Cellular automata as convolutional neural networks

William Gilpin*
Quantitative Biology Initiative, Harvard University, Cambridge, Massachusetts 02138, USA

(Received 17 March 2019; revised manuscript received 19 July 2019; published 4 September 2019)

Deep-learning techniques have recently demonstrated broad success in predicting complex dynamical systems ranging from turbulence to human speech, motivating broader questions about how neural networks encode and represent dynamical rules. We explore this problem in the context of cellular automata (CA), simple dynamical systems that are intrinsically discrete and thus difficult to analyze using standard tools from dynamical systems theory. We show that any CA may readily be represented using a convolutional neural network with a network-in-network architecture. This motivates the development of a general convolutional multilayer perceptron architecture, which we find can learn the dynamical rules for arbitrary CA when given videos of the CA as training data. In the limit of large network widths, we find that training dynamics are nearly identical across replicates, and that common patterns emerge in the structure of networks trained on different CA rulesets. We train ensembles of networks on randomly sampled CA, and we probe how the trained networks internally represent the CA rules using an information-theoretic technique based on distributions of layer activation patterns. We find that CA with simpler rule tables produce trained networks with hierarchical structure and layer specialization, while more complex CA produce shallower representations—illustrating how the underlying complexity of the CA's rules influences the specificity of these internal representations. Our results suggest how the entropy of a physical process can affect its representation when learned by neural networks.

DOI: 10.1103/PhysRevE.100.032402

I. INTRODUCTION

Recent studies have demonstrated the surprising ability of deep neural networks to learn predictive representations of dynamical systems [1–5]. For example, certain types of recurrent neural networks, when trained on short-timescale samples of a high-dimensional chaotic process, can learn transition operators for that process that rival traditional simulation techniques [2,6,7]. More broadly, neural networks can learn and predict general features of dynamical systems—ranging from turbulent energy spectra [8], to Hamiltonian ground states [9,10], to topological invariants [11]. Such successes mirror well-known findings in applied domains [12], which have convincingly demonstrated that neural networks may not only represent, but also learn, generators for processes ranging from speech generation [13] to video prediction [14]. However, open questions remain about how the underlying structure of a physical process affects its representation by a neural network trained using standard optimization techniques.

We aim to study such questions in the context of cellular automata (CA), among the simplest dynamical systems due to the underlying discreteness of both their domain and the dynamical variables that they model. The most widely known CA is Conway's Game of Life, which consists of an infinite square grid of sites ("cells") that can only take on a value of zero ("dead") or one ("alive"). Starting from an initial binary pattern, each cell is synchronously updated based on its current state, as well as its current number of living and nonliving neighbors. Despite its simple dynamical rules, the Game of Life has been found to exhibit remarkable properties ranging from self-replication to Turing universality [15]. Such versatility offers a vignette of broader questions in CA research, because many CA offer minimal examples of complexity emerging from apparent simplicity [16–20]. For this reason, CA have previously been natural candidates for evaluating the expressivity and capability of machine learning techniques such as genetic algorithms [21,22].

Here, we show that deep convolutional neural networks are capable of representing arbitrary cellular automata, and we demonstrate an example network architecture that smoothly and repeatably learns an arbitrary CA using standard loss gradient-based training. Our approach takes advantage of the "mean-field limit" for large networks [23–25], for which we find that trained networks express a universal sparse representation of CA based on depthwise consolidation of similar inputs. The effective depth of this representation, however, depends on the entropy of the CA's underlying rules.

II. EQUIVALENCE BETWEEN CELLULAR AUTOMATA AND CONVOLUTIONAL NEURAL NETWORKS

A. Cellular automata

We define a CA as a dynamical system with M possible states, which updates its value based on its current value and D other cells—usually its immediate neighbors in a square lattice. There are M^D possible unique M-ary input strings to a CA function, which we individually refer to as σ. A cellular automaton implements an operator $\mathcal{G}(\sigma)$ that is fully specified by a list of transition rules $\sigma \to m$, $m \in 0, 1, \ldots, M-1$, and

*Also at Department of Applied Physics, Stanford University, Stanford, California 94305, USA; wgilpin@stanford.edu

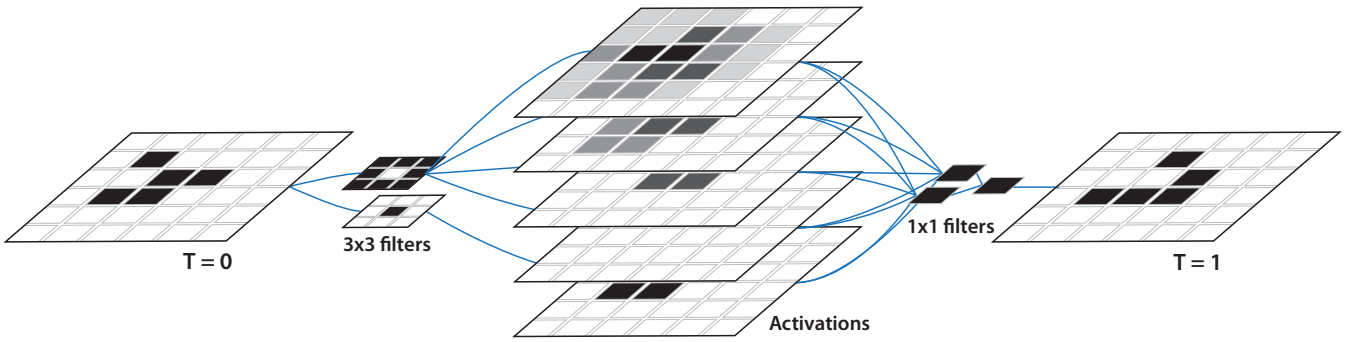

FIG. 1. Conway's Game of Life as a convolutional neural network. Two convolutional filters identify the value of the center pixel and count the number of neighbors. These features are then scored and summed to generate a prediction for the system at the next timepoint.

there are M^{M^D} possible unique $\mathcal{G}(\sigma)$, each implementing a different ruleset. For the Game of Life, $M = 2, D = 9$, and so $\mathcal{G}(\sigma)$ is a Boolean function that maps each of the $2^9 = 512$ possible 9-bit input strings to a single bit. A defining feature of CA is the locality of dynamical update rule, which ensures that the rule domain is small; the size of D thus sets an upper bound on the rate at which information propagates across space.

B. Convolutional neural networks

We define a convolutional neural network as a function that takes as an input a multichannel image, to which it applies a series of local convolutions via a trainable "kernel." The same kernel is applied to all pixels in the image, and each convolutional layer consolidates information within a fixed local radius of each pixel in the input image [12]. Many standard convolutional architectures include "pooling" layers, which downsample the previous layer and thereby consolidate local information across progressively larger spatial scales; however, all CNN discussed in this paper do not include downsampling steps, and thus preserve the full dimensionality of the input image.

C. Cellular automata as recurrent mlpconv networks

The primary analogy between cellular automata and traditional convolutional neural networks arises from (1) the locality of the dynamics and (2) simultaneous temporal updating of all spatial points. Because neural networks can, in principle, act as universal function approximators [26], a sufficiently complex neural network architecture can be used to fully approximate each rule $\sigma \rightarrow m$ that comprises the CA function $\mathcal{G}(\sigma)$. This single-neighborhood operator can then be implemented as a convolutional operator as part of a CNN, allowing it to be applied synchronously to all pixel neighborhoods in an input image.

Representing a CA with a CNN thus requires two steps: feature extraction to identify each of the M^D input cases describing each neighborhood, followed by association of each neighborhood with an appropriate output pixel. In the Appendix, we show explicitly how to represent *any* CA using a single convolutional layer, followed by repeated 1×1 convolutional layers. The appropriate weights can be found analytically using analysis of the CA itself, rather than via algorithmic training on input data. In fact, we find that many representations are possible; we show that one possible approach defines a shallow network that uniquely matches each of the M^D input σ against a template, while another approach treats layers of the network like levels in a tree search that iteratively narrows down each input σ to the desired output m. A key aspect of this approach is the usage of only one nonunity convolutional layer (with size 3×3 for the case of the Game of Life), which serves as the first hidden layer in the network. The receptive field of these convolutional neurons is equivalent to the neighborhood D of the CA. All subsequent layers consist of 1×1 convolutions, which do not consolidate any additional neighbor information.

Our use of 1×1 convolutions to implement the logic of the CA rule table is inspired by recent work showing that such layers can greatly increase network expressivity at low computational cost [27]. Moreover, because CA are explicitly local, the network requires no pooling layers—making the network the equivalent of fitting a small, convolutional multilayer perceptron or "mlpconv" to the CA [27,28]. Our general approach is comparable to previous uses of deep convolutional networks to parallelize simple operations such as binary arithmetic [29], and it differs from efforts using less-common network types with sigma-pi units, in which individual input bits can gate one another [30].

Figure 1 shows an example analytical mlpconv representation of the Game of Life, in which the two salient features for determining the CA evolution (the center pixel value and the number of neighbors) are extracted via an initial 3×3 convolution, the results of which are passed to additional 1×1 convolutional layers to generate a final output prediction (exact weights are given in the Supplemental Material [31]). The number of separate convolutions (four with the neighbor filter with different biases, and one with the identity filter) is affected by the choice of ReLU activations (the current best practice for deep convolutional networks) instead of traditional neurons with saturating nonlinearities [32]. Many alternative and equivalent representations may be defined, underscoring the expressivity of multilayer perceptrons when representing simple functions like CA.

III. A GENERAL NETWORK ARCHITECTURE FOR LEARNING ARBITRARY CELLULAR AUTOMATA

Having proven that arbitrary cellular automata may be analytically represented by convolutional perceptrons with finite layers and units, we next ask whether automated training

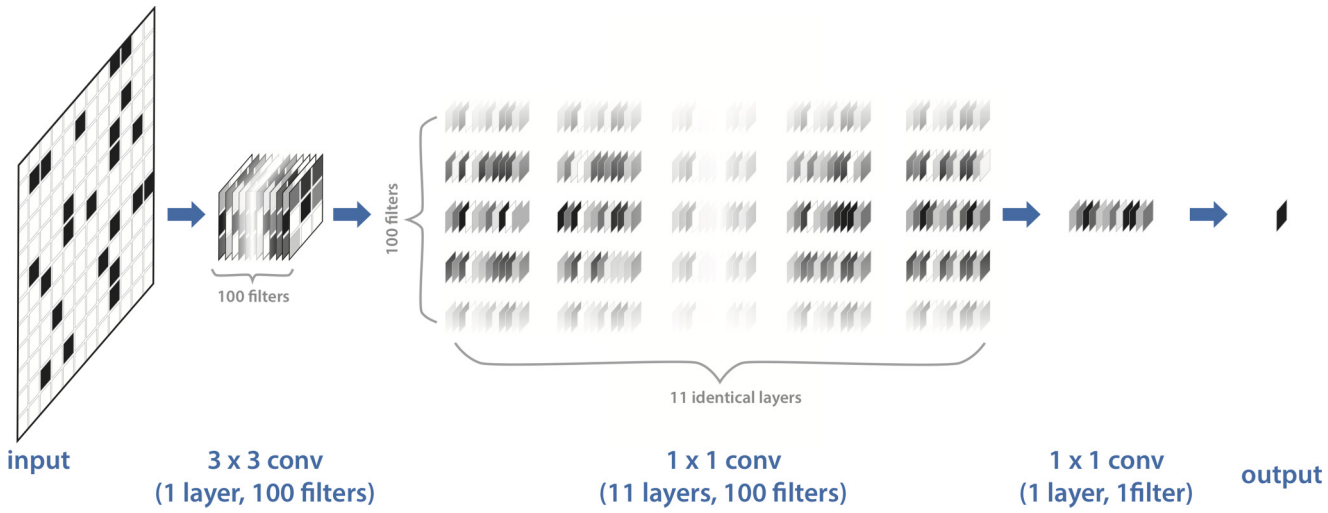

FIG. 2. Architecture of a trainable convolutional neural network for learning cellular automata. Dimensions, where not marked, are determined by the dimensionality of the previous layer.

of neural networks on time series of cellular automata images is sufficient to learn their rules. We investigate this process by training ensembles of convolutional neural networks on random images and random CA rulesets. We start by defining a CA as an explicit mapping between each of $2^9 = 512$ possible 3×3 pixel groups in a binary image, and a single output pixel value. We then apply this map to an ensemble of random binary images (the training data), to produce a new output binary image set (the training labels). Here, we use large enough images (10×10 pixels) and training data batches (500 images) to ensure that the training data contains at least one instance of each rule. On average, each image contains an equal number of black and white pixels; for sufficiently large images this ensures that each of the 512 input states is equally probable. We note that, in principle, training the network will proceed much faster if the network is shown an example of only one rule at a time. However, such a process causes the network structure to depend strongly on the order in which individual rules were shown, whereas presenting all input cases simultaneously forces the network to learn internal rule representations based on their relative importance for maximizing accuracy.

A. Network architecture and training parameters

Figure 2 shows the network used in the training experiments. Our network consists of a basic mlpconv architecture corresponding to a single 3×3 convolutional layer, followed by a variable number of 1×1 convolutional layers [28]. No pooling layers are used, and the parameters in the 3×3 and 1×1 layers are trained together. The final hidden layer consists of a weighted summation, which generates the predicted value for the next state of a lattice site. Empirically, including final "prediction" layer with softmax classifier accelerates training on binary CA by reducing the dependence of convergence on initial neuron weights; however, we omit this step here to allow the same architecture to readily be generalized for CA with $M > 2$. Our network may thus be considered a fully convolutional linear committee machine.

We trained the networks using the Adam optimizer with an L2 norm loss function, with hyperparameters (learning rate, initial weights, etc.) optimized via a grid search (see Appendix for all hyperparameters). Because generating new training data is computationally inexpensive, for each stage of hyper parameter tuning, a new, unseen validation dataset was generated. Additionally, validation was performed using randomly chosen, unseen CA rulesets to ensure that network hyperparameters were not tuned to specific CA rulesets. During training, a second validation dataset 20% of the size of the training data was generated from the same CA ruleset. Training was stopped when the network prediction accuracy reached 100% on this secondary validation dataset, after rounding predictions to the nearest integer. The loss used to compute gradients for the optimizer was not rounded. The final, trained networks were then applied to a new dataset of unseen test data (equal in size to five batches of training data).

We found that training successfully converged for all CA rulesets studied, and we note that the explicit use of a convolutional network architecture simplifies learning of the full rule table. Because we are primarily interested in using CNN as a way to study internal representations of CA rulesets, we emphasize that 100% performance on the second validation dataset a condition of stopping training. As a result, all trained networks had identical performance; however, the duration and dynamics of training varied considerably by CA ruleset (discussed below). Regardless of whether weight-based regularization was used during training, we found that performance on the unseen test data was within $\sim$0.3% of the training data for all networks studied (after outputs are rounded, performance reaches 100%, as expected). We caution, however, that this equal train-test performance should not be interpreted as a measure of generalizability, as would be the case for CNN used to classify images, etc. [33]. Rather, because a CA only has M^D possible input-output pairs (rather than an unlimited space of inputs), this result simply demonstrates that training was stopped at a point where the model had encountered and learned all inputs. In fact, we note that it would be impossible to train a network to represent an

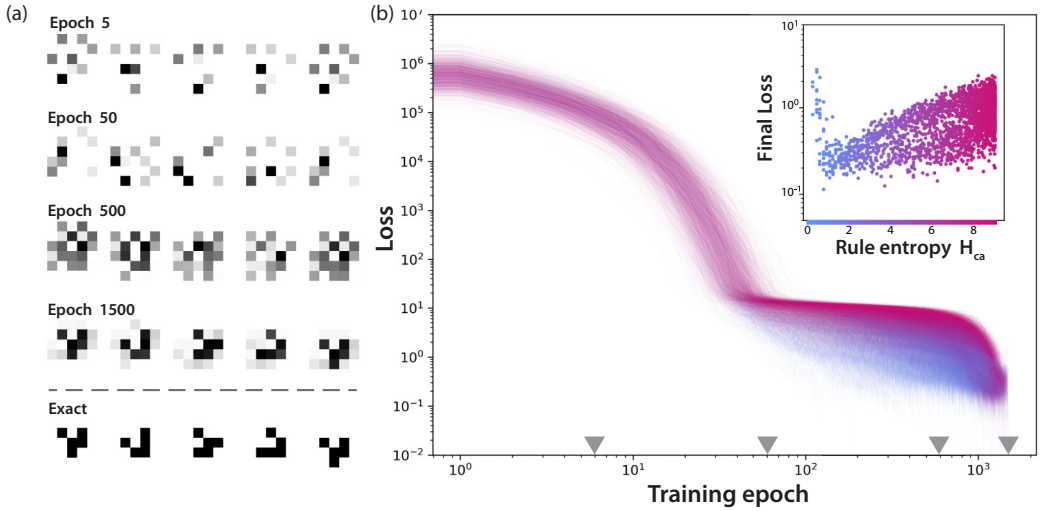

FIG. 3. Training 2560 convolutional neural networks on random cellular automata. (a) A network trained on the Game of Life for different durations, and then applied to images of each stage of the "glider" solution. (b) The loss versus time during training, colored by the rule entropy $\mathcal{H}_{\text{ca}}$. Groups of 512 related cellular automata were generated by iteratively choosing random $\sigma \to 0$ rules from the 512 possible input configurations, and setting those sites to $\sigma \to 1$. Five replicates were performed. Loss values represent the sum over the batch; values of 10 or smaller imply that only small rounding errors were present at the end of training. The entropy of the resulting rule table is characteristic of the CA, and it is indicated by $\mathcal{H}_{\text{ca}} = 0$ (blue, minimum entropy CA) to $\mathcal{H}_{\text{ca}} = 9$ (magenta, maximum entropy CA). (Inset) The final loss for each network at the end of training, shown as a function of $\mathcal{H}_{\text{ca}}$.

arbitrary CA without being exposed to all of its inputs: since an arbitrary CA can send any given input σ to any given output m, there is no way for a network to predict the output for an symbol without having encountered it previously. However, we note that a network could, in principle, encode a prior expectation for an unseen input symbol σ, if it was trained primarily on CA of a certain type.

In a previous work that used a one-layer network to learn the rules of a chaotic CA, it was found that training without weight-sharing prevents full learning, because different spatial regions on the system's attractor have different dynamical complexity [30]. In the results below, we deliberately use very large networks with 12 hidden layers—one 3×3 convolutional layer, followed by eleven 1×1 convolutional layers, all with 100 neurons per layer. These large networks ensure that the network can represent the CA ruleset in as shallow or deep a manner as it finds—and we expect and observe that many fewer neurons per layer are used than are available.

B. Training dynamics of networks

Consistent with prior reports that large networks approach a "mean-field" limit [24,25,34], we find that training is highly repeatable for the large networks that we study, even when different training data is used, different CA rules are learned, or the hyperparameters are altered slightly from their optimal values (although this extends the duration of training). We also find that doubling the depth and width of the networks does not qualitatively affect the results, consistent with a large-network limit. Additionally, we trained alternative networks using a different optimizer (vanilla stochastic gradient descent) and loss function (cross-entropy loss), and found nearly identical internal structure in the trained networks (as discussed below); however, the form of the loss curves during training was more concave for such networks. See the Supplemental Material [31] for further details of networks and training.

Figure 3(a) shows the results of training a single network on the Game of Life, and then applying the trained network to the "glider," a known solitonlike solution to the Game. During the early stages of the training, the activations appear random and intermittent. As training proceeds, the network adjusts to the scale of output values generated by the input data, and then begins to learn clusters of related rules—leading to tightening of the output image and trimming of spurious activation patterns.

IV. ANALYSIS OF TRAINED NETWORKS

We next consider the relevance of the training observations to the general properties of binary cellular automata. Intuition would suggest that certain sets of CA rules are intrinsically easier to learn, regardless of M and D; for example, a null CA that sends every input to zero in a single timestep requires a trivial network structure, while the Game of Life should require a structure like Fig. 1 that can identify each possible neighborhood count. We thus repeat the training data generation and CA network training process described above, except this time we sample CA at random from the $2^{2^9} \approx 10^{154}$ possible rulesets for binary CA. The complexity of the dynamics produced by a given rule are generally difficult to ascertain *a priori*, and typical efforts to systematically investigate the full CA rule space have focused on comparative simulations of different rules [16,17]. For example, the Game of Life is a member of a unique set of "Class IV" CA capable of both chaotic and regular dynamics depending on their initial state; membership in this class has been hypothesized to be a prerequisite to supporting computational universality [15,16].

General prediction of dynamical class is an ongoing question in the CA literature [21]; however, there is a known, approximate relationship between the complexity of simulated dynamics, and the relative fraction λ of transitions to zero and one among the full set of 512 possible input cases: $\lambda = 0$ and $\lambda = 1$ correspond to null CA, whereas $\lambda = 0.5$ corresponds to CA that sends equal numbers of input cases to 0 and 1 [17]. This captures the general intuition that CA typically display richer dynamics when they have a broader range of output symbols [18,20]. Here, instead of using λ directly, we parametrize the space of CA equivalently using the effective "rule entropy," $\mathcal{H}_{\text{ca}}$. We define $\mathcal{H}_{\text{ca}}$ by starting from a maximum-entropy image with a uniform distribution of input symbols ($p_\sigma \approx 1/M^D$ for all σ), to which we then apply the CA rule once and then record the new distribution of input cases, p'_σ. The residual Shannon entropy $\mathcal{H}_{\text{ca}} \equiv -\sum_\sigma p'_\sigma \log_2 p'_\sigma$ provides a measure of the degree to which the CA rules compress the space of available states. $\mathcal{H}_{\text{ca}}(\lambda)$ monotonically increases from $\mathcal{H}_{\text{ca}}(0) = 0$ until it reaches a global maximum at $\mathcal{H}_{\text{ca}}(1/2) = 9$, after which it symmetrically decreases back to $\mathcal{H}_{\text{ca}}(1) = 0$.

Figure 3(b) shows the result of training 2560 randomly sampled CA with different values of $\mathcal{H}_{\text{ca}}$. Ensembles of 512 related cellular automata were generated by randomly selecting single symbols in the input space to transition to 1 (starting with the null case $\sigma \to 0$ for all σ), one at a time, until reaching the case $\sigma \to 1$ for all σ. This "table walk" sampling approach [17] was then replicated 5 times for different starting conditions.

We observe that the initial 10–100 training epochs are universal across $\mathcal{H}_{\text{ca}}$. Detailed analysis of the activation patterns across the network (Supplemental Material [31]) suggests that this transient corresponds to initialization, wherein the network learns the scale and bounds of the input data. Recent studies of networks trained on real-world data suggest that this initialization period consists of the network finding an optimal representation of the input data [35]. During the next stage of training, the network begins to learn specific rules: the number of neurons activated in each layer begins to decrease, as the network becomes more selective regarding which inputs provoke nonzero network outputs (see Supplemental Material [31]). Because $\mathcal{H}_{\text{ca}}$ determines the sparsity of the rule table—and thus the degree to which the rules may be compressed—$\mathcal{H}_{\text{ca}}$ strongly affects the dynamics of this phase of training, with simpler CA learning faster and shallower representations of the rule table, resulting in smaller final loss values [Fig. 3(b), inset]. This behavior confirms general intuition that more complicated CA rules require more precise representations, making them harder to learn.

A key feature of using large networks to fit simple functions like CA is strong repeatability of training across different initializations and CA rulesets. In the Appendix, we reproduce all results shown in the main text using networks with different sizes and depths, and even a different optimizer, loss function, and other hyperparameters, and we report nearly identical results (for both training and test data) as those found using the network architecture described above. On both the training data and test data, we find similar universal training curves that depend on $\mathcal{H}_{\text{ca}}$, as well as distributions of activation patterns. This universality is not observed in "narrow" networks with fewer neurons per layer, for which training proceeds as a series of plateaus in the loss punctuated by large drops when the stochastic optimizer happens upon new rules. In this limit, randomly chosen CA rulesets will not consistently result in training successfully finding all correct rules and terminating. Moreover, small networks that do terminate do not display apparent patterns when their internal structure is analyzed using the approaches described below—consistent with a random search. Similar loss dynamics have previously been observed when CA are learned using genetic algorithms, in which the loss function remains mostly flat, punctuated by occasional leaps when a mutant encounters a new rule [21]. For gradient-based training, similar kinetic trapping occurs in the vicinity of shallow minima or saddle points [36,37], but these effects are reduced in larger networks such as those used here.

V. INFORMATION-THEORETIC QUANTIFICATION OF ACTIVATIONS

That training thousands of arbitrary CA yields extremely similar training dynamics suggests that deep networks trained using gradient optimizers learn a universal approach to approximating simple functions like CA. This motivates us to next investigate how exactly the trained networks represent the underlying CA rule table—do the networks simply match entire input patterns, or do they learn consolidated features such as neighbor counts? Because the intrinsic entropy of the CA rule table affects training, we reason that the entropy of activated representations at each layer is a natural heuristic for analyzing the internal states of the network. We thus define a binary measure of activity for each neuron in a fully trained network: When the network encounters a given input σ, any neurons that produce a nonzero output are marked as 1 (or 0 otherwise), resulting in a new set of binary strings $a(\sigma)$ denoting the rounded activation pattern for each input σ. For example, in an mlpconv network with only 3 layers, and 3 neurons per layer, an example activation pattern for a specific input σ_1 could yield $a(\sigma_1) = \{010, 000, 011\}$, with commas demarcating layers. Our approach constitutes a simplified version of efforts to study deep neural networks by inspecting activation pattern "images" of neurons in downstream layers when specific input images are fed into the network [25,38–40]. However, for this system binary strings (thresholded activation patterns) are sufficient to characterize the trained networks, due to the finite space of input-output pairs for binary CA, and the large size of the networks; in the investigations, no cases were found in which two different inputs (σ, σ') produced different unrounded activation patterns, but identical patterns after binarization $[a(\sigma), a(\sigma')]$.

Given the ensemble of input symbols $\sigma \in \{0, 1\}^D$, and a network consisting of L layers each containing N neurons, we can define separate symbol spaces representing activations of the entire network $a_{\mathsf{T}}(\sigma) \in \{0, 1\}^{LN}$; each individual layer, $a_{\mathsf{L},i}(\sigma) \in \{0, 1\}^N$, $i \in [0, L-1]$; and each individual neuron $a_{\mathsf{N},ij}(\sigma) \in \{0, 1\}$, $i \in [0, L-1]$, $j \in [0, N-1]$. Averaging over test data consisting of an equiprobable ensemble of all M^D unique input cases σ, we can then calculate the probability $p_{\alpha,k}$ for observing a given unique symbol a_k at a level $\alpha \in \{\mathsf{T}, \mathsf{L}, \mathsf{N}\}$ in the network. We quantify the uniformity of each activation symbol distribution p using the entropy

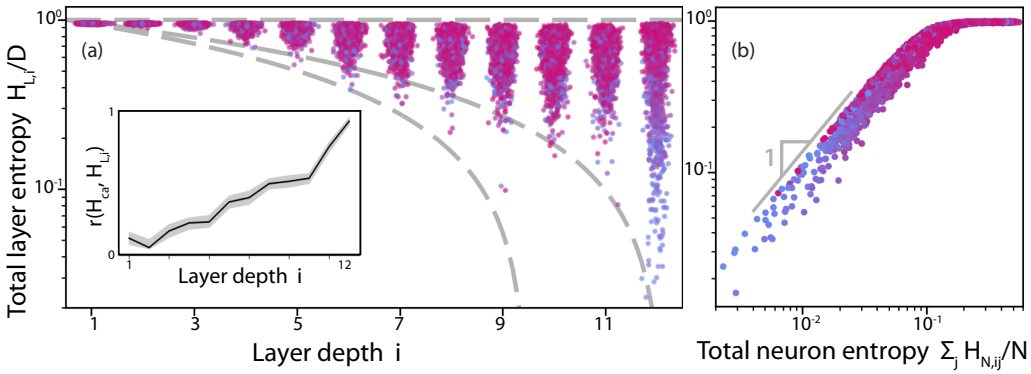

FIG. 4. Internal representations of cellular automata by trained networks. (a) The individual layerwise entropy ($\mathcal{H}_{\mathrm{L},i}/D$) for the 2560 networks shown in the previous figure. Noise has been added to the horizontal coordinates (layer index) to facilitate visualization. As in previous figures, coloration corresponds to the entropy $\mathcal{H}_{\mathrm{ca}}$ of the underlying CA. Dashed lines correspond to expected trends for theoretical networks that eliminates 0% of cases in each layer (i.e., a pattern-matching implementation), 45% of cases, and 50% (top to bottom) (Inset) The Pearson correlation coefficient r between the rule entropy $\mathcal{H}_{\mathrm{ca}}$ and layer entropy $\mathcal{H}_{\mathrm{L},i}$. Error range corresponds to bootstrapped 25%–75% quantiles. (b) The normalized layerwise entropy ($\mathcal{H}_{\mathrm{L},i}/D$) versus the normalized total layerwise neuron entropy ($\mathcal{H}_{\mathrm{N},ij}/N$), with the linear scaling annotated.

$\mathcal{H}_\alpha = -\sum_k p_{\alpha,k} \log_2 p_{\alpha,k}$, which satisfies $\mathcal{H}_\alpha \leqslant \dim(\alpha)$. We condense notation and refer to the activation entropies $\mathcal{H}_\mathsf{T}$, $\mathcal{H}_{\mathrm{L},i}$, $\mathcal{H}_{\mathrm{N},ij}$ as the total entropy, the entropy of ith layer, and the entropy of the jth neuron in the ith layer. We note that, in addition to readily quantifying the number of unique activation patterns and their uniformity across input cases, the Shannon entropy naturally discounts zero-entropy "dead neurons," a common artifact of training high-dimensional ReLU networks [32]. Our general analysis approach is related to a recently developed class of techniques for analyzing trained networks [41], in which an ensemble of training data (here, a uniform distribution of σ) is fed into a trained network to generate a new statistical observable (here, $\mathcal{H}$).

We expect and observe that $\langle \mathcal{H}_{\mathrm{N},ij} \rangle_{ij} < \langle \mathcal{H}_{\mathrm{L},i} \rangle_i \leqslant \mathcal{H}_\mathsf{T}$. Unsurprisingly, the maximum entropy of a single neuron is $\log_2 2 = 1$, and all multineuron layers generate more than two patterns across the test data. We also observe that $\mathcal{H}_\mathsf{T} \approx 9$ for all networks trained, suggesting that the overall firing patterns in the network differed for every unique input case—even for trivial rules like $\lambda = 0$ where a network with all zero weights and biases would both correctly represent the rule table, and have identical firing patterns for all inputs ($\mathcal{H}_\mathsf{T} = 0$). This effect directly arises from training using gradient-based methods, for which at least some early layers in the network produce unique activation patterns for each σ that are never condensed during later training stages. Accordingly, regularization using a total weight cost or dropout both reduce $\mathcal{H}_\mathsf{T}$.

Comparing $\mathcal{H}_{\mathrm{L},i}$ across models and layers demonstrates that early layers in the network tend to generate a broad set of activation patterns that closely follow the uniform input symbol distribution [Fig. 4(a)]. These early layers in the network thus remain saturated at $\mathcal{H}_{\mathrm{L},i} = \mathcal{H}_\mathsf{T} \approx 9$; however, in deeper layers progressively lower entropies are observed, consistent with fewer unique activation patterns (and a less uniform distribution across these strings) appearing in later layers. These trends depend strongly on the CA rules (coloration). In the figure, dashed lines allow comparison of $\mathcal{H}_{\mathrm{L},i}$ to theoretical predictions for the layerwise entropy for the different types of ways that a CNN can represent the CA. The uppermost dashed curve corresponds to a network that generates a maximum entropy set of 512 equiprobable activation patterns in each layer. This case corresponds to a "shallow" network that matches each input case to a unique template at each layer. Lower dashed curves correspond to predictions for networks that implement the CA as layerwise search, in which σ that map to the same output m are mapped to the same activation pattern at some point before the final layer. This corresponds to a progressive decrease in the number of unique activation patterns in each layer. The two dashed curves shown correspond to theoretical networks that eliminate 45% and 50% of unique activation patterns at each layer.

We find that higher entropy rules $\mathcal{H}_{\mathrm{ca}}$ (red points) tend to produce shallower networks due to the rule table being less intrinsically compressible; whereas simpler CA (blue points) produce networks with more binary treelike structure. This relationship has high variance in early layers, making it difficult to visually discern in the panel save for the last layer. However, explicit calculation of the Pearson correlation $r(\mathcal{H}_{\mathrm{ca}}, \mathcal{H}_{\mathrm{L},i})$ confirms its presence across all layers of the network, and that it becomes more prominent in deeper layers [Fig. 4(a), inset]. This trend is a consequence of training the network using backpropagation-based techniques, in which loss gradients computed at the final, Lth hidden layer are used to update the weights in the previous $(L-1)$th layer, which are then used to update the $(L-2)$th layer, and so forth [42]. During training, the entropy of the final layer increases continuously until it reaches a plateau determined by the network size and by $\mathcal{H}_{\mathrm{ca}}$. The penultimate layer then increases in entropy until reaching a plateau, and so forth until $\mathcal{H}_\mathsf{T} = 9$ across all σ—at which point training stops because the test error will reach zero (training dynamics are further analyzed in the Supplemental Material [31]). This general correlation between CA entropy and network structure is consistent with earlier studies in which networks were trained to label CA rulesets by their dynamical complexity class [43].

The role of $\mathcal{H}_{\mathrm{ca}}$ on internal representation distributions p_L can be further analyzed using Zipf plots of activation pattern a_k frequency versus rank (Supplemental Material [31]): the

resulting plots show that the distribution of activation symbols is initially uniform (because the training data has a uniform distribution of σ), but the distribution becomes progressively narrower and more peaked in later layers. This process occurs more sharply for networks trained on CA with smaller $\mathcal{H}_{\text{ca}}$.

We next consider how the entropy of the observed layer activation patterns relates to the entropy of the individual neurons $\mathcal{H}_{\mathsf{N},ij}$ that comprise them; we suspect there is a relation because the individual firing entropies determine the "effective" number of neurons in a layer, $N_{\text{eff}} = 2^{\sum_j \mathcal{H}_{\mathsf{N},ij}}$. Across all layers, we observe a linear relationship between $\mathcal{H}_{\mathsf{N},ij}$ and $\mathcal{H}_{\mathsf{L},i}$, which saturates when $\mathcal{H}_{\mathsf{L},i} \approx \mathcal{H}_{\mathsf{T}}$ [Fig. 4(b)]. The lower-$\mathcal{H}_{\text{ca}}$ CA lie within the linear portion of this plot, suggesting that variation in activation patterns in this regime results from layers recruiting varying numbers of neurons. Conversely, higher-entropy CA localize in a saturated region where each layer encodes a unique activation pattern for each unique input state, leading to no dependence on the total effective number of neurons. This plot explains the earlier observation that the dynamics of training do not depend on the exact network shape as long as the network has sufficiently many neurons: for low $\mathcal{H}_{\text{ca}}$, layers never saturate, and are free to recruit more neurons until they are able to pattern-match every unique input (at intermediate and large $\mathcal{H}_{\text{ca}}$). A CA with more possible input states (larger M or D) would thus require more neurons per layer to enter this large-network limit.

We also consider the degree to which the decrease $\mathcal{H}_{\mathsf{L},i}$ versus i arises from deeper layers becoming "specialized" to specific input features, a common observation for deep neural networks [12,39,42]. We quantify the layer specialization using the total correlation, a measure of the mutual information between the activation patterns of a layer, and the neurons within that layer: $\mathcal{I}_i = \sum_j \mathcal{H}_{\mathsf{N},ij} - \mathcal{H}_{\mathsf{L},i}$. This quantity is minimized ($\mathcal{I}_i = 0$) when the single neuron activations within a layer are independent of one another; conversely, at the maximum value individual neurons only activate jointly in the context of forming a specific layer activation pattern. Plots of $\mathcal{I}_i$ versus i (Supplemental Material [31]) reveal that during early layers, individual neurons tend to fire independently, consistent with multineuron features being unique to each input case. In these early layers, $\mathcal{I}_i$ is large because the number of possible activation patterns in a single layer of the large network (2^{100}) is much larger than the number of input cases (2^9). In later layers, however, the correlation begins to decrease, consistent with individual neurons being activated in the context of multiple input cases—indicating that these neurons are associated with features found in multiple input cases, like the states of specific neighbors. Calculation of $r(\mathcal{I}_i, \mathcal{H}_{\text{ca}})$ confirms that this effect varies with $\mathcal{H}_{\text{ca}}$.

VI. DISCUSSION

We have shown an analogy between convolutional neural networks and cellular automata, and demonstrated a type of network capable of learning arbitrary binary CA using standard techniques. Our approach uses a simple architecture that applies a single 3×3 convolutional layer to consolidate the neighborhood structure, followed by repeated 1×1 convolutions that perform local operations. This architecture is capable of predicting output states using a mixture of shallow pattern-matching and deep layerwise tree searching. After training an ensemble of networks on a variety of CA, we find that the networks structurally encode generic dynamical features of CA, such as the relative entropy of the rule table. Further work is necessary to determine whether neural networks can more broadly inform efforts to understand the dynamical space of CA, including fundamental efforts to relate a CA's *a priori* rules to its apparent dynamical complexity during simulation [16,18,22]—for example, do Class IV and other complex CA impose unique structures upon fitted neural networks, or can neural networks predict their computational complexity given a rule table? These problems and more general studies of dynamical systems will require more sophisticated approaches, such as unsupervised training and generative architectures (such as restricted Boltzmann machines). More broadly, we note that studying the bounded space of CA has motivated the development of general entropy-based approaches to probing trained neural networks. In future work we hope to relate our observations to more general patterns observed in studies of deep networks, such as the information bottleneck [35]. Such results may inform analysis of open-ended dynamical prediction tasks, such as video prediction, by showing a simple manner in which process complexity manifests as structural motifs.

ACKNOWLEDGMENTS

W.G. was supported by the NSF-Simons Center for Mathematical and Statistical Analysis of Biology at Harvard University, from NSF Grant No. DMS-1764269, and from the Harvard FAS Quantitative Biology Initiative. He was also supported by the U.S. Department of Defense through the NDSEG fellowship program, as well as by the Stanford EDGE-STEM fellowship program.

APPENDIX

1. Representing arbitrary CA with convolutional neural networks

Here we show explicitly how a standard *mlpconv* multilayer perceptron architecture with ReLU activation is capable of representing an arbitrary M state cellular automaton with a finite depth and neuron count [28]. We provide the following explicit examples primarily as an illustration of the ways in which 1×1 convolutions may be used to implement arbitrary CA using a perceptron; we note that real-world networks trained using optimizers will find many other heuristics and representations. We provide the two analytic cases below for concreteness, and to illustrate two important limits: pattern-matching templates for each unique input across the entire network, or using individual layers to eliminate cases until the appropriate output symbol has been identified.

a. Pattern-matching the rule table with a shallow network

An arbitrary M-state cellular automaton can first be converted into a one-hot binary representation. Given an $L \times L$ image, we seek to generate an $L \times L \times M$ stack of binary activation images:

(1) Convolve the input layer with M distinct 1×1 convolutional filters with unit weights, and with biases given by $1, 0, -1, \ldots -(M-1)$. Now apply ReLU activation

(2) Convolve the resulting image with M 1×1 convolutional filters with zero biases. Each of the first $(M-1)$ convolutional filters tests a different consecutive pair $[1, -b, 0, \ldots, 0]$, $[0, 1, -b, 0, \ldots, 0]$, $[0, 0, 1, -b, 0, \ldots, 0]$, $\ldots, [0, \ldots, 0, 1, -b]$, where b is any positive constant $b \geqslant M/(M-1)$. The last convolutional filter is the identity $[0, \ldots, 0, 1]$. Now apply ReLU activation again.

This conversion step is not necessary when working with a binary CA. It requires at total of $(1+M)+M^2$ parameters and two layers to produce an activation volume of dimensions $L\times L\times M$.

We now have an $L\times L\times(M-1)$ array corresponding the one-hot encoding of each pixel's value in an $L\times L$ lattice. We now pattern match each of the M^D possible inputs with its corresponding correct output value. We note that the steps we take below represent an upper bound; if the number of quiescent versus active states in the cellular automaton is known in advance ($=\lambda M^D$, where λ is Langton's parameter) [17], then the number of patterns to match (and thus total parameters) may be reduced by a factor of λ, because only the nonquiescent "active" rules that produce nonzero output values need to be matched.

(1) Construct a block of M^D $S\times S\times(M-1)$ convolutional filters, where S corresponds to the neighborhood size of the CA ($S=3$ for a standard CA with a Moore neighborhood). Each of the M^D filters simply corresponds to an image of each possible input state, with entries equaling one for each nonzero site, and large negative values [greater than $D(M-1)$] at each zero site. For cases when $M>2$, the depth of each convolutional kernel allows exact matching of different nonzero values.

(2) Assign a bias to each of the M^D filters based on the cellular automaton's rule table. For $S\times S\times(M-1)$ inputs that should map to a nonzero value q, assign a bias of $(q-1)-(L-1)$, where L is the number of nonzero sites in the neighborhood $L \leqslant D(M-1)$. This ensures that only exact matches to the rule will produce positive values under convolution. For inputs that should map to zero, assign any bias $\geqslant L$, such as $D(M-1)$.

(3) Apply the ReLU function.

b. Searching the rule table with a deep network

Another way to represent a cellular automaton with a multilayer perceptron constitutes searching a subset of all possible inputs in each layer. This approach requires all input cases σ that map to the same output symbol m, to also map to the same activation pattern at some layer of the network. This coalescence of different input states can occur at any point in the network before the final layer; here we outline a general approach for constructing maps to the same output symbol using large networks.

Assigning input cases to a unique binary strings. Assume there are N convolutional filters. If there are M^D unique input cases, then these filters can be used to generate an n-hot encoding of the input states. n should be chosen such that $\binom{N}{n} \geqslant M^D$. Here, we assume a binary CA with a Moore neighborhood ($M=2$, $D=9$). If $N=100$ neurons are present in each layer, then a two-hot binary string ($n=2$) is sufficient to uniquely represent every possible input state of a binary Moore CA, using the following steps

(1) The D pixel neighborhood is split into n subneighborhoods, with sizes we refer to as $D_1, D_2, .., D_n$. For example, for a the binary Moore CA, we can split the neighborhood into the first five pixels (counted from top-left to the center) and the remaining four pixels (the center pixel to the bottom right corner. Note that the number and dimensionality of these sub-neighborhoods must satisfy the condition: if $Q \equiv M^{D_1} + M^{D_2} + \cdots + M^{D_n}$, then $\binom{N}{Q} \geqslant M^D$.

(2) Define $M^{D_1} + M^{D_2} + \cdots + M^{D_n}$ filters, which match each possible sub-neighborhood. For example, for the neighborhood reading 101000111 from upper-left to bottom-right, two filters can be defined that will match subneighborhoods consisting of the first 5 bits and the last 4 bits, using the approach described above for pattern-matching. In this case, these filters would be $1, -100, 1, -100, -100, -100, 0, 0, 0$ with a bias of -1, and $0, 0, 0, -100, -100, -100, 1, 1, 1$ with a bias of -2.

(3) Apply ReLU activation.

(4) The resulting activation map will be an n-hot binary encoding of the input state, because each unique input case will match the same n filters from the set of N, thus creating a unique representation.

Assigning input case binary strings to matching output symbols. At this stage in the network, each input case has been mapped to a unique N digit binary string with exactly n ones within it. Successive 1×1 convolutional filters may now be used to combine different inputs into the same activation pattern. As a simple example, if $N=5$, then the possible input cases are $\sigma \in \{10001,\ 10010,10100,\ 11000,\ 01001,\ 01010,\ 01100,\ 00101,\ 00110,\ 00011\}$. Many of these cases can be uniquely matched by applying a filter consisting of three ones, followed by a bias of $b=2$. For example, using the filter $W=(-1,-2,-1,0,-2)$ to perform the operation $h = RELU(W\sigma + b)$ will result in an output of 1 for the cases $\{10010, 00110\}$ only. To match strings with no overlapping bits, more than two cases must be merged simultaneously. In general, to merge H cases using this approach, two strings must have $H-1$ overlapping bits.

For the case of binary CA with a Moore radius, an example of a network analogous to a simple binary search would consist of filters that reduce the 512 input cases to 512 2-hot strings (in the first 3×3 convolutional layer). Subsequent 1×1 convolutions could then map these states to 256 unique cases, then 128, and so forth until there are only two unique activation patterns left—the first for input states that map to one, and the second for input states that map to zero. Depending on the λ parameter of the CA rule table, the depth (and thus minimum number of layers) to perform this search would be a maximum of $\log_2 512 - 1 = 8$ layers when $\lambda = 0.5$ (i.e., when there are equal numbers of ones and zero outputs in the rule table). This case comprises just one example of performing a search using the depth of a network. However, many variations are possible, because coalescence of two input states may occur in any layer. Moreover, while the above examples describe two input states being combined together for each filter in a given layer, it is not difficult to

construct alternative filters that can combine more than two states together. We thus expect that there is considerably flexibility in the different ways that a network trained algorithmically can internally represent input states with similar features and similar outputs, but that these different approaches manifest as an overall decrease in the number of unique activation patterns observed across the depth of the network.

c. Network representation of the Game of Life

We note that there are many other ways to implement a CA that are not exactly layerwise depth search, nor a shallow pattern match, depending on the number and type of features being checked at each layer of the network. For example, each of the D pixels in the neighborhood of the CA can be checked with separate convolutional kernels all in the first layer, and then different combinations of these values could be checked in subsequent steps. The shallow network described above represents an extreme case, in which every value of the full input space is explicitly checked in the first layer. This implementation is efficient for many CA, because of the low cost of performing multiple numerical convolutions. However, for CA with large M or D, the layer-wise search method may be preferable.

For the Game of Life, we can use knowledge of the structure of a CA to design a better implementation. The Game of Life is an outer totalistic CA, meaning that the next state of the system is fully determined by the current value of the center pixel, and the total number of ones and zeros among its immediate neighbors. For this reason, only two unique convolutional filters are needed.

The first filter is the identity, which is applied with bias 0.

0	0	0
0	1	0
0	0	0

The second filter is the neighbor counting filter

1	1	1
1	0	1
1	1	1

Due specifically to the use of ReLU activation functions throughout the networks (rather than sigmoids), several copies of this filter must be applied to detect different specific neighbor counts. In particular, because the Game of Life rules require specific information about whether the total number of "alive" neighbors is <2, 2, 3, or $\geqslant 4$, we need four duplicates of the neighbor counting filter, with biases $(-1, -2, -3, -4)$, to produce unique activation patterns for each neighbor total after the ReLU activation is applied.

We thus perform a single convolution of an $L\times L$ binary input image with 5 total $3\times 3\times 1$ convolutional filters, producing an $L\times L\times 5$ activation volume. Hereafter, we assume that the identity filter is the lowest-indexed filter in the stack, followed by the filters that count the successively increasing numbers of neighbors <2, $=2$, $=3$, and $\geqslant 4$.

Each 5×1 pixel across the $L\times L$ face of the activation volume now contains a unique activation pattern that can be matched against the appropriate output case. In the next layer of the network, two 1×1 convolutional filters with depth 5 are applied,

$$(0, 0, 4/3, -8/3, -1/3),$$
$$(3/2, 5/4, -5, -1/4, -1/4),$$

which are combined with biases $-1/3, -7/4$ and then activated with ReLU activation, resulting in an $L\times L\times 2$ activation volume. To generate a final $L\times L$ output corresponding to the next state of the automaton, this volume is summed along its depth—which can be performed efficiently as a final convolution with a 1×1 filter with value (1,1) along its depth, and no bias. This will produce an $L\times L$ output image corresponding to the next state of the Game.

For an example implementation of this algorithm in TensorFlow, see the function

`ca_funcs.make_game_of_life()`

in https://github.com/williamgilpin/convoca/blob/master/ca_funcs.py.

In principle, this architecture can work for any outer totalistic cellular automaton, such as Life without Death, High Life, etc.—although depending on the number of unique neighbor count and center pixel pairings that determine the ruleset, the number of neighbor filters may need to be adjusted. For example, in the Game of Life the cases of 0 living and 1 living neighbors do not need to be distinguished by the network, because both cases result in the center pixel having a value of zero in the next timestep.

Likewise, for a purely totalistic cellular automaton (such as a majority vote rule), only a single convolutional filter (consisting of 9 identical values) is necessary, because the value of the center pixel does not need to be resolved by the network.

2. Neural network training details

Convolutional neural networks were implemented in Python 3.4 using TensorFlow 1.8 [44]. Source code is available at https://github.com/williamgilpin/convoca.

For all convolutions, periodic boundary conditions were implemented by manually copying pixel values from each

TABLE I. Hyperparameters for networks used in the main text.

Parameter	Value
Input dimensions	10×10 px
Number of layers	12
Neurons per layer	100
Input samples	500 images
Batch size	10 images
Weight initialization	He Normal [45]
Weight scale	1
Learning rate	10^{-4}
Max train epochs	1500
Optimizer	Adam
Loss	L2

TABLE II. Hyperparameters for the large network.

Parameter	Value
Input dimensions	10×10 px
Number of layers	24
Neurons per layer	200
Input samples	500 images
Batch size	10 images
Weight initialization	He Normal [45]
Weight scale	1
Learning rate	10^{-4}
Max train epochs	1500
Optimizer	Adam
Loss	L2

TABLE III. Hyperparameters for the alternative network.

Parameter	Value
Input dimensions	10×10 px
Number of layers	12
Neurons per layer	100
Input samples	500 images
Batch size	20 images
Weight initialization	He Normal [45]
Weight scale	5×10^{-1}
Learning rate	5×10^{-4}
Max train epochs	3000
Optimizer	S. G. D.
Loss	cross-entropy

edge of the input image, and then appending them onto the opposite edges. The padding option "VALID" was then used for the first convolutional filter layer in the TensorFlow graph.

Hyperparameters for the large networks described in the main text were optimized using a grid search. For each training run performed while optimizing hyperparameters, a new validation set of unseen binary images associated with an unseen cellular automaton ruleset was created, to prevent the cellular automaton ruleset from biasing the choice of hyperparameters. Once hyperparameters were chosen, and training on arbitrary cellular automata started, an additional validation set of binary images was generated for each ruleset. These images were used to determine when to stop training. Finally, an unseen set of binary images was used as a test partition, to compute the final accuracy of the trained networks. The training and test accuracies (before rounding the CNN output to the nearest integer) were within 0.3% for all networks studied, which is a direct consequence of the network's ability to represent all input cases exactly. After rounding the CNN output to the nearest integer, both the train and test datasets had 100% accuracy. The unrounded train and test performance during the training of one network are shown as a function of training epoch in Fig. S1 of the Supplemental Material [31].

The default networks contained one 3×3 convolutional layer followed by 11 layers of 1×1 convolutions. The convolutional layer, as well as the 1×1 layers, each had 100 filters. A depth of 12 layers was chosen for the network ensembles analyzed in the main text, to facilitate analysis of hidden layers across a variety of depths. Network and training parameters are given in Table I.

We also considered the degree to which the exact dimensions of the "large network" affect the results. We trained another ensemble of networks with loss function, hyperparameters, and optimizer identical to the main text, but with the number of layers and the number of neurons per layer doubled (Table II). As we observe in the main text, our results remain almost identical (Fig. S2, left panel, of the Supplemental Material [31]). We attribute this to the relatively small number of unique input cases that the networks need to learn (512) as compared to the potential expressivity of large networks.

As a control against the choice of optimizer and loss affecting training, we also trained a replicate ensemble of networks that had the same network shapes (12 layers with 100 neurons each) but a different loss function and optimizer, for which different optimal hyperparameters were found using a new grid search (Table III). We compare results using this alternative network to the default network described in the main text and find the results are nearly identical.

Figures S2 (right panel) and S3 of the Supplemental Material [31] show the results of training a network using these parameters. The shape of the training curve is slightly different, with the universal transient (during which the network learns general features of the input data such as the range and number of unique cases) being much longer for this network. However, the later phases of training continue similarly to the standard network, with $\mathcal{H}_{\text{ca}}$ strongly affecting the later stages of training and the final loss. Moreover, after training has concluded, the dependence of the internal representations of the network on $\mathcal{H}_{\text{ca}}$ (Fig. S2 of the Supplemental Material [31]) matches the patterns seen in the default network above.

[1] L. Zdeborová, Nat. Phys. **13**, 420 (2017).
[2] J. Pathak, B. Hunt, M. Girvan, Z. Lu, and E. Ott, Phys. Rev. Lett. **120**, 024102 (2018).
[3] J. Carrasquilla and R. G. Melko, Nat. Phys. **13**, 431 (2017).
[4] E. P. Van Nieuwenburg, Y.-H. Liu, and S. D. Huber, Nat. Phys. **13**, 435 (2017).
[5] G. Torlai, G. Mazzola, J. Carrasquilla, M. Troyer, R. Melko, and G. Carleo, Nat. Phys. **14**, 447 (2018).
[6] H. Jaeger and H. Haas, Science **304**, 78 (2004).
[7] Y. Bar-Sinai, S. Hoyer, J. Hickey, and M. P. Brenner, Proc. Natl. Acad. Sci. USA **116**, 15344 (2019).
[8] J. N. Kutz, J. Fluid Mech. **814**, 1 (2017).
[9] G. Carleo and M. Troyer, Science **355**, 602 (2017).
[10] G. Torlai and R. G. Melko, Phys. Rev. B **94**, 165134 (2016).
[11] P. Zhang, H. Shen, and H. Zhai, Phys. Rev. Lett. **120**, 066401 (2018).

[12] Y. LeCun, Y. Bengio, and G. Hinton, Nature **521**, 436 (2015).
[13] A. Van Den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. W. Senior, and K. Kavukcuoglu, arXiv:1609.03499v2.
[14] M. Mathieu, C. Couprie, and Y. LeCun, arXiv:1511.05440.
[15] A. Adamatzky, *Game of Life Cellular Automata* (Springer, Berlin, 2010), Vol. 1.
[16] S. Wolfram, Rev. Mod. Phys. **55**, 601 (1983).
[17] C. G. Langton, Phys. D **42**, 12 (1990).
[18] D. P. Feldman, C. S. McTague, and J. P. Crutchfield, Chaos: Interdisc. J. Nonlin. Sci. **18**, 043106 (2008).
[19] E. Fredkin, Phys. D **45**, 254 (1990).
[20] A. Adamatzky and J. Durand-Lose, in *Handbook of Natural Computing* (Springer, Berlin, 2012), pp. 1949–1978.
[21] M. Mitchell, J. P. Crutchfield, R. Das *et al.*, in *Proceedings of the 1st International Conference on Evolutionary Computation and Its Applications (EvCA'96)* (Russian Academy of Sciences, Moscow, 1996), Vol. 8.
[22] M. Mitchell, J. P. Crutchfield and P. T. Hraber, arXiv:adap-org/9306003 (1993).
[23] P.-M. Nguyen, arXiv:1902.02880.
[24] R. M. Neal, *Bayesian Learning for Neural Networks* (Springer Science & Business Media, Berlin, 2012), Vol. 118.
[25] M. Chen, J. Pennington, and S. S. Schoenholz, arXiv:1806.05394.
[26] G. Cybenko, Math. Control Signals Syst. **2**, 303 (1989).
[27] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (IEEE, Piscataway, NJ, 2015), pp. 1–9.
[28] M. Lin, Q. Chen, and S. Yan, arXiv:1312.4400.
[29] I. Sutskever, J. Martens, and G. E. Hinton, in *Proceedings of the 28th International Conference on Machine Learning (ICML)* (ACM, New York, 2011).
[30] N. H. Wulff and J. A. Hertz, in *Advances in Neural Information Processing Systems* (MIT Press, Cambridge, MA, 1993), pp. 631–638.
[31] See Supplemental Material at http://link.aps.org/supplemental/10.1103/PhysRevE.100.032402 for supplemental data.
[32] V. Nair and G. E. Hinton, in *Proceedings of the 27th International Conference on Machine Learning (ICML)* (ACM, New York, 2010), pp. 807–814.
[33] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning* (MIT Press, Cambridge, MA, 2016).
[34] H. Sompolinsky, A. Crisanti, and H.-J. Sommers, Phys. Rev. Lett. **61**, 259 (1988).
[35] R. Shwartz-Ziv and N. Tishby, arXiv:1703.00810.
[36] D. Saad and S. A. Solla, Phys. Rev. E **52**, 4225 (1995).
[37] Y. N. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio, in *Advances in Neural Information Processing Systems* (MIT Press, Cambridge, MA, 2014), pp. 2933–2941.
[38] S. S. Schoenholz, J. Gilmer, S. Ganguli, and J. Sohl-Dickstein, in *Proceedings of the 34th International Conference on Machine Learning* (PMLR, 2017).
[39] D. Erhan, Y. Bengio, A. Courville, and P. Vincent, Visualizing higher-layer features of a deep network, Technical Report 1341, University of Montreal, June 2009, also presented at the ICML 2009 Workshop on Learning Feature Hierarchies, Montreal, Canada.
[40] B. Poole, S. Lahiri, M. Raghu, J. Sohl-Dickstein, and S. Ganguli, in *Advances in Neural Information Processing Systems* (MIT Press, Cambridge, MA, 2016), pp. 3360–3368.
[41] M. Raghu, J. Gilmer, J. Yosinski, and J. Sohl-Dickstein, in *Advances in Neural Information Processing Systems* (MIT Press, Cambridge, MA, 2017), pp. 6076–6085.
[42] S. Arora, A. Bhaskara, R. Ge, and T. Ma, in *Proceedings of the 31st International Conference on Machine Learning (ICML)* (ACM, New York, 2014), pp. 584–592.
[43] J. Gorodkin, A. Sørensen, and O. Winther, Complex Syst. **7**, 1 (1993).
[44] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. A. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zhang, arXiv:1605.08695.
[45] K. He, X. Zhang, S. Ren, and J. Sun, in *Proceedings of the IEEE International Conference on Computer Vision* (IEEE, Piscataway, NJ, 2015), pp. 1026–1034.