assigned one of the 256 logics to each of the nodes and then determined the global function, repeating this a million times. These are plotted in Fig. 5B (points). We include errors bars, but these are negligible in comparison to the point size. 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. As n increases, true and false again dominate, causing the

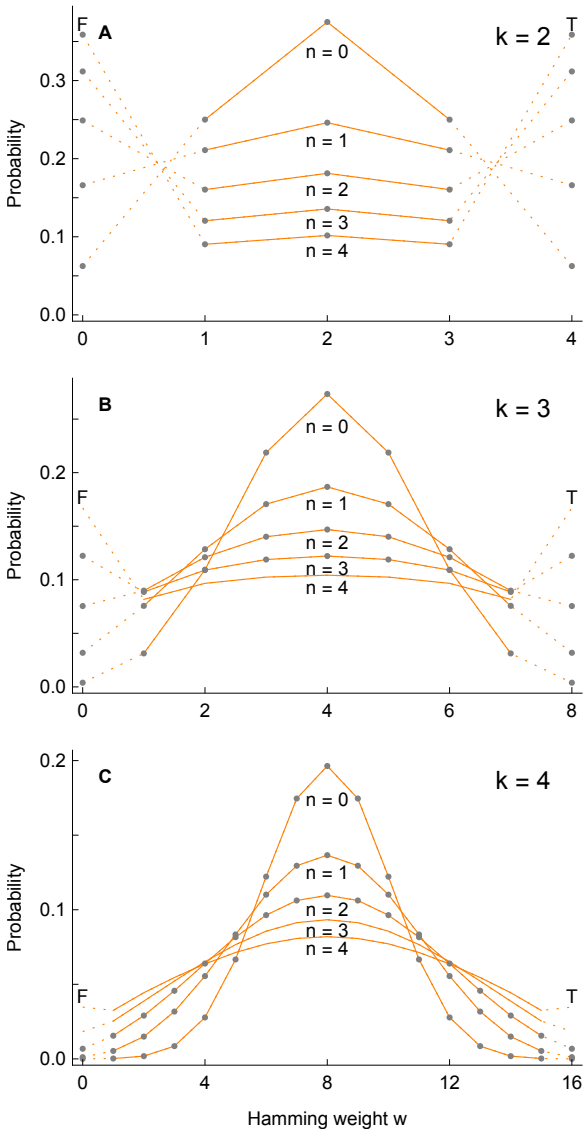

FIG. 5: **Computational confirmation of our theory.** We compare our theory (orange lines) with computer experiment (gray dots) for various values of k (the number of variables) and n (the network depth). The y-axis measures the probability of obtaining a given function, where we group together functions with the same Hamming weight w, which is the number of 1s in its truth table. For example, for $k=2$, the probabilities correspond to the values in Fig. 2. **A** For $k=2$, we enumerated all of the configurations up to network depth $n=4$. As n increases, the distribution of the non-constant functions flattens out. But for true and false ($w=0$ and $w=4$), the probabilities approach $1/2$. **B** For $k=3$, we show exact results for $n=0$ and 1, and sample the configurations for $n=2$. For $k=4$, we show exact results for $n=0$, and sample the configurations for $n=1$ and 2.

non-constant functions to fall.

For $k = 4$ variables (Fig. 1C), there are $65,536^5$ and $65,536^9$ programs for network depths $n = 1$ and 2. These are too many to enumerate, so we sampled from the configurations: a million samples for $n = 1$ and the same for $n = 2$. These are plotted in Fig. 5C, again with errors bars. Once again, our theory predicts the experiments to within statistical significance. We also show our $n = 3$ and 4 predictions for comparison.

Discussion

The main result of this paper is that the composition of logic functions on a fully-connected network architecture is biased towards simple functions. This bias becomes stronger as the network becomes deeper. In the limit of large depth n, the probability that an arbitrary function can be compressed by s bits is constant, rather than 2^{-s} as is the case for randomly chosen functions.

Two time scales. As we increase the network depth n, the simplicity distribution changes in two ways, which we can think of as independent. The first is that the non-constant functions (everything apart from true and false) flatten out, and the second is that true and false start to dominate. The first happens relatively quickly and the second happens relatively slowly.

The two effects are shown in Fig. 6 for different values of the number of variables k. The orange curves show the distribution of the non-constant functions when the probability of true and false are negligible. The gray curves show the distribution as the probability of true and false start to dominate, causing the distribution (which continues to flatten) to descend. For example, for $k = 6$, the probability of true for $n = 9$, 27, 81, 243 and 729 layers is 0.001, 0.056, 0.299, 0.484 and 0.5000, with the same for false. As k increases, the times scales for flattening and descending diverge: the distribution flattens before it descends. We have computationally confirmed this behavior for up to $k = 9$.

Related work. This paper reveals a bias towards simplicity for a fully-connected architecture, in which the variables in the logic functions are all the same. It is the theoretical follow-up to a paper about the computational evidence for a simplicity bias for a fully-connected architecture and a branching architecture [14].

This paper considers the fully-connected architecture, In a separate paper [13], we consider the opposite regime: a regularly branching architecture, in which the variables in the logic functions are all different.

Open questions. This research raises a number of open questions. One is the principle eigenvector of the transition matrix B. Right now we are able to bound the variation in the components to be within $(1-e)/e$, which guarantees its flatness. But we do not have an analytic form, A second question is a theoretical understanding of the two time scales described above, in which the distribution flattens before it descends. This will be tied to the spectrum of eigenvalues of the transition matrix B.

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

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

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

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

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

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

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

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

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

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

[11] T. Fink, Exact dynamics of the critical Kauffman model with connectivity one, arXiv:2302.05314.

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

[13] T. Fink Computational evidence that the iteration of simple rules imposes Occam's razor (2025).

[14] T. Fink The iteration of simple rules imposes Occam's razor: branching architecture, (2025).

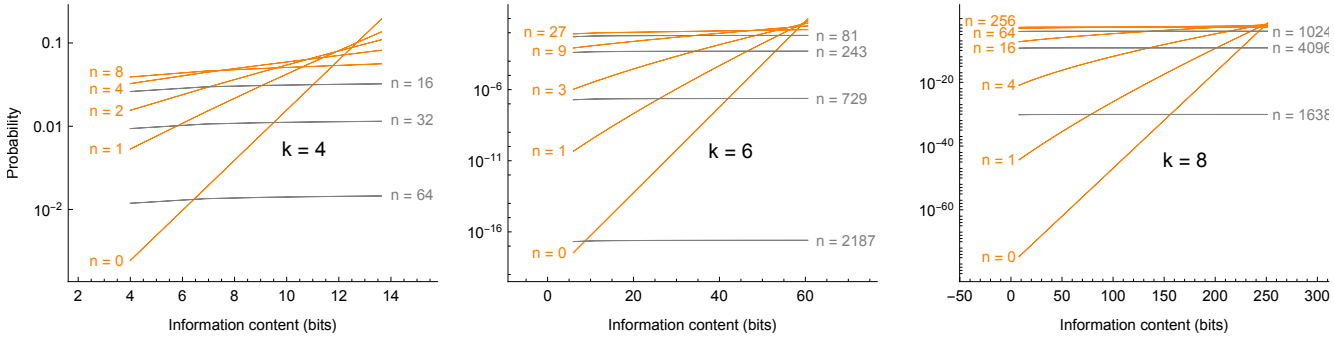

FIG. 6: **The simplicity distribution depends on the depth of the network.** As the network depth n increases, the simplicity distribution of the non-constant functions (everything but true and false) flattens out. On a slower time scale, the probability of true and false start to dominate as they each approach one half, causing the distribution to fall. The orange curves show the distribution before true and false dominate, and the gray curves show it after they dominate. As the number of variables k increases, these time scales diverge: the distribution flattens out before it falls.