Dynamic compression and expansion in a classifying recurrent network

Matthew Farrell[1-2], Stefano Recanatesi[1], Guillaume Lajoie[3-4], and Eric Shea-Brown[1-2]

[1] *Computational Neuroscience Center, University of Washington*
[2] *Department of Applied Mathematics, University of Washington*
[3] *Mila—Québec AI Institute*
[4] *Dept. of Mathematics and Statistics, Université de Montréal*

Abstract

Recordings of neural circuits in the brain reveal extraordinary dynamical richness and high variability. At the same time, dimensionality reduction techniques generally uncover low-dimensional structures underlying these dynamics when tasks are performed. In general, it is still an open question what determines the dimensionality of activity in neural circuits, and what the functional role of this dimensionality in task learning is. In this work we probe these issues using a recurrent artificial neural network (RNN) model trained by stochastic gradient descent to discriminate inputs. The RNN family of models has recently shown promise in revealing principles behind brain function. Through simulations and mathematical analysis, we show how the dimensionality of RNN activity depends on the task parameters and evolves over time and over stages of learning. We find that common solutions produced by the network naturally compress dimensionality, while variability-inducing chaos can expand it. We show how chaotic networks balance these two factors to solve the discrimination task with high accuracy and good generalization properties. These findings shed light on mechanisms by which artificial neural networks solve tasks while forming compact representations that may generalize well.

Introduction

Dynamics shape computation in brain circuits. These dynamics arise from highly recurrent and complex networks of interconnected neurons, and neural trajectories observed in cortical areas are correspondingly rich and variable across stimuli, time, and trials. Despite this high degree of variability in neural responses, repeatable and reliable activity structure is often unveiled by dimensionality reduction procedures [11, 42]. Rather than being set by, say, the number of neurons engaged in the circuit, the *effective* dimensionality of the activity (often called neural "representation") seems to be intimately linked to the complexity of the function, or behavior, that the neural circuit fulfills or produces [16, 44, 49, 10]. These findings appear to show some universality: similar task-dependent dimensional representations can manifest in artificial networks used in machine learning systems trained using optimization algorithms (e.g., [38, 10, 53]). This connection is especially intriguing in light of fundamental ideas about the dimension of representations in machine learning. On the one hand, high-dimensional representations can subserve complex and general computations that nonlinearly combine many features of inputs [9, 15, 46, 52]. On the other, low-dimensional representations that preserve only essential features needed for specific tasks can allow learning based on fewer parameters and examples, and hence with better "generalization" (for reviews, see [6, 15, 51, 8]). Bridging between machine learning and neuroscience, artificial networks are powerful tools for investigating dynamical representations in controlled settings, and enable us to test theoretical hypotheses that can be leveraged to formulate experimental predictions (see [5] for a review of this approach).

What are the mechanisms by which a network regulates its activity's effective dimension across many trials and inputs? One feature frequently encountered in recurrent neural network (RNN) models of brain function is dynamical chaos [47, 50, 27, 34, 54], whereby tiny changes in internal states are amplified by unstable, but deterministic, dynamics. Chaos provides a parsimonious explanation for both repeatable structure as well as internally generated variability seen in highly recurrent brain networks such as cortical circuits [27, 36, 48, 14]. In the reservoir computing framework, chaos in an RNN can increase the diversity of patterns and dimension of the activity the network produces through time in response to inputs, even if these inputs are simple and low-dimensional themselves [22, 37, 31]. The dimensionality expansion is likely a consequence of chaos driving neural trajectories to "expand out" in more directions of neural space, so that they collectively occupy more space. While chaos as determined by the initial weights influences the dimensionality of an RNN's response to inputs, this dimensionality can change as the recurrent weights evolve through training.

The goal of the present work is to investigate how initialization and training impacts activity dimension in RNNs, in a highly simplified setting. We initialize networks with varying degrees of chaos and train them to solve a simple and well-known task – delayed discrimination (classification) using working memory. Specifically, the RNN receives a static input sample and is then given time to process it during a delay period by using its undriven internal dynamics. After the delay period, the input's class is inferred by a linear readout of the network state. We employ the most basic and widely used training rule for neural networks: stochastic gradient descent with backpropagation (through time). While not biologically plausible in its simplest form, the characteristics of networks trained by this algorithm can still resemble dynamics in the brain [38, 53, 4, 41, 5].

Our framework directly addresses how dimensionality of the input and output spaces affects representation dimensionality and classification performance in RNNs. First, we observe that RNNs with high and low degrees of initial chaos both learn to compress high-dimensional, linearly separable inputs into a low-dimensional internal representation by the time this information is sent to the readout. Second, in the case of low-dimensional inputs with highly nonlinear separation boundaries, we observe differences in behavior depending on degree of chaos. Highly chaotic networks initially expand the effective dimensionality of their neural activations, resulting in representations that are simpler from the perspective of a linear readout and allowing the networks to learn the task with high accuracy. Less chaotic networks initialized on the "edge-of-chaos" are less adept at forming high-dimensional representations, and suffer reduced performance as a result. In both cases, the expansion is limited, suggesting a broad regime where chaotic RNNs trained with stochastic gradient descent can balance dimensionality expansion and compression. Finally, we investigate a special case (inputs that are only shown to a subset of neurons) where the benefits of strong chaos are even more pronounced, and evidence of active dimension compression appears more limited. We discuss how chaos impacts dimension using ideas from dynamical systems theory, and also provide analytical arguments outlining mechanisms by which stochastic gradient descent can naturally promote dimensionality compression.

We conclude that chaotic RNNs trained through stochastic gradient descent, without any regularization or special features or design, can often learn to automatically balance dimensionality expansion and compression – suggesting a flexible compromise between guiding principles of separation and generalization. This finding could shed light on the highly variable nature of neural circuits and inspire new ideas for improving the flexibility and accuracy of artificial neural networks. In addition, our analysis of stochastic gradient descent's tendency to compress dimensionality is a step toward revealing why it is often successful at finding solutions that generalize (e.g. [18, 30, 1, 25, 32]).

Results

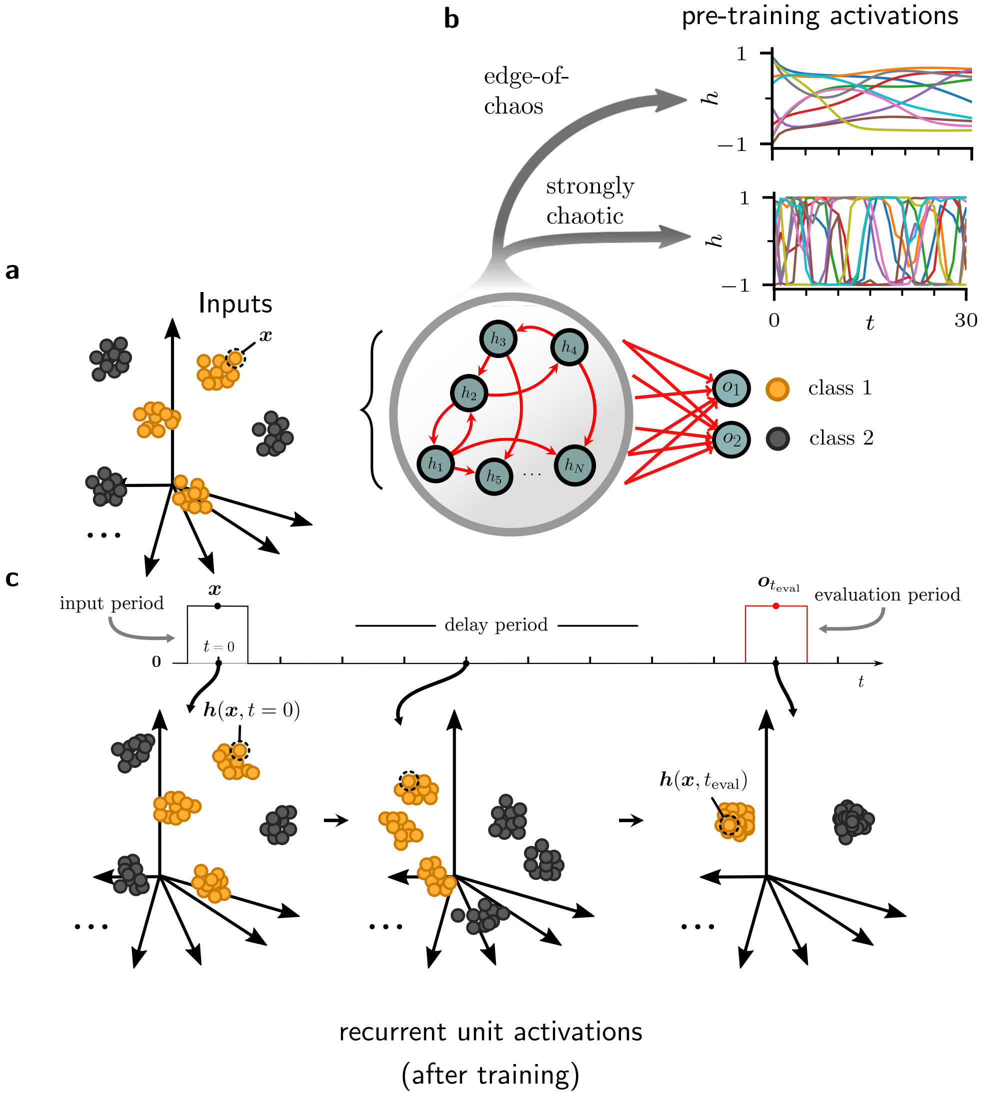

Figure 1: Task and model schematic. **a.** Input clusters are distributed in a d-dimensional subspace of neural space and presented to the network. **b.** Example trajectories of ten recurrent units of the untrained network responding to an input sample. Top: Trajectories for a network initialized on the edge of chaos. Bottom: Trajectories for a network initialized in the strongly chaotic regime. **c.** After a delay period in which the network evolves undriven, the network state (representation) is read out and used to classify the inputs. Recurrent and output weights are both modified by training. Bottom: Response of the network to the ensemble of inputs, viewed at snapshots in time. After training, the evolution of the RNN representation reflects an attempt by the network to help the readout be more successful. In the process, the effective dimensionality can change (here the representation goes from being high to low-dimensional). The dashed circle tracks a single input sample and the resulting network response. Points are schematic and not from simulation data.

Model and task overview. We investigate the dynamics of recurrent networks learning to classify static inputs. Our network model is based on standard RNN models used in machine learning [20, 33]. Interactions between $N = 200$ neural units are determined by a randomly initialized recurrent connectivity matrix, and unit activations pass through a hyperbolic tangent nonlinearity. The dynamics of the pre-trained network can be modulated from stable to chaotic by the average magnitude of the initial neural coupling strength increasing, and vice-versa. We compare chaotic networks initialized near the transition point to chaos (said to be on the "edge of chaos"), to "strongly chaotic" networks well past the transition point, as they learn to solve the task described below.

Inputs are N-dimensional constant vectors. They are selected from Gaussian-distributed clusters whose means are distributed randomly within a d-dimensional, random subspace of the N-dimensional neural activity space. We call this subspace the input's *ambient space* and consider two values for the ambient dimension: $d = 2$ and $d = N$. Each cluster is assigned one of two class labels, at random, as illustrated in Figure 1a (while this schematic only shows six clusters for clarity, in our simulations we use 60 clusters). The task proceeds as follows: (1) a random input is selected from one of the clusters and presented to the network for one timestep; (2) the network's dynamics evolve undriven during a delay period; (3) a linear readout of the network's state is used to classify the input into one of two classes. The delay period gives the network multiple timesteps to engage its dynamics and process the input (Figure 1c). The readout occurs at a single timestep ($t_{\text{eval}} = 10$). Recurrent and output weights are adjusted via a stochastic gradient descent routine to minimize a cross entropy loss function, and classification performance is evaluated on novel inputs not used for training. We find our results qualitatively robust to small changes in the choice of model parameters. (See Methods for a more detailed discussion.)

Networks compress high-dimensional inputs. We start by considering the classification of high-dimensional inputs, where input ambient space dimensionality d is equal to the number N of recurrent units (see Figure 2a for a visualization). While classification in networks is often viewed from the perspective of making inputs linearly separable, in our scenario the data are already linearly separable in the input space. This focuses our attention on what, if anything, networks learn to do beyond this.

Figure 2b measures the degree of chaos of the edge-of-chaos and strongly chaotic regimes by plotting the top 15 *Lyapunov exponents* χ_k of the networks through training (see Methods and Figure 5b for a description of this measure). In these spectrum plots, positive values of χ_k indicate chaotic dynamics. If all χ_k are negative, then the dynamics are non-chaotic (stable), meaning that trajectories converge to stable fixed points or stable limit cycles. The edge-of-chaos network is weakly chaotic before training and becomes stable after training, while the strongly chaotic network is chaotic both before and after training. While each of these plots only shows the exponents for a single network realization and a particular positioning of input clusters, they capture the general qualitative behavior of the two dynamical regimes for $d = N$.

As shown in Figure 2c, both networks easily achieve perfect testing accuracy on the task. What is now of interest is how properties of the network's *internal representation* change over the course of training. Figure 2d tracks the *effective dimensionalities* of the two dynamical regimes at the evaluation time t_{eval} through training. Effective dimensionality is closely related to dimensionality reduction via principal component analysis, and can be thought of as the number of principal components needed to explain most of the variance of the data (see Methods and Figure 5a for a precise definition). Networks in both regimes experience dimensionality compression, though the strongly chaotic network requires more training to do so. The degree of these trends depends somewhat on the learning procedure, with more aggressive weight updates resulting in faster dimensionality compression (data not shown). However, the general behavior is robust to small changes in the learning algorithm parameters.

To get a better sense for the processing of inputs driven by the recurrent dynamics, we investigate the dimensionality of network representations through time t in Figure 2e. The effective dimensionalities of the trained networks are approximately equal to that of the input at time $t = 0$, since the initial states of the network only differ from the inputs by one application of the nonlinearity (see Methods). The dimensionality drops with increasing t and is highly compressed at the evaluation

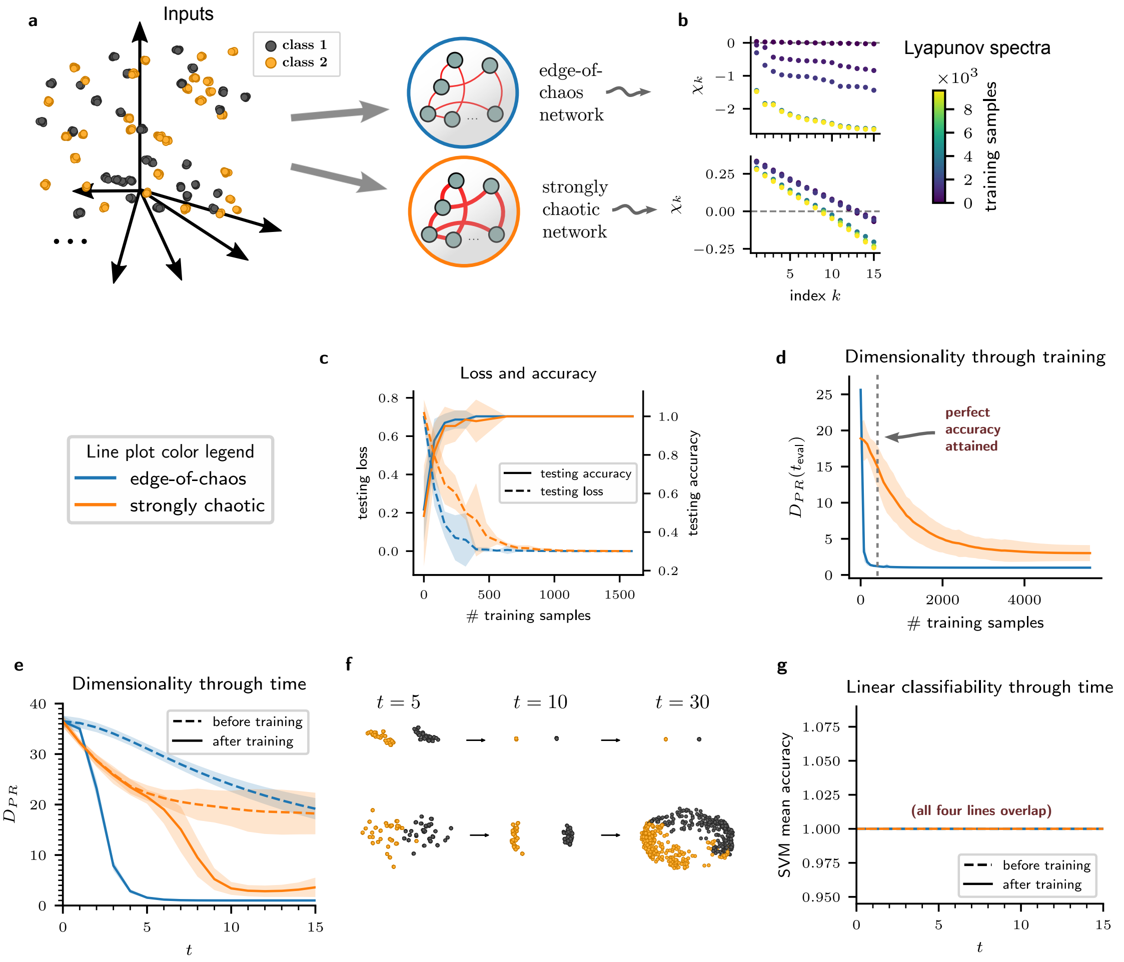

Figure 2: Comparison of edge-of-chaos and strongly chaotic networks classifying high-dimensional inputs. Input color (red or gray) denotes true class label. In all panels, blue denotes the edge-of-chaos network and orange the strongly chaotic network. Shaded regions indicate two standard deviations of a sample of six network and input realizations. **a.** Schematic of the task. **b.** Lyapunov exponents measured through training. Each plot is of a single network and input realization. Error bars (too small to see) denote standard error of the mean. **c.** Testing loss (dashed lines) and accuracy (solid) measured through training. **d.** Effective dimensionality through training, measured at the evaluation time $t_{\text{eval}} = 10$. Both networks compress dimensionality to a relatively low value, but the edge-of-chaos network does so more quickly. Dashed vertical line denotes point in training when networks achieve perfect accuracy. **e.** Effective dimensionality measured through time. The dashed and solid lines denote the dimensionality of the networks before and after training, respectively. Trained networks develop compressed representations at the readout time $t_{\text{eval}} = 10$. **f.** Activations of recurrent units responding to an ensemble of 600 inputs, plotted as “snapshots” in time in principal component space. The top row corresponds to the edge-of-chaos network and the bottom row to the strongly chaotic network. **g.** Mean accuracy of a support vector machine linear classifier trained on the recurrent unit activations at each timepoint t.

time $t_{\text{eval}} = 10$. This phenomenon becomes clearer upon looking at the top two principal components of the network activations at snapshots in time (see Methods) in Figure 2f. These scatter plots reveal that the low-dimensionality of the representation is a result of the networks forming two well-separated, spatially compressed attractors, one for each class. At time $t = 5$, both networks are using their dynamics to separate the points in the top principal component space. At the evaluation time $t_{\text{eval}} = 10$, not only have the classes been separated in this low-dimensional space, but the points belonging to each class have been compressed together in all dimensions. Looking at the representation at time $t_{\text{eval}} = 30$ gives an indication as to the long-time behavior of the network. The edge-of-chaos network has formed stable fixed point attractors, while the strongly chaotic network has formed chaotic attractors. The fixed points remain separate in principal component space indefinitely, while the trajectories in the chaotic attractors eventually mix back together. Figure 2g shows the mean accuracy of a support vector machine trained to classify the network representation at each timepoint. This plot reveals how the linearly separable inputs are kept linearly separable by the network dynamics, even as they compress dimensionality and in the process promote better generalization. We emphasize that in this case, separating the inputs does not require either dimensionality expansion or compression; nevertheless, trained networks do strongly compress their inputs.

Chaos expands low-dimensional inputs. We next turn our attention to inputs embedded in a two-dimensional ambient space, $d = 2$ (see Figure 3a for a visualization). In this case, the two classes are generally far from being linearly separable (classification boundaries must be curved and nonlinear to navigate around the 60 clusters randomly distributed in two-dimensional space). As a consequence, it is difficult for the network to classify without first increasing the dimensionality of its representation.

Figure 3 compares the behavior of the edge-of-chaos and strongly chaotic networks trained on this task. In this case, the edge-of-chaos network becomes very near the edge of stability through training (Figure 3b), with the top Lyapunov exponent close to 0. The strongly chaotic network remains strongly chaotic. The strongly chaotic network still achieves near-perfect accuracy, while the edge-of-chaos network is not as successful (Figure 3c).

We again track the effective dimensionalities of the network representations at the evaluation time $t_{\text{eval}} = 10$ through training (Figure 3d). In this case, the strongly chaotic network is higher-dimensional before training, and experiences a dimensionality compression through training. The edge-of-chaos network generally experiences a dimensionality expansion through training from an initially low value. In looking at the dimensionality of the trained networks through time, we find that both initially expand dimensionality until about $t = 7$ (Figure 3e, solid lines), with the expansion being much more dramatic for the strongly chaotic network. The strongly chaotic network then reverses course to compress dimensionality up to time $t_{\text{eval}} = 10$. The dimensionality expansion of the strongly chaotic network up to $t = 7$ seems to follow from its natural tendency to expand dimensionality before training (Figure 3e, dashed lines). The compression from $t = 7$ to $t = 10$ is learned through training. Looking at the response ensemble at the evaluation time (Figure 3f, $t = 10$, top), we see that the strongly chaotic network has neatly sorted out the two classes in the top principal component space. The edge-of-chaos network has not separated out the classes as cleanly (bottom). After $t_{\text{eval}} = 10$, the edge-of-chaos network settles into limit cycles or stable fixed points while the points of the strongly chaotic network expand as they move along the chaotic attractor.

Concerning the linear classifiability of the network representation (Figure 3g), we see that the strongly chaotic network's pre-training expansion of dimensionality is effective at creating a representation that is linearly separable, while the edge-of-chaos network's representation is far from being linearly separable before training. This highlights the efficacy of the dimensionality expansion strategy as used by the strongly chaotic network in achieving linear separability and better performance on the task. The network balances this expansion with an equally pronounced dimensionality compression through time, suggesting good generalization.

If we assume that input is only shown to a subset of neurons (Figure 4a), then dimensionality expansion can have the added benefit of enlisting more neurons to aid in the computation [28]. In this case, we might expect the strongly chaotic network to have an even more pronounced advantage over

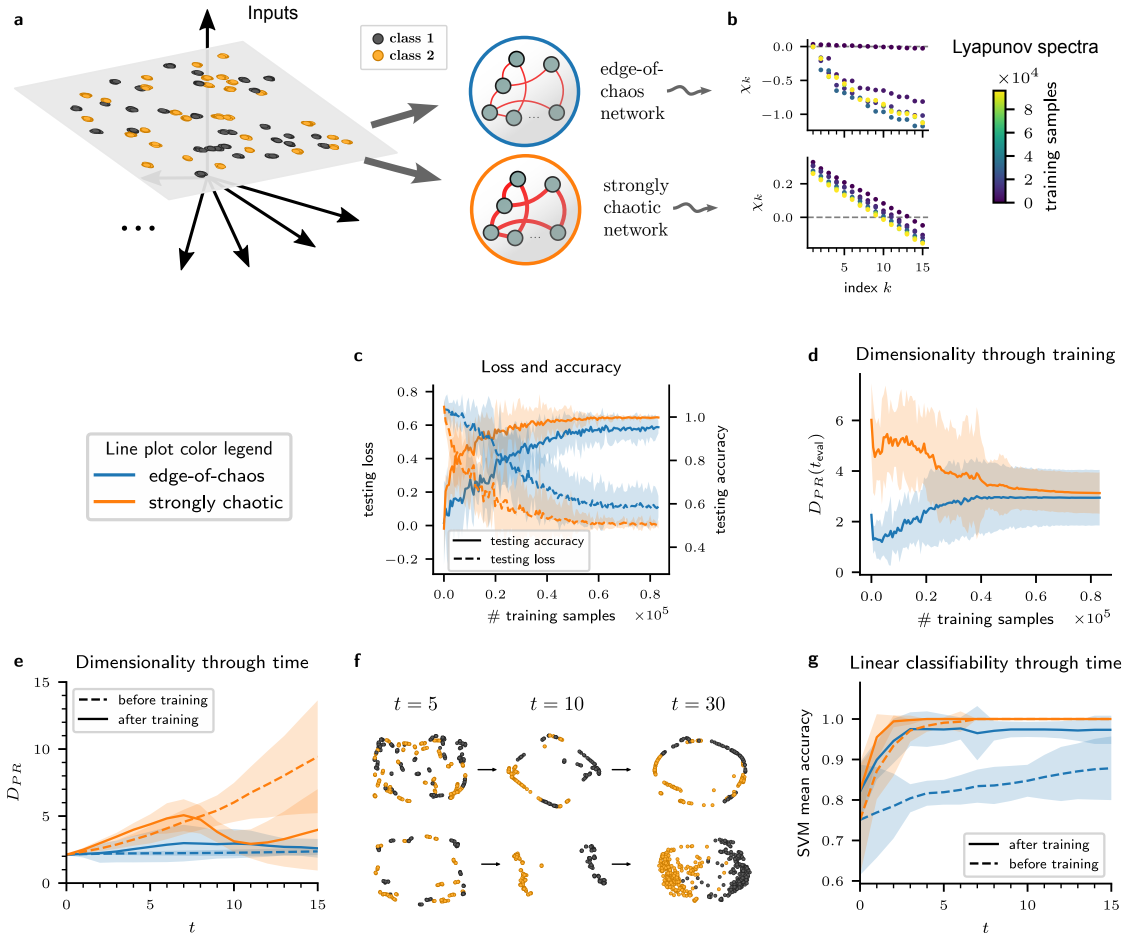

Figure 3: Comparison of edge-of-chaos and strongly chaotic networks classifying low-dimensional inputs. Details are similar to Figure 2. **a.** Inputs lie on a two-dimensional plane cutting through recurrent unit space. **b.** Lyapunov exponents. **c.** Testing loss and accuracy. **d.** Effective dimensionality measured through training. **e.** Effective dimensionality measured through time. **f.** Activations of recurrent units. The top row corresponds to the edge-of-chaos network and the bottom row to the strongly chaotic network. **g.** Mean accuracy of a support vector machine (SVM) linear classifier. The strongly chaotic network achieves linear separability before training by virtue of its intrinsic dynamics.

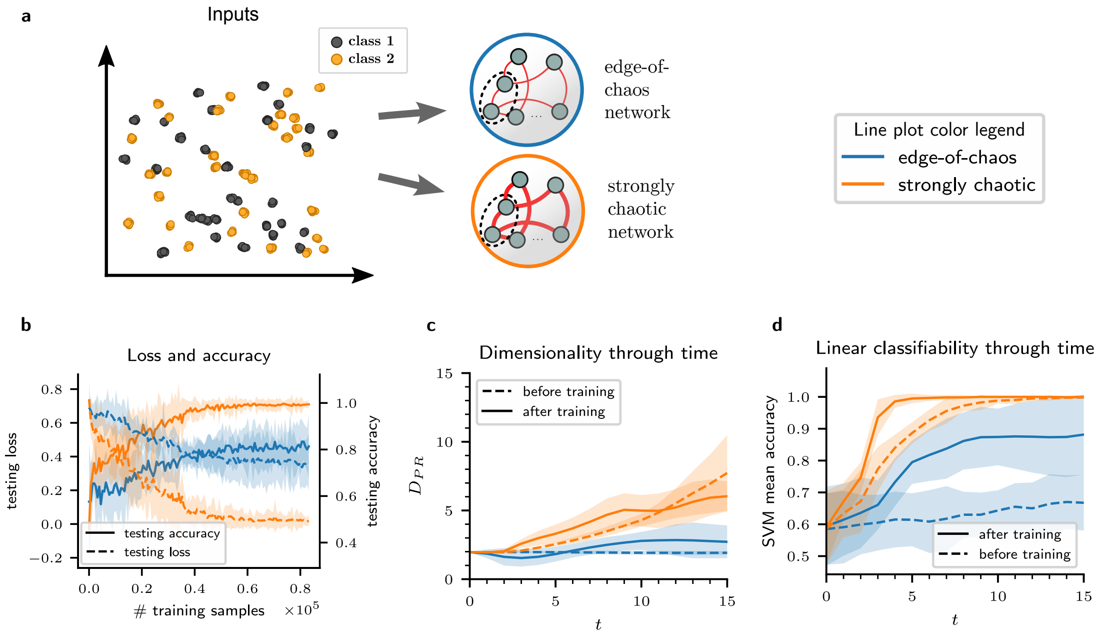

Figure 4: Comparison of edge-of-chaos and strongly chaotic networks classifying two-dimensional inputs fed to two neurons. Details are similar to Figures 2 and 3. **a.** Schematic of the task. Two-dimensional input is delivered to two randomly selected neurons in the network. **b.** Testing loss and accuracy measured through training. **c.** Effective dimensionality measured through time. **d.** Mean accuracy of a support vector machine (SVM) linear classifier.

the edge-of-chaos network. Figure 4 shows the behavior of the two networks in this scenario. We find that the strongly chaotic network is still able to solve the task near-perfectly, while the edge-of-chaos network's performance remains at about 80% on average (Figure 4b). If we look at the effective dimensionality, we find that the edge-of-chaos network is not able to learn to significantly expand the dimensionality of its representation, while the strongly chaotic network is higher-dimensional before training and learns to expand its dimensionality even further (Figure 4c). This expansion again results in good linear separability, even before training (Figure 4d). In this rather extreme case, the strongly chaotic network expands dimensionality through time without a clear compression phase.

Mechanistic explanations for compression and expansion. Our networks exhibit dimensionality compression, dimensionality expansion, or both depending on dynamical regime and the dimensionality of the inputs. While we have discussed the benefits that both phenomena can provide in the context of classification, in this section we give mechanistic explanations for why they occur. We first show how stochastic gradient descent-based learning rules can lead to weight changes that compress the dimensionality of internal unit representations (see [21] for a more thorough analysis in the case of networks trained by a particular unsupervised learning rule, and [24] for a similar flavor of compression induced by the "Pseudo-Inverse" learning rule). We then indicate how weights that lead to chaos also lead to increases in representation dimensionality in a subset of directions in neural state space.

To shed light on the compression behavior illustrated in Figure 2f – where points that belong to the same class are brought close together by the dynamics of the trained network – we show how compression can occur in a simplified scenario. In particular, we consider the case of a linear, single-hidden-layer feedforward network. The argument loosely applies to our RNN model since it can be viewed as a sequence of identical feedforward layers. It also approximately applies to networks with hyperbolic tangent activation functions in the case where the activations are small, since the hyperbolic tangent is approximately linear near 0.

In a single-layer linear network, the internal neural activations responding to a single input sample $\boldsymbol{x}$ are $\boldsymbol{h} = \boldsymbol{W}^{in}\boldsymbol{x} + \boldsymbol{b}$, and the scalar output is $o = \boldsymbol{w}^T\boldsymbol{h}$. In the case of a squared error loss function $\mathcal{L}_{\boldsymbol{W}}(\boldsymbol{x}) = \frac{1}{2}\left(o - o_{\text{target}}\right)^2$, where o_{target} is the target corresponding to the class label for $\boldsymbol{x}$, the learning updates $\boldsymbol{W}^{in} \leftarrow \boldsymbol{W}^{in} + \delta\boldsymbol{W}^{in}$ and $\boldsymbol{b} \leftarrow \boldsymbol{b} + \delta\boldsymbol{b}$ after the presentation of the single input sample $\boldsymbol{x}$ is:

$$\delta\boldsymbol{W}^{in} = -\eta\left(o - o_{\text{target}}\right)\boldsymbol{w}\boldsymbol{x}^T \tag{1}$$

$$\delta\boldsymbol{b} = -\eta\left(o - o_{\text{target}}\right)\boldsymbol{w} \tag{2}$$

Here η is the *learning rate* and $\boldsymbol{w}\boldsymbol{x}^T$ is the outer product of $\boldsymbol{w}$ and $\boldsymbol{x}$. This is one step of stochastic gradient descent with batch size 1.

Given an initial $\boldsymbol{h}$, the update rule results in a corresponding update $\boldsymbol{h} \leftarrow \boldsymbol{h} + \delta\boldsymbol{h}$. In the following, we suppose $\boldsymbol{h}$ to be a random variable and compute statistics of $\boldsymbol{h} + \delta\boldsymbol{h}$ induced by the update rule. Note that randomness of $\boldsymbol{h}$ can come from randomness in $\boldsymbol{x}$ as well as the weights $\boldsymbol{W}^{in}$ and bias $\boldsymbol{b}$.

Assuming for the moment that the bias is fixed, we note that $\delta\boldsymbol{h} = \delta\boldsymbol{W}^{in}\boldsymbol{x} \propto \boldsymbol{w}$, so that changes in the hidden representation point in the direction defined by the output weights $\boldsymbol{w}$. This is also approximately the case if the component of $\delta\boldsymbol{b}$ orthogonal to $\boldsymbol{w}$ is small. If we first assume that output weights $\boldsymbol{w}$ are fixed, the update rule can modulate the response, but only along this single dimension. This modulation can be expected to cause compression, as learning converges when $o_{\text{target}} = \boldsymbol{w}^T\boldsymbol{h}$, implying that $\boldsymbol{h}$ has minimized its variability along $\boldsymbol{w}$. To see reduction of variability in all dimensions like that exhibited by Figure 2f, rather than only along a single dimension, we need to consider how the direction of compression can change across learning. We turn to this next.

One natural factor that could drive compression of the hidden layer representation in multiple directions is changes in the output weights $\boldsymbol{w}$. In practice – including in our simulations above – the variability underlying these changes can have many different sources. One is that the output weights do evolve across stages of learning, being simultaneously trained under random batches of inputs via stochastic gradient descent. A second, more complicated effect is that the linearization taken in writing Equation (1) is in practice taken around different points in the hidden unit state space; this introduces fluctuations in the approximation above that could also change the “effective” directions in which hidden units are read out. Here, we take a simple approach to modeling these sources of variability by assuming that $\boldsymbol{w}$ is a normally distributed random variable with mean $\langle\boldsymbol{w}\rangle = \mu_w\boldsymbol{e}_1$ and covariance $\boldsymbol{C}_{\boldsymbol{w}} = \sigma_w^2\boldsymbol{I}$ (the choice of mean direction doesn't result in loss of generality, since we can rotate our coordinate system). We further assume that (1) $\boldsymbol{C}_{\boldsymbol{h}}$ is diagonal, (2) $\boldsymbol{h}$ is independent of $\boldsymbol{w}$, and (3) the input has constant norm, $\|\boldsymbol{x}\| = \text{const}$. With these assumptions, a straightforward calculation (see Supplementary Material) shows that $\boldsymbol{C}_{\boldsymbol{h}+\delta\boldsymbol{h}}$ is a diagonal matrix up to order $O(\eta^2)$, with diagonal entries

$$\text{var}\left(h_k + \delta h_k\right) = \alpha_k\,\text{var}(h_k) + O(\eta^2) \tag{3}$$

where the α_k are scalars less than one. Here k ranges from 1 to the number of hidden units.

Taken together, these equations reveal that the variability of $\boldsymbol{h}$ is reduced in all directions by the gradient descent weight update. Since $\boldsymbol{C}_{\boldsymbol{h}+\delta\boldsymbol{h}}$ is again approximately diagonal, the same argument can be used for subsequent steps of gradient descent, suggesting that continuing the gradient descent procedure also continues to compress $\boldsymbol{h}$ to smaller variability in all directions (at least until our independence assumption between the hidden representation and w breaks down). This analysis also indicates that the rate of compression depends on the learning rate, which we have observed (data not shown). See Figure 6 in Supplementary Material for a visualization of compression driven by variability in $\boldsymbol{w}$.

While far from a rigorous proof, this analysis gives intuition for how stochastic gradient descent can lead to representations in which each class of input is mapped into a tightly clustered set. This, in turn, gives rise to an overall dimension for the representation that is determined by the number of classes.

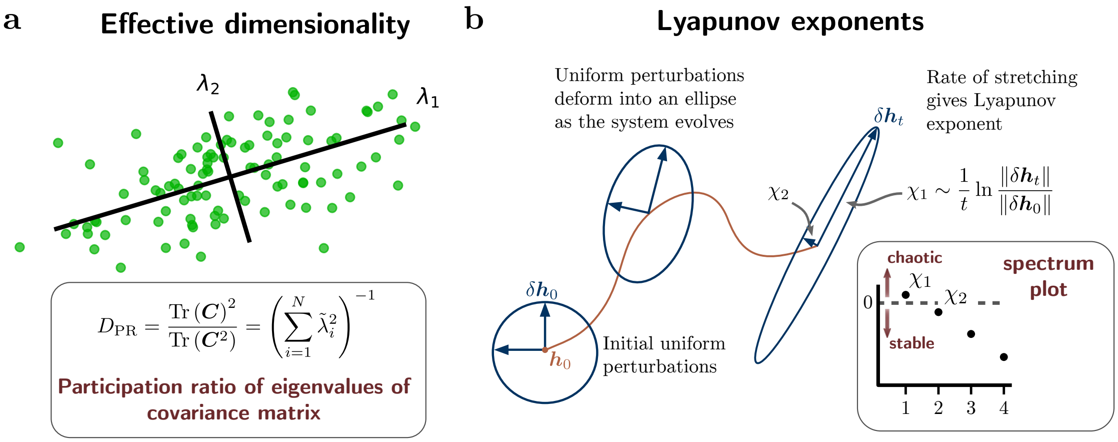
a
Effective dimensionality
λ_2
λ_1
$D_{\mathrm{PR}} = \frac{\mathrm{Tr}\,(\boldsymbol{C})^2}{\mathrm{Tr}\,(\boldsymbol{C}^2)} = \left(\sum_{i=1}^{N} \tilde{\lambda}_i^2\right)^{-1}$
Participation ratio of eigenvalues of covariance matrix
b
Lyapunov exponents
Uniform perturbations deform into an ellipse as the system evolves
$\delta \boldsymbol{h}_t$
Rate of stretching gives Lyapunov exponent
χ_2
$\chi_1 \sim \frac{1}{t} \ln \frac{\|\delta \boldsymbol{h}_t\|}{\|\delta \boldsymbol{h}_0\|}$
$\delta \boldsymbol{h}_0$
$\boldsymbol{h}_0$
Initial uniform perturbations
chaotic
χ_1
χ_2
spectrum plot
0
stable
1
2
3
4

Figure 5: Two metrics used to characterize network behavior. **a.** The PR Dimensionality of a set of points measures how many axes are needed to capture most of the variance of the points. The quantity is derived from the eigenvalues of the covariance matrix of the points. The quantity is maximized when all eigenvalues are equal, and minimized when one eigenvalue is much larger than the rest. **b.** The Lyapunov exponents of a trajectory measure the rate at which neighboring trajectories diverge/converge to the reference trajectory. Having a positive exponent indicates chaotic dynamics. To measure chaos, we plot a subset of the largest Lyapunov exponents from biggest to smallest.

We now turn our attention to a mechanism of dimensionality expansion: chaos. Experimentally we have seen that strongly chaotic networks naturally transform low-dimensional inputs into a higher-dimensional representation (Figures 3e and 4c). This can be understood through the definition of Lyapunov exponents (see Methods and Figure 5b). Positive exponents indicate the existence of directions in which nearby dynamical trajectories are pulled apart in time. This persistent pulling apart results in an attractor that has a nonzero “volume” in some subspace: all of the trajectories that collapse to this attractor do not collapse toward the same point, nor a single orbit, and so occupy a nonzero extent of space. This volume is constrained by the negative exponents, which indicate directions in which trajectories are being pulled together. Overall, chaotic attractors also have complex geometry, forming fractal patterns in some directions, and smooth manifolds in others, the interplay of which is a result of mixed expansion and contraction of nearby trajectories (measured by positive and negative Lyapunov exponents, respectively). See [19] for details of how this volume is measured, and [27, 40, 14] for investigations of this measure in recurrent networks. If the effective dimensionality of the representation is dominated by a single attractor, then we can expect the effective dimensionality to increase as the number of expansion directions (i.e. the number of positive Lyapunov exponents) increases. This assumption can be violated when there is more than one separate attractor; for instance, different inputs may lead to distinct attractors and thus jointly occupy some subspace of nonzero effective dimension, even if their respective attractors are stable fixed points (which individually have zero volume).

In the case of random (untrained), undriven chaotic networks, one expects that dynamics generally accumulate on a single, connected attractor. The network dynamics return to this attractor after being perturbed by transient inputs. Hence, a strongly chaotic network that is initialized with low-dimensional inputs can be expected to produce dimensionality expansion, as the trajectories return to the higher-dimensional intrinsic chaotic attractor. Note that compression due to negative Lyapunov exponents typically constrains the dimension of the chaotic attractor to be significantly smaller than N [27, 14].

Conclusions and Discussion

RNN models have rich computational capacity, including the ability to expand and compress dimensionality in order solve a classification task with efficient, compact representations. This behavior as it evolves through learning depends on the initial dynamics of the network. Here we study networks that are initialized at the edge of chaos and in the strongly chaotic regime.

We find that both networks compress high-dimensional inputs – i.e., those that are initially linearly separable – into two compact regions, one for each class. We suggest, through mathematical reasoning based on a simple linear case, that stochastic gradient descent itself naturally promotes the observed compression of points belonging to the same class.

The compressed representations can arise either from the formation of stable fixed points (for networks initially at the edge of chaos) or chaotic attractors (for networks in the strongly chaotic regime). This has implications for coding: the edge-of-chaos networks learn to solve the task for all time while the strongly chaotic networks slowly "reset" after the evaluation period. The former may lend itself to long-term memory storage; the latter could be useful in flexibly learning new tasks.

In the case of low-dimensional (non-linearly separable) inputs, only the strongly chaotic network is able to reliably learn the task. This corresponds to the network expanding the dimensionality of its representation both before and after training, an attribute not shared by the edge-of-chaos network. As predicted by fundamental ideas in classification [9], higher-dimensional representations are more likely to permit a hyperplane that separates classes. This feature of dimensionality expansion has been explored in feed-forward models such as kernel learning machines [46] and models of olfactory, cerebellar, and visual circuits [3, 39, 2, 35, 7, 49], as well as reservoir computing models where recurrent weights are random and untrained [31].

On the other hand, these studies have also pointed out the need to constrain dimensionality expansion to enable generalization [31, 49]. In the cases of both high and low-dimensional inputs studied here, sufficiently chaotic RNNs trained by stochastic gradient descent appear to satisfy these two desiderata, producing representations that are both separable and relatively compressed. In the most striking example, strongly chaotic networks trained to classify distributed two-dimensional inputs evolve through time to initially expand, and then to compress dimensionality. In the extreme case of spatially localized inputs constrained to two neurons, the networks typically only expand dimensionality, without subsequently compressing it. However, this expansion is still small relative to the size of the network. In all of these cases, strongly chaotic networks produce ongoing variability, a property shared by more biological models of neural circuits [50, 34, 45, 29] as well as experimental recordings [48, 36].

While these findings are interesting, our study has limitations in its scope and analysis. Foremost is that we consider only a single, extremely simple task (see [10] for important progress in understanding the dimensionality of RNNs trained on a more complicated task). A clear need in future work is also to consider a wider range of task and model parameters: for example, input dimension between 2 and N, and to consider higher-dimensional outputs specified by more than two class labels. For a start, in preliminary work we have found that in the case of high-dimensional inputs, the number of class labels strongly modulates the readout dimensionality of the RNN, while the number of input clusters does not (see Figure 7 in Supplementary Material). In addition, our theoretical arguments for compression induced by stochastic gradient descent, while providing some intuition, are highly simplified. It remains to extend these or other arguments to the full nonlinear, recurrent network trained over many samples from multiple classes, and to prove explicitly that this compression results in a reduction of the dimensionality of the network response to an ensemble of inputs. This could build on the thorough analysis in the case of deep networks trained by a particular unsupervised learning rule performed by [21]. It is also worth noting that other mechanisms for dimensionality expansion exist for recurrent networks and would be interesting to explore in future works. Examples include training the network with an explicit dimensionality term in the loss function or implicitly via a highly variable "target" network as in [26, 12]. Indeed, we have used only a single, basic type of "vanilla RNN" network model, and extensions toward more complex models is important if we are to generalize our findings to other machine learning settings, and to make more confident predictions about the brain's circuits.

Taken together, we find that chaos is a flexible (allowing for classification in all regimes) as well as functional (enabling classification of low-dimensional inputs) mechanism for RNNs learning to perform classification. The observation that stochastic gradient descent naturally promotes dimensionality compression sheds light on its surprising success in generalizing well (see [25, 1, 32] for other approaches to explaining this). These findings also lend support for the hypothesis that low-dimensional representations in the brain arise naturally from synaptic modifications driven by learning, and highlight a scenario where chaotic variability and salient low-dimensional structure coexist (see [27] for a more detailed discussion of this phenomenon).

Methods

Model and Task. The model is a recurrent neural network RNN with one hidden layer trained to perform a delayed classification task (Figure 1a). The primary observables of interest are the activations $\boldsymbol{h}_t$ of the hidden layer units at each time step t as the network solves the task presented to it. The equation for $\boldsymbol{h}_t$ is

$$\boldsymbol{h}_t = \phi\left(\boldsymbol{W}^{rec}\boldsymbol{h}_{t-1} + \boldsymbol{x}_t + \boldsymbol{b}^{in}\right) \tag{4}$$

where $\boldsymbol{W}^{rec}$ is the matrix of recurrent connection weights, $\boldsymbol{x}_t$ is the input, and $\boldsymbol{b}^{in}$ is a bias term. The nonlinearity ϕ is taken to be $\phi = \tanh$. The network is initialized to have zero activation, $\boldsymbol{h}_{-1} = \boldsymbol{0}$. We write N as the number of hidden units, so for fixed t we have $\boldsymbol{h}_t \in \mathbb{R}^N$.

The output of the network is

$$\boldsymbol{o}_t = \boldsymbol{W}^{out}\boldsymbol{h}_t + \boldsymbol{b}^{out}.$$

The output $\boldsymbol{o} = \boldsymbol{o}_{t_{\text{eval}}}$ at the evaluation timestep is passed to a categorical cross-entropy loss function to produce a scalar loss signal to be minimized. The equation for this loss is $\text{CCE}(\boldsymbol{o}, k) = \log(p_k)$ where $p_k = \exp(o_k) / \left(\sum_j \exp(o_j)\right)$ represents the probability that $\boldsymbol{o}$ indicates class k. This loss is used to update the recurrent weights $\boldsymbol{W}^{rec}$ and the output weights $\boldsymbol{W}^{out}$ via backpropagation-through-time using the pytorch 0.4.1 implementation of the RMSProp optimization scheme with a batch size of 10 and learning rate of .001 (loss is summed over the batch). We reduce the learning rate through the training process via a reduce-on-plateau strategy. When the loss fails to decrease by more than $1e-7$ for five epochs in a row, the learning rate is halved. This conservative but fairly standard "reduce only when necessary" approach to the learning rate is meant to help ensure that the quantities we measure over training plateau for reasons other than a vanishing learning rate, while still allowing the network to find the best solution it is able to. The trends and measures analyzed are qualitatively robust to small changes in the learning parameters. Perhaps the most sensitive parameter is the learning rate, which directly impacts the strength of the network's compression and the degree to which Lyapunov exponents are pushed negative. In some cases, halving the learning rate is enough to make the compression phenomenon in Figure 3e considerably fainter, while doubling it can have a strong effect in inhibiting the dimensionality expansion. Considering the learning rate's primary role in our estimated equation for compression, this sensitivity is not surprising. We found our choice of learning rate to roughly maximize the testing accuracy of the networks.

The network receives a static input for a number of timesteps (the input period), and is then asked to classify this input after a delay period. In our simulations, the input period is 1 timestep and the delay period is 9 timesteps, after which comes an evaluation period of 1 timestep. Details about the input, delay, and evaluation periods are described in the main text.

The inputs consist of isotropic gaussian clusters distributed randomly through an ambient space of dimension d, where each cluster's covariance is the $d \times d$ matrix $\sigma^2\boldsymbol{I}$. The means of the clusters are distributed uniformly at random in a d-dimensional hypercube of side length $\ell = 4$ centered at the origin, with a minimum separation s between means being enforced to ensure that clusters do not overlap. Here ℓ is chosen so that the average magnitude of the scalar elements of all the input samples is ~ 1 (we found this value to roughly maximize the performance of both the edge-of-chaos and strongly chaotic networks). The minimum separation s is chosen so that all points belonging to a cluster fall within a distance s of the mean with a confidence of 99.9999% (i.e. the clusters are non-overlapping with probability close to 1). To form an input sample $\boldsymbol{x}$, we first select a center

$\boldsymbol{c} \in \mathcal{C}$ and draw $\boldsymbol{x}$ from the isotropic gaussian distribution centered at $\boldsymbol{c}$ (which is contained in the ambient space of the input). This is commonly expressed by the notation $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{c}, \sigma^2 \boldsymbol{I})$. In our case, we choose a standard deviation of $\sigma = 0.02$. To embed these inputs in the N-dimensional space of the recurrent units, we multiply the inputs by an orthogonal transformation. In our simulations, we consider the case of 60 clusters, $\#\mathcal{C} = 60$. Each cluster center is randomly associated with one of two class labels (30 clusters for each label), and the training samples drawn from this cluster are assigned this label. Over training, new samples are always drawn, so the network never sees the same training example twice. Test inputs are drawn from the same distribution and have not been seen by the network during training. Our results are not sensitive to small changes in the above parameters.

The (intrinsic) dynamics of the network depend primarily on $\boldsymbol{W}^{rec}$. Strong random coupling between recurrent units leads to chaotic dynamics, whereas weak random coupling results in dynamics that converge onto stable attractors (for large networks these attractors are typically fixed points; see the seminal work by Sompolinsky, Crisanti, and Sommers [47] for an analysis of a rate model network with similar properties). We investigate these dynamical regimes by initializing $\boldsymbol{W}^{rec}$ as $\boldsymbol{W}^{rec} = (1-\varepsilon)\boldsymbol{I} + \varepsilon \boldsymbol{J}$ where $\varepsilon = .01$ sets the timescale of the dynamics to be slow (so that the discrete update Equation (4) produces visually smooth trajectories). The matrix $\boldsymbol{J}$ has normally distributed entries that scale in magnitude with a coupling strength parameter γ, $J_{ij} \sim \mathcal{N}(0, \gamma^2/N)$. The first regime we consider is at the "edge of chaos" with $\gamma = 20$, where the network is weakly chaotic before training and dynamically stable after training. The second regime we consider is "strongly chaotic" with $\gamma = 250$, where the dynamics are chaotic both before and after training. While our equations are not identical to those analyzed in [47], we find that the trajectories produced and qualitative properties are similar. In particular, the top Lyapunov exponent of our system grows with the gain γ. In terms of the top Lyapunov exponent, a value of $\gamma = 20$ in our model roughly corresponds to a gain of 2 in the Sompolinsky rate model, while $\gamma = 250$ corresponds to a gain of about 5. The second regime we consider is "strongly chaotic" with $\gamma = 250$, where the dynamics are chaotic both before and after training.

The input is transformed by a random $N \times d$ orthogonal transformation, where each column has norm $0.4/\sqrt{d}$ (this transformation preserves the geometry of the inputs). The one exception is input given to two neurons in the network, where the input is transformed by a matrix with 1 in the $(1, 1)$ and $(2, 2)$ position, and zero everywhere else. The output weights are initialized as a random matrix $\boldsymbol{J'}$ where $J'_{ij} \sim \mathcal{N}(0, .4^2/N)$.

To plot snapshots in time (e.g. Figure 2f), we compute, independently at each time point, the principal component vectors of the network's responses to the inputs. This means we have a new set of principal component vectors at each time point, and each of these principal vectors is determined only up to sign. To align the points from one time point to the next, we take the signs of the principal vectors that maximize the overlaps between the two time points.

Measures. The degree of chaos present in the network is measured through numerical estimates of *Lyapunov exponents.* For a dynamical map $\boldsymbol{h}_{t+1} = f(\boldsymbol{h}_t)$, the Lyapunov exponents χ are defined as

$$\chi(\boldsymbol{h}_0, \delta\boldsymbol{h}_0) = \lim_{t\to\infty} \frac{1}{t} \ln \frac{\|\delta\boldsymbol{h}_{t+1}\|}{\|\delta\boldsymbol{h}_0\|} = \lim_{t\to\infty} \frac{1}{t} \ln \|f'(\boldsymbol{h}_t) \cdots f'(\boldsymbol{h}_0)\, \delta\boldsymbol{h}_0\| .$$

where $\boldsymbol{h}_0$ is the initial condition for a trajectory and $\delta\boldsymbol{h}_0$ is a perturbation of $\boldsymbol{h}_0$ in a particular direction. The vector $\delta\boldsymbol{h}_{t+1}$ arises from evolving the perturbation $\delta\boldsymbol{h}_0$ forward in time. The equation measures the rate of expansion/contraction of the neighboring trajectory induced by $\delta\boldsymbol{h}_0$ as the dynamics run forward (Figure 5b). For a particular trajectory with initial condition $\boldsymbol{h}_0$, there are N Lyapunov exponents (since the system is N-dimensional), depending on choice of $\delta\boldsymbol{h}_0$. The largest exponent χ_1 has particular importance, as $\chi_1 > 0$ indicates chaotic dynamics, and $\chi_1 < 0$ indicates contraction to stable fixed points or stable limit cycles. Since different initial conditions can converge onto different attractors, the Lyapunov exponents can differ with choice of $\boldsymbol{h}_0$. In practice, we find these differences to be small. When we report exponents, we show standard error of the mean over ten randomly chosen initial conditions $\boldsymbol{h}_0$. We use the discrete $\boldsymbol{QR}$ method to numerically compute the exponents [13, 17].

To characterize the inputs, as well as the network responses to these inputs, we measure their *effective dimensionalities* (see [23] for an introduction of the measure in physics, and [43] for the first use in computational neuroscience). The equation for the effective dimensionality of a set of S points $\boldsymbol{V} = [v_{si}] \in \mathbb{R}^{S\times d}$ with ambient dimension d is

$$D_{\text{PR}}\left(\boldsymbol{V}\right) = \frac{\text{Tr}\left(\boldsymbol{C}\right)^2}{\text{Tr}\left(\boldsymbol{C}^2\right)} = \left(\sum_{i=1}^{N} \tilde{\lambda}_i^2\right)^{-1} \tag{5}$$

where $C_{ij} = \left\langle v_{si}v_{sj}\right\rangle_s - \left\langle v_{si}\right\rangle_s \left\langle v_{sj}\right\rangle_s$ is the covariance matrix of $\boldsymbol{V}$ and the $\tilde{\lambda}_i = \lambda_i / \sum_j \lambda_j$ are the normalized eigenvalues of $\boldsymbol{C}$. A visual intuition for this quantity is shown in Figure 5. To measure the effective dimensionality of the network representation, at each time point t we compute Equation (5) with covariance matrix $C_{ij}(t) = \left\langle h_i(t)h_j(t)\right\rangle_s - \left\langle h_i(t)\right\rangle_s \left\langle h_j(t)\right\rangle_s$ where $\left\langle\cdot\right\rangle_s$ denotes an average over input samples. Here we have written $\boldsymbol{h}(t)$ instead of $\boldsymbol{h}_t$ to avoid confusion with the other index of $\boldsymbol{h}$. Note that this quantity is time-dependent, i.e. we get a covariance matrix $\boldsymbol{C} = \boldsymbol{C}\left(t\right)$ for each time point t and corresponding participation ratio $D_{PR} = D_{PR}\left(t\right)$.

Acknowledgements

MF acknowledges the support of the National Science Foundation Graduate Research Fellowship under Grant No. DGE-1256082. GL is funded by an NSERC Discovery Grant (RGPIN-2018-04821), an FRQNT Young Investigator Startup Program (2019-NC-253251), and an FRQS Research Scholar Award, Junior 1 (LAJGU0401-253188). ESB acknowledges the support of NSF DMS Grant 1514743. We thank Merav Stern, Dmitri Chklovskii, and Ali Weber for their insights and suggestions. MF would also like to thank Haim Sompolinsky and SueYeon Chung for their mentorship and inspiration.

References

[1] Madhu S. Advani and Andrew M. Saxe. High-dimensional dynamics of generalization error in neural networks. *arXiv:1710.03667 [physics, q-bio, stat]*, October 2017. arXiv: 1710.03667.

[2] James S. Albus. A theory of cerebellar function. *Mathematical Biosciences*, 10(1):25–61, February 1971.

[3] Baktash Babadi and Haim Sompolinsky. Sparseness and Expansion in Sensory Representations. *Neuron*, 83(5):1213–1226, September 2014.

[4] Andrea Banino, Caswell Barry, Benigno Uria, Charles Blundell, Timothy Lillicrap, Piotr Mirowski, Alexander Pritzel, Martin J. Chadwick, Thomas Degris, Joseph Modayil, Greg Wayne, Hubert Soyer, Fabio Viola, Brian Zhang, Ross Goroshin, Neil Rabinowitz, Razvan Pascanu, Charlie Beattie, Stig Petersen, Amir Sadik, Stephen Gaffney, Helen King, Koray Kavukcuoglu, Demis Hassabis, Raia Hadsell, and Dharshan Kumaran. Vector-based navigation using grid-like representations in artificial agents. *Nature*, 557(7705):429, May 2018.

[5] Omri Barak. Recurrent neural networks as versatile tools of neuroscience research. *Current Opinion in Neurobiology*, 46:1–6, 2017. Publisher: Elsevier Ltd.

[6] Y. Bengio, A. Courville, and P. Vincent. Representation Learning: A Review and New Perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, August 2013.

[7] N. Alex Cayco-Gajic, Claudia Clopath, and R. Angus Silver. Sparse synaptic connectivity is required for decorrelation and pattern separation in feedforward networks. *Nature Communications*, 8(1):1116, October 2017.

[8] Vladimir Cherkassky and Filip M. Mulier. *Learning from Data: Concepts, Theory, and Methods*. Wiley-IEEE Press, Hoboken, N.J, 2 edition edition, August 2007.

[9] Thomas M. Cover. Geometrical and Statistical Properties of Systems of Linear Inequalities with Applications in Pattern Recognition. *IEEE Transactions on Electronic Computers*, EC-14(3):326–334, June 1965.

[10] Christopher J. Cueva, Encarni Marcos, Alex Saez, Aldo Genovesio, Mehrdad Jazayeri, Ranulfo Romo, C. Daniel Salzman, Michael N. Shadlen, and Stefano Fusi. Low dimensional dynamics for working memory and time encoding. *bioRxiv*, page 504936, January 2019.

[11] John P. Cunningham and Byron M. Yu. Dimensionality reduction for large-scale neural recordings. *Nature Neuroscience*, 17(11):1500–1509, November 2014.

[12] Brian DePasquale, Christopher J. Cueva, Kanaka Rajan, G. Sean Escola, and L. F. Abbott. full-FORCE: A Target-Based Method for Training Recurrent Networks. pages 1–20, 2017. arXiv: 1710.03070.

[13] J. P. Eckmann and D. Ruelle. Ergodic theory of chaos and strange attractors. *Reviews of Modern Physics*, 57(3):617–656, July 1985.

[14] Rainer Engelken and Fred Wolf. Dimensionality and entropy of spontaneous and evoked rate activity. *Bulletin of the American Physical Society*, 2017. Publisher: American Physical Society.

[15] Stefano Fusi, Earl K Miller, and Mattia Rigotti. Why neurons mix: high dimensionality for higher cognition. *Current Opinion in Neurobiology*, 37:66–74, April 2016.

[16] Peiran Gao, Eric Trautmann, Byron M Yu, Gopal Santhanam, Stephen Ryu, Krishna Shenoy, and Surya Ganguli. A theory of multineuronal dimensionality, dynamics and measurement. page 214262, 2017.

[17] Karlheinz Geist, Ulrich Parlitz, and Werner Lauterborn. Comparison of Different Methods for Computing Lyapunov Exponents. *Progress of Theoretical Physics*, 83(5):875–893, 1990.

[18] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[19] Peter Grassberger and Itamar Procaccia. Measuring the strangeness of strange attractors. *Physica D: Nonlinear Phenomena*, 9(1):189–208, October 1983.

[20] Alex Graves. *Supervised Sequence Labelling with Recurrent Neural Networks*. Studies in Computational Intelligence. Springer-Verlag, Berlin Heidelberg, 2012.

[21] Haiping Huang. Mechanisms of dimensionality reduction and decorrelation in deep neural networks. *Physical Review E*, 98(6), December 2018.

[22] Herbert Jaeger. The" echo state" approach to analysing and training recurrent neural networks-with an erratum note'. *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, 148, 2001.

[23] R. J.Bell and P. Dean. Atomic vibrations in vitreous silica. *Discussions of the Faraday Society*, 50(0):55–61, 1970.

[24] Jonathan Kadmon and Haim Sompolinsky. Optimal Architectures in a Solvable Model of Deep Networks. *Advances in Neural Information Processing Systems 29*, pages 4781–4789, 2016.

[25] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima. *ICLR*, page 16, 2017.

[26] Rodrigo Laje and Dean V Buonomano. Robust timing and motor patterns by taming chaos in recurrent neural networks. *Nature neuroscience*, 16(7):925–33, 2013. arXiv: NIHMS150003 Publisher: Nature Publishing Group ISBN: 1097-6256.

[27] Guillaume Lajoie, Kevin Lin, and Eric Shea-Brown. Chaos and reliability in balanced spiking networks with temporal drive. *Phys Rev E Stat Nonlin Soft Matter Phys*, 87(5):2432–2437, 2013.

[28] Guillaume Lajoie, Kevin K. Lin, Jean-Philippe Thivierge, and Eric Shea-Brown. Encoding in Balanced Networks: Revisiting Spike Patterns and Chaos in Stimulus-Driven Systems. *PLOS Computational Biology*, 12(12):e1005258, 2016.

[29] Itamar Daniel Landau and Haim Sompolinsky. Coherent chaos in a recurrent neural network with structured connectivity. *bioRxiv*, page 350801, October 2018.

[30] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, May 2015.

[31] Robert Legenstein and Wolfgang Maass. Edge of chaos and prediction of computational performance for neural circuit models. *Neural Networks*, 20(3):323–334, 2007. ISBN: 0893-6080.

[32] Yuanzhi Li and Yingyu Liang. Learning Overparameterized Neural Networks via Stochastic Gradient Descent on Structured Data. *arXiv:1808.01204 [cs, stat]*, August 2018. arXiv: 1808.01204.

[33] Zachary C. Lipton, John Berkowitz, and Charles Elkan. A Critical Review of Recurrent Neural Networks for Sequence Learning. *arXiv:1506.00019 [cs]*, May 2015. arXiv: 1506.00019.

[34] Ashok Litwin-Kumar and Brent Doiron. Slow dynamics and high variability in balanced cortical networks with clustered connections. *Nature Neuroscience*, 15(11):1498–1505, November 2012.

[35] Ashok Litwin-Kumar, Kameron Decker Harris, Richard Axel, Haim Sompolinsky, and L. F. Abbott. Optimal Degrees of Synaptic Connectivity. *Neuron*, 93(5):1153–1164.e7, March 2017.

[36] M. London, A. Roth, L. Beeren, M. Husser, and P. E. Latham. Sensitivity to perturbations in vivo implies high noise and suggests rate coding in cortex. *Nature*, 466(7302):123–7, 2010. ISBN: 1476-4687.

[37] Wolfgang Maass, Thomas Natschlger, and Henry Markram. Real-Time Computing Without Stable States: A New Framework for Neural Computation Based on Perturbations. *Neural Computation*, 14(11):2531–2560, November 2002. Publisher: MIT Press 238 Main St., Suite 500, Cambridge, MA 02142-1046 USA journals-info@mit.edu.

[38] Valerio Mante, David Sussillo, Krishna V. Shenoy, and William T. Newsome. Context-dependent computation by recurrent dynamics in prefrontal cortex. *Nature*, 503(7474):78–84, November 2013.

[39] David Marr. A theory of cerebellar cortex. *The Journal of Physiology*, 202(2):437–470.1, June 1969.

[40] Michael Monteforte and Fred Wolf. Dynamical entropy production in spiking neuron networks in the balanced state. *Physical Review Letters*, 105(26), 2010. arXiv: 1003.4410 ISBN: 0031-9007\r1079-7114.

[41] A. Emin Orhan and Wei Ji Ma. A diverse range of factors affect the nature of neural representations underlying short-term memory. *Nature Neuroscience*, 22(2):275, February 2019.

[42] Rich Pang, Benjamin J. Lansdell, and Adrienne L. Fairhall. Dimensionality reduction in neuroscience. *Current Biology*, 26(14):R656–R660, July 2016.

[43] Kanaka Rajan, Larry Abbot, and Haim Sompolinsky. Inferring Stimulus Selectivity from the Spatial Structure of Neural Network Dynamics. *Advances in Neural Information Processing Systems 23*, 23:1–9, 2010.

[44] Mattia Rigotti, Omri Barak, Melissa R. Warden, Xiao-Jing Wang, Nathaniel D. Daw, Earl K. Miller, and Stefano Fusi. The importance of mixed selectivity in complex cognitive tasks. *Nature*, 497(7451):585–590, 2013.

[45] Robert Rosenbaum, Matthew A Smith, Adam Kohn, Jonathan E Rubin, and Brent Doiron. The spatial structure of correlated neuronal variability. *Nature Neuroscience*, (October):1–35, 2016. ISBN: 1546-1726 (Electronic)\r1097-6256 (Linking).

[46] B. Schlkopf and AJ. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Adaptive Computation and Machine Learning. Biologische Kybernetik, Cambridge, MA, USA, December 2002.

[47] H. Sompolinsky, A. Crisanti, and H. J. Sommers. Chaos in random neural networks. *Physical Review Letters*, 61(3):259–262, 1988. arXiv: 1011.1669v3 ISBN: 1079-7114 (Electronic)\r0031-9007 (Linking).

[48] C. J. Stam. Nonlinear dynamical analysis of EEG and MEG: Review of an emerging field. *Clinical Neurophysiology*, 116(10):2266–2301, October 2005.

[49] Carsen Stringer, Marius Pachitariu, Nicholas Steinmetz, Matteo Carandini, and Kenneth D. Harris. High-dimensional geometry of population responses in visual cortex. *bioRxiv*, page 374090, August 2018.

[50] C van Vreeswijk and H Sompolinsky. Chaotic balanced state in a model of cortical circuits. *Neural computation*, 10(6):1321–71, August 1998.

[51] Vladimir N. Vapnik. *Statistical Learning Theory*. Wiley-Interscience, New York, 1 edition edition, September 1998.

[52] Vladimir N. Vapnik. *The Nature of Statistical Learning Theory*. Springer New York, New York, NY, 2000.

[53] Jing Wang, Devika Narain, Eghbal A. Hosseini, and Mehrdad Jazayeri. Flexible timing by temporal scaling of cortical responses. *Nature Neuroscience*, 21(1):102, January 2018.

[54] Fred Wolf, Rainer Engelken, Maximilian Puelma-Touzel, Juan Daniel Flrez Weidinger, and Andreas Neef. Dynamical models of cortical circuits. *Current Opinion in Neurobiology*, 25:228–236, April 2014.

Supplementary material

Additional figures.

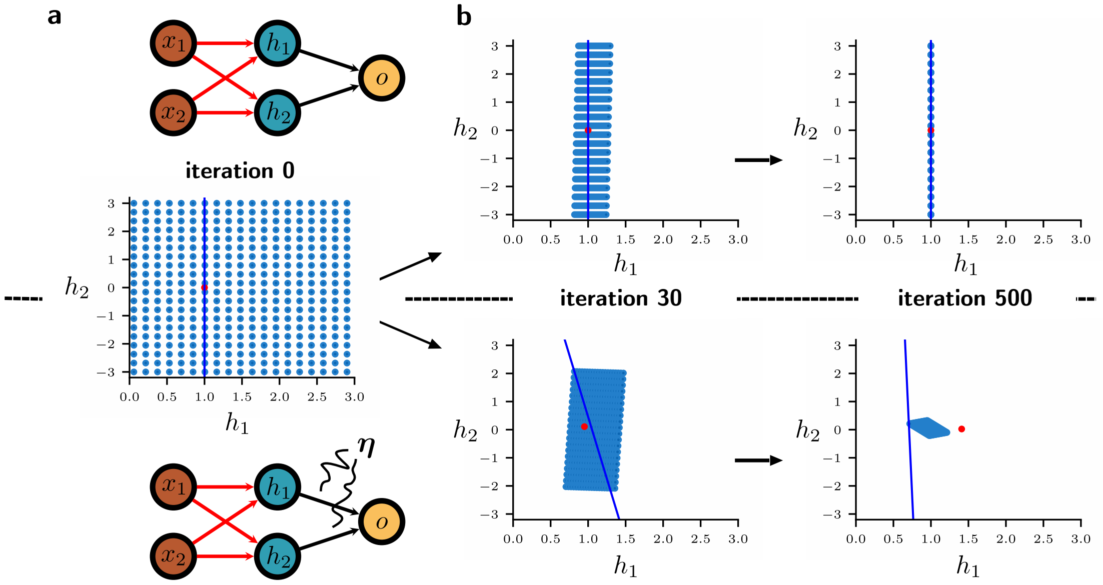

Figure 6: Example of noise in output weights driving compression of the hidden representation in a linear network with two hidden layer units. The equation for the network is $\boldsymbol{h} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$ with output $o = \boldsymbol{w}^T\boldsymbol{h}$. The input weights (red) are initialized to the 2×2 identity matrix, and bias is initialized as $(1, 0)$. The inputs are placed on a grid from $x = -1$ to $x = 2$ and from $y = -3$ to $y = 3$ (not shown). Network output o is trained to minimize the squared error loss $0.5(o-1)^2$. Input samples are chosen randomly, and input weights are updated via stochastic gradient descent with batch size 1. **a.** Top: Diagram of network where input weights are trained and output weights are fixed. Bottom: Diagram of network where input weights are trained and output weights are normally distributed with mean $(1, 0)$ and covariance $0.05\boldsymbol{I}$. In the figure, $\boldsymbol{\eta}$ represents additive white noise. Middle: hidden unit responses (blue circles) to the inputs before training (iteration 0). Red dot denotes the output weight vector, and the blue line is the affine subspace of points that w maps to 1. **b.** Evolution of the hidden layer response to inputs (representation) as input weights are trained. Top: Representation of the network where output weights are fixed. The iteration number denotes the number of training samples that have been used to update the weights. Activations compress to the space orthogonal to $\boldsymbol{w}$, shifted by $(1, 0)$. Bottom: Representation of the network where output weights are randomly drawn at every input sample presentation. Activations compress to a compact, localized space. The direction of compression is both along and orthogonal to $\boldsymbol{w}$.

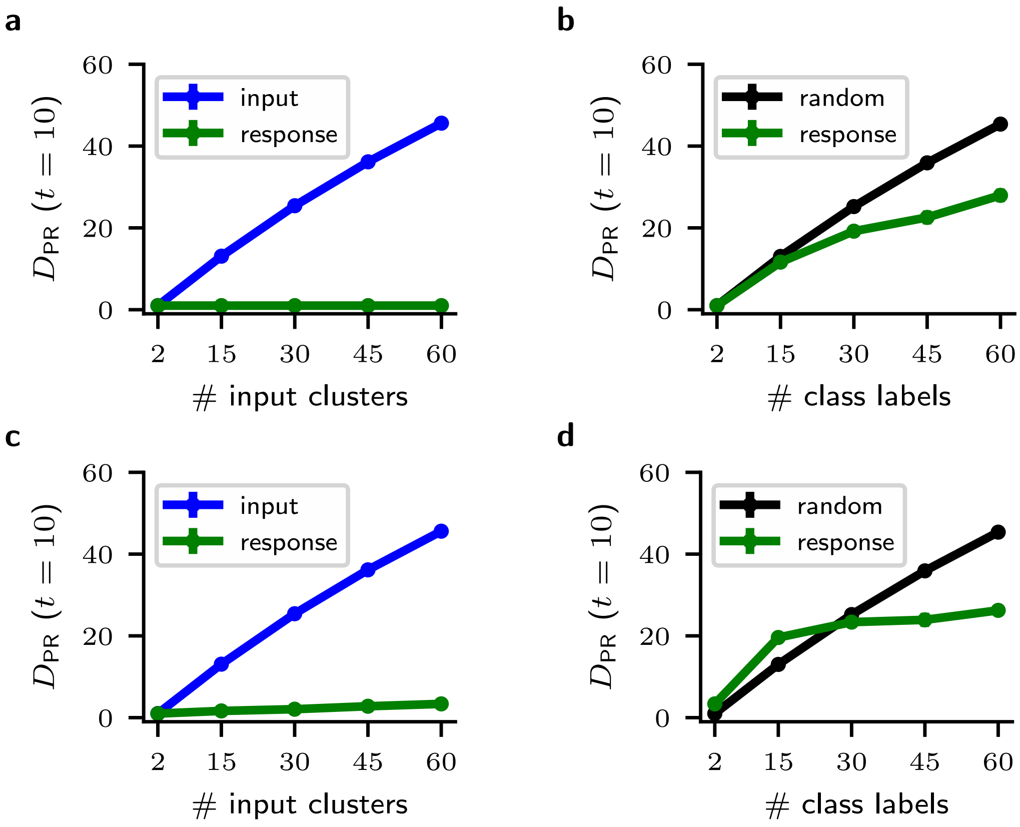

Figure 7: Effective dimensionalities of the trained network responses to inputs distributed in N-dimensional ambient space, measured at the evaluation time $t_{eval} = 10$. Error bars denote two standard deviations of three initializations of task and networks (in all panels they are too small to see). **a.** Edge-of-chaos networks. Blue: Effective dimensionality of the inputs. Green: effective dimensionality of the network representation as a function of the number of input clusters. Dimensionality remains flat and small. **b.** Edge-of-chaos networks. Green: effective dimensionality of the network representation as a function of the number of class labels. Black: Effective dimensionality of points distributed uniformly at random in an N-dimensional ball. The number of points drawn is determined by the number of class labels. This is to roughly measure what the effective dimensionality of the network would be if it formed a fixed point for every class label, and distributed these fixed points randomly in space. **c.** Strongly chaotic networks. Legend as in **a**. **d.** Strongly chaotic networks. Legend as in **b**.

Theoretical arguments for compression. Here we show the full steps of the derivation of the compression result Equation (3). Our assumptions again are of a linear, single-layer, feedforward network $\boldsymbol{h} = \boldsymbol{W}^{in}\boldsymbol{x} + \boldsymbol{b}$ with scalar output $o = \boldsymbol{w}^T\boldsymbol{h}$. The loss for this network is $\mathcal{L}_{\boldsymbol{W}}(x) = 0.5\left(o - o_{\text{target}}\right)^2$, where o_{target} encodes the class label for input sample $\boldsymbol{x}$. The update rule for $\boldsymbol{W}^{in}$ after presentation of the random input sample $\boldsymbol{x}$ (i.e. batch size 1) is then $\boldsymbol{W}^{in} \leftarrow \boldsymbol{W}^{in} + \delta\boldsymbol{W}^{in}$ where

$$\delta\boldsymbol{W}^{in} = -\eta\left(o - o_{\text{target}}\right)\boldsymbol{w}\boldsymbol{x}^T, \tag{6}$$

and the update for the bias is $\boldsymbol{b} \leftarrow \boldsymbol{b} + \delta\boldsymbol{b}$ where

$$\delta\boldsymbol{b} = -\eta\left(o - o_{\text{target}}\right)\boldsymbol{w}. \tag{7}$$

Here η is the *learning rate* for the gradient descent routine and $\boldsymbol{w}\boldsymbol{x}^T$ is the outer product of $\boldsymbol{w}$ and $\boldsymbol{x}$.

We suppose that $\boldsymbol{h}$ is a random variable and compute statistics of $\boldsymbol{h} + \delta\boldsymbol{h}$ induced by the update rule Equations (6) and (7). Instead of updating the output weights $\boldsymbol{w}$ via gradient descent, we instead assume that $\boldsymbol{w}$ is normally distributed with mean $\langle\boldsymbol{w}\rangle = \mu_w\boldsymbol{e}_1$ and covariance $\boldsymbol{C}_w = \sigma_w^2\boldsymbol{I}$ (the choice of mean doesn't result in loss of generality, since we can always rotate our coordinate system). We further assume that (1) $\boldsymbol{C}_{\boldsymbol{h}}$ is diagonal, (2) $\boldsymbol{h}$ is independent of $\boldsymbol{w}$, and (3) the norm $\|\boldsymbol{x}\|$ of the input is constant. (These assumptions are satisfied, for instance, by a 3×3 input matrix $\boldsymbol{W}^{in} = \boldsymbol{I}$, zero bias, and $\boldsymbol{x}$ uniformly distributed on the 3-dimensional unit sphere with first coordinate $x_1 = 1/\sqrt{3}$.)

Our goal is to compute $\boldsymbol{C}_{\boldsymbol{h}+\delta\boldsymbol{h}}$ and relate it to $\boldsymbol{C}_{\boldsymbol{h}}$. We start by computing that

$$\begin{aligned}
\boldsymbol{h} + \delta\boldsymbol{h} &= \boldsymbol{h} + \delta\boldsymbol{W}^{in}\boldsymbol{x} + \delta\boldsymbol{b}\\
&= \boldsymbol{h} - \eta\left(o - o_{\text{target}}\right)\boldsymbol{w}\boldsymbol{x}^T\boldsymbol{x} - \eta\left(o - o_{\text{target}}\right)\boldsymbol{w}\\
&= \boldsymbol{h} - \eta\left(1 + \|\boldsymbol{x}\|^2\right)\left(\boldsymbol{w}^T\boldsymbol{h} - o_{\text{target}}\right)\boldsymbol{w}\\
&= \boldsymbol{h} - \eta\left(1 + \|\boldsymbol{x}\|^2\right)\left(\boldsymbol{w}\boldsymbol{w}^T\boldsymbol{h} - o_{\text{target}}\boldsymbol{w}\right)
\end{aligned}$$

Let $a = 1 + \|x\|^2$ for convenience. Using that a is constant, we compute

$$\begin{aligned}
\left\langle(\boldsymbol{h} + \delta\boldsymbol{h})(\boldsymbol{h} + \delta\boldsymbol{h})^T\right\rangle &= \left\langle\left(\boldsymbol{h} - \eta a\left(\boldsymbol{w}\boldsymbol{w}^T\boldsymbol{h} - o_{\text{target}}\boldsymbol{w}\right)\right)\left(\boldsymbol{h} - \eta a\left(\boldsymbol{w}\boldsymbol{w}^T\boldsymbol{h} - o_{\text{target}}\boldsymbol{w}\right)\right)^T\right\rangle\\
&= \left\langle\left(\boldsymbol{h} - \eta a\left(\boldsymbol{w}\boldsymbol{w}^T\boldsymbol{h} - o_{\text{target}}\boldsymbol{w}\right)\right)\left(\boldsymbol{h}^T - \eta a\left(\boldsymbol{h}^T\boldsymbol{w}\boldsymbol{w}^T - o_{\text{target}}\boldsymbol{w}^T\right)\right)\right\rangle\\
&= \left\langle\boldsymbol{h}\boldsymbol{h}^T - \eta a\boldsymbol{h}\left(\boldsymbol{h}^T\boldsymbol{w}\boldsymbol{w}^T - o_{\text{target}}\boldsymbol{w}^T\right) - \eta a\left(\boldsymbol{w}\boldsymbol{w}^T\boldsymbol{h} - o_{\text{target}}\boldsymbol{w}\right)\boldsymbol{h}^T\right\rangle\\
&\quad + O\left(\eta^2\right)\\
&= \left\langle\boldsymbol{h}\boldsymbol{h}^T\right\rangle - \eta a\left\langle\boldsymbol{h}\boldsymbol{h}^T\boldsymbol{w}\boldsymbol{w}^T - o_{\text{target}}\boldsymbol{h}\boldsymbol{w}^T\right\rangle\\
&\quad - \eta a\left\langle\boldsymbol{w}\boldsymbol{w}^T\boldsymbol{h}\boldsymbol{h}^T - o_{\text{target}}\boldsymbol{w}\boldsymbol{h}^T\right\rangle + O\left(\eta^2\right)\\
&= \left\langle\boldsymbol{h}\boldsymbol{h}^T\right\rangle - \eta a\left(\left\langle\boldsymbol{h}\boldsymbol{h}^T\boldsymbol{w}\boldsymbol{w}^T\right\rangle + \left\langle\boldsymbol{w}\boldsymbol{w}^T\boldsymbol{h}\boldsymbol{h}^T\right\rangle\right)\\
&\quad + o_{\text{target}}\eta a\left(\left\langle\boldsymbol{h}\boldsymbol{w}^T\right\rangle + \left\langle\boldsymbol{w}\boldsymbol{h}^T\right\rangle\right) + O\left(\eta^2\right)\\
&= \left\langle\boldsymbol{h}\boldsymbol{h}^T\right\rangle - \eta a\left(\left\langle\boldsymbol{h}\boldsymbol{h}^T\right\rangle\left\langle\boldsymbol{w}\boldsymbol{w}^T\right\rangle + \left\langle\boldsymbol{w}\boldsymbol{w}^T\right\rangle\left\langle\boldsymbol{h}\boldsymbol{h}^T\right\rangle\right)\\
&\quad + o_{\text{target}}\eta a\left(\langle\boldsymbol{h}\rangle\langle\boldsymbol{w}\rangle^T + \langle\boldsymbol{w}\rangle\langle\boldsymbol{h}\rangle^T\right) + O\left(\eta^2\right)
\end{aligned}$$

Now we compute

$$\begin{aligned}
\langle \boldsymbol{h}+\delta\boldsymbol{h}\rangle \langle \boldsymbol{h}+\delta\boldsymbol{h}\rangle^T &= \left(\langle \boldsymbol{h}\rangle - \eta a \left(\langle \boldsymbol{w}\boldsymbol{w}^T\rangle \langle \boldsymbol{h}\rangle - o_{\text{target}} \langle \boldsymbol{w}\rangle\right)\right)\left(\langle \boldsymbol{h}\rangle - \eta a \left(\langle \boldsymbol{w}\boldsymbol{w}^T\rangle \langle \boldsymbol{h}\rangle - o_{\text{target}} \langle \boldsymbol{w}\rangle\right)\right)^T \\
&= \left(\langle \boldsymbol{h}\rangle - \eta a \left(\langle \boldsymbol{w}\boldsymbol{w}^T\rangle \langle \boldsymbol{h}\rangle - o_{\text{target}} \langle \boldsymbol{w}\rangle\right)\right)\left(\langle \boldsymbol{h}\rangle^T - \eta a \left(\langle \boldsymbol{h}\rangle^T \langle \boldsymbol{w}\boldsymbol{w}^T\rangle^T - o_{\text{target}} \langle \boldsymbol{w}\rangle^T\right)\right) \\
&= \langle \boldsymbol{h}\rangle \langle \boldsymbol{h}\rangle^T - \eta a \left(\langle \boldsymbol{h}\rangle \langle \boldsymbol{h}\rangle^T \langle \boldsymbol{w}\boldsymbol{w}^T\rangle - o_{\text{target}} \langle \boldsymbol{h}\rangle \langle \boldsymbol{w}\rangle^T\right) \\
&\qquad - \eta a \left(\langle \boldsymbol{w}\boldsymbol{w}^T\rangle \langle \boldsymbol{h}\rangle \langle \boldsymbol{h}\rangle^T - o_{\text{target}} \langle \boldsymbol{w}\rangle \langle \boldsymbol{h}\rangle^T\right) + O\left(\eta^2\right) \\
&= \langle \boldsymbol{h}\rangle \langle \boldsymbol{h}\rangle^T - \eta a \left(\langle \boldsymbol{h}\rangle \langle \boldsymbol{h}\rangle^T \langle \boldsymbol{w}\boldsymbol{w}^T\rangle + \langle \boldsymbol{w}\boldsymbol{w}^T\rangle \langle \boldsymbol{h}\rangle \langle \boldsymbol{h}\rangle^T\right) \\
&\qquad + o_{\text{target}}\eta a \left(\langle \boldsymbol{h}\rangle \langle \boldsymbol{w}\rangle^T + \langle \boldsymbol{w}\rangle \langle \boldsymbol{h}\rangle^T\right) + O\left(\eta^2\right)
\end{aligned}$$

Putting these together and using our assumptions of diagonal covariances,

$$\begin{aligned}
\boldsymbol{C}_{\boldsymbol{h}+\delta\boldsymbol{h}} &= \left\langle (\boldsymbol{h}+\delta\boldsymbol{h})(\boldsymbol{h}+\delta\boldsymbol{h})^T\right\rangle - \langle \boldsymbol{h}+\delta\boldsymbol{h}\rangle \langle \boldsymbol{h}+\delta\boldsymbol{h}\rangle^T \\
&= \left\langle \boldsymbol{h}\boldsymbol{h}^T\right\rangle - \langle \boldsymbol{h}\rangle \langle \boldsymbol{h}\rangle^T \\
&\qquad - \eta a \left(\left\langle \boldsymbol{h}\boldsymbol{h}^T\right\rangle \langle \boldsymbol{w}\boldsymbol{w}^T\rangle + \langle \boldsymbol{w}\boldsymbol{w}^T\rangle \left\langle \boldsymbol{h}\boldsymbol{h}^T\right\rangle - \left(\langle \boldsymbol{h}\rangle \langle \boldsymbol{h}\rangle^T \langle \boldsymbol{w}\boldsymbol{w}^T\rangle + \langle \boldsymbol{w}\boldsymbol{w}^T\rangle \langle \boldsymbol{h}\rangle \langle \boldsymbol{h}\rangle^T\right)\right) \\
&\qquad + O\left(\eta^2\right). \\
&= \left\langle \boldsymbol{h}\boldsymbol{h}^T\right\rangle - \langle \boldsymbol{h}\rangle \langle \boldsymbol{h}\rangle^T \\
&\qquad - \eta a \left(\left(\left\langle \boldsymbol{h}\boldsymbol{h}^T\right\rangle - \langle \boldsymbol{h}\rangle \langle \boldsymbol{h}\rangle^T\right)\langle \boldsymbol{w}\boldsymbol{w}^T\rangle + \langle \boldsymbol{w}\boldsymbol{w}^T\rangle \left(\left\langle \boldsymbol{h}\boldsymbol{h}^T\right\rangle - \langle \boldsymbol{h}\rangle \langle \boldsymbol{h}\rangle^T\right)\right) + O\left(\eta^2\right) \\
&= \boldsymbol{C}_{\boldsymbol{h}} - \eta a \boldsymbol{C}_{\boldsymbol{h}} \langle \boldsymbol{w}\boldsymbol{w}^T\rangle + \langle \boldsymbol{w}\boldsymbol{w}^T\rangle \boldsymbol{C}_{\boldsymbol{h}} + O\left(\eta^2\right) \\
&= \boldsymbol{C}_{\boldsymbol{h}} - 2\eta a \langle \boldsymbol{w}\boldsymbol{w}^T\rangle \boldsymbol{C}_{\boldsymbol{h}} + O\left(\eta^2\right) \\
&= \left(\boldsymbol{I} - 2\eta a \langle \boldsymbol{w}\boldsymbol{w}^T\rangle\right)\boldsymbol{C}_{\boldsymbol{h}} + O\left(\eta^2\right).
\end{aligned}$$

This matrix is diagonal (up to order $O(\eta^2)$), with diagonal entries

$$\operatorname{var}\left(h_k + \delta h_k\right) = \alpha_k \operatorname{var}(h_k) + O(\eta^2) \tag{8}$$

where $\alpha_1 = 1 - 2\eta\left(1 + \|\boldsymbol{x}\|^2\right)\left(\sigma_w^2 + \mu_w^2\right)$ and $\alpha_k = 1 - 2\eta\left(1 + \|\boldsymbol{x}\|^2\right)\sigma_w^2$ for $1 < k \leq N$.

Note that, while we started with the assumption of independence of h and w, the hidden state generally becomes increasingly more correlated with w over training, so that our arguments only hold for the first iterations of training.