Why does deep and cheap learning work so well?

Henry W. Lin and Max Tegmark
Dept. of Physics, Harvard University, Cambridge, MA 02138 and
Dept. of Physics & MIT Kavli Institute, Massachusetts Institute of Technology, Cambridge, MA 02139

(Dated: August 31, 2016)

We show how the success of deep learning depends not only on mathematics but also on physics: although well-known mathematical theorems guarantee that neural networks can approximate arbitrary functions well, the class of functions of practical interest can be approximated through "cheap learning" with exponentially fewer parameters than generic ones, because they have simplifying properties tracing back to the laws of physics. The exceptional simplicity of physics-based functions hinges on properties such as symmetry, locality, compositionality and polynomial log-probability, and we explore how these properties translate into exceptionally simple neural networks approximating both natural phenomena such as images and abstract representations thereof such as drawings. We further argue that when the statistical process generating the data is of a certain hierarchical form prevalent in physics and machine-learning, a deep neural network can be more efficient than a shallow one. We formalize these claims using information theory and discuss the relation to renormalization group procedures. Various "no-flattening theorems" show when these efficient deep networks cannot be accurately approximated by shallow ones without efficiency loss — even for linear networks.

I. INTRODUCTION

Deep learning works remarkably well, and has helped dramatically improve the state-of-the-art in areas ranging from speech recognition, translation and visual object recognition to drug discovery, genomics and automatic game playing [1]. However, it is still not fully understood *why* deep learning works so well. In contrast to GOFAI ("good old-fashioned AI") algorithms that are hand-crafted and fully understood analytically, many algorithms using artificial neural networks are understood only at a heuristic level, where we empirically know that certain training protocols employing large data sets will result in excellent performance. This is reminiscent of the situation with human brains: we know that if we train a child according to a certain curriculum, she will learn certain skills — but we lack a deep understanding of how her brain accomplishes this.

This makes it timely and interesting to develop new analytic insights on deep learning and its successes, which is the goal of the present paper. Such improved understanding is not only interesting in its own right, and for potentially providing new clues about how brains work, but it may also have practical applications. Better understanding the shortcomings of deep learning may suggest ways of improving it, both to make it more capable and to make it more robust [2].

A. The swindle: why does "cheap learning" work?

Throughout this paper, we will adopt a physics perspective on the problem, to prevent application-specific details from obscuring simple general results related to dynamics, symmetries, renormalization, *etc.*, and to exploit useful similarities between deep learning and statistical mechanics.

For concreteness, let us focus on the task of approximating functions. As illustrated in Figure 1, this covers most core sub-fields of machine learning, including unsupervised learning, classification and prediction. For example, if we are interested in classifying faces, then we may want our neural network to implement a function where we feed in an image represented by a million greyscale pixels and get as output the probability distribution over a set of people that the image might represent.

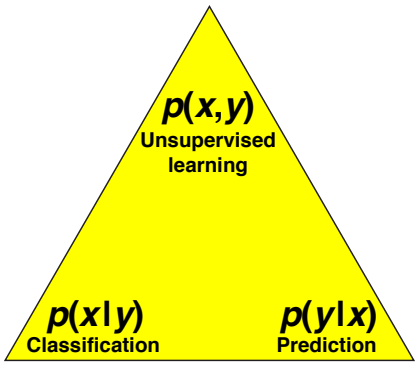

FIG. 1: Neural networks can approximate probability distributions. Given many samples of random vectors $\mathbf{x}$ and $\mathbf{y}$, both classification and prediction involve viewing $\mathbf{y}$ as a stochastic function of $\mathbf{x}$ and attempting to estimate the probability distributions for $\mathbf{x}$ given $\mathbf{y}$ and $\mathbf{y}$ given $\mathbf{x}$, respectively. In contrast, unsupervised learning attempts to approximate the joint probability distribution of $\mathbf{x}$ and $\mathbf{y}$ without making any assumptions about causality. In all three cases, the neural network searches for patterns in the data that can be used to better model the probability distribution.

When investigating the quality of a neural net, there are

several important factors to consider:

- **Expressibility:** What class of functions can the neural network express?
- **Efficiency:** How many resources (neurons, parameters, *etc.*) does the neural network require to approximate a given function?
- **Learnability:** How rapidly can the neural network learn good parameters for approximating a function?

This paper is focused on expressibility and efficiency, and more specifically on the following paradox: *How can neural networks approximate functions well in practice, when the set of possible functions is exponentially larger than the set of practically possible networks?* For example, suppose that we wish to classify megapixel greyscale images into two categories, *e.g.*, cats or dogs. If each pixel can take one of 256 values, then there are $256^{1000000}$ possible images, and for each one, we wish to compute the probability that it depicts a cat. This means that an arbitrary function is defined by a list of $256^{1000000}$ probabilities, *i.e.*, way more numbers than there are atoms in our universe (about 10^{78}). Yet neural networks with merely thousands or millions of parameters somehow manage to perform such classification tasks quite well. How can deep learning be so "cheap", in the sense of requiring so few parameters?

We will see in below that neural networks perform a combinatorial swindle, replacing exponentiation by multiplication: if there are say $n = 10^6$ inputs taking $v = 256$ values each, this swindle cuts the number of parameters from v^n to $v \times n$ times some constant factor. We will show that this success of this swindle depends fundamentally on physics: although neural networks only work well for an exponentially tiny fraction of all possible inputs, the laws of physics are such that the data sets we care about for machine learning (natural images, sounds, drawings, text, *etc.*) are also drawn from an exponentially tiny fraction of all imaginable data sets. Moreover, we will see that these two tiny subsets are remarkably similar, enabling deep learning to work well in practice.

The rest of this paper is organized as follows. In Section II, we present results for shallow neural networks with merely a handful of layers, focusing on simplifications due to locality, symmetry and polynomials. In Section III, we study how increasing the depth of a neural network can provide polynomial or exponential efficiency gains even though it adds nothing in terms of expressivity, and we discuss the connections to renormalization, compositionality and complexity. We summarize our conclusions in Section IV and discuss a technical point about renormalization and deep learning in Appendix V.

II. EXPRESSIBILITY AND EFFICIENCY OF SHALLOW NEURAL NETWORKS

Let us now explore what classes of probability distributions p are the focus of physics and machine learning, and how accurately and efficiently neural networks can approximate them. Although our results will be fully general, it will help illustrate key points of we give the mathematical notation from Figure 1 concrete interpretations. For a machine-learning example, we might interpret x as an element of some set of animals $\{\text{cat}, \text{dog}, \text{rabbit}, ...\}$ and $\mathbf{y}$ as the vector of pixels in an image depicting such an animal, so that $p(\mathbf{y}|x)$ for $x = cat$ gives the probability distribution of images of cats with different coloring, size, posture, viewing angle, lighting condition, electronic camera noise, *etc.* For a physics example, we might interpret x as an element of some set of metals $\{\text{iron}, \text{aluminum}, \text{copper}, ...\}$ and $\mathbf{y}$ as the vector of magnetization values for different parts of a metal bar. The prediction problem from Figure 1 is then to evaluate $p(\mathbf{y}|x)$, whereas the classification problem is to evaluate $p(x|\mathbf{y})$.

Because of the above-mentioned "swindle", accurate approximations are only possible for a tiny subclass of all probability distributions. Fortunately, as we will explore below, the function $p(\mathbf{y}|x)$ often has many simplifying features enabling accurate approximation, because it follows from some simple physical law or some generative model with relatively few free parameters: for example, its dependence on $\mathbf{y}$ may exhibit symmetry, locality and/or be of a simple form such as the exponential of a low-order polynomial. In contrast, the dependence of $p(x|\mathbf{y})$ on x tends to be more complicated; it makes no sense to speak of symmetries or polynomials involving a variable $x = cat$.

Let us therefore start by tackling the more complicated case of modeling $p(x|\mathbf{y})$. This probability distribution $p(x|\mathbf{y})$ is determined by the hopefully simpler function $p(\mathbf{y}|x)$ via Bayes' theorem:

$$p(x|\mathbf{y}) = \frac{p(\mathbf{y}|x)p(x)}{\sum_{x'} p(\mathbf{y}|x')(x')}, \tag{1}$$

where $p(x)$ is the probability distribution over x (animals or metals, say) *a priori*, before examining the data vector $\mathbf{y}$.

A. Probabilities and Hamiltonians

It is useful to introduce the negative logarithms of two of these probabilities:

$$\begin{aligned} H_x(\mathbf{y}) &\equiv -\ln p(\mathbf{y}|x), \\ \mu_x &\equiv -\ln p(x). \end{aligned} \tag{2}$$

Statisticians refer to $-\ln p$ as "self-information" or "surprisal", and statistical physicists refer to $H_x(\mathbf{y})$ as the *Hamiltonian*, quantifying the energy of $\mathbf{y}$ (up to an arbitrary and irrelevant additive constant) given the parameter x. These definitions transform equation (1) into the Boltzmann form

$$p(x|\mathbf{y}) = \frac{1}{N(\mathbf{y})} e^{-[H_x(\mathbf{y})+\mu_x]}, \tag{3}$$

where

$$N(\mathbf{y}) \equiv \sum_x e^{-[H_x(\mathbf{y})+\mu_x]}. \tag{4}$$

This recasting of equation (1) is useful because the Hamiltonian tends to have properties making it simple to evaluate. We will see in Section III that it also helps understand the relation between deep learning and renormalization.

B. Bayes theorem as a softmax

Since the variable x takes one of a discrete set of values, we will often write it as an index instead of as an argument, as $p_x(\mathbf{y}) \equiv p(x|\mathbf{y})$. Moreover, we will often find it convenient to view all values indexed by x as elements of a vector, written in boldface, thus viewing p_x, H_x and μ_x as elements of the vectors $\mathbf{p}$, $\mathbf{H}$ and $\boldsymbol{\mu}$, respectively. Equation (3) thus simplifies to

$$\mathbf{p}(\mathbf{y}) = \frac{1}{N(\mathbf{y})} e^{-[\mathbf{H}(\mathbf{y})+\boldsymbol{\mu}]}, \tag{5}$$

using the standard convention that a function (in this case exp) applied to a vector acts on its elements.

We wish to investigate how well this vector-valued function $\mathbf{p}(\mathbf{y})$ can be approximated by a neural net. A standard n-layer feedforward neural network maps vectors to vectors by applying a series of linear and nonlinear transformations in succession. Specifically, it implements vector-valued functions of the form [1]

$$\mathbf{f}(\mathbf{y}) = \boldsymbol{\sigma}_n \mathbf{A}_n \cdots \boldsymbol{\sigma}_2 \mathbf{A}_2 \boldsymbol{\sigma}_1 \mathbf{A}_1 \mathbf{y}, \tag{6}$$

where the $\boldsymbol{\sigma}_i$ are relatively simple nonlinear operators on vectors and the $\mathbf{A}_i$ are affine transformations of the form $\mathbf{A}_i\mathbf{y} = \mathbf{W}_i\mathbf{y}+\mathbf{b}_i$ for matrices $\mathbf{W}_i$ and so-called bias vectors $\mathbf{b}_i$. Popular choices for these nonlinear operators $\boldsymbol{\sigma}_i$ include

- *Local function* (apply some nonlinear function σ to each vector element),
- *Max-pooling* (compute the maximum of all vector elements),
- *Softmax* (exponentiate all vector elements and normalize them to so sum to unity).

The softmax operator is therefore defined by

$$\boldsymbol{\sigma}(\mathbf{y}) \equiv \frac{e^{\mathbf{y}}}{\sum_i e^{y_i}}. \tag{7}$$

This allows us to rewrite equation (5) in the extremely simple form

$$\mathbf{p}(\mathbf{y}) = \boldsymbol{\sigma}[-\mathbf{H}(\mathbf{y}) - \boldsymbol{\mu}]. \tag{8}$$

This means that if we can compute the Hamiltonian vector $\mathbf{H}(\mathbf{y})$ with some n-layer neural net, we can evaluate the desired classification probability vector $\mathbf{p}(\mathbf{y})$ by simply adding a softmax layer. The $\boldsymbol{\mu}$-vector simply becomes the bias term in this final layer.

C. What Hamiltonians can be approximated by feasible neural networks?

It has long been known that neural networks are universal approximators [3, 4], in the sense that networks with virtually all popular nonlinear activation functions $\sigma(y)$ can approximate any smooth function to any desired accuracy — even using merely a single hidden layer. However, these theorems do not guarantee that this can be accomplished with a network of feasible size, and the following simple example explains why they cannot: There are 2^{2^n} different Boolean functions of n variables, so a network implementing a generic function in this class requires at least 2^n bits to describe, *i.e.*, more bits than there are atoms in our universe if $n > 260$.

The fact that neural networks of feasible size are nonetheless so useful therefore implies that the class of functions we care about approximating is dramatically smaller. We will see below in Section II D that both physics and machine learning tend to favor Hamiltonians that are polynomials[1] — indeed, often ones that are sparse, symmetric and low-order. Let us therefore focus our initial investigation on Hamiltonians that can be expanded as a power series:

$$H_x(\mathbf{y}) = h + \sum_i h_i y_i + \sum_{i\le j} h_{ij} y_i y_j + \sum_{i\le j\le k} h_{ijk} y_i y_j y_k + \cdots. \tag{9}$$

If the vector $\mathbf{y}$ has n components ($i = 1, ..., n$), then there are $(n+d)!/(n!d!)$ terms of degree up to d.

[1] The class of functions that can be exactly expressed by a neural network must be invariant under composition, since adding more layers corresponds to using the output of one function as the input to another. Important such classes include linear functions, affine functions, piecewise linear functions (generated by the popular Rectified Linear unit "ReLU" activation function $\sigma(y) = \max[0, y]$), polynomials, continuous functions and smooth functions whose n^{th} derivatives are continuous. According to the Stone-Weierstrass theorem, both polynomials and piecewise linear functions can approximate continuous functions arbitrarily well.

1. Continuous input variables

If we can accurately approximate multiplication using a small number of neurons, then we can construct a network efficiently approximating any polynomial $H_x(\mathbf{y})$ by repeated multiplication and addition. We will now see that we can, using any smooth but otherwise arbitrary non-linearity σ that is applied element-wise. The popular logistic sigmoid activation function $\sigma(y) = 1/(1+e^{-y})$ will do the trick.

Theorem: Let $\mathbf{f}$ be a neural network of the form $\mathbf{f} = \mathbf{A}_2\boldsymbol{\sigma}\mathbf{A}_1$, where $\boldsymbol{\sigma}$ acts elementwise by applying some smooth non-linear function σ to each element. Let the input layer, hidden layer and output layer have sizes 2, 4 and 1, respectively. Then $\mathbf{f}$ can approximate a multiplication gate arbitrarily well.

To see this, let us first Taylor-expand the function σ around the origin:

$$\sigma(u) = \sigma_0 + \sigma_1 u + \sigma_2 u^2 + \mathcal{O}(u^3). \tag{10}$$

Without loss of generality, we can assume that $\sigma_2 \neq 0$: since σ is non-linear, it must have a non-zero second derivative at some point, so we can use the biases in $\mathbf{A}_1$ to shift the origin to this point to ensure $\sigma_2 \neq 0$. Equation (10) now implies that

$$\begin{aligned} m(u,v) &\equiv \frac{\sigma(u+v)+\sigma(-u-v)-\sigma(u-v)-\sigma(-u+v)}{8\sigma_2} \\ &= uv\left[1+\mathcal{O}\left(u^2+v^2\right)\right], \end{aligned} \tag{11}$$

where we will term $m(x,y)$ the *multiplication approximator*. Taylor's theorem guarantees that $m(x,y)$ is an arbitrarily good approximation of xy for arbitrarily small $|x|$ and $|y|$. However, we can always make $|x|$ and $|y|$ arbitrarily small by scaling $\mathbf{A}_1 \to \lambda\mathbf{A}_1$ and then compensating by scaling $\mathbf{A}_2 \to \lambda^{-2}\mathbf{A}_2$. In the limit that $\lambda \to \infty$, this approximation becomes exact. In other words, arbitrarily accurate multiplication can always be achieved using merely 4 neurons. Figure 2 illustrates such a multiplication approximator using a logistic sigmoid σ.

Corollary: For any given multivariate polynomial and any tolerance $\epsilon > 0$, there exists a neural network of fixed finite size N (independent of ϵ) that approximates the polynomial to accuracy better than ϵ. Furthermore, N is bounded by the complexity of the polynomial, scaling as the number of multiplications required times a factor that is typically slightly larger than 4.[2]

[2] In addition to the four neurons required for each multiplication, additional neurons may be deployed to copy variables to higher layers bypassing the nonlinearity in σ. Such linear "copy gates" implementing the function $u \to u$ are of course trivial to implement using a simpler version of the above procedure: using $\mathbf{A}_1$ to shift and scale down the input to fall in a tiny range where $\sigma'(u) \neq 0$, and then scaling it up and shifting accordingly with $\mathbf{A}_2$.

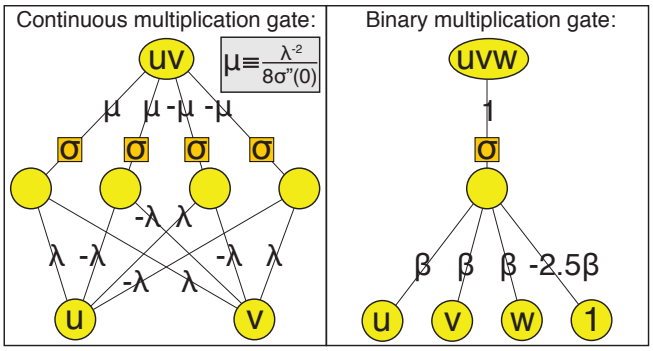

FIG. 2: Multiplication can be efficiently implemented by simple neural nets, becoming arbitrarily accurate as $\lambda \to 0$ (left) and $\beta \to \infty$ (right). Squares apply the function σ, circles perform summation, and lines multiply by the constants labeling them. The "1" input implements the bias term. The left gate requires $\sigma''(0) \neq 0$, which can always be arranged by biasing the input to σ. The right gate requires the sigmoidal behavior $\sigma(x) \to 0$ and $\sigma(x) \to 1$ as $x \to -\infty$ and $x \to \infty$, respectively.

This is a stronger statement than the classic universal universal approximation theorems for neural networks [3, 4], which guarantee that for every ϵ there exists some $N(\epsilon)$, but allows for the possibility that $N(\epsilon) \to \infty$ as $\epsilon \to 0$. An approximation theorem in [5] provides an ϵ-independent bound on the size of the neural network, but at the price of choosing a pathological function σ.

2. Discrete input variables

For the simple but important case where $\mathbf{y}$ is a vector of bits, so that $y_i = 0$ or $y_i = 1$, the fact that $y_i^2 = y_i$ makes things even simpler. This means that only terms where all variables are different need be included, which simplifies equation (9) to

$$H_x(\mathbf{y}) = h + \sum_i h_i y_i + \sum_{i<j} h_{ij} y_i y_j + \sum_{i<j<k} h_{ijk} y_i y_j y_k + \cdots. \tag{12}$$

The infinite series equation (9) thus gets replaced by a finite series with 2^n terms, ending with the term $h_{1\ldots n} y_1 \cdots y_n$. Since there are 2^n possible bit strings $\mathbf{y}$, the 2^n $h-$parameters in equation (12) suffice to exactly parametrize an arbitrary function $H_x(\mathbf{y})$.

The efficient multiplication approximator above multiplied only two variables at a time, thus requiring multiple layers to evaluate general polynomials. In contrast, $H(\mathbf{y})$ for a bit vector $\mathbf{y}$ can be implemented using merely three layers as illustrated in Figure 2, where the middle layer evaluates the bit products and the third layer takes a linear combination of them. This is because bits allow an accurate multiplication approximator that takes the product of an arbitrary number of bits at once, exploiting the fact that a product of bits can be trivially

determined from their sum: for example, the product $y_1y_2y_3 = 1$ if and only if the sum $y_1 + y_2 + y_3 = 3$. This sum-checking can be implemented using one of the most popular choices for a nonlinear function σ: the logistic sigmoid $\sigma(y) = \frac{1}{1+e^{-y}}$ which satisfies $\sigma(y) \approx 0$ for $y \ll 0$ and $\sigma(y) \approx 1$ for $y \gg 1$. To compute the product of some set of k bits described by the set K (for our example above, $K = \{1, 2, 3\}$), we let $\mathbf{A}_1$ and $\mathbf{A}_2$ shift and stretch the sigmoid to exploit the identity

$$\prod_{i\in K} y_i = \lim_{\beta\to\infty} \sigma\left[-\beta\left(k - \frac{1}{2} - \sum_{y\in K} y_i\right)\right]. \qquad (13)$$

Since σ decays exponentially fast toward 0 or 1 as β is increased, modestly large β-values suffice in practice; if, for example, we want the correct answer to $D = 10$ decimal places, we merely need $\beta > D\ln 10 \approx 23$. In summary, when $\mathbf{y}$ is a bit string, an *arbitrary* function $p_x(\mathbf{y})$ can be evaluated by a simple 3-layer neural network: the middle layer uses sigmoid functions to compute the products from equation (12), and the top layer performs the sums from equation (12) and the softmax from equation (8).

D. What Hamiltonians do we want to approximate?

We have seen that polynomials can be accurately approximated by neural networks using a number of neurons scaling either as the number of multiplications required (for the continuous case) or as the number of terms (for the binary case). But polynomials *per se* are no panacea: with binary input, all functions are polynomials, and with continuous input, there are $(n + d)!/(n!d!)$ coefficients in a generic polynomial of degree d in n variables, which easily becomes unmanageably large. We will now see how exceptionally simple polynomials that are sparse, symmetric and/or low-order play a special role in physics and machine-learning.

1. Low polynomial order

For reasons that are still not fully understood, our universe can be accurately described by polynomial Hamiltonians of low order d. At a fundamental level, the Hamiltonian of the standard model of particle physics has $d = 4$. There are many approximations of this quartic Hamiltonian that are accurate in specific regimes, for example the Maxwell equations governing electromagnetism, the Navier-Stokes equations governing fluid dynamics, the Alvén equations governing magnetohydrodynamics and various Ising models governing magnetization — all of these approximations have Hamiltonians that are polynomials in the field variables, of degree d ranging from 2 to 4. This means that the number of polynomial coefficients is not infinite as in equation (9) or exponential in n as in equation (12), merely of order n^2, n^3 or n^4.

Thanks to the Central Limit Theorem [6], many probability distributions in machine-learning and statistics can be accurately approximated by multivariate Gaussians, *i.e.*, of the form

$$p(\mathbf{y}) = e^{h+\sum_i h_j y_i - \sum_{ij} h_{ij} y_i y_j}, \qquad (14)$$

which means that the Hamiltonian $H = -\ln p$ is a quadratic polynomial. More generally, the maximum-entropy probability distribution subject to constraints on some of the lowest moments, say expectation values of the form $\langle y_1^{\alpha_1} y_2^{\alpha_2} \cdots y_n^{\alpha_n}\rangle$ for some integers $\alpha_i \geq 0$ would lead to a Hamiltonian of degree no greater than $d \equiv \sum_i \alpha_i$ [7].

Image classification tasks often exploit invariance under translation, rotation, and various nonlinear deformations of the image plane that move pixels to new locations. All such spatial transformations are linear function functions ($d = 1$ polynomials) of the pixel vector $\mathbf{y}$. Functions implementing convolutions and Fourier transforms are also $d = 1$ polynomials.

2. Locality

One of the deepest principles of physics is *locality*: that things directly affect only what is in their immediate vicinity. When physical systems are simulated on a computer by discretizing space onto a rectangular lattice, locality manifests itself by allowing only nearest-neighbor interaction. In other words, almost all coefficients in equation (9) are forced to vanish, and the total number of non-zero coefficients grows only linearly with n. For the binary case of equation (9), which applies to magnetizations (spins) that can take one of two values, locality also limits the degree d to be no greater than the number of neighbors that a given spin is coupled to (since all variables in a polynomial term must be different).

This can be stated more generally and precisely using the Markov network formalism [8]. View the spins as vertices of a Markov network; the edges represent dependencies. Let N_c be the *clique cover number* of the network (the smallest number of cliques whose union is the entire network) and let S_c be the size of the largest clique. Then the number of required neurons is $\leq N_c 2^{S_c}$. For fixed S_c, N_c is proportional to the number of vertices, so locality means that the number of neurons scales only linearly with the number of spins n.

3. Symmetry

Whenever the Hamiltonian obeys some symmetry (is invariant under some transformation), the number of in-

dependent parameters required to describe it is further reduced. For instance, many probability distributions in both physics and machine learning are invariant under translation and rotation. As an example, consider a vector $\mathbf{y}$ of air pressures y_i measured by a microphone at times $i = 1, ..., n$. Assuming that the Hamiltonian describing it has $d = 2$ reduces the number of parameters N from ∞ to $(n + 1)(n + 2)/2$. Further assuming locality (nearest-neighbor couplings only) reduces this to $N = 2n$, after which requiring translational symmetry reduces the parameter count to $N = 3$. Taken together, the constraints on locality, symmetry and polynomial order reduce the number of continuous parameters in the Hamiltonian of the standard model of physics to merely 32 [9].

Symmetry can reduce not merely the parameter count, but also the computational complexity. For example, if a linear vector-valued function $\mathbf{f}(\mathbf{y})$ mapping a set of n variables onto itself happens to satisfy translational symmetry, then it is a convolution (implementable by a convolutional neural net; "convnet"), which means that it can be computed with $n \log_2 n$ rather than n^2 multiplications using Fast Fourier transform.

III. WHY DEEP?

Above we investigated how probability distributions from physics and computer science applications lent themselves to "cheap learning", being accurately and efficiently approximated by neural networks with merely a handful of layers. Let us now turn to the separate question of depth, *i.e.*, the success of deep learning: what properties of real-world probability distributions cause efficiency to further improve when networks are made deeper? This question has been extensively studied from a mathematical point of view [10–12], but mathematics alone cannot fully answer it, because part of the answer involves physics. We will argue that the answer involves the hierarchical/compositional structure of generative processes together with inability to efficiently "flatten" neural networks reflecting this structure.

A. Hierarchical processess

One of the most striking features of the physical world is its hierarchical structure. Spatially, it is an object hierarchy: elementary particles form atoms which in turn form molecules, cells, organisms, planets, solar systems, galaxies, *etc.* Causally, complex structures are frequently created through a distinct sequence of simpler steps.

Figure 3 gives two examples of such causal hierarchies generating data vectors $x_0 \mapsto x_1 \mapsto ... \mapsto x_n$ that are relevant to physics and image classification, respectively. Both examples involve a Markov chain[3] where the probability distribution $p(x_i)$ at the i^{th} level of the hierarchy is determined from its causal predecessor alone:

$$\mathbf{p}_i \mathbf{M}_i \mathbf{p}_{i-1}, \tag{15}$$

where the probability vector $\mathbf{p}_i$ specifies the probability distribution of $p(x_i)$ according to $(\mathbf{p}_i)_x \equiv p(x_i)$ and the Markov matrix $\mathbf{M}_i$ specifies the transition probabilities between two neighboring levels, $p(x_i|x_{i-1})$. Iterating equation (15) gives

$$\mathbf{p}_n = \mathbf{M}_n \mathbf{M}_{n-1} \cdots \mathbf{M}_1 \mathbf{p}_0, \tag{16}$$

so we can write the combined effect of the the entire generative process as a matrix product.

In our physics example (Figure 3, left), a set of cosmological parameters x_0 (the density of dark matter, *etc.*) determines the power spectrum x_1 of density fluctuations in our universe, which in turn determines the pattern of cosmic microwave background radiation x_2 reaching us from our early universe, which gets combined with foreground radio noise from our Galaxy to produce the frequency-dependent sky maps (x_3) that are recorded by a satellite-based telescope that measures linear combinations of different sky signals and adds electronic receiver noise. For the recent example of the Planck Satellite [13], these datasets x_i, $x_2, ...$ contained about 10^1, 10^4, 10^8, 10^9 and 10^{12} numbers, respectively.

More generally, if a given data set is generated by a (classical) statistical physics process, it must be described by an equation in the form of equation (16), since dynamics in classical physics is fundamentally Markovian: classical equations of motion are always first order differential equations in the Hamiltonian formalism. This technically covers essentially all data of interest in the machine learning community, although the fundamental Markovian nature of the generative process of the data may be an in-efficient description.

Our toy image classification example (Figure 3, right) is deliberately contrived and over-simplified for pedagogy: x_0 is a single bit signifying "cat or dog", which determines a set of parameters determining the animal's coloration, body shape, posture, *etc.* using approxiate probability distributions, which determine a 2D image via ray-tracing, which is scaled and translated by random amounts before a randomly background is added.

In both examples, the goal is to reverse this generative hierarchy to learn about the input $x \equiv x_0$ from the output $x_n \equiv y$, specifically to provide the best possibile estimate

[3] If the next step in the generative hierarchy requires knowledge of not merely of the present state but also information of the past, the present state can be redefined to include also this information, thus ensuring that the generative process is a Markov process.

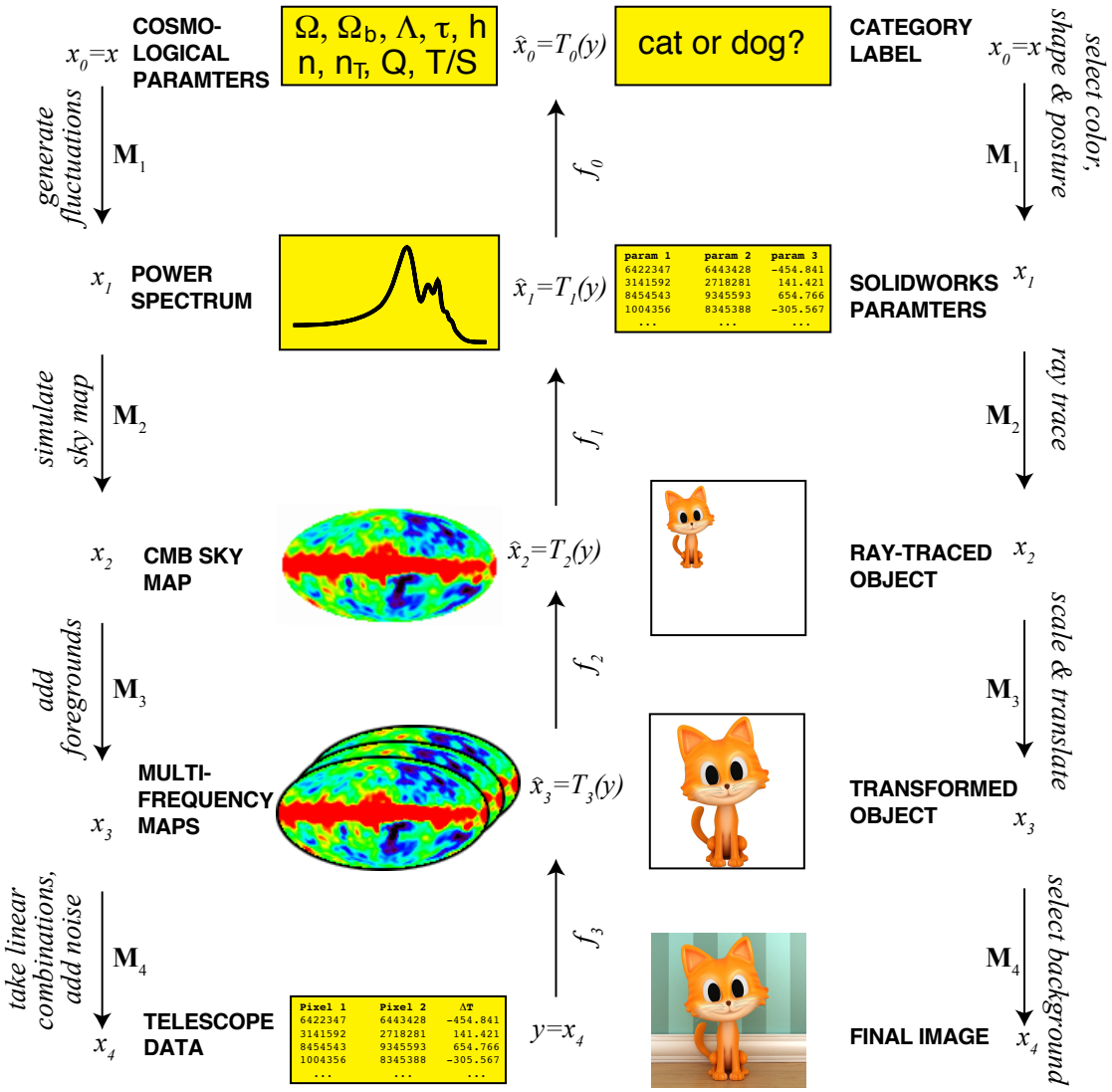

FIG. 3: Causal hierarchy examples relevant to physics (left) and image classification (right). As information flows down the hierarchy $x_0 \to x_1 \to ... \to x_n = y$, some of it is destroyed by random Markov processes. However, no further information is lost as information flows optimally back up the hierarchy as $\widehat{x}_{n-1} \to ... \to \widehat{x}_0$. The right example is deliberately contrived and over-simplified for pedagogy; for example, translation and scaling are more naturally performed before ray tracing, which in turn breaks down into multiple steps.

of the probability distribution $p(x|y) = p(x_0|x_n)$ — *i.e.*, to determine the probability distribution for the cosmological parameters and to determine the probability that the image is a cat, respectively.

B. Resolving the swindle

This decomposition of the generative process into a hierarchy of simpler steps helps resolve the "swindle" paradox from the introduction: although the number of parameters required to describe an arbitrary function of the input data y is beyond astronomical, the generative process can be specified by a more modest number of parameters, because each of its steps can. Whereas specifying an arbitrary probability distribution over multi-megapixel images y requires far more bits than there are atoms in our universe, the information specifying how to compute the probability distribution $p(y|x)$ for a microwave background map fits into a handful of published journal articles or software packages [14–20]. For a megapixel image of a galaxy, its entire probability distribution is defined by the standard model of particle physics with its 32 parameters [9], which together specify the process transforming primordial hydrogen gas into galaxies.

The same parameter-counting argument can also be applied to all artificial images of interest to machine learn-

ing: for example, giving the simple low-information-content instruction *"draw a cute kitten"* to a random sample of artists will produce a wide variety of images y with a complicated probability distribution over colors, postures, *etc.*, as each artist makes random choices at a series of steps. Even the pre-stored information about cat probabilities in these artists' brains is modest in size.

Note that a random resulting image typically contains much more information than the generative process creating it; for example, the simple instruction *"generate a random string of* 10^9 *bits"* contains much fewer than 10^9 bits. Not only are the typical steps in the generative hierarchy specified by a non-astronomical number of parameters, but as discussed in Section II D, it is plausible that neural networks can implement each of the steps efficiently.[4]

A deep neural network stacking these simpler networks on top of one another would then implement the entire generative process efficiently. In summary, the data sets and functions we care about form a minuscule minority, and it is plausible that they can also be efficiently implemented by neural networks reflecting their generative process. So what is the remainder? Which are the data sets and functions that we do *not* care about?

Almost all images are indistinguishable from random noise, and almost all data sets and functions are indistinguishable from completely random ones. This follows from Borel's theorem on normal numbers [22], which states that almost all real numbers have a string of decimals that would pass any randomness test, *i.e.*, are indistinguishable from random noise. Simple parameter counting shows that deep learning (and our human brains, for that matter) would fail to implement almost all such functions, and training would fail to find any useful patterns. To thwart pattern-finding efforts. cryptography therefore aims to produces random-looking patterns. Although we might expect the Hamiltonians describing human-generated data sets such as drawings, text and music to be more complex than those describing simple physical systems, we should nonetheless expect them to resemble the natural data sets that inspired their creation much more than they resemble random functions.

[4] Although our discussion is focused on describing probability distributions, which are not random, stochastic neural networks can generate random variables as well. In biology, spiking neurons provide a good random number generator, and in machine learning, stochastic architectures such as restricted Boltzmann machines [21] do the same.

C. Sufficient statistics and hierarchies

The goal of deep learning classifiers is to reverse the hierarchical generative process as well as possible, to make inferences about the input x from the output y. Let us now treat this hierarchical problem more rigorously using information theory.

Given $P(x|y)$, a *sufficient statistic* $T(y)$ is defined by the equation $P(x|y) = P(x|T(y))$ and has played an important role in statistics for almost a century [23]. All the information about x contained in y is contained in the sufficient statistic. A *minimal sufficient statistic* [23] is some sufficient statistic T_* which is a sufficient statistic for all other sufficient statistics. This means that if $T(y)$ is sufficient, then there exists some function f such that $T_*(y) = f(T(y))$. As illustrated in Figure 3, T_* can be thought of as a an information distiller, optimally compressing the data so as to retain all information relevant to determining x and discarding all irrelevant information.

The sufficient statistic formalism enables us to state some simple but important results that apply to any hierarchical generative process cast in the Markov chain form of equation (16).

Theorem 2: Given a Markov chain described by our notation above, let T_i be a minimal sufficient statistic of $P(x_i|x_n)$. Then there exists some functions f_i such that $T_i = f_i \circ T_{i+1}$. More casually speaking, the generative hierarchy of Figure 3 can be optimally reversed one step at a time: there are functions f_i that optimally undo each of the steps, distilling out all information about the level above that was not destroyed by the Markov process.

Here is the proof. Note that for any $k \geq 1$, "backwards" Markovity $P(x_i|x_{i+1}, x_{i+k}) = P(x_i|x_{i+1})$ follows from Markovity via Bayes' theorem:

$$\begin{aligned} P(x_i|x_{i+k}, x_{i+1}) &= \frac{P(x_{i+k}|x_i, x_{i+1})P(x_i|x_{i+1})}{P(x_{i+k}|x_{i+1})} \\ &= \frac{P(x_{i+k}|x_{i+1})P(x_i|x_{i+1})}{P(x_{i+k}|x_{i+1})} \\ &= P(x_i|x_{i+1}). \end{aligned} \tag{17}$$

Using this fact, we see that

$$\begin{aligned} P(x_i|x_n) &= \sum_{x_{i+1}} P(x_i|x_{i+1}x_n)P(x_{i+1}|x_n) \\ &= \sum_{x_{i+1}} P(x_i|x_{i+1})P(x_{i+1}|T_{i+1}(x_n)). \end{aligned} \tag{18}$$

Since the above equation depends on x_n only through $T_{i+1}(x_n)$, this means that T_{i+1} is a sufficient statistic for $P(x_i|x_n)$. But since T_i is the minimal sufficient statistic, there exists a function f_i such that $T_i = f_i \circ T_{i+1}$.

Corollary 2: With the same assumptions and notation as theorem 2, define the function $f_0(T_0) = P(x_0|T_0)$ and

$f_n = T_{n-1}$. Then

$$P(x_0|x_n) = (f_0 \circ f_1 \circ \cdots \circ f_n)(x_n), \tag{19}$$

The proof is easy. By induction,

$$T_0 = f_1 \circ f_2 \circ \cdots \circ T_{n-1}, \tag{20}$$

which implies the corollary.

Roughly speaking, Corollary 2 states that *the structure of the inference problem reflects the structure of the generative process*. In this case, we see that the neural network trying to approximate $P(x|y)$ must approximate a compositional function. We will argue below in Section III F that in many cases, this can only be accomplished efficiently if the neural network has $\gtrsim n$ hidden layers.

In neuroscience parlance, the functions f_i compress the data into forms with ever more *invariance* [24], containing features invariant under irrelevant transformations (for example background substitution, scaling and translation).

Let us denote the distilled vectors $\widehat{x}_i \equiv f_i(\widehat{x}_{i+1})$, where $\widehat{x}_n \equiv y$. As summarized by Figure 3, as information flows down the hierarchy $x = x_0 \to x_1 \to ...\mathbf{x}_n = y$, some of it is destroyed by random processes. However, no further information is lost as information flows optimally back up the hierarchy as $y \to \widehat{x}_{n-1} \to ... \to \widehat{x}_0$.

D. Approximate information distillation

Although minimal sufficient statistics are often difficult to calculate in practice, it is frequently possible to come up with statistics which are nearly sufficient in a certain sense which we now explain.

An equivalent characterization of a sufficient statistic is provided by information theory [25, 26]. The *data processing inequality* [26] states that for any function f and any random variables x, y,

$$I(x, y) \geq I(x, f(y)), \tag{21}$$

where I is the *mutual information*:

$$I(x, y) = \sum_{x,y} p(x, y) \log \frac{p(x, y)}{p(x)p(y)}. \tag{22}$$

A sufficient statistic $T(y)$ is a function $f(y)$ for which "$\geq$" gets replaced by "$=$" in equation (21), *i.e.*, a function retaining all the information about x.

Even information distillation functions f that are not strictly sufficient can be very useful as long as they distill out *most* of the relevant information and are computationally efficient. For example, it may be possible to trade some loss of mutual information with a dramatic reduction in the complexity of the Hamiltonian; e.g., $H_x(f(y))$ may be considerably easier to implement in a neural network than $H_x(y)$. Precisely this situation applies to the physics example from Figure 3, where a hierarchy of efficient near-perfect information distillers f_i have been found, the numerical cost of f_3 [19, 20], f_2 [17, 18], f_1 [15, 16] and f_0 [13] scaling with the number of inputs parameters n as $\mathcal{O}(n)$, $\mathcal{O}(n^{3/2})$, $\mathcal{O}(n^2)$ and $\mathcal{O}(n^3)$, respectively.

E. Distillation and renormalization

The systematic framework for distilling out desired information from unwanted "noise" in physical theories is known as Effective Field Theory [27]. Typically, the desired information involves relatively large-scale features that can be experimentally measured, whereas the noise involves unobserved microscopic scales. A key part of this framework is known as the *renormalization group* (RG) transformation [27, 28]. Although the connection between RG and machine learning has been studied or alluded to repeatedly [29–33], there are significant misconceptions in the literature concerning the connection which we will now attempt to clear up.

Let us first review a standard working definition of what renormalization is in the context of statistical physics, involving three ingredients: a vector y of random variables, a course-graining operation R and a requirement that this operation leaves the Hamiltonian invariant except for parameter changes. We think of y as the microscopic degrees of freedom — typically physical quantities defined at a lattice of points (pixels or voxels) in space. Its probability distribution is specified by a Hamiltonian $H_x(y)$, with some parameter vector x. We interpret the map $R : y \to y$ as implementing a coarse-graining[5] of the system. The random variable $R(y)$ also has a Hamiltonian, denoted $H'(R(y))$, which we require to have the same functional form as the original Hamiltonian H_x, although the parameters x may change. In other words, $H'(R(y)) = H_{r(x)}(R(y))$ for some function r. Since the domain and the range of R coincide, this map R can be iterated n times $R^n = R \circ R \circ \cdots R$, giving a Hamiltonian $H_{r^n(x)}(R^n(y))$ for the repeatedly renormalized data.

[5] A typical renormalization scheme for a lattice system involves replacing many spins (bits) with a single spin according to some rule. In this case, it might seem that the map R could not possibly map its domain onto itself, since there are fewer degrees of freedom after the coarse-graining. On the other hand, if we let the domain and range of R differ, we cannot easily talk about the Hamiltonian as having the same functional form, since the renormalized Hamiltonian would have a different domain than the original Hamiltonian. Physicists get around this by taking the limit where the lattice is infinitely large, so that R maps an infinite lattice to an infinite lattice.

Similar to the case of sufficient statistics, $P(x|R^n(y))$ will then be a compositional function.

Contrary to some claims in the literature, effective field theory and the renormalization group have little to do with the idea of unsupervised learning and pattern-finding. Instead, the standard renormalization procedures in statistical physics and quantum field theory are essentially a feature extractor for *supervised* learning, where the features typically correspond to long-wavelength/macroscopic degrees of freedom. In other words, effective field theory only makes sense if we specify what features we are interested in. For example, if we are given data y about the position and momenta of particles inside a mole of some liquid and is tasked with predicting from this data whether or not Alice will burn her finger when touching the liquid, a (nearly) sufficient statistic is simply the temperature of the object, which can in turn be obtained from some very coarse-grained degrees of freedom (for example, one could use the fluid approximation instead of working directly from the positions and momenta of $\sim 10^{23}$ particles).

To obtain a more quantitative link between renormalization and deep-learning-style feature extraction, let us consider as a toy model for natural images (functions of a 2D position vector $\mathbf{r}$) a generic two-dimensional Gaussian random field $y(\mathbf{r})$ whose Hamiltonian satisfies translational and rotational symmetry:

$$H_x(y) = \int \left[x_0 y^2 + x_1 (\nabla y)^2 + x_2 \left(\nabla^2 y\right)^2 + \cdots \right] d^2 r. \tag{23}$$

Thus the fictitious classes of images that we are trying to distinguish are all generated by Hamiltonians H_x with the same above form but different parameter vectors x. We assume that the function $y(\mathbf{r})$ is specified on pixels that are sufficiently close that derivatives can be well-approximated by differences. Derivatives are linear operations, so they can be implemented in the first layer of a neural network. The translational symmetry of equation (23) allows it to be implemented with a convnet. If can be shown [27] that for any course-graining operation that replaces each block of $b \times b$ pixels by its average and divides the result by b^2, the Hamiltonian retains the form of equation (23) but with the parameters x_i replaced by

$$x_i' = b^{2-2i} x_i. \tag{24}$$

This means that all paramters x_i with $i \geq 2$ decay exponentially with b as we repeatedly renormalize and b keeps increasing, so that for modest b, one can neglect all but the first few x_i's.In this example, the parameters x_0 and x_1 would be called "relevant operators" by physicists and "signal" by machine-learners, whereas the remaining parameters would be called "irrelevant operators" by physicists and "noise" by machine-learners.

In summary, renormalization is a special case of feature extraction and nearly sufficient statistics, typically treating small scales as noise. This makes it a special case of supervised learning, not unsupervised learning. We elaborate on this further in Appendix A, where we construct a counter-example to a recent claim [32] that a so-called "exact" RG is equivalent to perfectly reconstructing the empirical probability distribution in an unsupervised problem. The information-distillation nature of renormalization is explicit in many numerical methods, where the purpose of the renormalization group is to efficiently and accurately evaluate the free energy of the system as a function of macroscopic variables of interest such as temperature and pressure. Thus we can only sensibly talk about the accuracy of an RG-scheme once we have specified what macroscopic variables we are interested in.

A subtlety regarding the above statements is presented by the Multi-scale Entanglement Renormalization Ansatz (MERA) [34]. MERA can be viewed as a variational class of wave functions whose parameters can be tuned to to match a given wave function as closely as possible. From this perspective, MERA is as an unsupervised machine learning algorithm, where classical probability distributions over many variables are replaced with quantum wavefunctions. Due to the special tensor network structure found in MERA, the resulting variational approximation of a given wavefunction has an interpretation as generating an RG flow. Hence this is an example of an unsupervised learning problem whose solution gives rise to an RG flow. This is only possible due to the extra mathematical structure in the problem (the specific tensor network found in MERA); a generic variational Ansatz does not give rise to any RG interpretation and vice versa.

F. No-flattening theorems

Above we discussed how Markovian generative models cause $p(y|x)$ to be a composition of a number of simpler functions f_i. Suppose that we can approximate each function f_i with an efficient neural network for the reasons given in Section II. Then we can simply stack these networks on top of each other, to obtain an deep neural network efficiently approximating $p(y|x)$.

But is this the most efficient way to represent $p(y|x)$? Since we know that there are shallower networks that accurately approximate it, are any of these shallow networks as efficient as the deep one, or does flattening necessarily come at an efficiency cost?

To be precise, for a neural network $\mathbf{f}$ defined by equation (6), we will say that the neural network $\mathbf{f}_\epsilon^\ell$ is the *flattened* version of $\mathbf{f}$ if its number ℓ of hidden layers is smaller and $\mathbf{f}_\epsilon^\ell$ approximates $\mathbf{f}$ within some error ϵ (as measured by some reasonable norm). We say that $\mathbf{f}_\epsilon^\ell$ is a *neuron-efficient flattening* if the sum of the dimensions of its hidden layers (sometimes referred to as the number of neurons N_n) is less than for $\mathbf{f}$. We say that $\mathbf{f}_\epsilon^\ell$ is a

synapse-efficient flattening if the number N_s of non-zero entries (sometimes called synapses) in its weight matrices is less than for $\mathbf{f}$. This lets us define the *flattening cost* of a network $\mathbf{f}$ as the two functions

$$C_n(\mathbf{f},\ell,\epsilon) \equiv \min_{\mathbf{f}_\epsilon^\ell} \frac{N_n(\mathbf{f}_\epsilon^\ell)}{N_n(\mathbf{f})}, \tag{25}$$

$$C_s(\mathbf{f},\ell,\epsilon) \equiv \min_{\mathbf{f}_\epsilon^\ell} \frac{N_s(\mathbf{f}_\epsilon^\ell)}{N_s(\mathbf{f})}, \tag{26}$$

specifying the factor by which optimal flattening increases the neuron count and the synapse count, respectively. We refer to results where $C_n > 1$ or $C_s > 1$ for some class of functions $\mathbf{f}$ as *"no-flattening theorems"*, since they imply that flattening comes at a cost and efficient flattening is impossible. A complete list of no-flattening theorems would show exactly when deep networks are more efficient than shallow networks.

There has already been very interesting progress in this spirit, but crucial questions remain. On one hand, it has been shown that deep is not always better, at least empirically for some image classification tasks [35]. On the other hand, many functions $\mathbf{f}$ have been found for which the flattening cost is significant. Certain deep Boolean circuit networks are exponentially costly to flatten [36]. Two families of multivariate polynomials with an exponential flattening cost C_n are constructed in[10]. [11, 12, 37] focus on functions that have tree-like hierarchical compositional form, concluding that the flattening cost C_n is exponential for almost all functions in Sobolev space. For the ReLU activation function, [38] finds a class of functions that exhibit exponential flattening costs; [39] study a tailored complexity measure of deep versus shallow ReLU networks. [40] shows that given weak conditions on the activation function, there always exists at least one function that can be implemented in a 3-layer network which has an exponential flattening cost. Finally, [41, 42] study the differential geometry of shallow versus deep networks, and find that flattening is exponentially neuron-inefficient. Further work elucidating the cost of flattening various classes of functions will clearly be highly valuable.

G. Linear no-flattening theorems

In the mean time, we will now see that interesting no-flattening results can be obtained even in the simpler-to-model context of *linear* neural networks [43], where the $\boldsymbol{\sigma}$ operators are replaced with the identity and all biases are set to zero such that $\mathbf{A}_i$ are simply linear operators (matrices). Every map is specified by a matrix of real (or complex) numbers, and composition is implemented by matrix multiplication.

One might suspect that such a network is so simple that the questions concerning flattening become entirely trivial: after all, successive multiplication with n different matrices is equivalent to multiplying by a single matrix (their product). While the effect of flattening is indeed trivial for *expressibility* ($\mathbf{f}$ can express any linear function, independently of how many layers there are), this is not the case for the *learnability*, which involves nonlinear and complex dynamics despite the linearity of the network [43]. We will show that the *efficiency* of such linear networks is also a very rich question.

Neuronal efficiency is trivially attainable for linear networks, since all hidden-layer neurons can be eliminated without accuracy loss by simply multiplying all the weight matrices together. We will instead consider the case of synaptic efficiency and set $\ell = \epsilon = 0$.

Many divide-and-conquer algorithms in numerical linear algebra exploit some factorization of a particular matrix $\mathbf{A}$ in order to yield significant reduction in complexity. For example, when $\mathbf{A}$ represents the discrete Fourier transform (DFT), the fast Fourier transform (FFT) algorithm makes use of a sparse factorization of $\mathbf{A}$ which only contains $\mathcal{O}(n\log n)$ non-zero matrix elements instead of the naive single-layer implementation, which contains n^2 non-zero matrix elements. This is our first example of a linear no-flattening theorem: fully flattening a network that performs an FFT of n variables increases the synapse count N_s from $\mathcal{O}(n\log n)$ to $\mathcal{O}(n^2)$, *i.e.*, incurs a flattening cost $C_s = \mathcal{O}(n/\log n) \sim \mathcal{O}(n)$. This argument applies also to many variants and generalizations of the FFT such as the Fast Wavelet Transform and the Fast Walsh-Hadamard Transform.

Another important example illustrating the subtlety of linear networks is matrix multiplication. More specifically, take the input of a neural network to be the entries of a matrix $\mathbf{M}$ and the output to be $\mathbf{NM}$, where both $\mathbf{M}$ and $\mathbf{N}$ have size $n\times n$. Since matrix multiplication is linear, this can be exactly implemented by a 1-layer linear neural network. Amazingly, the naive algorithm for matrix multiplication, which requires n^3 multiplications, is not optimal: the Strassen algorithm [44] requires only $\mathcal{O}(n^\omega)$ multiplications (synapses), where $\omega = \log_2 7 \approx 2.81$, and recent work has cut this scaling exponent down to $\omega \approx 2.3728639$ [45]. This means that fully optimized matrix multiplication on a deep neural network has a flattening cost of at least $C_s = \mathcal{O}(n^{0.6271361})$.

Low-rank matrix multiplication gives a more elementary no-flattening theorem. If $\mathbf{A}$ is a rank-k matrix, we can factor it as $\mathbf{A} = \mathbf{BC}$ where $\mathbf{B}$ is a $k\times n$ matrix and $\mathbf{C}$ is an $n\times k$ matrix. Hence the number of synapses is n^2 for an $\ell = 0$ network and $2nk$ for an $\ell = 1$-network, giving a flattening cost $C_s = n/2k > 1$ as long as the rank $k < n/2$.

Finally, let us consider flattening a network $\mathbf{f} = \mathbf{AB}$, where $\mathbf{A}$ and $\mathbf{B}$ are random sparse $n\times n$ matrices such that each element is 1 with probability p and 0 with probability $1-p$. Flattening the network results in a matrix $F_{ij} = \sum_k A_{ik}B_{kj}$, so the probability that $F_{ij} = 0$ is $(1-p^2)^n$. Hence the number of non-zero components

will on average be $\left(1-(1-p^2)^n\right)n^2$, so

$$C_s = \frac{\left[1-(1-p^2)^n\right]n^2}{2n^2p} = \frac{1-(1-p^2)^n}{2p}. \quad (27)$$

Note that $C_s \leq 1/2p$ and that this bound is asymptotically saturated for $n \gg 1/p^2$. Hence in the limit where n is very large, flattening multiplication by sparse matrices $p \ll 1$ is horribly inefficient.

IV. CONCLUSIONS

We have shown that the success of deep and cheap (low-parameter-count) learning depends not only on mathematics but also on physics, which favors certain classes of exceptionally simple probability distributions that deep learning is uniquely suited to model. We argued that the success of shallow neural networks hinges on symmetry, locality, and polynomial log-probability in data from or inspired by the natural world, which favors sparse low-order polynomial Hamiltonians that can be efficiently approximated. Whereas previous universality theorems guarantee that there exists a neural network that approximates any smooth function to within an error ϵ, they cannot guarantee that the size of the neural network does not grow to infinity with shrinking ϵ or that the activation function σ does not become pathological. We show constructively that given a multivariate polynomial and any generic non-linearity, a neural network with a *fixed* size and a generic smooth activation function can indeed approximate the polynomial highly efficiently.

Turning to the separate question of depth, we have argued that the success of deep learning depends on the ubiquity of hierarchical and compositional generative processes in physics and other machine-learning applications. By studying the sufficient statistics of the generative process, we showed that the inference problem requires approximating a compositional function of the form $f_1 \circ f_2 \circ f_2 \circ \cdots$ that optimally distills out the information of interest from irrelevant noise in a hierarchical process that mirrors the generative process. Although such compositional functions can be efficiently implemented by a deep neural network as long as their individual steps can, it is generally *not* possible to retain the efficiency while flattening the network. We extend existing "no-flattening" theorems [10–12] by showing that efficient flattening is impossible even for many important cases involving *linear* networks.

Strengthening the analytic understanding of deep learning may suggest ways of improving it, both to make it more capable and to make it more robust. One promising area is to prove sharper and more comprehensive no-flattening theorems, placing lower and upper bounds on the cost of flattening networks implementing various classes of functions. A concrete example is placing tight lower and upper bounds on the number of neurons and synaptic weights needed to approximate a given polynomial. We conjecture that approximating a multiplication gate $x_1x_2\cdots x_n$ will require exponentially many neurons in n using non-pathological activation functions, whereas we have shown that allowing for $\log_2 n$ layers allows us to use only $\sim 4n$ neurons.

Physics	Machine learning
Hamiltonian	Surprisal $-\ln p$
Simple H	Cheap learning
Quadratic H	Gaussian p
Locality	Sparsity
Translationally symmetric H	Convnet
Computing p from H	Softmaxing
Spin	Bit
Free energy difference	KL-divergence
Effective theory	Nearly lossless data distillation
Irrelevant operator	Noise
Relevant operator	Feature

TABLE I: Physics-ML dictionary.

Acknowledgements: This work was supported by the Foundational Questions Institute `http://fqxi.org/`. We thank Tomaso Poggio and Bart Selman for helpful discussions and suggestions and the Center for Brains, Minds, and Machines (CBMM) for hospitality.

V. APPENDIX

A. Why matching partition functions do not imply matching probability distributions

Let us interpret the random variable $\mathbf{y}$ as as describing degrees of freedom which are in thermal equilibrium at unit temperature with respect to some Hamiltonian $H(\mathbf{y})$:

$$p(\mathbf{y}) = \frac{1}{Z}e^{-H(\mathbf{y})}, \quad (28)$$

where the normalization $Z \equiv \sum_{\mathbf{y}} e^{-H(\mathbf{y})}$. (Unlike before, we only require in this section that $H(\mathbf{y}) = -\ln p(\mathbf{y}) +$ constant.) Let $H(\mathbf{y},\mathbf{y}')$ be a Hamiltonian of two random variables $\mathbf{y}$ and $\mathbf{y}'$, *e.g.*,

$$\tilde{p}(\mathbf{y},\mathbf{y}') = \frac{1}{Z_{\rm tot}}e^{-H(\mathbf{y},\mathbf{y}')}, \quad (29)$$

where the normalization $Z_{\rm tot} \equiv \sum_{\mathbf{y}\mathbf{y}'} e^{-H(\mathbf{y},\mathbf{y}')}$.

It has been claimed [32] that $Z_{\rm tot} = Z$ implies $\tilde{p}(\mathbf{y}) \equiv \sum_{\mathbf{y}'} \tilde{p}(\mathbf{y},\mathbf{y}') = p(\mathbf{y})$. We will construct a family of counterexamples where $Z_{\rm tot} = Z$, but $\tilde{p}(\mathbf{y}) \neq p(\mathbf{y})$.

Let $\mathbf{y}'$ belong to the same space as $\mathbf{y}$ and take any non-constant function $K(\mathbf{y})$. We choose the joint Hamiltonian

$$H(\mathbf{y},\mathbf{y}') = H(\mathbf{y}) + H(\mathbf{y}') + K(\mathbf{y}) + \ln \tilde{Z}, \quad (30)$$

where $\tilde{Z} = \sum_{\mathbf{y}} e^{-[H(\mathbf{y})+K(\mathbf{y})]}$. Then

$$\begin{aligned} Z_{\rm tot} &= \sum_{\mathbf{y}\mathbf{y}'} e^{-H(\mathbf{y},\mathbf{y}')} \\ &= \frac{1}{\tilde{Z}} \sum_{\mathbf{y}\mathbf{y}'} e^{-\left[H(\mathbf{y})+K(\mathbf{y})+H(\mathbf{y}')\right]} \\ &= \frac{1}{\tilde{Z}} \sum_{\mathbf{y}} e^{-[H(\mathbf{y})+K(\mathbf{y})]} \sum_{\mathbf{y}'} e^{-H(\mathbf{y}')} \\ &= \frac{1}{\tilde{Z}} \cdot \tilde{Z} \cdot \sum_{\mathbf{y}'} e^{-H(\mathbf{y}')} = \sum_{\mathbf{y}} e^{-H(\mathbf{y})} = Z. \end{aligned} \tag{31}$$

So the partition functions agree. However, the marginalized probability distributions do not:

$$\begin{aligned} \tilde{p}(\mathbf{y}) &= \frac{1}{Z_{\rm tot}} \sum_{\mathbf{y}'} e^{-H(\mathbf{y},\mathbf{y}')} \\ &= \frac{1}{\tilde{Z}} e^{-[H(\mathbf{y})+K(\mathbf{y})]} \neq p(\mathbf{y}). \end{aligned} \tag{32}$$

Hence the claim that $Z = Z_{\rm tot}$ implies $\tilde{p}(\mathbf{y}) = p(\mathbf{y})$ is false. Note that our counterexample generalizes immediately to the case where there are one or more parameters x in the Hamiltonian $H(\mathbf{y}) \to H_x(\mathbf{y})$ that we might want to vary. For example, x could be one component of an external magnetic field. In this case, we simply choose $H_x(\mathbf{y},\mathbf{y}') = H_x(\mathbf{y}) + H_x(\mathbf{y}') + K(\mathbf{y}) + \ln \tilde{Z}_x$. This means that all derivatives of $\ln Z$ and $Z_{\rm tot}$ with respect to x can agree despite the fact that $\tilde{p} \neq p$. This is important because all macroscopic observables such as the average energy, magnetization, *etc.* can be written in terms of derivatives of $\ln Z$. This illustrates the point that an exact Kadanoff RG scheme that can be accurately used to compute physical observables nevertheless can fail to accomplish any sort of unsupervised learning. In retrospect, this is unsurprising since the point of renormalization is to compute macroscopic quantites, not to solve an unsupervised learning problem in the microscopic variables.

[1] Y. LeCun, Y. Bengio, and G. Hinton, Nature **521**, 436 (2015).
[2] S. Russell, D. Dewey, and M. Tegmark, AI Magazine **36** (2015).
[3] K. Hornik, M. Stinchcombe, and H. White, Neural networks **2**, 359 (1989).
[4] G. Cybenko, Mathematics of control, signals and systems **2**, 303 (1989).
[5] A. Pinkus, Acta Numerica **8**, 143 (1999).
[6] B. Gnedenko, A. Kolmogorov, B. Gnedenko, and A. Kolmogorov, Amer. J. Math. **105**, 28 (1954).
[7] E. T. Jaynes, Physical review **106**, 620 (1957).
[8] R. Kindermann and J. L. Snell (1980).
[9] M. Tegmark, A. Aguirre, M. J. Rees, and F. Wilczek, Physical Review D **73**, 023505 (2006).
[10] O. Delalleau and Y. Bengio, in *Advances in Neural Information Processing Systems* (2011), pp. 666–674.
[11] H. Mhaskar, Q. Liao, and T. Poggio, ArXiv e-prints (2016), 1603.00988.
[12] H. Mhaskar and T. Poggio, arXiv preprint arXiv:1608.03287 (2016).
[13] R. Adam, P. Ade, N. Aghanim, Y. Akrami, M. Alves, M. Arnaud, F. Arroja, J. Aumont, C. Baccigalupi, M. Ballardini, et al., arXiv preprint arXiv:1502.01582 (2015).
[14] U. Seljak and M. Zaldarriaga, arXiv preprint astro-ph/9603033 (1996).
[15] M. Tegmark, Physical Review D **55**, 5895 (1997).
[16] J. Bond, A. H. Jaffe, and L. Knox, Physical Review D **57**, 2117 (1998).
[17] M. Tegmark, A. de Oliveira-Costa, and A. J. Hamilton, Physical Review D **68**, 123523 (2003).
[18] P. Ade, N. Aghanim, C. Armitage-Caplan, M. Arnaud, M. Ashdown, F. Atrio-Barandela, J. Aumont, C. Baccigalupi, A. J. Banday, R. Barreiro, et al., Astronomy & Astrophysics **571**, A12 (2014).
[19] M. Tegmark, The Astrophysical Journal Letters **480**, L87 (1997).
[20] G. Hinshaw, C. Barnes, C. Bennett, M. Greason, M. Halpern, R. Hill, N. Jarosik, A. Kogut, M. Limon, S. Meyer, et al., The Astrophysical Journal Supplement Series **148**, 63 (2003).
[21] G. Hinton, Momentum **9**, 926 (2010).
[22] M. Émile Borel, Rendiconti del Circolo Matematico di Palermo (1884-1940) **27**, 247 (1909).
[23] R. A. Fisher, Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character **222**, 309 (1922).
[24] M. Riesenhuber and T. Poggio, Nature neuroscience **3**, 1199 (2000).
[25] S. Kullback and R. A. Leibler, Ann. Math. Statist. **22**, 79 (1951), URL `http://dx.doi.org/10.1214/aoms/1177729694`.
[26] T. M. Cover and J. A. Thomas, *Elements of information theory* (John Wiley & Sons, 2012).
[27] M. Kardar, *Statistical physics of fields* (Cambridge University Press, 2007).
[28] J. Cardy, *Scaling and renormalization in statistical physics*, vol. 5 (Cambridge university press, 1996).
[29] J. K. Johnson, D. M. Malioutov, and A. S. Willsky, ArXiv e-prints (2007), 0710.0013.
[30] C. Bény, ArXiv e-prints (2013), 1301.3124.
[31] S. Saremi and T. J. Sejnowski, Proceedings of the National Academy of Sciences **110**, 3071 (2013), http://www.pnas.org/content/110/8/3071.full.pdf, URL `http://www.pnas.org/content/110/8/3071.abstract`.
[32] P. Mehta and D. J. Schwab, ArXiv e-prints (2014), 1410.3831.
[33] E. Miles Stoudenmire and D. J. Schwab, ArXiv e-prints (2016), 1605.05775.
[34] G. Vidal, Physical Review Letters **101**, 110501 (2008), quant-ph/0610099.
[35] J. Ba and R. Caruana, in *Advances in neural information processing systems* (2014), pp. 2654–2662.
[36] J. Hastad, in *Proceedings of the eighteenth annual ACM symposium on Theory of computing* (ACM, 1986), pp.

6–20.
[37] T. Poggio, F. Anselmi, and L. Rosasco, Tech. Rep., Center for Brains, Minds and Machines (CBMM) (2015).
[38] M. Telgarsky, arXiv preprint arXiv:1509.08101 (2015).
[39] G. F. Montufar, R. Pascanu, K. Cho, and Y. Bengio, in *Advances in neural information processing systems* (2014), pp. 2924–2932.
[40] R. Eldan and O. Shamir, arXiv preprint arXiv:1512.03965 (2015).
[41] B. Poole, S. Lahiri, M. Raghu, J. Sohl-Dickstein, and S. Ganguli, ArXiv e-prints (2016), 1606.05340.
[42] M. Raghu, B. Poole, J. Kleinberg, S. Ganguli, and J. Sohl-Dickstein, ArXiv e-prints (2016), 1606.05336.
[43] A. M. Saxe, J. L. McClelland, and S. Ganguli, arXiv preprint arXiv:1312.6120 (2013).
[44] V. Strassen, Numerische Mathematik **13**, 354 (1969).
[45] F. Le Gall, in *Proceedings of the 39th international symposium on symbolic and algebraic computation* (ACM, 2014), pp. 296–303.