Overparameterized neural networks implement associative memory

Adityanarayanan Radhakrishnan[a,b], Mikhail Belkin[c,1], and Caroline Uhler[a,b,2]

[a]Laboratory for Information & Decision Systems, Massachusetts Institute of Technology, Cambridge, MA 02139; [b]Institute for Data, Systems, and Society, Massachusetts Institute of Technology, Cambridge, MA 02139; and [c]Department of Computer Science and Engineering, The Ohio State University, Columbus, OH 43210

Edited by Nathan Kallus, Cornell University, New York, NY, and accepted by Editorial Board Member Donald B. Rubin September 10, 2020 (received for review March 16, 2020)

Identifying computational mechanisms for memorization and retrieval of data is a long-standing problem at the intersection of machine learning and neuroscience. Our main finding is that standard overparameterized deep neural networks trained using standard optimization methods implement such a mechanism for real-valued data. We provide empirical evidence that 1) overparameterized autoencoders store training samples as attractors and thus iterating the learned map leads to sample recovery, and that 2) the same mechanism allows for encoding sequences of examples and serves as an even more efficient mechanism for memory than autoencoding. Theoretically, we prove that when trained on a single example, autoencoders store the example as an attractor. Lastly, by treating a sequence encoder as a composition of maps, we prove that sequence encoding provides a more efficient mechanism for memory than autoencoding.

associative memory | neural networks | autoencoders | sequence encoders | overparameterization

Developing computational models of associative memory, a system that can recover stored patterns from corrupted inputs, is a long-standing problem at the intersection of machine learning and neuroscience.

An early example of a computational model for memory dates back to the introduction of Hopfield networks (1, 2). Hopfield networks are an example of an attractor network, a system that allows for the recovery of patterns by storing them as attractors of a dynamical system. In order to write patterns into memory, Hopfield networks construct an energy function with local minima corresponding to the desired patterns. To retrieve these stored patterns, the constructed energy function is iteratively minimized starting from a new input pattern until a local minimum is discovered and returned.

While Hopfield networks can only store binary patterns, the simplicity of the model allowed for a theoretical analysis of capacity (3). In order to implement a form of associative memory for more complex data modalities, such as images, the idea of storing training examples as the local minima of an energy function was extended by several recent works (4–8). Unlike Hopfield networks, these modern methods do not guarantee that a given pattern can be stored and typically lack the capacity to store patterns exactly (e.g., ref. 4).

Our main finding is that standard overparameterized neural networks trained using standard optimization methods can implement associative memory. In contrast to energy-based methods, the storage and retrieval mechanisms are automatic consequences of training and do not require constructing and minimizing an energy function.

Interpolation Alone Is Not Sufficient for Implementing Associative Memory

While in recent machine learning literature (e.g., refs. 9 and 10), the term memorization is often used interchangeably with interpolation, the ability of a model to perfectly fit training data; note that memorization is stronger and also requires a model to be able to recover training data. In general, interpolation does not guarantee the ability to recover training data nor does it guarantee the ability to associate new inputs with training examples. Fig. 1*A* shows an example of a function that interpolates training data but does not implement associative memory: there is no apparent method to recover the training examples from the function alone. On the other hand, Fig. 1*B* gives an example of a function that implements memory: the training examples are retrieved as the range of the function.

While it has been observed (e.g., ref. 10) that overparameterized networks can interpolate the training data, there is no a priori reason why it should be possible to recover the training data from the network. In this work, we show that, remarkably, the retrieval mechanism also follows naturally from training: the examples can be recovered simply by iterating the learned map. A depiction of the memorization and retrieval mechanisms is presented in Fig. 1 *C* and *D*. More precisely, given a set of training examples $\{x^{(i)}\}_{i=1}^{n} \subset \mathbb{R}^d$ and an overparameterized neural network implementing a family of continuous functions $\mathcal{F} = \{f : \mathbb{R}^d \to \mathbb{R}^d\}$, we show that minimizing the following autoencoding objective with gradient descent (GD) methods leads to training examples being stored as attractors:

Significance

Development of computational models of memory is a subject of long-standing interest at the intersection of machine learning and neuroscience. Our main finding is that overparameterized neural networks trained using standard optimization methods provide a simple mechanism for implementing associative memory. Remarkably, this mechanism allows for the storage and retrieval of sequences of examples. This finding also sheds light on inductive biases in overparameterized networks: while there are many functions that can achieve zero training loss in the overparameterized regime, our result shows that increasing depth and width in neural networks leads to maps that are more contractive around training examples, thereby allowing for storage and retrieval of more training examples.

Author contributions: A.R., M.B., and C.U. designed research, performed research, analyzed data, and wrote the paper.

The authors declare no competing interest.

This article is a PNAS Direct Submission. N.K. is a guest editor invited by the Editorial Board.

[1]Present address: Halcoğlu Data Science Institute, University of California San Diego, La Jolla, CA 92093.

[2]To whom correspondence should be addressed. Email: cuhler@mit.edu.

This article contains supporting information online at https://www.pnas.org/lookup/suppl/doi:10.1073/pnas.2005013117/-/DCSupplemental.

First published October 16, 2020.

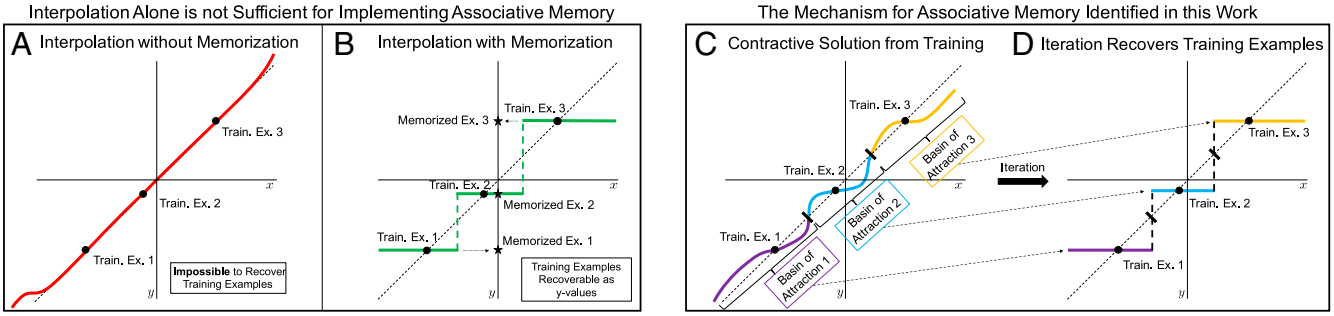


Fig. 1. The difference between associative memory and interpolation is described in *A* and *B*; the mechanism identified in this work by which overparameterized autoencoders implement associative memory is described in *C* and *D*. Training examples are shown as black points, the identity function is shown as dotted lines, and functions are represented using solid colored lines. (*A*) An example of a function that interpolates the training data but does not memorize training data: training data are not recoverable from just the function alone. (*B*) An example of a function that interpolates and memorizes training data: training data are recoverable as the range of the function. (*C*) An example of an interpolating function for which the training examples are attractors; the basis of attraction is shown. (*D*) Iteration of the function from *C* leads to a function that is piecewise constant almost everywhere, with the training examples corresponding to the nontrivial constant regions. The fact that the training examples are attractors implies that iteration provides a retrieval mechanism.

$$\arg\min_{f\in\mathcal{F}} \sum_{i=1}^{n} \|f(x^{(i)}) - x^{(i)}\|^2. \qquad [1]$$

Interestingly, attractors arise without any specific regularization to the above loss function. We demonstrate this phenomenon by presenting a wealth of empirical evidence, including a network that stores 500 images from ImageNet-64 (11) as attractors. In addition, we present a proof of this phenomenon for overparameterized networks trained on single examples.

Furthermore, we show that a slight modification of the objective Eq. **1** leads to an implementation of associative memory for sequences. More precisely, given a sequence of training examples $\{x^{(i)}\}_{i=1}^{n} \subset \mathbb{R}^d$, minimizing the following sequence encoding objective with GD methods leads to the training sequence being stored as a stable limit cycle:

$$\arg\min_{f\in\mathcal{F}} \sum_{i=1}^{n} \left\| f\left(x^{((i \bmod n)+1))}\right) - x^{(i)} \right\|^2. \qquad [2]$$

Multiple cycles can be encoded similarly (*SI Appendix, SI Materials and Methods A*). In particular, we provide several examples of networks storing video and audio samples as limit cycles. Interestingly, these experiments suggest that sequence encoding provides a more efficient mechanism for memorization and retrieval of training examples than autoencoding. By considering a sequence encoder as a composition of maps, we indeed prove that sequence encoders are more contractive to a sequence of examples than autoencoders are to individual examples.

Related Work

Autoencoders (12) are commonly used for manifold learning, and the autoencoder architecture and objective (Eq. **1**) have been modified in several ways to improve their ability to represent data manifolds. Two variations, contractive and denoising autoencoders, add specific regularizers to the objective function in order to make the functions implemented by the autoencoder contractive toward the training data (13–15). However, these autoencoders are typically used in the underparameterized regime, where they do not have the capacity to interpolate (fit exactly) the training examples and hence, cannot store the training examples as fixed points.

On the other hand, it is well known that overparameterized neural networks can interpolate the training data when trained with GD methods (10, 16–18). As a consequence, overparameterized autoencoders can store training examples as fixed points. In particular, recent work empirically studied overparameterized autoencoders in the setting with one training example (19).

In this paper, we take a dynamical systems perspective to study overparameterized autoencoders and sequence encoders. In particular, we show that not only do overparameterized autoencoders (sequence encoders) trained using standard methods store training examples (sequences) as fixed points (limit cycles) but that these fixed points (limit cycles) are attractors (stable; i.e., they can be recovered via iteration). While energy-based methods have also been shown to be able to recall sequences as stable limit cycles (20, 21), the mechanism identified here is unrelated: it does not require setting up an energy function and is a direct consequence of training an overparameterized network.

Background from Dynamical Systems

We now introduce tools related to dynamical systems that we will use to analyze autoencoders and sequence encoders.

Attractors in Dynamical Systems. Let $f:\mathbb{R}^d \to \mathbb{R}^d$ denote the function learned by an autoencoder trained on a dataset $X = \{x^{(i)}\}_{i=1}^{n} \subset \mathbb{R}^d$. Consider the sequence $\{f^k(x)\}_{k\in\mathbb{N}}$ where $f^k(x)$ denotes k compositions of f applied to $x \in \mathbb{R}^d$. A point $x \in \mathbb{R}^d$ is a fixed point of f if $f(x) = x$; in this case, the sequence $\{f^k(x)\}_{k\in\mathbb{N}}$ trivially converges to x.

Since overparameterized autoencoders interpolate the training data, it holds that $f(x^{(i)}) = x^{(i)}$ for each training example $x^{(i)} \in X$; hence, all training examples are fixed points of f.* We now formally define what it means for a fixed point to be an attractor and provide a sufficient condition for this property.

***Definition 1*:** A fixed point $x^* \in \mathbb{R}^d$ is an attractor of $f:\mathbb{R}^d \to \mathbb{R}^d$ if there exists an open neighborhood, $\mathcal{O}$, of x^*, such that for any $x \in \mathcal{O}$, the sequence $\{f^k(x)\}_{k\in\mathbb{N}}$ converges to x^* as $k \to \infty$. The set S of all such points is called the basin of attraction of x^*.

Proposition 1. *A fixed point $x^* \in \mathbb{R}^d$ is an attractor of a differentiable map f if all eigenvalues of the Jacobian of f at x^* are strictly less than one in absolute value. If any of the eigenvalues are greater than one, x^* cannot be an attractor.*

Proposition 1 is a well-known condition in the theory of dynamical systems (chapter 6 of ref. 22). The condition intuitively means that the function f is "flatter" around an attractor x^*. Since training examples are fixed points in overparameterized autoencoders, from Proposition 1, it follows that a training example is an attractor if the maximum eigenvalue (in absolute value) of the Jacobian at the example is less than one. Since attractors are recoverable through iteration, autoencoders that

*To ensure $f(x^{(i)}) \approx x^{(i)}$, it is essential to train until the loss is very small; we used $\leq 10^{-8}$.

store training examples as attractors guarantee recoverability of these examples. Energy-based methods also allow for verification of whether a training example is an attractor. However, this requires checking the second-order condition that the Hessian is positive definite at the training example, which is more computationally expensive than checking the first-order condition from Proposition 1.

Discrete Limit Cycles in Dynamical Systems. Discrete limit cycles can be considered the equivalent of an attractor for sequence encoding, and a formal definition is provided below.

***Definition 2*:** A finite set $X^* = \{x^{(i)}\}_{i=1}^n \subset \mathbb{R}^d$ is a stable discrete limit cycle of a smooth function $f : \mathbb{R}^d \to \mathbb{R}^d$ if 1) $f(x^{(i)}) = x^{(i \bmod n)+1}$ $\forall i \in \{1, \dots n\}$. 2) There exists an open neighborhood, $\mathcal{O}$, of X^* such that for any $x \in \mathcal{O}$, X^* is the limit set of $\{f^k(x)\}_{k=1}^\infty$.

The equivalent of Proposition 1 for verifying that a finite sequence of points forms a limit cycle is provided below.

Proposition 2. *Let network $f : \mathbb{R}^d \to \mathbb{R}^d$ be trained on a given sequence $x^{(1)}, \dots, x^{(n)}$ such that $f(x^{(i)}) = x^{((i \bmod n)+1)}$. Then, the sequence $\{x^{(i)}\}_{i=1}^n$ forms a stable discrete limit cycle if the largest eigenvalue of the Jacobian of $f^n(x^{(i)})$ is (in absolute value) less than one for any i.*

This follows directly by applying Proposition 1 to the map f^n since $x^{(i)} = f^n(x^{(i)})$ and $f(x^{(i)}) = x^{((i \bmod n)+1)}$.

Before presenting our results, we provide the following important remark.

Why the Emergence of Attractors in Autoencoders Is Notable. Proposition 1 states that for a fixed point to be an attractor, all eigenvalues of the Jacobian at that point must be less than one in absolute value. Since the number of eigenvalues of the Jacobian equals the dimension of the space, this means that the angle of the derivative is less than $\pi/4$ in every eigendirection of the Jacobian. This is a highly restrictive condition since intuitively, we expect the "probability" of such an event to be $1/2^d$. Hence, a fixed point of an arbitrary high-dimensional map is unlikely to be an attractor. Indeed, as we show below in Corollary, fixed points of neural networks are not generally attractors. While not yet fully understood, the emergence and indeed, proliferation of attractors in autoencoding are not due solely to architectures but to specific inductive biases of the training procedures.

Empirical Findings

Training Examples Are Stored as Attractors in Overparameterized Autoencoders. In the following, we present a range of empirical evidence that attractors arise in autoencoders across common architectures and optimization methods. Details on the specific architectures and optimization schemes used for each experiment are in *SI Appendix*, Fig. S1.

Storing Images as Attractors. In Fig. 2, we present an example of an overparameterized autoencoder storing 500 images from ImageNet-64 (11) as attractors. This was achieved by training an autoencoder with depth 10, width 1,024, and cosid nonlinearity (23) on 500 training examples using the Adam (24) optimizer to loss $\leq 10^{-8}$. We verified that all 500 training images were stored as attractors by checking that the magnitudes of all eigenvalues of the Jacobian matrix at each example were less than one. Indeed, Fig. 2*A* demonstrates that iteration of the trained autoencoder map starting from corrupted inputs converges to individual training examples. A common practice for measuring recoverability of training patterns is to input corrupted versions of the patterns and verify that the system is able to recover the original patterns. From Proposition 1, provided that a corrupted example is in the basin of attraction of the original example, iteration is guaranteed to converge to the original example. In examples 5 and 6 in Fig. 2*A*, the corrupted images are not in the basin of attraction for the original examples, and so, iteration converges to a different (but contextually similar) training example. Fig. 2*B* provides further examples of correct recovery from corrupted images. Fig. 2*C* presents a quantitative analysis of the recovery rate of training examples under various forms of corruption. Overall, the recovery rate is remarkably high: even when 50% of the image is corrupted, the recovery rate of the network is significantly higher than expected by chance.

Examples of autoencoders storing training examples as attractors when trained on 2,000 images from Modified National Institute of Standards and Technology (MNIST) (25) and 1,000 black-and-white images from Canadian Institute For Advanced Research (CIFAR10) (26) are presented in *SI Appendix*, Figs. S2 and S3, respectively. The MNIST autoencoder presented in *SI Appendix*, Fig. S2 stores 2,000 training examples as attractors. Note that one iteration of the learned map on test examples can look similar to the identity function, but in fact, iterating until convergence yields a training example (*SI Appendix*, Fig. S2).

Spurious Attractors. While in these examples, we verified that the training examples were stored as attractors by checking the eigenvalue condition, there could be spurious attractors (i.e., attractors other than the training examples). In fact, spurious attractors are known to exist for Hopfield networks (27). To investigate whether there are additional attractors outside of the training examples, we iterated the map from sampled test images and randomly generated images until convergence. More precisely, we declared convergence of the map at iteration k for some image x when $\|f^{k+1}(x) - f^k(x)\|_2 < 10^{-8}$ and concluded that $f^k(x)$ had converged to the training example $x^{(i)}$ if $\|f^k(x) - x^{(i)}\|_2 < 10^{-7}$.

In general, spurious attractors can exist for overparameterized autoencoders, and we provide examples in *SI Appendix*, Fig. S4. However, remarkably, for the network presented in Fig. 2, we could not identify any spurious attractors even after iterating the trained autoencoder map from $40,000$ test examples from ImageNet-64, $10,000$ examples of uniform random noise, and $10,000$ examples of Gaussian noise with variance 4.

Attractors Arise across Architectures, Training Methods, and Initialization Schemes. We performed a thorough analysis of the attractor phenomenon identified above across a number of common architectures, optimization methods, and initialization schemes. Starting with fully connected autoencoders, we analyzed the number of training examples stored as attractors when trained on 100 black-and-white images from CIFAR10 (26) under the following nonlinearities, initializations, and optimization methods.

- Nonlinearities: rectified linear unit (ReLU), Leaky ReLU, scaled exponential linear units, cosid ($\cos x - x$), Swish (23, 28–30), and sinusoidal ($x + (\sin 10x)/5$).
- Optimization methods: GD, GD with momentum, GD with momentum and weight decay, rms propagation, and Adam (chapter 8 of ref. 31).
- Initialization schemes: random uniform initialization, namely $U[-a, a]$, per weight for $a \in \{0.01, 0.02, 0.05, 0.1, 0.15\}$. These initialization schemes subsume the PyTorch (version 0.4) default, Xavier initialization, and Kaiming initialization (32–34).

In *SI Appendix*, Tables S1 and S2, we provide the number of training examples stored as attractors for all possible combinations of 1) nonlinearity and optimization method listed above and 2) nonlinearity and initialization scheme listed above. These tables demonstrate that attractors arise in all settings for which training converged to a sufficiently low loss within $1,000,000$ epochs. In *SI Appendix*, Figs. S5 and S6 and *SI Materials and Methods F*, we also present examples of convolutional and

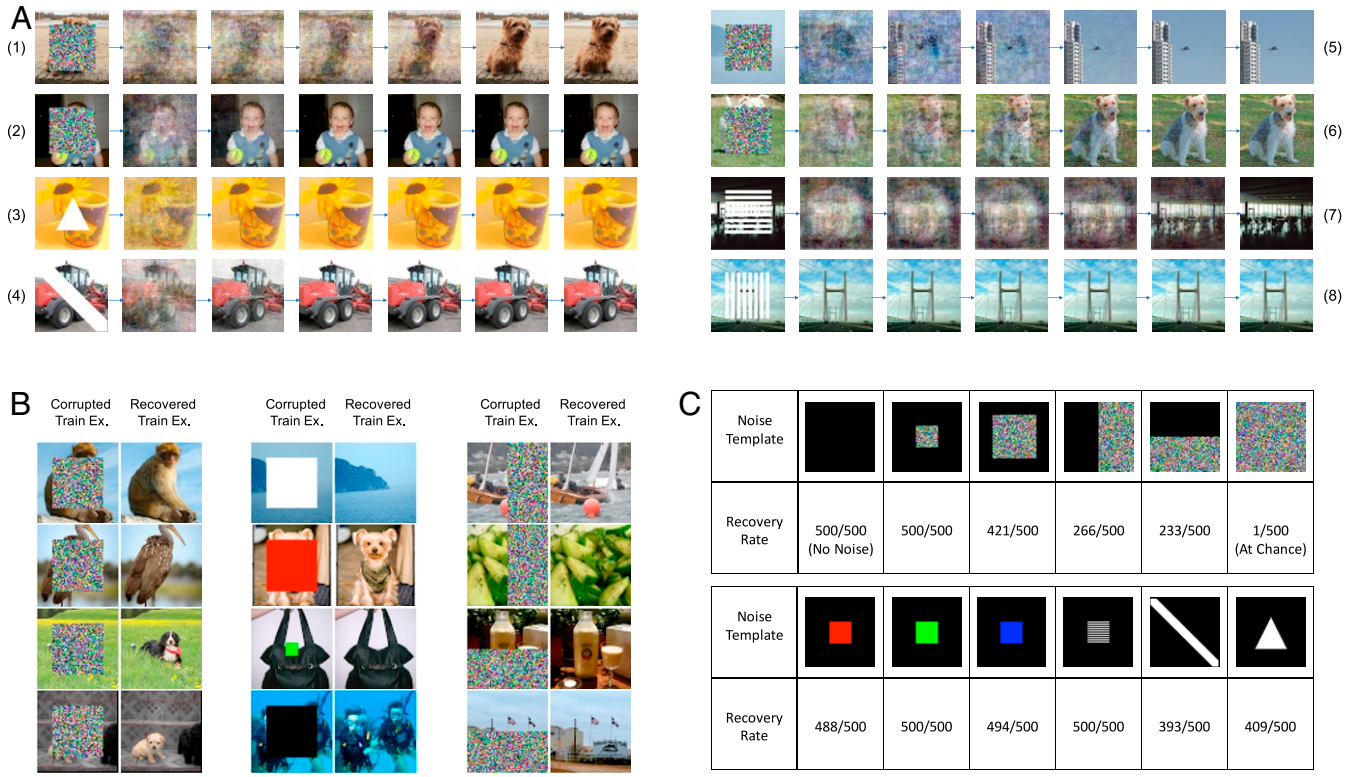


Fig. 2. Example of an overparameterized autoencoder storing 500 images from ImageNet-64 as attractors after training to a reconstruction error of less than 10^{-8}. Architecture and optimizer details are provided in *SI Appendix*, Fig. S1. (*A*) By iterating the trained autoencoder on corrupted versions of training samples, individual training samples are recovered. (*B*) Samples that are corrupted by uniform random noise or squares of varying color and size are recovered via iteration. (*C*) Fraction of samples recovered correctly from different noise applied to the training images. A sample is considered recovered when the error between the original sample and the recovered sample is less than 10^{-7}.

recurrent networks that store training examples as attractors, thereby demonstrating that this phenomenon is not limited to fully connected networks and occurs in all commonly used network architectures.

Visualizing Attractors in Two Dimensions. In order to better understand the attractor phenomenon, we present an example of an overparameterized autoencoder storing training examples as attractors in the two-dimensional (2D) setting, where

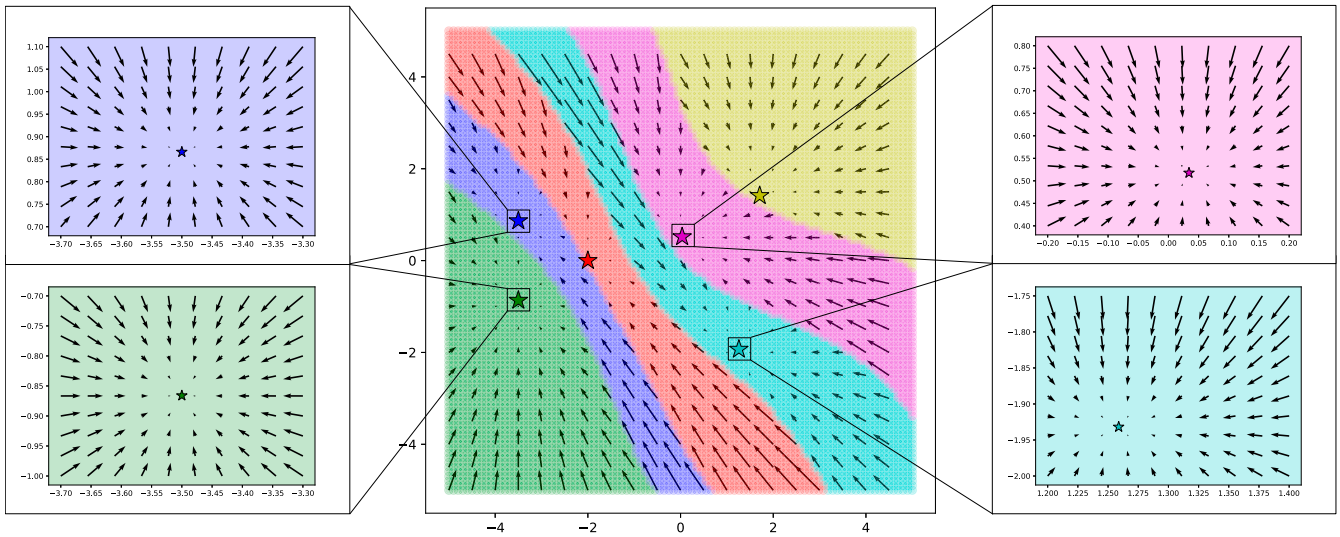


Fig. 3. Example of an overparameterized autoencoder in the 2D setting storing training examples (represented as stars) as attractors. Basins of attraction for each sample are colored by sampling 10, 000 points in a grid around the training examples, taking the limit of the iteration for each point, and assigning a color to the point based on the training example indicated by the limit. The vector field indicates the direction of motion given by iteration, and *Insets* indicate that iteration leads to training examples for all points in an open set around each example.

the basins of attraction can easily be visualized (Fig. 3). We trained an autoencoder to store six training examples as attractors. Their basins of attraction were visualized by iterating the trained autoencoder map starting from $10,000$ points on a grid until convergence. The vector field indicates the direction of motion given by iteration. Also in this experiment, we found no spurious attractors. Each training example and corresponding basin of attraction is colored differently. Interestingly, the example in Fig. 3 shows that the metric learned by the autoencoder to separate the basins of attraction is not Euclidean distance, which would be indicated by a Voronoi diagram.

Overparameterized Sequence Encoders Store Training Examples as Stable Limit Cycles and Are More Efficient at Memorizing and Retrieving Examples than Autoencoders. We have thus far analyzed the occurrence of attractors in overparameterized autoencoders. In this section, we demonstrate via various examples that by modifying the autoencoder objective to encode sequences (e.g., Eq. **2**), we can implement a form of associative memory for sequences. Details on the specific architectures and optimization schemes used for each experiment are in *SI Appendix*, Fig. S1.

Storing Sequences as Limit Cycles. We trained a network to encode 389 frames of size 128×128 from the Disney film "Steamboat Willie" by mapping frame i to frame $i+1 \mod 389$. Fig. 4*A* and Movies S1 and S2 show that iterating the trained network starting from random noise yields the original video.

As a second example, we encoded two 10-digit sequences from MNIST: one counting upward from digit 0 to 9 and the other counting down from digit 9 to 0. The maximal eigenvalues of the Jacobian of the trained encoder composed 10 times are 0.0034 and 0.0033 for the images from the first and second sequences, respectively. Hence, by Proposition 2, both sequences form limit cycles. Indeed, as shown in Fig. 4*B* and Movies S3 and S4, iteration from Gaussian noise leads to the recovery of both training sequences.

Finally, in Fig. 4*C*, we visualized the vector field and basins of attraction for four cycles in the 2D setting. Unlike autoencoding where points near a training example are pushed toward it via iteration, the points now move following the cycles. An animation of this process is shown in Movie S5. In *SI Appendix*, *SI Materials and Methods G*, we also trained a sequence encoder that stores 10 s of speech as a limit cycle. Movies S6 and S7 demonstrate that iterating the trained network from random noise recovers the original audio.

Efficiency of Sequence Encoding. In Fig. 5, we analyze the network sizes (width and depth) needed to store and retrieve 100 training images from MNIST using fully connected autoencoders and sequence encoders. Interestingly, our experiments indicate that memorization and retrieval of training examples can be performed more efficiently through sequence encoding than autoencoding. In particular, Fig. 5*A* shows the number of training examples (of 100) that are attractors for different width and depth of the network. Note that a depth of 31 and a width of 512 are needed to store almost all (99) training examples. If we instead encode the same data using 20 sequences of length 5, all 20 sequences (and thus, all 100 examples) can be recovered using a much smaller network with a depth of six and 512 hidden units per layer (Fig. 5*B*). Extending this idea further (Fig. 5 *C*–*E*), if we chain all 100 examples as a single sequence, the entire sequence is stored using a network with only one hidden layer and 512 hidden units.

Increasing Depth and Width Leads to More Attractors/Limit Cycles. The experiments in Fig. 5 indicate that increasing network depth and width leads to an increase in the number of training examples/sequences stored as attractors/limit cycles. For overparameterized autoencoders, this implies that the maximum eigenvalue of the Jacobian is less than one for a greater number of training examples upon increasing network depth and width (Proposition 1) (i.e., the network becomes more contractive around the training examples). Indeed, by analyzing the histogram of the maximum eigenvalue of the Jacobian at each of the training examples, we observed that as network depth and width increase, the mode of these histograms shifts closer to zero (*SI Appendix*, Fig. S7). Additionally, when considering the distribution of the top 1% of Jacobian eigenvalues, we find that as network width increases, the variance of the distribution of Jacobian eigenvalues decreases, and when depth increases, the mode of the distribution shifts closer to zero (*SI Appendix*, Fig. S8). In the following, we prove this phenomenon for a single training example (i.e., we prove that autoencoders trained on a single example become more contractive at the training example with increasing depth and width).

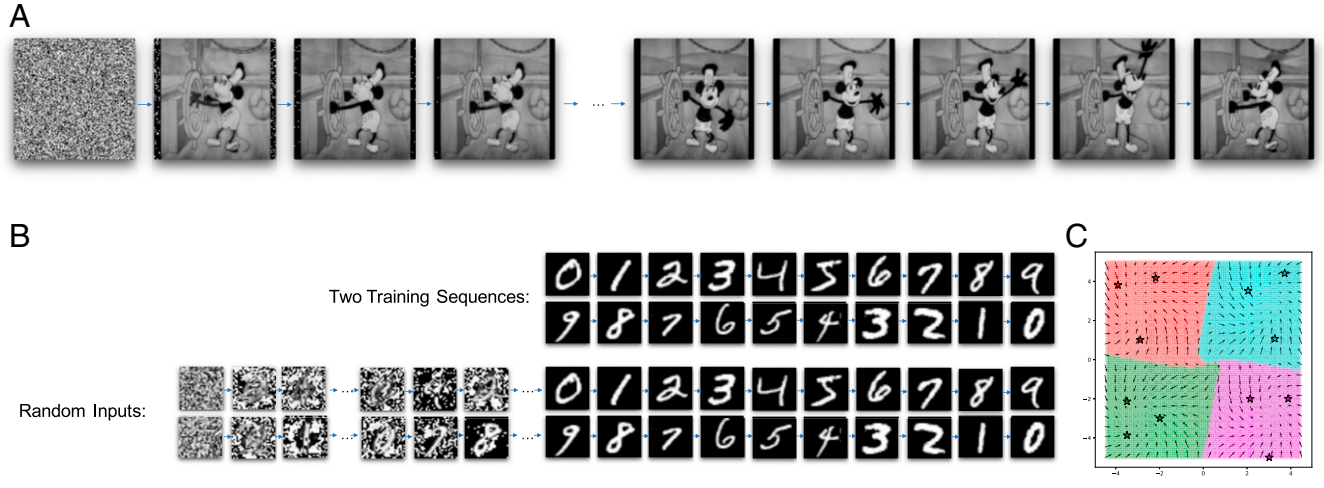
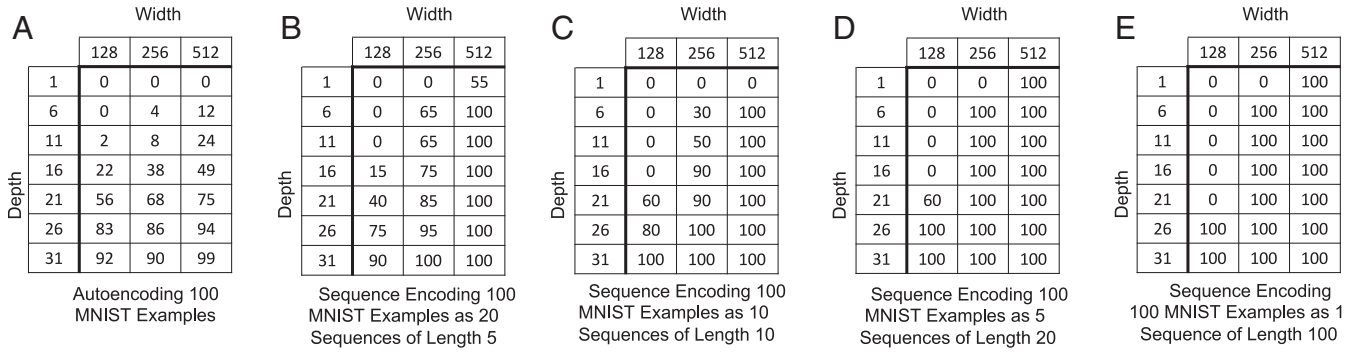


Fig. 4. Examples of overparameterized sequence encoders storing training sequences as limit cycles. Architecture and optimizer details are provided in *SI Appendix*, Fig. S1. (*A*) When trained on 389 frames of size 128×128 from the Disney film "Steamboat Willie," the entire movie was stored as a limit cycle. Hence, iteration from random noise leads to recovery of the entire sequence. (*B*) When trained on two sequences of length 10 from MNIST, each sequence was stored as a limit cycle. Hence, iteration from random noise leads to the recovery of each individual sequence. (*C*) Visualization of the basins of attraction for a sequence encoder storing four sequences as limit cycles in the 2D setting. The vector field indicates the direction of motion given by iteration.



A	128	256	512
1	0	0	0
6	0	4	12
11	2	8	24
16	22	38	49
21	56	68	75
26	83	86	94
31	92	90	99

B	128	256	512
1	0	0	55
6	0	65	100
11	0	65	100
16	15	75	100
21	40	85	100
26	75	95	100
31	90	100	100

C	128	256	512
1	0	0	0
6	0	30	100
11	0	50	100
16	0	90	100
21	60	90	100
26	80	100	100
31	100	100	100

D	128	256	512
1	0	0	100
6	0	100	100
11	0	100	100
16	0	100	100
21	60	100	100
26	100	100	100
31	100	100	100

E	128	256	512
1	0	0	100
6	0	100	100
11	0	100	100
16	0	100	100
21	0	100	100
26	100	100	100
31	100	100	100

Fig. 5. Sequence encoders are more efficient at implementing associative memory than autoencoders. Numbers of training examples recovered are out of 100; architecture and optimizer details are provided in *SI Appendix*, Fig. S1. (*A*) Number of recovered images when autoencoding 100 examples from MNIST individually; a network of depth 31 and width 512 recovers 99 images of 100. (*B–E*) Sequence encoding the same 100 MNIST examples as sequences of different lengths improves the recovery rates; in particular, a network of depth 1 and width 512 recovers the full 10 images when encoded as five sequences of length 20 (*D*) or one sequence of length 100 (*E*).

Theoretical Analysis of Special Cases

We now provide theoretical support for our empirical findings. Complete proofs are given in *SI Appendix*, *SI Materials and Methods B–E*.

Proof That When Trained on a Single Example, Overparameterized Autoencoders Store the Example as an Attractor. We outline the proof for the one-hidden layer setting. The complete proof for the multilayer setting is given in *SI Appendix*, *SI Materials and Methods C*.

Let $f(z) = W_1\phi(W_2 z)$ represent a one-hidden layer autoencoder with elementwise nonlinearity ϕ and weights $W_1 \in \mathbb{R}^{k_0 \times k}$ and $W_2 \in \mathbb{R}^{k \times k_0}$, applied to $z \in \mathbb{R}^{k_0}$. We analyze the function learned by GD with learning rate γ by minimizing the following autoencoding loss on one training example x:

$$\mathcal{L}(x, f) = \frac{1}{2}\|x - f(x)\|_2^2. \qquad [3]$$

Let $W_1^{(t)}$, $W_2^{(t)}$ denote the values of the weights after t steps of GD. To prove that x is an attractor of f after training, we solve for $W_1^{(\infty)}$, $W_2^{(\infty)}$ and compute the top eigenvalue of the Jacobian of f at x [denoted $\lambda_1(\mathbf{J}(f(x)))$].

In order to solve for W_1, W_2, we first identify two invariants of GD (proved in *SI Appendix*, *SI Materials and Methods B*):

***Invariant 1*:** If W_1 and W_2 are initialized to be rank 1 matrices of the forms $x u^{(0)\,T}$ and $v^{(0)} x^T$, respectively, then $W_1^{(t)} = x u^{(t)\,T}$ and $W_2^{(t)} = v^{(t)} x^T$ for all time steps $t > 0$.

***Invariant 2*:** If, in addition, all weights in each row of W_1 and W_2 are initialized to be equal, they remain equal throughout training.

Invariant 1 implies that autoencoders trained on one example produce outputs that are multiples of the training example. Generalizing this result, in *SI Appendix*, *SI Materials and Methods D*, we prove that autoencoders trained on multiple examples produce outputs in the span of the training data. Invariant 2 reduces GD dynamics to the one-dimensional setting. Using Invariants 1 and 2 combined with gradient flow (i.e., taking the limit as the learning rate $\gamma \to 0$), we can solve for $W_1^{(\infty)}$ and $W_2^{(\infty)}$.

Theorem 1. *Let $f(z) = W_1\phi(W_2 z)$ denote a one-hidden layer network with elementwise nonlinearity ϕ and weights $W_1 \in \mathbb{R}^{k_0 \times k}$ and $W_2 \in \mathbb{R}^{k \times k_0}$, applied to $z \in \mathbb{R}^{k_0}$. Let $x \in \mathbb{R}^{k_0}$ be a training example with $\|x\|_2 = 1$. Assuming $\frac{\phi(z)}{\phi'(z)} < \infty$ $\forall z \in \mathbb{R}$, then under Invariants 1 and 2, GD with learning rate $\gamma \to 0$ applied to minimize the autoencoding loss in Eq. 3 leads to a rank 1 solution $W_1^{(\infty)} = x u^T$ and $W_2^{(\infty)} = v x^T$ with $u, v \in \mathbb{R}^k$ satisfying*

$$\frac{u_i^2 - {u_i^{(0)}}^2}{2} = \int_{v_i^{(0)}}^{v_i} \frac{\phi(z)}{\phi'(z)}\,dz \quad \text{and} \quad u_i\phi(v_i) = \frac{1}{k},$$

and $u_i = u_j$, $v_i = v_j$ for all $i, j \in [k]$, where $u^{(0)}$ and $v^{(0)}$ are such that $W_1^{(0)} = x\,u^{(0)\,T}$ and $W_2^{(0)} = v^{(0)} x^T$.

Theorem 1 allows us to compute the top eigenvalue of the Jacobian at x, denoted by $\lambda_1(\mathbf{J}(f(x)))$.

Theorem 2. *Under the setting of Theorem 1, it holds that*

$$\lambda_1(\mathbf{J}(f(x))) = \frac{\phi'(v_i)v_i}{\phi(v_i)}.$$

Using Theorem 2, we can explicitly determine whether a training example x is an attractor, when given a nonlinearity ϕ, initial values for $u^{(0)}$ and $v^{(0)}$, and the width of the network k. We note that for all nonpiecewise nonlinearities used thus far, we can make any training example an attractor by selecting values for $u^{(0)}$, $v^{(0)}$, and k appropriately.

Example: Let x be a training example in $\mathbb{R}^{k_0}$. Suppose $\phi(z) = \frac{1}{1+e^{-z}}$ for $z \in \mathbb{R}$, $k = 2$, and $u_i^{(0)} = v_i^{(0)} = 1$ for all i. Then, by Theorems 1 and 2, it holds after training that

$$\frac{u_i^2 - 1}{2} = \int_1^{v_i} \left(\frac{1}{1 - \phi(z)}\right) dz \quad \text{and} \quad \frac{u_i}{1 + e^{-v_i}} = \frac{1}{2}$$

with $u_i \approx .697$, $v_i \approx .929$, and $\lambda_1(\mathbf{J}(f(x))) \approx .263$. Since $\lambda_1(\mathbf{J}(f(x))) < 1$, x is an attractor. We also confirmed this result (up to the third decimal place) experimentally by training a network using GD with learning rate 10^{-4}.

Importantly, the analysis of Theorem 2 implies that attractors arise as a result of training and are not simply a consequence of interpolation by a neural network with a certain architecture; see the following corollary.

Corollary. *Let $x \in \mathbb{R}^{k_0}$ with $\|x\|_2 = 1$ and $f(z) = x u^T \phi(v x^T z)$, where $u, v \in \mathbb{R}^k$ and ϕ is a smooth elementwise nonlinearity with $\frac{\phi'(z)}{\phi(z)} < \infty$ for all $z \in \mathbb{R}$, $\left|\frac{\phi'(z)z}{\phi(z)}\right| > 1$ for z in an open interval $\mathcal{O} \subset \mathbb{R}$. Then, there exist infinitely many $v \in \mathbb{R}^k$, such that $f(x) = x$ and x is not an attractor for f.*

The condition, $|\phi'(z)z/\phi(z)| > 1$ for z in an open interval, holds for all smooth nonlinearities considered in this paper. The proof is presented in *SI Appendix*, *SI Materials and Methods B*.

We note that while the linear setting with $\phi(z) = z$ has been studied extensively using gradient flow (35–37), our results extend to the nonlinear setting.

Remarks on the Multiple Sample Setting. While we extend Invariant 1 to the multiple example setting in *SI Appendix*, *SI Materials and Methods D*, a similar extension of Invariant 2 is required in order to generalize Theorem 1 to multiple examples. We believe such an extension may be possible for orthonormal training examples. Under random initialization, it may be possible to prove the attractor phenomenon by analyzing autoencoders in the Neural Tangent Kernel (NTK) regime (38). However, the disadvantage of such an analysis is that it relies on computing a closed form for the NTK in the limiting case of network width approaching infinity. On the other hand, Theorem 1 holds for a general class of nonlinearities and for finite width and depth.

Remarks on Similarity to Power Iteration. The attractor phenomenon identified in this work appears similar to that of Fast Independent Component Analysis (39) or more general nonlinear power iteration (40), where every "eigenvector" (corresponding to a training example in our setting) of a certain iterative map has its own basin of attraction. In particular, increasing network depth may play a similar role to increasing the number of iterations in those methods. While the mechanism may be different, understanding this connection is an important direction for future work.

Proof That Sequence Encoding Provides a More Efficient Mechanism for Memory than Autoencoding by Analyzing Sequence Encoders as a Composition of Maps. We start by generalizing Invariants 1 and 2 and Theorem 1 to the case of training a network to map an example $x^{(i)} \in \mathbb{R}^{k_0}$ to an example $x^{(i+1)} \in \mathbb{R}^{k_0}$ as follows.

Theorem 3. *Let $f(z) = W_1\phi(W_2 z)$ denote a one-hidden layer network with elementwise nonlinearity ϕ and weights $W_1 \in \mathbb{R}^{k_0 \times k}$ and $W_2 \in \mathbb{R}^{k \times k_0}$, applied to $z \in \mathbb{R}^{k_0}$. Let $x^{(i)}, x^{(i+1)} \in \mathbb{R}^{k_0}$ be training examples with $\|x^{(i)}\|_2 = \|x^{(i+1)}\|_2 = 1$. Assuming that $\frac{\phi(z)}{\phi'(z)} < \infty \ \forall z \in \mathbb{R}$ and there exist $u^{(0)}, v^{(0)} \in \mathbb{R}^k$ such that $W_1^{(0)} = x^{(i+1)} u^{(0)\,T}$ and $W_2^{(0)} = v^{(0)} x^{(i)\,T}$ with $u_i^{(0)} = u_j^{(0)}, v_i^{(0)} = v_j^{(0)} \ \forall i, j \in [k]$. Then, GD with learning rate $\gamma \to 0$ applied to minimize*

$$\mathcal{L}(x, f) = \frac{1}{2}\left\| x^{(i+1)} - f(x^{(i)}) \right\|_2^2 \qquad [4]$$

leads to a rank 1 solution $W_1^{(\infty)} = x^{(i+1)} u^T$ and $W_2^{(\infty)} = v x^{(i)\,T}$ with $u, v \in \mathbb{R}^k$ satisfying

$$\frac{u_i^2 - u_i^{(0)\,2}}{2} = \int_{v_i^{(0)}}^{v_i} \frac{\phi(z)}{\phi'(z)}\, dz, \quad \text{and} \quad u_i \phi(v_i) = \frac{1}{k},$$

and $u_i = u_j$, $v_i = v_j$ for all $i, j \in [k]$.

The proof is analogous to that of Theorem 1. Sequence encoding can be viewed as a composition of individual networks f_i that are trained to map example $x^{(i)}$ to example $x^{((i \bmod n)+1)}$. The following theorem provides a sufficient condition for when the composition of these individual networks stores the sequence of training examples $\{x^{(i)}\}_{i=1}^n$ as a stable limit cycle.

Theorem 4. *Let $\{x^{(i)}\}_{i=1}^n$ be n training examples with $\|x^{(i)}\|_2 = 1$ for all $i \in [n]$, and let $\{f_i\}_{i=1}^n$ denote n one-hidden layer networks satisfying the assumptions in Theorem 3 and trained on the loss in Eq. 4. Then, the composition $f = f_n \circ f_{n-1} \circ \ldots \circ f_1$ satisfies*

$$\lambda_1\left(\mathbf{J}(f(x^{(1)}))\right) = \prod_{i=1}^{n} \left(\frac{\phi'(v_j^{(i)}) v_j^{(i)}}{\phi(v_j^{(i)})} \right). \qquad [5]$$

The proof is presented in *SI Appendix*, *SI Materials and Methods E*. Theorem 4 shows that sequence encoding provides a more efficient mechanism for memory than autoencoding. If each of the networks f_i autoencoded example x_i for $i \in [n]$, then Theorem 2 implies that each of the n training examples is an attractor (and thus, recoverable) if each term in the product in Eq. **5** is less than one. This in turn implies that the product itself is less than one, and hence, all training examples are stored by the corresponding sequence encoder, f, as a stable limit cycle.

Discussion

We have shown that standard overparameterized neural networks trained using standard optimization methods implement associative memory. In particular, we empirically showed that autoencoders store training examples as attractors and that sequence encoders store training sequences as stable limit cycles. We then demonstrated that sequence encoders provide a more efficient mechanism for memorization and retrieval of data than autoencoders. In addition, we mathematically proved that when trained on a single example, nonlinear fully connected autoencoders store the example as an attractor. By modeling sequence encoders as a composition of maps, we showed that such encoders provide a more efficient mechanism for implementing memory than autoencoders, a finding that fits with our empirical evidence. We end by discussing implications and possible future extensions of our results.

Inductive Biases. In the overparameterized regime, neural networks can fit the training data exactly for different values of parameters. In general, such interpolating autoencoders do not store data as attractors (Corollary). Yet, as we showed in this paper, this is typically the case for parameter values chosen by gradient-based optimization methods. Thus, our work identifies an inductive bias of the specific solutions selected by the training procedure. Furthermore, increasing depth and width leads to networks becoming more contractive around the training examples, as demonstrated in Fig. 5.

While our paper concentrates on the question of implementing associative memory, we employ the same training procedures and similar network architectures to those used in standard supervised learning tasks. We believe that our finding on the existence and ubiquity of attractors in these maps may shed light on the important question of inductive biases in interpolating neural networks for classification (41).

Generalization. While generalization in autoencoding often refers to the ability of a trained autoencoder to reconstruct test data with low error (19), this notion of generalization may be problematic for the following reason. The identity function achieves zero test error and thus, "generalizes," although no training is required for implementing this function. In general, it is unclear how to formalize generalization for autoencoding, and alternate notions of generalization may better capture the desired properties. An alternative definition of generalization is the ability of an autoencoder to map corrupted versions of training examples back to their originals (as in Fig. 2 *A* and *B*). Under this definition, overparameterized autoencoders storing training examples as attractors generalize (Fig. 2*C*), while the identity function does not generalize. Given this issue with the current notion of generalization for autoencoding, it is an important line of future work to provide a definition of generalization that appropriately captures desired properties of trained autoencoders. Lastly, another important direction of future work is to build on the properties of autoencoders and sequence encoders identified in this work to understand generalization properties of networks used for classification and regression.

Metrics Used by Nonlinear Networks. In Fig. 3, we provided a visualization of how the basins of attraction for individual training examples subdivide the space of inputs. The picture appears very different from the Voronoi tessellation corresponding to the one-nearest neighbor (1-NN) predictor, where each input is associated with its closest training point in Euclidean distance. Yet, this may be different in high dimension. In *SI Appendix*, Fig. S9, we compare the recovery rate of our network from Fig. 2 with that of a 1-NN classifier and observe remarkable similarity, leading us to conjecture that the basins of attraction of high-dimensional fully connected neural networks may be closely related to the tessellations produced by 1-NN predictors. Thus, understanding the geometry of attractors in high-dimensional neural networks is an important direction of future research.

Connection to Biological Systems. Finally, another avenue for future exploration [and a key motivation for the original work on Hopfield networks (1)] is the connection of autoencoding and sequence encoding in neural nets to memory mechanisms in biological systems. Since overparameterized autoencoders and sequence encoders recover stored patterns via iteration, the retrieval mechanism presented here is biologically plausible. However, back propagation is not believed to be a biologically plausible mechanism for storing patterns (42). An interesting avenue for future research is to identify storage mechanisms that are biologically plausible and to see whether similar attractor phenomena arise in other, more biologically plausible, optimization methods.

Materials and Methods

An overview of all experimental details, including datasets, network architectures, initialization schemes, random seeds, and training hyperparameters, considered in this work is provided in *SI Appendix*, Fig. S1 and Tables S1 and S2. Briefly, we used the PyTorch library (32) and two NVIDIA Titan Xp graphics processing units for training all neural networks. In our autoencoding experiments on the image datasets ImageNet-64 (11), CIFAR10 (26), and MNIST (25), we trained both fully connected networks and U-Net convolutional networks (43). For Figs. 3, 4 *B* and *C*, and 5 as well as for training sequence encoder models on audio and video samples (attached as Movies S1 and S6), we used fully connected networks. For all these experiments, we used the Adam optimizer with a learning rate of 10^{-4} until the mean squared error dropped below 10^{-8}. For *SI Appendix*, Tables S1 and S2, we fixed the architecture width and depth while varying the initialization scheme, optimization method, and activation function.

Data Availability. All study data are included in the article and *SI Appendix*.

ACKNOWLEDGMENTS. We thank the Simons Institute at University of California, Berkeley for hosting us during the summer 2019 program on "Foundations of Deep Learning," which facilitated this work. A.R. and C.U. were partially supported by NSF Grant DMS-1651995, Office of Naval Research Grants N00014-17-1-2147 and N00014-18-1-2765, IBM, and a Simons Investigator Award (to C.U.). M.B. acknowledges support from NSF Grants IIS-1815697 and IIS-1631460, and a Google Faculty Research Award. The Titan Xp used for this research was donated by the NVIDIA Corporation.

1. J. J. Hopfield, Neural networks and physical systems with emergent collective computational abilities. *Proc. Natl. Acad. Sci. U.S.A.* **79**, 2554–2558 (1982).
2. W. A. Little, The existence of persistent states in the brain. *Math. Biosci.* **19**, 101–120 (1974).
3. R. McEliece, E. Posner, E. R. Rodemich, S. S. Venkatesh, The capacity of the Hopfield associative memory. *IEEE Trans. Inf. Theor.* **33**, 461–482 (1987).
4. S. Bartunov, J. Rae, S. Osindero, T. Lillicrap, "Meta-learning deep energy-based memory models" in *International Conference on Learning Representation* (ICLR, 2020).
5. G. E. Hinton, O. Simon, Y.-W. Teh, A fast learning algorithm for deep beliefnets. *Neural Comput.* **18**, 1527–1554 (2006).
6. G. E. Hinton, "A practical guide to training restricted Boltzmann machines" in *Neural Networks: Tricks of the Trade*, G. Montavon, G. B. Orr, K. R. Müller, Eds. (Lecture Notes in Computer Science, vol. 7700, Springer, Berlin, Germany, 2012), pp. 599–619.
7. Y. Du, I. Mordatch, Implicit generation and generalization in energy-based models. arXiv:1903.08689 (230 June 2020).
8. R. Salakhutdinov, H. Larochelle, "Efficient learning of deep Boltzmann machines" in *International Conference on Artificial Intelligence and Statistics*, Y. W. Teh, M. Titterington, Eds. (JMLR, 2010), pp. 233–242.
9. D. Arpit *et al.*, "A closer look at memorization in deep networks" in *International Conference on Machine Learning*, D. Precup, Y. W. Teh, Eds. (Proceedings of Machine Learning Research, 2017), pp. 233–242.
10. C. Zhang, S. Bengio, M. Hardt, B. Recht, O. Vinyals, "Understanding deep learning requires rethinking generalization" in *International Conference on Learning Representations* (ICLR, 2017).
11. A. van den Oord, N. Kalchbrenner, K. Kavukcuglu, "Pixel recurrent neural networks" in International Conference on Machine Learning, M. F. Balcan, K. Q. Weinberger, Eds. (Proceedings of Machine Learning Research, 2016), pp. 1747–1756.
12. P. Baldi, "Autoencoders, unsupervised learning, and deep architectures" in *International Conference on Machine Learning*, I. Guyon, G. Dror, V. Lemaire, G. Taylor, D. Silver, Eds. (Proceedings of Machine Learning Research, 2012), pp. 37–49.
13. P. Vincent, H. Larochelle, Y. Bengio, P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders" in *International Conference on Machine Learning* (Association for Computing Machinery, New York, NY, 2008), pp. 1096–1103.
14. S. Rifai, P. Vincent, X. Muller, X. Glorot, Y. Bengio, "Contractive auto-encoders: Explicit invariance during feature extraction" in *International Conference on Machine Learning*, L. Getoor, T. Scheffer, Eds. (Omnipress, Madison, WI, 2011), pp. 833–840.
15. G. Alain, Y. Bengio, What regularized auto-encoders learn from the data-generating distribution. *J. Mach. Learn. Res.* **15**, 3743–3773 (2014).
16. S. S. Du, X. Zhai, B. Poczos, A. Singh, "Gradient descent provably optimizes over-parameterized neural networks" in *International Conference on Learning Representations* (ICLR, 2019).
17. S. S. Du, J. D. Lee, H. Li, L. Wang, X. Zhai, "Gradient descent finds global minima of deep neural networks" in *International Conference on Machine Learning*, K. Chaudhuri, R. Salakhutdinov, Eds. (Proceedings of Machine Learning Research, 2019), pp. 1675–1685.
18. X. Wu, S. S. Du, R. Ward, Global convergence of adaptive gradient methods for an over-parameterized neural network? arXiv:1902.07111 (19 October 2019).
19. C. Zhang, S. Bengio, M. Hardt, Y. Singer, Identity crisis: Memorization and generalization under extreme overparameterization. arXiv:1902.04698 (9 January 2020).
20. B. Kosko, Bidirectional associative memories. *IEEE Trans. Syst. Man Cybern.* **18**, 49–60 (1988).
21. J. Buhmann, K. Schulten, "Storing sequences of biased patterns in neural networks with stochastic dynamics" in *Neural Computers*, R. Eckmiller, C. v. d. Malsburg, Eds. (Springer Study Edition, vol. 41, Springer, Berlin, Germany, 1989), 231–242.
22. S. Strogatz, *Nonlinear Dynamics and Chaos* (Westview Press, 2015), vol. 2.
23. S. Eger, P. Youssef, I. Gurevych, "Is it time to swish? Comparing deep learning activation functions across NLP tasks" in *Empirical Methods in Natural Language Processing* (Association for Computational Linguistics, 2018), pp. 4415–4424.
24. D. P. Kingma, J. Ba, "Adam: A method for stochastic optimization" in *International Conference on Learning Representations* (ICLR, 2015).
25. Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition. *Proc. IEEE* **86**, 2278–2324 (1998).
26. A. Krizhevsky, "Learning multiple layers of features from tiny images," Master's thesis, University of Toronto, Toronto, Canada (2009).
27. J. J. Hopfield, D. I. Feinstein, R. G. Palmer, 'Unlearning' has a stabilizing effect in collective memories. *Nature* **304**, 158–159 (1983).
28. P. Ramachandran, B. Zoph, Q. V. Le, "Searching for activation functions" in *International Conference on Learning Representations* (ICLR, 2017).
29. B. Xu, N. Wang, T. Chen, M. Li, Empirical evaluation of rectified activations in convolution network. arXiv:1505.00853 (27 November 2015).
30. G. Klambauer, T. Unterthiner, A. Mayr, S. Hochreiter, Self-normalizing neural networks. arXiv:1706.02515v5 (7 September 2017).
31. I. Goodfellow, Y. Bengio, A. Courville, *Deep Learning* (MIT Press, 2016), vol. 1.
32. A. Paszke *et al.*, "PyTorch: An imperative style, high-performance deep learning library" in *Advances in Neural Information Processing Systems 32*, H. Wallach *et al.*, Eds. (Curran Associates, Inc., Red Hook, NY, 2019), pp. 8024–8035.
33. K. He, X. Zhang, S. Ren, J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification" in *International Conference on Computer Vision* (IEEE Computer Society, Washington, DC, 2015), pp. 1026–1034.
34. X. Glorot, Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks" in *International Conference on Artificial Intelligence and Statistics*, Y. W. Teh, M. Titterington, Eds. (JMLR, 2010), pp. 249–256.
35. S. Gunasekar, B. E. Woodworth, S. Bhojanapalli, B. Neyshabur, N. Srebro, "Implicit regularization in matrix factorization" in *Advances in Neural Information Processing Systems 30*, I. Guyon *et al.*, Eds. (Curran Associates, Inc., Red Hook, NY, 2017), pp. 6151–6159.
36. S. Arora, N. Cohen, E. Hazan, "On the optimization of deep networks: Implicit acceleration by overparameterization" in *International Conference on Machine Learning*, J. Dy, A. Krause, Eds. (Proceedings of Machine Learning Research, 2018), pp. 244–253.

37. S. Arora, N. Cohen, N. Golowich, W. Hu, "A convergence analysis of gradient descent for deep linear neural networks" in *International Conference on Learning Representations* (ICLR, 2019).
38. A. Jacot, F. Gabriel, C. Hongler, "Neural tangent kernel: Convergence and generalization in neural networks" in *Neural Information Processing Systems* (Curran Associates, Inc., Red Hook, NY, 2018), pp. 8571–8580.
39. A. Hyvärinen, E. Oja, A fast fixed-point algorithm for independent component analysis. *Neural Comput.* **9**, 1483–1492 (1997).
40. M. Belkin, L. Rademacher, J. Voss, Eigenvectors of orthogonally decomposable functions. *SIAM J. Comput.* **47**, 547–615 (2018).
41. M. Belkin, D. Hsu, S. Ma, S. Mandal, Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proc. Natl. Acad. Sci. U.S.A.* **116**, 15849–15854 (2019).
42. S. Grossberg, Competitive learning: From interactive activation to adaptive resonance. *Cognit. Sci.* **11**, 23–63 (1987).
43. O. Ronneberger, P. Fischer, T. Brox, "U-Net: Convolutional networks for biomedical image segmentation" in *International Conference on Medical Image Computing and Computer Assisted Intervention*, N. Navab, J. Hornegger, W. M. Wells, A. F. Frangi, Eds. (Springer International Publishing, New York, NY, 2015), pp. 234–241.