CrossMark

Why Does Deep and Cheap Learning Work So Well?

Henry W. Lin[1] · **Max Tegmark[2]** · **David Rolnick[3]**

© Springer Science+Business Media, LLC 2017

Abstract We show how the success of deep learning could depend 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 frequently be approximated through "cheap learning" with exponentially fewer parameters than generic ones. We explore how properties frequently encountered in physics such as symmetry, locality, compositionality, and polynomial log-probability translate into exceptionally simple neural networks. 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 the renormalization group. We prove various "no-flattening theorems" showing when efficient linear deep networks cannot be accurately approximated by shallow ones without efficiency loss; for example, we show that n variables cannot be multiplied using fewer than 2^n neurons in a single hidden layer.

1 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 [1,2]. 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

✉ Henry W. Lin
  henrylin@college.harvard.edu

1   Department of Physics, Harvard University, Cambridge, MA 02138, USA

2   Department of Physics, Massachusetts Institute of Technology, Cambridge, MA 02139, USA

3   Department of Mathematics, Massachusetts Institute of Technology, Cambridge, MA 02139, USA

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 [3].

1.1 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.

The task of approximating functions of many variables is central to most applications of machine learning, including unsupervised learning, classification and prediction, as illustrated in Fig. 1. 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.

When investigating the quality of a neural net, there are several important factors to consider:
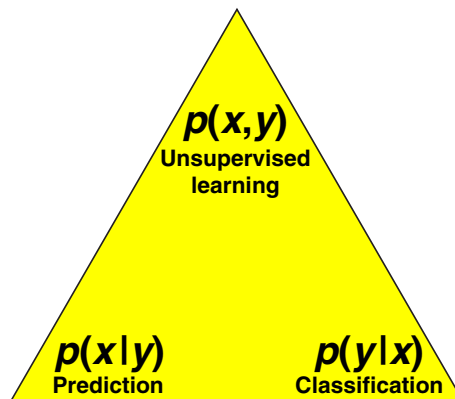

Fig. 1 In this paper, we follow the machine learning convention that **x** refers to data (e.g., an image) and **y** refers to underlying information about that data (such as a label for the image). Neural networks can be used to estimate (or sample from) probability distributions with respect to **x** and **y**, given many samples. *Classification* involves approximating the probability distribution of **y** given **x**, in the case that **y** is discrete-valued. This problem may also be called *prediction*, e.g. when **x** is earlier data in a time series. *Generation* involves approximating the probability distribution for **x** given **y**, or drawing samples from this distribution. *Unsupervised learning* attempts to approximate or model the probability distribution of **x**, without any knowledge of **y**

- *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 well-known [4–6] problem: *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 $256^{1000,000}$ 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 $256^{1000,000}$ 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 Sect. 2, we present results for shallow neural networks with merely a handful of layers, focusing on simplifications due to locality, symmetry and polynomials. In Sect. 3, 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 Sect. 4.

2 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. We will be interested in probability distributions $p(\mathbf{x}|y)$, where $\mathbf{x}$ ranges over some sample space and y will be interpreted either as another variable being conditioned on or as a model parameter. For a machine learning example, we might interpret y as an element of some set of animals {cat, dog, rabbit, …} and $\mathbf{x}$ as the vector of pixels in an image depicting such an animal, so that $p(\mathbf{x}|y)$ for $y = cat$ 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 y as an element of some set of metals {iron, aluminum, copper, …} and $\mathbf{x}$ as the vector of magnetization values for different parts of a metal bar. The prediction problem is then to evaluate $p(\mathbf{x}|y)$, whereas the classification problem is to evaluate $p(y|\mathbf{x})$.

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 $p(\mathbf{x}|y)$ 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 $\mathbf{x}$ 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 $p(y|\mathbf{x})$ on y tends to be more complicated; it makes no sense to speak of symmetries or polynomials involving a variable $y = cat$.

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

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

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

2.1 Probabilities and Hamiltonians

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

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

Table 1 is a brief dictionary translating between physics and machine-learning terminology.

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

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

where

$$N(\mathbf{x}) \equiv \sum_y e^{-[H_y(\mathbf{x})+\mu_y]}. \tag{4}$$

Table 1 Physics-ML dictionary

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

This recasting of Eq. (1) is useful because the Hamiltonian tends to have properties making it simple to evaluate. We will see in Sect. 3 that it also helps understand the relation between deep learning and renormalization [7].

2.2 Bayes' Theorem as a Softmax

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

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

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 $\mathbf{p}(\mathbf{x})$ 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]

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

where the $\boldsymbol{\sigma}_i$ are relatively simple nonlinear operators on vectors and the $\mathbf{A}_i$ are affine transformations of the form $\mathbf{A}_i \mathbf{x} = \mathbf{W}_i \mathbf{x} + \mathbf{b}_i$ 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

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

(We use $\tilde{\sigma}$ to indicate the softmax function and σ to indicate an arbitrary non-linearity, optionally with certain regularity requirements).

This allows us to rewrite Eq. (5) as

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

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

2.3 What Hamiltonians can be Approximated by Feasible Neural Networks?

It has long been known that neural networks are universal[1] approximators [8,9], in the sense that networks with virtually all popular nonlinear activation functions $\sigma(x)$ can approximate any smooth function to any desired accuracy—even using merely a single hidden layer.

---

[1] Neurons are universal analog computing modules in much the same way that NAND gates are universal digital computing modules: any computable function can be accurately evaluated by a sufficiently large network of them. Just as NAND gates are not unique (NOR gates are also universal), nor is any particular neuron implementation—indeed, *any* generic smooth nonlinear activation function is universal [8,9].

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 Sect. 2.4 that both physics and machine learning tend to favor Hamiltonians that are polynomials[2]—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:

$$H_y(\mathbf{x}) = h + \sum_i h_i x_i + \sum_{i \leq j} h_{ij} x_i x_j + \sum_{i \leq j \leq k} h_{ijk} x_i x_j x_k + \cdots . \tag{9}$$

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

2.3.1 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 $H_y(\mathbf{x})$ 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 $\sigma(x) = 1/(1 + e^{-x})$ will do the trick.

Theorem 1 *Let f be a neural network of the form $f = A_2 \sigma A_1$, where σ 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 f can approximate a multiplication gate arbitrarily well.*

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

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

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 A_1 to shift the origin to this point to ensure $\sigma_2 \neq 0$. Equation (10) now implies that

$$m(u, v) \equiv \frac{\sigma(u + v) + \sigma(-u - v) - \sigma(u - v) - \sigma(-u + v)}{4\sigma_2}$$
$$= uv \left[1 + \mathcal{O}\left(u^2 + v^2\right) \right], \tag{11}$$

where we will term $m(u, v)$ the *multiplication approximator*. Taylor's theorem guarantees that $m(u, v)$ is an arbitrarily good approximation of uv for arbitrarily small $|u|$ and $|v|$. However, we can always make $|u|$ and $|v|$ arbitrarily small by scaling $A_1 \rightarrow \lambda A_1$ and

---

[2] 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 $\sigma(x) = \max[0, x]$), 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.
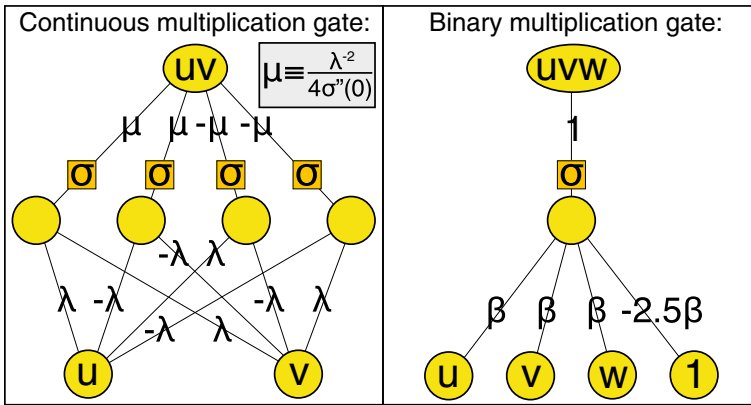
Fig. 2 Multiplication can be efficiently implemented by simple neural nets, becoming arbitrarily accurate as $\lambda \to 0$ (*left*) and $\beta \to \infty$ (*right*). Squares apply the function σ, circles perform summation, and lines multiply by the constants labeling them. The "1" input implements 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

then compensating by scaling $\mathbf{A}_2 \to \lambda^{-2}\mathbf{A}_2$. In the limit that $\lambda \to \infty$, this approximation becomes exact.[3] In other words, arbitrarily accurate multiplication can always be achieved using merely 4 neurons. Figure 2 illustrates such a multiplication approximator. (Of course, a practical algorithm like stochastic gradient descent cannot achieve arbitrarily large weights, though a reasonably good approximation can be achieved already for $\lambda^{-1} \sim 10$.)

Corollary 1 *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.*[4]

This is a stronger statement than the classic universal universal approximation theorems for neural networks [8,9], 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 [10] provides an ϵ-independent bound on the size of the neural network, but at the price of choosing a pathological function σ.

2.3.2 Discrete Input Variables

For the simple but important case where $\mathbf{x}$ is a vector of bits, so that $x_i = 0$ or $x_i = 1$, the fact that $x_i^2 = x_i$ makes things even simpler. This means that only terms where all variables

---

[3] The limit where $\lambda \to \infty$ but $|A_1|^2 |A_2|$ is held constant is very similar in spirit to the 't Hooft limit in large N quantum field theories where $g^2 N$ is held fixed but $N \to \infty$. The extra terms in the Taylor series which are suppressed at large λ are analogous to the suppression of certain Feynman diagrams at large N. The authors thank Daniel Roberts for pointing this out.

[4] 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 $\mathbf{A}_2$.

are different need be included, which simplifies Eq. (9) to

$$H_y(\mathbf{x}) = h + \sum_i h_i x_i + \sum_{i<j} h_{ij} x_i y_j + \sum_{i<j<k} h_{ijk} x_i x_j x_k + \cdots . \tag{12}$$

The infinite series Eq. (9) thus gets replaced by a finite series with 2^n terms, ending with the term $h_{1\dots n} x_1 \cdots x_n$. Since there are 2^n possible bit strings $\mathbf{x}$, the 2^n $h-$parameters in Eq. (12) suffice to exactly parametrize an arbitrary function $H_y(\mathbf{x})$.

The efficient multiplication approximator above multiplied only two variables at a time, thus requiring multiple layers to evaluate general polynomials. In contrast, $H(\mathbf{x})$ for a bit vector $\mathbf{x}$ can be implemented using merely three layers as illustrated in Fig. 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 $x_1 x_2 x_3 = 1$ if and only if the sum $x_1 + x_2 + x_3 = 3$. This sum-checking can be implemented using one of the most popular choices for a nonlinear function σ: the logistic sigmoid $\sigma(x) = \frac{1}{1+e^{-x}}$ which satisfies $\sigma(x) \approx 0$ for $x \ll 0$ and $\sigma(x) \approx 1$ for $x \gg 1$. 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

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

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 $\mathbf{x}$ is a bit string, an *arbitrary* function $p_y(\mathbf{x})$ can be evaluated by a simple 3-layer neural network: the middle layer uses sigmoid functions to compute the products from Eq. (12), and the top layer performs the sums from Eq. (12) and the softmax from Eq. (8).

2.4 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 discuss situations in which exceptionally simple polynomials that are sparse, symmetric and/or low-order play a special role in physics and machine learning.

2.4.1 Low Polynomial Order

The Hamiltonians that show up in physics are not random functions, but tend to be polynomials of very low order, typically of degree ranging from 2 to 4. The simplest example is of course the harmonic oscillator, which is described by a Hamiltonian that is quadratic in both position and momentum. There are many reasons why low order polynomials show up in physics. Two of the most important ones are that sometimes a phenomenon can be studied perturbatively, in which case, Taylor's theorem suggests that we can get away with a low order polynomial approximation. A second reason is renormalization: higher order

terms in the Hamiltonian of a statistical field theory tend to be negligible if we only observe macroscopic variables.

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 in many examples is not infinite as in Eq. (9) or exponential in n as in Eq. (12), merely of order $O(n^4)$.

There are additional reasons why we might expect low order polynomials. Thanks to the Central Limit Theorem [11], many probability distributions in machine learning and statistics can be accurately approximated by multivariate Gaussians, *i.e.*, of the form

$$p(\mathbf{x}) = e^{h + \sum_i h_j x_i - \sum_{ij} h_{ij} x_i x_j},\tag{14}$$

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 $\langle x_1^{\alpha_1} x_2^{\alpha_2} \cdots x_n^{\alpha_n} \rangle$ for some integers $\alpha_i \geq 0$ would lead to a Hamiltonian of degree no greater than $d \equiv \sum_i \alpha_i$ [12].

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 functions ($d = 1$ polynomials) of the pixel vector $\mathbf{x}$. Functions implementing convolutions and Fourier transforms are also $d = 1$ polynomials.

Of course, such arguments do not imply that we should expect to see low-order polynomials in every application. If we consider some data set generated by a very simple Hamiltonian (say the Ising Hamiltonian), but then discard some of the random variables, the resulting marginalized distribution can become quite complicated and of high order. Similarly, if we do not observe the random variables directly, but observe some generic functions of the random variables, the result will generally be a mess. These arguments, however, might indicate that the probability of encountering a Hamiltonian described by a low-order polynomial in some application might be significantly higher than what one might expect from some naive prior. For example, a uniform prior on the space of all polynomials of degree N would suggest that a randomly chosen polynomial would almost always have degree N, but this might be a bad prior for real-world applications.

We should also note that even if a Hamiltonian is described exactly by a low-order polynomial, we would not expect the corresponding neural network to reproduce a low-order polynomial Hamiltonian exactly in any practical scenario for a host of possible reasons including limited data, the requirement of infinite weights for infinite accuracy, and the failure of practical algorithms such as stochastic gradient descent to find the global minimum of a cost function in many scenarios. So looking at the weights of a neural network trained on actual data may not be a good indicator of whether or not the underlying Hamiltonian is a polynomial of low degree or not.

2.4.2 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 Eq. (9) are forced to vanish, and the total number of non-zero coefficients grows only linearly with n. For the binary case of Eq. (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).

Again, the applicability of these considerations to particular machine learning applications must be determined on a case by case basis. Certainly, an arbitrary transformation of a collection of local random variables will result in a non-local collection. (This might ruin locality in certain ensembles of images, for example). But there are certainly cases in physics where locality is still approximately preserved, for example in the simple block-spin renormalization group, spins are grouped into blocks, which are then treated as random variables. To a high degree of accuracy, these blocks are only coupled to their nearest neighbors. Such locality is famously exploited by both biological and artificial visual systems, whose first neuronal layer performs merely fairly local operations.

2.4.3 Symmetry

Whenever the Hamiltonian obeys some symmetry (is invariant under some transformation), the number of independent 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 $\mathbf{x}$ of air pressures y_i measured by a microphone at times $i = 1, \ldots, n$. 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 [13]. Naturally, this does not mean that modeling a real physical system requires merely 32 parameters - the objects involved must be modeled also; there too, however, symmetry allows us to abstract away from the information contained in individual particles to that summarizing components of the system.

Symmetry can reduce not merely the parameter count, but also the computational complexity. For example, if a linear vector-valued function $\mathbf{f}(\mathbf{x})$ 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.

3 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 [14–16], 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.

3.1 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 $y_0 \mapsto y_1 \mapsto \ldots \mapsto y_n$ that are relevant to physics and image classification, respectively. Both examples involve a Markov chain[5] where the probability distribution $p(y_i)$ at the ith 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 $p(y_i)$ according to $(\mathbf{p}_i)_y \equiv p(y_i)$ and the Markov matrix $\mathbf{M}_i$ specifies the transition probabilities between two neighboring levels, $p(y_i|y_{i-1})$. Iterating Eq. (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 (Fig. 3, left), a set of cosmological parameters y_0 (the density of dark matter, *etc*) determines the power spectrum y_1 of density fluctuations in our universe, which in turn determines the pattern of cosmic microwave background radiation y_2 reaching us from our early universe, which gets combined with foreground radio noise from our Galaxy to produce the frequency-dependent sky maps (y_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 [17], these datasets y_i, y_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 Eq. (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 (Fig. 3, right) is deliberately contrived and over-simplified for pedagogy: y_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 generated background is added.

In both examples, the goal is to reverse this generative hierarchy to learn about the input $y \equiv y_0$ from the output $y_n \equiv x$, specifically to provide the best possibile estimate of the probability distribution $p(y|y) = p(y_0|y_n)$—*i.e.*, to determine the probability distribution for the cosmological parameters and to determine the probability that the image is a cat, respectively.

---

[5] 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.
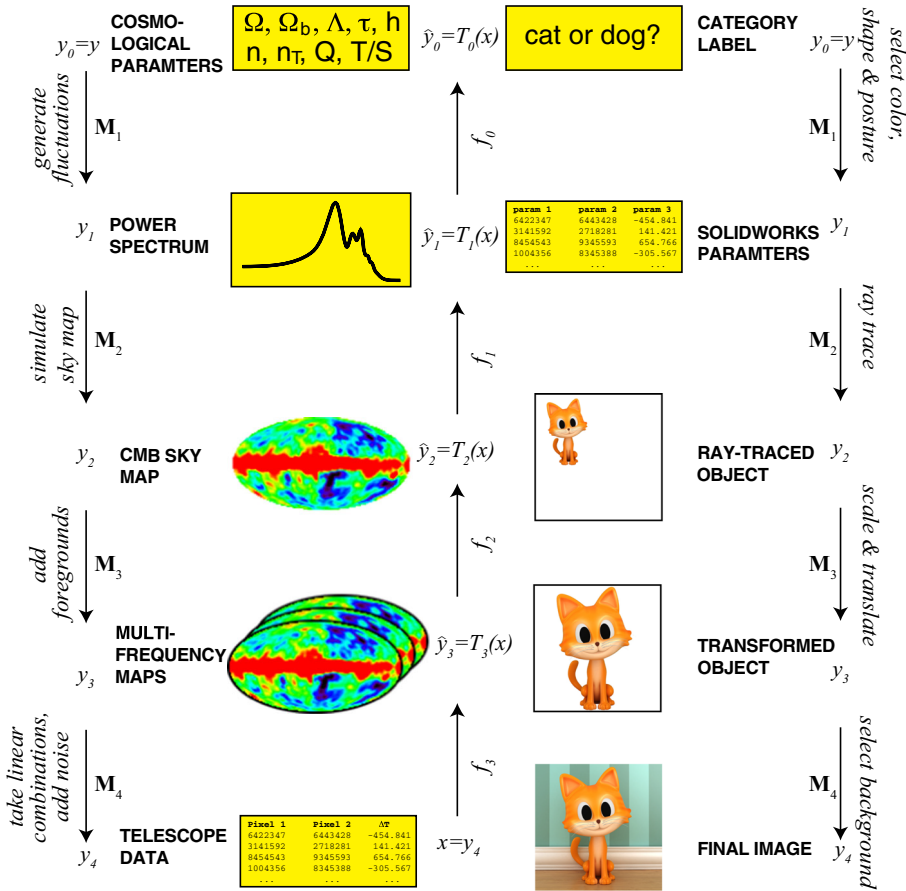
Fig. 3 Causal hierarchy examples relevant to physics (*left*) and image classification (*right*). As information flows down the hierarchy $y_0 \to y_1 \to \ldots \to y_n = y$, some of it is destroyed by random Markov processes. However, no further information is lost as information flows optimally back up the hierarchy as $\widehat{y}_{n-1} \to \ldots \to \widehat{y}_0$. 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.

3.2 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 x requires far more bits than there are atoms in our universe, the information specifying how to compute the probability distribution $p(x|y)$ for a microwave background map fits into a handful of published journal articles or software packages [18–24]. For a megapixel image of a galaxy, its entire probability distribution is defined by the standard model of particle physics with its 32 parameters [13], 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 learning: 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 Sect. 2.4, it is plausible that neural networks can implement each of the steps efficiently.[6]

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 [26], 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.

3.3 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 y from the output x. Let us now treat this hierarchical problem more rigorously using information theory.

Given $P(y|x)$, a *sufficient statistic* $T(x)$ is defined by the equation $P(y|x) = P(y|T(x))$ and has played an important role in statistics for almost a century [27]. All the information about y contained in x is contained in the sufficient statistic. A *minimal sufficient statistic* [27] 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 Fig. 3, T_* can be thought of as a an information distiller, optimally compressing the data so as to retain all information relevant to determining y 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 Eq. (16).

Theorem 2 *Given a Markov chain described by our notation above, let T_i be a minimal sufficient statistic of $P(y_i|y_n)$. Then there exists some functions f_i such that $T_i = f_i \circ T_{i+1}$.*

---

[6] 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 [25] do the same.

More casually speaking, the generative hierarchy of Fig. 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 $k \geq 1$, the "backwards" Markov property $P(y_i|y_{i+1}, y_{i+k}) = P(y_i|y_{i+1})$ follows from the Markov property via Bayes' theorem:

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

Using this fact, we see that

$$
\begin{aligned}
P(y_i|y_n) &= \sum_{y_{i+1}} P(y_i|y_{i+1}y_n)P(y_{i+1}|y_n) \\
&= \sum_{y_{i+1}} P(y_i|y_{i+1})P(y_{i+1}|T_{i+1}(y_n)).
\end{aligned}
\tag{18}
$$

Since the above equation depends on y_n only through $T_{i+1}(y_n)$, this means that T_{i+1} is a sufficient statistic for $P(y_i|y_n)$. 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 $f_0(T_0) = P(y_0|T_0)$ and $f_n = T_{n-1}$. Then*

$$
P(y_0|y_n) = (f_0 \circ f_1 \circ \cdots \circ f_n)(y_n).
\tag{19}
$$

The proof is easy. By induction,

$$
T_0 = f_1 \circ f_2 \circ \cdots \circ T_{n-1}, \text{ which implies the corollary.}
\tag{20}
$$

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 $P(y|x)$ must approximate a compositional function. We will argue below in Sect. 3.6 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 [28], containing features invariant under irrelevant transformations (for example background substitution, scaling and translation).

Let us denote the distilled vectors $\widehat{y}_i \equiv f_i(\widehat{y}_{i+1})$, where $\widehat{y}_n \equiv y$. As summarized by Fig. 3, as information flows down the hierarchy $y = y_0 \rightarrow y_1 \rightarrow \ldots ex_n = x$, some of it is destroyed by random processes. However, no further information is lost as information flows optimally back up the hierarchy as $y \rightarrow \widehat{y}_{n-1} \rightarrow \cdots \rightarrow \widehat{y}_0$.

3.4 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 [29,30]. The *data processing inequality* [30] 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 $T(x)$ is a function $f(x)$ for which "$\geq$" gets replaced by "$=$" in Eq. (21), *i.e.*, a function retaining all the information about y.

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., $H_y(f(x))$ may be considerably easier to implement in a neural network than $H_y(x)$. Precisely this situation applies to the physical example described in Fig. 3, where a hierarchy of efficient near-perfect information distillers f_i have been found, the numerical cost of f_3 [23,24], f_2 [21,22], f_1 [19,20] and f_0 [17] 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. More abstractly, the procedure of renormalization, ubiquitous in statistical physics, can be viewed as a special case of approximate information distillation, as we will now describe.

3.5 Distillation and Renormalization

The systematic framework for distilling out desired information from unwanted "noise" in physical theories is known as Effective Field Theory [31]. 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 [31,32]. Although the connection between RG and machine learning has been studied or alluded to repeatedly [7,33–36], 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 $H_y(x)$, with some parameter vector y. We interpret the map $R : y \rightarrow y$ as implementing a coarse-graining[7] 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_y, although the parameters y may change. In other words, $H'(R(x)) = H_{r(y)}(R(x))$ 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 $H_{r^n(y)}(R^n(x))$ for the repeatedly renormalized data. Similar to the case of sufficient statistics, $P(y|R^n(x))$ will then be a compositional function.

---

[7] 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.

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 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 x 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). But without specifying that we wish to predict (long-wavelength physics), there is nothing natural about an effective field theory approximation.

To be more explicit about the link between renormalization and deep-learning, consider a toy model for natural images. Each image is described by an intensity field $\phi(\mathbf{r})$, where $\mathbf{r}$ is a 2-dimensional vector. We assume that an ensemble of images can be described by a quadratic Hamiltonian of the form

$$H_y(\phi) = \int \left[y_0\phi^2 + y_1(\nabla\phi)^2 + y_2\left(\nabla^2\phi\right)^2 + \cdots \right] d^2r. \tag{23}$$

Each parameter vector y defines an ensemble of images; we could imagine that the fictitious classes of images that we are trying to distinguish are all generated by Hamiltonians H_y with the same above form but different parameter vectors y. We further assume that the function $\phi(\mathbf{r})$ 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 Eq. (23) allows it to be implemented with a convnet. If can be shown [31] 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 Eq. (23) but with the parameters y_i replaced by

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

This means that all parameters y_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 y_i's. What would have taken an arbitrarily large neural network can now be computed on a neural network of finite and bounded size, assuming that we are only interested in classifying the data based only on the coarse-grained variables. These insufficient statistics will still have discriminatory power if we are only interested in discriminating Hamiltonians which all differ in their first few C_k. In this example, the parameters y_0 and y_1 correspond to "relevant operators" by physicists and "signal" by machine-learners, whereas the remaining parameters correspond to "irrelevant operators" by physicists and "noise" by machine-learners.

The fixed point structure of the transformation in this example is very simple, but one can imagine that in more complicated problems the fixed point structure of various transformations might be highly non-trivial. This is certainly the case in statistical mechanics problems where renormalization methods are used to classify various phases of matters; the point here is that the renormalization group flow can be thought of as solving the pattern-recognition problem of classifying the long-range behavior of various statistical systems.

In summary, renormalization can be thought of as a type of supervised learning,[8] where the large scale properties of the system are considered the features. If the desired features are not large-scale properties (as in most machine learning cases), one might still expect the a generalized formalism of renormalization to provide some intuition to the problem by replacing a scale transformation with some other transformation. But calling some procedure renormalization or not is ultimately a matter of semantics; what remains to be seen is whether or not semantics has teeth, namely, whether the intuition about fixed points of the renormalization group flow can provide concrete insight into machine learning algorithms. In many numerical methods, 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.

3.6 No-Flattening Theorems

Above we discussed how Markovian generative models cause $p(x|y)$ 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 Sect. 2. Then we can simply stack these networks on top of each other, to obtain an deep neural network efficiently approximating $p(x|y)$.

But is this the most efficient way to represent $p(x|y)$? 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 Eq. (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.

---

[8] A subtlety regarding the above statements is presented by the Multi-scale Entanglement Renormalization Ansatz (MERA) [37]. 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.

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 [38]. On the other hand, many functions **f** have been found for which the flattening cost is significant. Certain deep Boolean circuit networks are exponentially costly to flatten [39]. Two families of multivariate polynomials with an exponential flattening cost C_n are constructed in [14]. Poggio et al. [6], Mhaskar et al. [15], Mhaskar and Poggio [16] 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, [40] finds a class of functions that exhibit exponential flattening costs; [41] study a tailored complexity measure of deep versus shallow ReLU networks. Eldan and Shamir [42] 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, [43,44] 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.

3.7 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 [45], where the **σ** 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* (**f** can express any linear function, independently of how many layers there are), this is not the case for the *learnability*, which involves non-linear and complex dynamics despite the linearity of the network [45]. 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 **A** in order to yield significant reduction in complexity. For example, when **A** represents the discrete Fourier transform (DFT), the fast Fourier transform (FFT) algorithm makes use of a sparse factorization of **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. As first pointed out in [46], this is an example where depth helps and, in our terminology, 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 **M** and the output to be **NM**, where both **M** and **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 opti-

mal: the Strassen algorithm [47] 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 $\omega \approx 2.3728639$ [48]. 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

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

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.

3.8 A Polynomial No-Flattening Theorem

In Sect. 2, we saw that multiplication of two variables could be implemented by a flat neural network with 4 neurons in the hidden layer, using Eq. (11) as illustrated in Fig. 2. In Appendix A, we show that Eq. (11) is merely the $n = 2$ special case of the formula

$$\prod_{i=1}^n x_i = \frac{1}{2^n} \sum_{\{s\}} s_1 \ldots s_n \sigma(s_1 x_1 + \cdots + s_n x_n), \tag{28}$$

where the sum is over all possible 2^n configurations of $s_1, \cdots s_n$ where each s_i can take on values ± 1. In other words, multiplication of n variables can be implemented by a flat network with 2^n neurons in the hidden layer. We also prove in Appendix A that this is the best one can do: no neural network can implement an n-input multiplication gate using fewer than 2^n neurons in the hidden layer. This is another powerful no-flattening theorem, telling us that polynomials are exponentially expensive to flatten. For example, if n is a power of two, then the monomial $x_1 x_2 \ldots x_n$ can be evaluated by a deep network using only $4n$ neurons arranged in a deep neural network where n copies of the multiplication gate from Fig. 2 are arranged in a binary tree with $\log_2 n$ layers (the 5th top neuron at the top of Fig. 2 need not be counted, as it is the input to whatever computation comes next). In contrast, a functionally equivalent flattened network requires a whopping 2^n neurons. For example, a deep neural network can multiply 32 numbers using $4n = 160$ neurons while a shallow one requires $2^{32} = 4,294,967,296$ neurons. Since a broad class of real-world functions can be well approximated by polynomials, this helps explain why many useful neural networks cannot be efficiently flattened.

4 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. These arguments should be particularly relevant for explaining the success of machine learning applications to physics, for example using a neural network to approximate a many-body wavefunction [49]. 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 [14–16] by showing that efficient flattening is impossible even for many important cases involving *linear* networks. In particular, we prove that flattening polynomials is exponentially expensive, with 2^n neurons required to multiply n numbers using a single hidden layer, a task that a deep network can perform using only $\sim 4n$ neurons.

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.

Appendix A: The Polynomial No-Flattening Theorem

We saw above that a neural network can compute polynomials accurately and efficiently at linear cost, using only about 4 neurons per multiplication. For example, if n is a power of two, then the monomial $\prod_{i=1}^{n} x_i$ can be evaluated using $4n$ neurons arranged in a binary tree network with $\log_2 n$ hidden layers. In this appendix, we will prove a no-flattening theorem demonstrating that flattening polynomials is exponentially expensive.

Theorem *Suppose we are using a generic smooth activation function $\sigma(x) = \sum_{k=0}^{\infty} \sigma_k x^k$, where $\sigma_k \neq 0$ for $0 \leq k \leq n$. Then for any desired accuracy $\epsilon > 0$, there exists a neural network that can implement the function $\prod_{i=1}^{n} x_i$ using a single hidden layer of 2^n neurons.*

Furthermore, this is the smallest possible number of neurons in any such network with only a single hidden layer.

This result may be compared to problems in Boolean circuit complexity, notably the question of whether $TC^0 = TC^1$ [50]. Here circuit depth is analogous to number of layers, and the number of gates is analogous to the number of neurons. In both the Boolean circuit model and the neural network model, one is allowed to use neurons/gates which have an unlimited number of inputs. The constraint in the definition of TC^i that each of the gate elements be from a standard universal library (AND, OR, NOT, Majority) is analogous to our constraint to use a particular nonlinear function. Note, however, that our theorem is weaker by applying only to depth 1, while TC^0 includes all circuits of depth $O(1)$.

A.1 Proof that 2^n Neurons are Sufficient

A neural network with a single hidden layer of m neurons that approximates a product gate for n inputs can be formally written as a choice of constants a_{ij} and w_j satisfying

$$\sum_{j=1}^{m} w_j \sigma \left(\sum_{i=1}^{n} a_{ij} x_i \right) \approx \prod_{i=1}^{n} x_i. \tag{A1}$$

Here, we use $\approx$ to denote that the two sides of (A1) have identical Taylor expansions up to terms of degree n; as we discussed earlier in our construction of a product gate for two inputs, this exables us to achieve arbitrary accuracy ϵ by first scaling down the factors x_i, then approximately multiplying them and finally scaling up the result.

We may expand (A1) using the definition $\sigma(x) = \sum_{k=0}^{\infty} \sigma_k x^k$ and drop terms of the Taylor expansion with degree greater than n, since they do not affect the approximation. Thus, we wish to find the minimal m such that there exist constants a_{ij} and w_j satisfying

$$\sigma_n \sum_{j=1}^{m} w_j \left(\sum_{i=1}^{n} a_{ij} x_i \right)^n = \prod_{i=1}^{n} x_i, \tag{A2}$$

$$\sigma_k \sum_{j=1}^{m} w_j \left(\sum_{i=1}^{n} a_{ij} x_i \right)^k = 0, \tag{A3}$$

for all $0 \leq k \leq n - 1$. Let us set $m = 2^n$, and enumerate the subsets of $\{1, \ldots, n\}$ as $S_1, \ldots, S_m$ in some order. Define a network of m neurons in a single hidden layer by setting a_{ij} equal to the function $s_i(S_j)$ which is -1 if $i \in S_j$ and $+1$ otherwise, setting

$$w_j \equiv \frac{1}{2^n n! \sigma_n} \prod_{i=1}^{n} a_{ij} = \frac{(-1)^{|S_j|}}{2^n n! \sigma_n}. \tag{A4}$$

In other words, up to an overall normalization constant, all coefficients a_{ij} and w_j equal ± 1, and each weight w_j is simply the product of the corresponding a_{ij}.

We must prove that this network indeed satisfies Eqs. (A2) and (A3). The essence of our proof will be to expand the left hand side of Eq. (A1) and show that all monomial terms except $x_1 \cdots x_n$ come in pairs that cancel. To show this, consider a single monomial $p(\mathbf{x}) = x_1^{r_1} \cdots x_n^{r_n}$ where $r_1 + \ldots + r_n = r \leq n$.

If $p(\mathbf{x}) \neq \prod_{i=1}^{n} x_i$, then we must show that the coefficient of $p(\mathbf{x})$ in $\sigma_r \sum_{j=1}^{m} w_j \left(\sum_{i=1}^{n} a_{ij} x_i \right)^r$ is 0. Since $p(\mathbf{x}) \neq \prod_{i=1}^{n} x_i$, there must be some i_0 such that $r_{i_0} = 0$. In other words, $p(\mathbf{x})$ does not depend on the variable x_{i_0}. Since the sum in Eq. (A1) is over all

combinations of $\pm$ signs for all variables, every term will be canceled by another term where the (non-present) x_{i_0} has the opposite sign and the weight w_j has the opposite sign:

$$\sigma_r \sum_{j=1}^{m} w_j \left(\sum_{i=1}^{n} a_{ij} x_i \right)^r$$

$$= \sigma_r \sum_{S_j} \frac{(-1)^{|S_j|}}{2^n n! \sigma_r} \left(\sum_{i=1}^{n} s_i(S_j) x_i \right)^r$$

$$= \sigma_r \sum_{S_j \not\ni i_0} \left[\frac{(-1)^{|S_j|}}{2^n n! \sigma_r} \left(\sum_{i=1}^{n} s_i(S_j) x_i \right)^r \right.$$

$$\left. + \frac{(-1)^{|S_j \cup \{i_0\}|}}{2^n n! \sigma_r} \left(\sum_{i=1}^{n} s_i(S_j \cup \{i_0\}) x_i \right)^r \right]$$

$$= \sum_{S_j \not\ni i_0} \frac{(-1)^{|S_j|}}{2^n n!} \left[\left(\sum_{i=1}^{n} s_i(S_j) x_i \right)^r \right.$$

$$\left. - \left(\sum_{i=1}^{n} s_i(S_j \cup \{i_0\}) x_i \right)^r \right]$$

Observe that the coefficient of $p(\mathbf{x})$ is equal in $\left(\sum_{i=1}^{n} s_i(S_j) x_i \right)^r$ and $\left(\sum_{i=1}^{n} s_i(S_j \cup \{i_0\}) x_i \right)^r$, since $r_{i_0} = 0$. Therefore, the overall coefficient of $p(\mathbf{x})$ in the above expression must vanish, which implies that (A3) is satisfied.

If instead $p(\mathbf{x}) = \prod_{i=1}^{n} x_i$, then all terms have the coefficient of $p(\mathbf{x})$ in $\left(\sum_{i=1}^{n} a_{ij} x_i \right)^n$ is $n! \prod_{i=1}^{n} a_{ij} = (-1)^{|S_j|} n!$, because all $n!$ terms are identical and there is no cancellation. Hence, the coefficient of $p(\mathbf{x})$ on the left-hand side of (A2) is

$$\sigma_n \sum_{j=1}^{m} \frac{(-1)^{|S_j|}}{2^n n! \sigma_n} (-1)^{|S_j|} n! = 1,$$

completing our proof that this network indeed approximates the desired product gate.

From the standpoint of group theory, our construction involves a representation of the group $G = \mathbb{Z}_2^n$, acting upon the space of polynomials in the variables $x_1, x_2, \ldots, x_n$. The group G is generated by elements g_i such that g_i flips the sign of x_i wherever it occurs. Then, our construction corresponds to the computation

$$\mathbf{f}(x_1, \ldots, x_n) = (1 - g_1)(1 - g_2) \cdots (1 - g_n) \sigma(x_1 + x_2 + \ldots + x_n).$$

Every monomial of degree at most n, with the exception of the product $x_1 \cdots x_n$, is sent to 0 by $(1 - g_i)$ for at least one choice of i. Therefore, $\mathbf{f}(x_1, \ldots, x_n)$ approximates a product gate (up to a normalizing constant).

A.2 Proof that 2^n Neurons are Necessary

Suppose that S is a subset of $\{1, \ldots, n\}$ and consider taking the partial derivatives of (A2) and (A3), respectively, with respect to all the variables $\{x_h\}_{h \in S}$. Then, we obtain the equalities

$$\frac{n! \sigma_n}{(n - |S|)!} \sum_{j=1}^{m} w_j \prod_{h \in S} a_{hj} \left(\sum_{i=1}^{n} a_{ij} x_i \right)^{n-|S|} = \prod_{h \notin S} x_h, \tag{A5}$$

$$\frac{k! \sigma_k}{(k - |S|)!} \sum_{j=1}^{m} w_j \prod_{h \in S} a_{hj} \left(\sum_{i=1}^{n} a_{ij} x_i \right)^{k-|S|} = 0, \tag{A6}$$

for all $0 \leq k \leq n - 1$. Let $\mathbf{A}$ denote the $2^n \times m$ matrix with elements

$$A_{Sj} \equiv \prod_{h \in S} a_{hj}. \tag{A7}$$

We will show that $\mathbf{A}$ has full row rank. Suppose, towards contradiction, that $\mathbf{c}^t \mathbf{A} = \mathbf{0}$ for some non-zero vector $\mathbf{c}$. Specifically, suppose that there is a linear dependence between rows of $\mathbf{A}$ given by

$$\sum_{\ell=1}^{r} c_\ell A_{S_\ell, j} = 0, \tag{A8}$$

where the S_ℓ are distinct and $c_\ell \neq 0$ for every ℓ. Let s be the maximal cardinality of any S_ℓ. Defining the vector $\mathbf{d}$ whose components are

$$d_j \equiv w_j \left(\sum_{i=1}^{n} a_{ij} x_i \right)^{n-s}, \tag{A9}$$

taking the dot product of Eq. (A8) with $\mathbf{d}$ gives

$$\begin{aligned} 0 = \mathbf{c}^t \mathbf{A} \mathbf{d} &= \sum_{\ell=1}^{r} c_\ell \sum_{j=1}^{m} w_j \prod_{h \in S_\ell} a_{hj} \left(\sum_{i=1}^{n} a_{ij} x_i \right)^{n-s} \\ &= \sum_{\ell | (|S_\ell| = s)} c_\ell \sum_{j=1}^{m} w_j \prod_{h \in S_\ell} a_{hj} \left(\sum_{i=1}^{n} a_{ij} x_i \right)^{n-|S_\ell|} \\ &\quad + \sum_{\ell | (|S_\ell| < s)} c_\ell \sum_{j=1}^{m} w_j \prod_{h \in S_\ell} a_{hj} \left(\sum_{i=1}^{n} a_{ij} x_i \right)^{(n+|S_\ell|-s)-|S_\ell|}. \end{aligned} \tag{A10}$$

Applying Eq. (A6) (with $k = n + |S_\ell| - s$) shows that the second term vanishes. Substituting Eq. (A5) now simplifies Eq. (A10) to

$$0 = \sum_{\ell | (|S_\ell| = s)} \frac{c_\ell (n - |S_\ell|)!}{n! \sigma_n} \prod_{h \notin S_\ell} x_h, \tag{A11}$$

i.e., to a statement that a set of monomials are linearly dependent. Since all distinct monomials are in fact linearly independent, this is a contradiction of our assumption that the S_ℓ are distinct and c_ℓ are nonzero. We conclude that $\mathbf{A}$ has full row rank, and therefore that $m \geq 2^n$, which concludes the proof.

References

1. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature **521**, 436–444 (2015)
2. Bengio, Y.: Learning deep architectures for AI, foundations and trends®. Mach. Learn. **2**, 1–127 (2009)
3. Russell, S., Dewey, D., Tegmark, M.: Research priorities for robust and beneficial artificial intelligence. AI Mag. **36**, 105–114 (2015)
4. Herbrich, R., Williamson, R.C.: Algorithmic luckiness. J. Mach. Learn. Res. **3**, 175–212 (2002)
5. Shawe-Taylor, J., Bartlett, P.L., Williamson, R.C., Anthony, M.: Structural risk minimization over data-dependent hierarchies. IEEE Trans. Inf. Theory **44**, 1926–1940 (1998)
6. Poggio, T., Anselmi, F., Rosasco, L.: I-theory on depth vs width: hierarchical function composition. Center Brains Minds Mach. (2015). Technical Reports
7. Mehta, P., Schwab, D.J.: An exact mapping between the variational renormalization group and deep learning. arXiv:1410.3831 (2014)
8. Hornik, K., Stinchcombe, M., White, H.: Multilayer feedforward networks are universal approximators. Neural Netw. **2**, 359–366 (1989)
9. Cybenko, G.: Approximation by superpositions of a sigmoidal function. Math. Control Signals Syst. **2**, 303–314 (1989)
10. Pinkus, A.: Approximation theory of the MLP model in neural networks. Acta Numer. **8**, 143–195 (1999)
11. Gnedenko, B., Kolmogorov, A., Gnedenko, B., Kolmogorov, A.: Limit distributions for sums of independent. Am. J. Math. **105**, 28–35 (1954)
12. Jaynes, E.T.: Information theory and statistical mechanics. Phys. Rev. **106**, 620 (1957)
13. Tegmark, M., Aguirre, A., Rees, M.J., Wilczek, F.: Dimensionless constants, cosmology, and other dark matters. Phys. Rev. D **73**, 023505 (2006)
14. Delalleau, O., Bengio, Y.: Shallow vs. deep sum-product networks. In: Advances in Neural Information Processing Systems, pp. 666–674 (2011)
15. Mhaskar, H., Liao, Q., Poggio, T.: Learning functions: when is deep better than shallow. arXiv:1603.00988 (2016)
16. Mhaskar, H., Poggio, T.: Deep vs. shallow networks: an approximation theory perspective. arXiv:1608.03287 (2016)
17. Adam, R., Ade, P., Aghanim, N., Akrami, Y., Alves, M., Arnaud, M., Arroja, F., Aumont, J., Baccigalupi, C., Ballardini, M., et al.: arXiv:1502.01582 (2015)
18. Seljak, U., Zaldarriaga, M.: A line of sight approach to cosmic microwave background anisotropies. arXiv:astro-ph/9603033 (1996)
19. Tegmark, M.: How to measure CMB power spectra without losing information. Phys. Rev. D **55**, 5895 (1997)
20. Bond, J., Jaffe, A.H., Knox, L.: Estimating the power spectrum of the cosmic microwave background. Phys. Rev. D **57**, 2117 (1998)
21. Tegmark, M., de Oliveira-Costa, A., Hamilton, A.J.: High resolution foreground cleaned CMB map from WMAP. Phys. Rev. D **68**, 123523 (2003)
22. Ade, P., Aghanim, N., Armitage-Caplan, C., Arnaud, M., Ashdown, M., Atrio-Barandela, F., Aumont, J., Baccigalupi, C., Banday, A.J., Barreiro, R., et al.: Planck 2013 results. XII. Diffuse component separation. Astron. Astrophys. **571**, A12 (2014)
23. Tegmark, M.: How to make maps from cosmic microwave background data without losing information. Astrophys. J. Lett. **480**, L87 (1997)
24. Hinshaw, G., Barnes, C., Bennett, C., Greason, M., Halpern, M., Hill, R., Jarosik, N., Kogut, A., Limon, M., Meyer, S., et al.: First-year Wilkinson microwave anisotropy probe (WMAP) WMAP is the result of a partnership between Princeton University and the NASA Goddard Space Flight Center. Scientific guidance is provided by the WMAP Science Team. Observations: data processing methods and systematic error limits. Astrophys. J. Suppl. Ser. **148**, 63 (2003)
25. Hinton, G.: A practical guide to training restricted Boltzmann machines. Momentum **9**, 926 (2010)
26. Émile Borel, M.: Les probabilités dénombrables et leurs applications arithmétiques. Rendiconti del Circolo Matematico di Palermo (1884–1940) **27**, 247–271 (1909)
27. Fisher, R.A.: On the mathematical foundations of theoretical statistics. Philos. Trans. R. Soc. Lond. Ser. A Contain. Pap. Math. Phys. Charact. **222**, 309–368 (1922)
28. Riesenhuber, M., Poggio, T.: Models of object recognition. Nat. Neurosci. **3**, 1199–1204 (2000)
29. Kullback, S., Leibler, R.A.: On information and sufficiency. Ann. Math. Stat. **22**, 79–86 (1951). doi:10.1214/aoms/1177729694
30. Cover, T.M., Thomas, J.A.: Elements of Information Theory. Wiley, New York (2012)
31. Kardar, M.: Statistical Physics of Fields. Cambridge University Press, Cambridge (2007)

32. Cardy, J.: Scaling and Renormalization in Statistical Physics, vol. 5. Cambridge University Press, Cambridge (1996)
33. Johnson, J.K., Malioutov, D.M., Willsky, A.S.: Lagrangian relaxation for MAP estimation in graphical models. arXiv:0710.0013 (2007)
34. Bény, C.: Deep learning and the renormalization group. arXiv:1301.3124 (2013)
35. Saremi, S., Sejnowski, T.J.: Hierarchical model of natural images and the origin of scale invariance. Proc. Natl. Acad. Sci. **110**, 3071–3076 (2013). http://www.pnas.org/content/110/8/3071.full.pdf, http://www.pnas.org/content/110/8/3071.abstract
36. Miles Stoudenmire, E., Schwab, D.J.: Supervised learning with quantum-inspired tensor networks. arXiv:1605.05775 (2016)
37. Vidal, G.: Class of quantum many-body states that can be efficiently simulated. Phys. Rev. Lett. **101**, 110501 (2008). arXiv:quant-ph/0610099
38. Ba, J., Caruana, R.: Do deep nets really need to be deep? In: Advances in Neural Information Processing Systems, pp. 2654–2662 (2014)
39. Hastad, J.: Almost optimal lower bounds for small depth circuits. In: Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing, pp. 6–20. Organization ACM (1986)
40. Telgarsky, M.: Representation benefits of deep feedforward networks. arXiv:1509.08101 (2015)
41. Montufar, G.F., Pascanu, R., Cho, K., Bengio, Y.: On the number of linear regions of deep neural networks. In: Advances in Neural Information Processing Systems, pp. 2924–2932 (2014)
42. Eldan, R., Shamir, O.: The power of depth for feedforward neural networks. arXiv:1512.03965 (2015)
43. Poole, B., Lahiri, S., Raghu, M., Sohl-Dickstein, J., Ganguli, S.: Exponential expressivity in deep neural networks through transient chaos. arXiv:1606.05340 (2016)
44. Raghu, M., Poole, B., Kleinberg, J., Ganguli, S., Sohl-Dickstein, J.: On the expressive power of deep neural networks. arXiv:1606.05336 (2016)
45. Saxe, A.M., McClelland, J.L., Ganguli, S.: Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. arXiv:1312.6120 (2013)
46. Bengio, Y., LeCun, Y., et al.: Scaling learning algorithms towards AI. Large Scale Kernel Mach. **34**, 1–41 (2007)
47. Strassen, V.: Gaussian elimination is not optimal. Numer. Math. **13**, 354–356 (1969)
48. Le Gall, F.: In: Proceedings of the 39th international symposium on symbolic and algebraic computation. Organization ACM, pp. 296–303 (2014)
49. Carleo, G., Troyer, M.: Solving the quantum many-body problem with artificial neural networks. arXiv:1606.02318 (2016)
50. Vollmer, H.: Introduction to Circuit Complexity: A Uniform Approach. Springer, Berlin (2013)