Recurrent neural networks learn robust representations by dynamically balancing compression and expansion

Matthew Farrell,[1,2] Stefano Recanatesi,[1,3] Timothy Moore,[1] Guillaume Lajoie,[4,5] and Eric Shea-Brown[1,2]

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

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. What determines the dimensionality of activity in neural circuits? What is the functional role of this dimensionality in behavior and task learning? In this work we address these questions using recurrent neural network (RNN) models, which have recently shown promise in predicting and explaining brain dynamics. Through simulations and mathematical analysis, we show how the dimensionality of RNN activity evolves over time and over stages of learning. We find that RNNs can learn to balance tendencies to expand and compress dimensionality in a way that matches task demands and further generalizes to new data. Strongly chaotic networks appear particularly adept in learning this balance in the case of classifying low-dimensional inputs, combining the natural tendency of chaos to expand dimensionality with opportunistic compression driven by stochastic gradient descent to form representations with good generalization properties. These findings shed light on fundamental dynamical mechanisms by which neural networks solve tasks with robust representations that generalize to new cases.

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 [15, 47]. Rather than being set by, say, the number of neurons in the circuit, the *effective* dimensionality of the neural activity (or neural "representation") seems to be intimately linked to the complexity of the function, or behavior, that the neural circuit fulfills or produces [14, 19, 50, 57]. Similar task-dependent trends in dimensionality can manifest in artificial networks used in machine learning systems trained using optimization algorithms (e.g., [14, 42, 62]). Bridging between machine learning and neuroscience, artificial networks are powerful tools for investigating dynamical representations in controlled settings, and enable tests of theoretical hypotheses that can be leveraged to formulate experimental predictions (reviewed in [6]).

The connection between task complexity and representation dimension is especially intriguing in light of fundamental ideas in learning theory. On the one hand, high-dimensional representations can subserve complex and general computations that nonlinearly combine many features of inputs [8, 13, 18, 39, 52, 61]. 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 [7, 9, 18, 60], and see [10, 11] for important steps toward characterizing the coding properties of general representation geometries).

What are the mechanisms that regulate the effective dimensionality of network activity across many trials and inputs? While dimensionality modulation can occur due to hard structural constraints such as "bottleneck" layers with a small number of neurons [23], here we consider the case where the number of neurons is not a constraining factor. We consider two factors: chaos and learning. Frequently encountered in recurrent neural network (RNN) models of brain function, dynamical chaos (whereby tiny changes in internal states are amplified by unstable, but deterministic, dynamics) provides a parsimonious explanation for both repeatable structure as well as internally generated variability seen in highly recurrent brain networks such as cortical circuits [17, 30, 38, 40, 55, 56, 59, 63]. In the reservoir computing framework, chaos in an RNN can increase the diversity of patterns and dimension of activity the network produces through time in response to inputs, in some cases increasing the system's computational abilities [26, 35, 41]. While chaos-driven expansion as determined by fixed recurrent weights has been explored in simulations [35], the attributes of this phenomenon as recurrent weights evolve through training are less understood (but see [16, 58]).

The goal of the current work is to elucidate how an RNN's representation and dynamics are shaped by learning to balance the two desiderata of compressing and expanding the dimension of input data. Notably, we find evidence that stochastic gradient descent (SGD) is predisposed to compress dimension, even when expansion is necessary to solve the task. Chaos in our case provides a key ingredient for effective learning: directions of expansion that coexist with many directions of compression. SGD is often effective at making use of this expansion while still compressing opportunistically without sacrificing task performance. This results in learned representations that balance expansion and compression and that consequently lend themselves to good generalization. The same balance of expansion

and compression in chaotic networks may account for the highly variable nature of neural activity, and suggests that further optimizing this trade-off may be a method for improving the flexibility and accuracy of artificial neural networks. Finally, we analyze how SGD compresses dimensionality in neural networks. Our work is similar to the information compression phenomenon explored in [53, 54], but instead focusing explicitly on the geometry of the representation. In addition, while the noise generated by SGD in [53] is treated as additive and Gaussian, here we consider the true noise generated by two steps of SGD in a more simplified model. This contributes to revealing why this ubiquitous algorithm often finds solutions that generalize (e.g. [2, 4, 20, 29, 33, 36, 53, 54]).

RESULTS

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 [22, 37]. 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 network initialized in this way can be modulated from stable to chaotic by increasing the average magnitude of the initial neural coupling strength, 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. To measure chaos, we numerically compute the *Lyapunov exponents* of the system (see Supplemental Information for more details).

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 scenarios for the ambient dimension: $d = 2$ and $d = N$. Each cluster is assigned one of two class labels, at random, as illustrated in Fig. 1. While this schematic only shows six clusters for clarity, in our simulations we use 60 clusters. Each cluster is assigned one of two class labels: 30 clusters for label 1, and 30 for label 2. 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 of 9 timesteps; (3) a linear readout of the network's state at timestep 10 is used to classify the input into one of the two classes. Recurrent and output weights are adjusted via an SGD 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 moderate changes in the choice of model parameters, as well as using squared error loss instead of cross entropy. See Fig. S4, where we train our network on the MNIST digit classification database [34].

SGD trains networks to compress high-dimensional inputs

We start by considering the classification of high-dimensional inputs, where the input ambient space dimensionality d is equal to the number N of recurrent units (see Fig. 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

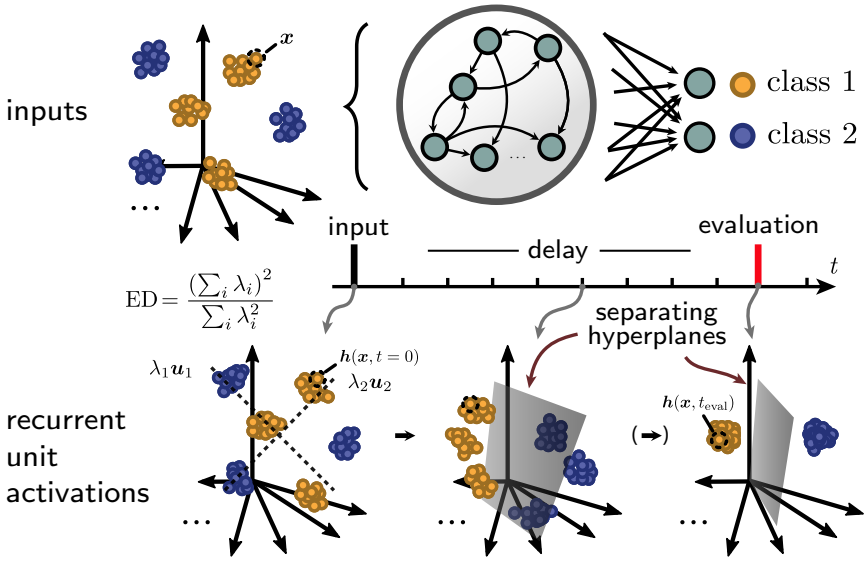

FIG. 1. Task and model schematic. Input clusters are distributed in a d-dimensional subspace of neural space. An input $\boldsymbol{x}$ is shown to the network for one timestep, and after a delay period in which the network evolves undriven, the network state $\boldsymbol{h}(\boldsymbol{x}, t_{\mathrm{eval}})$ is linearly read out at a timepoint t_{eval} in order to classify the input. Bottom: Schematic of the response of the network to the ensemble of inputs, viewed at snapshots in time. Through training, the network attempts to form a representation that allows a separating hyperplane. The network may also bring points belonging to the same class together to form a more compact representation. Bottom left: Dashed lines depict the top two eigenvectors of the covariance of the representation, scaled by the corresponding eigenvalues. To capture the degree to which activations fill space, we use the effective dimensionality (ED), which measures the participation ratio of all of the eigenvalues.

in the input space. This focuses our attention on what, if anything, networks learn to do beyond producing linear separability.

We study networks in two dynamical regimes: a strongly chaotic network and one that is initialized at the edge of chaos. Fig. 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. In these 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 the plots in Fig. 2b 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 Fig. 2d, both networks easily achieve perfect testing accuracy on the delayed classification task. What is then of interest is how properties of the network's *internal representation* change over the course of training. In Fig. 2c we plot the top two principal components of the network responses to 600 input samples at snapshots in time. The top row corresponds to the edge-of-chaos network, where we see that points belonging to different classes are pulled apart while points within the same class are compressed together in all dimensions (see Fig S1 for a quantification of this phenomenon). The case is similar for the

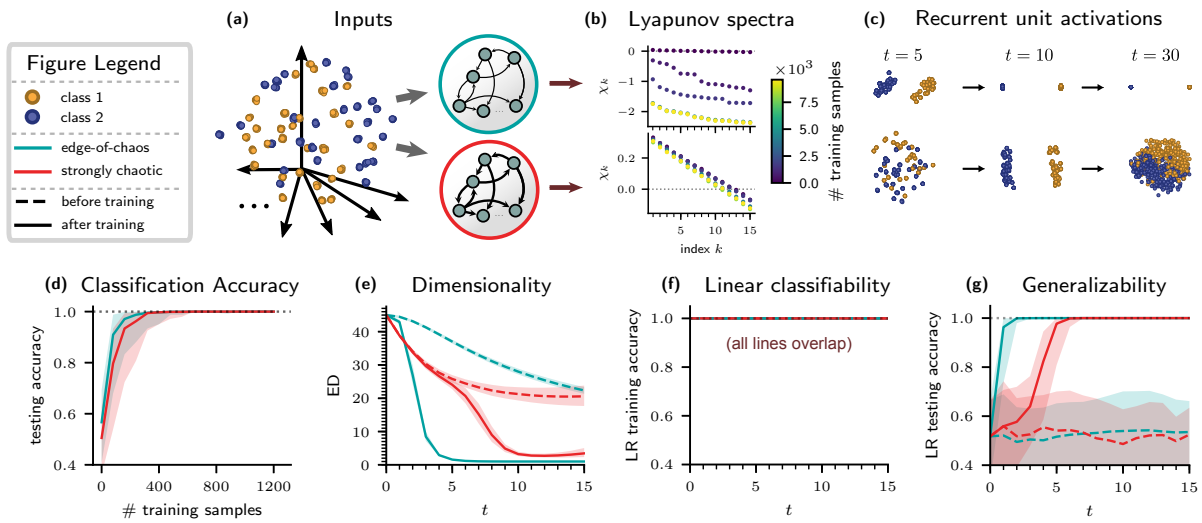

FIG. 2. Comparison of edge-of-chaos (blue-green) and strongly chaotic (red) networks classifying high-dimensional inputs. Input color (yellow-orange or purple) denotes true class label. Shaded regions indicate 75% probability mass of gamma distributions fit to values of the dependent variables over a sample of 30 network and input realizations, with solid lines indicating medians. "After training" designates networks that have been trained on 9,600 input samples. (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. Top: Edge-of-chaos network. Bottom: strongly chaotic network. (c) Activations of recurrent units responding to an ensemble of 600 inputs, plotted as "snapshots" in time in principal component space. Top: Edge-of-chaos network. Bottom: strongly chaotic network. (d) Testing accuracy measured through training. (e) Effective dimensionality (ED) of the network representation through time t. (f) Mean accuracy of a logistic regression (LR) linear classifier trained on the recurrent unit activations at each timepoint t. (g) Mean testing accuracy of an LR classifier. The model was trained on data drawn from 80% of the input clusters (816 input points) and tested on held-out data from the remaining 20% of input clusters (204 input points).

strongly chaotic network (bottom row), except that the classes begin to mix back together in the top two PCs after the evaluation time $t_{\text{eval}} = 10$. Our analysis of the Lyapunov exponents indicates that the edge-of-chaos network forms fixed points that persistently solve the task, while the strongly chaotic network forms chaotic attractors that transiently solve the task.

This compression phenomenon is partly captured by measuring the *effective dimensionality* (ED) of the representation [27, 48]. The equation for the ED of a set of S points $\boldsymbol{V} = [v_{si}] \in \mathbb{R}^{S \times d}$ with ambient dimension d is

$$ED\left(\boldsymbol{V}\right) = \frac{\operatorname{Tr}\left(\boldsymbol{C}\right)^2}{\operatorname{Tr}\left(\boldsymbol{C}^2\right)} = \frac{\left(\sum_{i=1}^{d} \lambda_i\right)^2}{\sum_{i=1}^{d} \lambda_i^2} = \left(\sum_{i=1}^{d} \tilde{\lambda}_i^2\right)^{-1} \tag{1}$$

where $C_{ij} = \langle v_{si} v_{sj} \rangle_s - \langle v_{si} \rangle_s \langle v_{sj} \rangle_s$ 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 Fig. 1. ED can be roughly thought of as the number of principal components needed to capture most of the variance of the data [19].

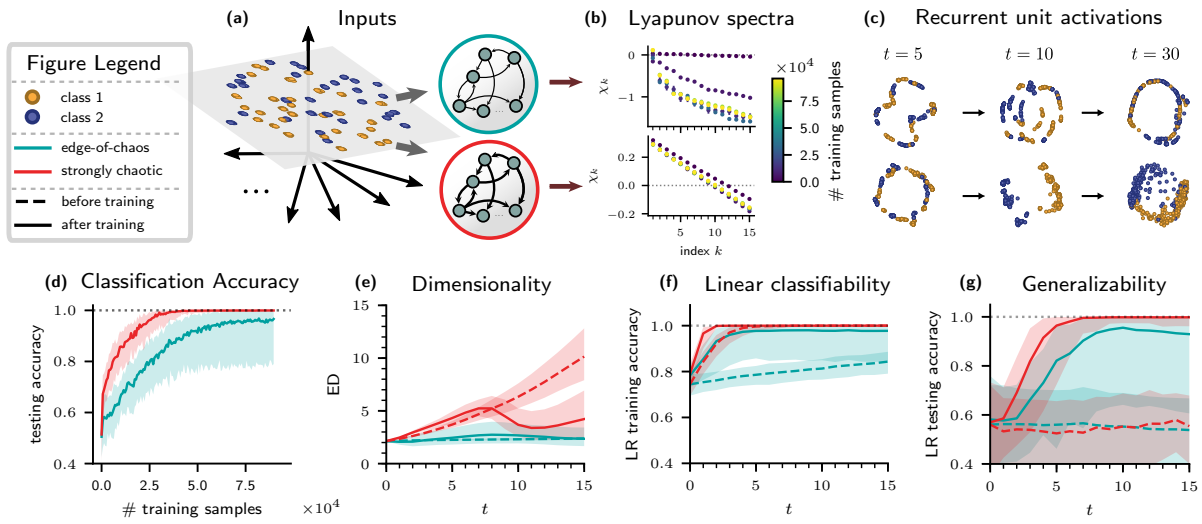

FIG. 3. Comparison of edge-of-chaos and strongly chaotic networks classifying low-dimensional inputs. Details are similar to Fig. 2. "After training" designates networks that have been trained on 96,000 input samples. (a) Inputs lie on a two-dimensional plane cutting through recurrent unit space. (b-g) Descriptions as in Fig. 2.

In Fig. 2e the ED is plotted through time t. For each time point t the matrix $\boldsymbol{V} = \boldsymbol{V}(t)$ used to compute ED has dimensions $S \times N$ where S is the number of input samples shown to the network. The EDs of the trained networks are approximately equal to that of the input at time $t = 0$, since the initial states 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 time $t_{\text{eval}} = 10$. This compression results both from increasing distances between different classes as well as decreasing distances within classes (see Fig. 2c and Fig S1). 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 moderate changes in the learning algorithm parameters. This compression can be viewed through the lens of building *invariance* in the network representation [1, 20, 21], in this case invariance to input cluster identity (data not shown).

We next study the coding properties of these representations. Fig. 2f shows the mean accuracy of a logistic regression (LR) linear classifier trained to classify the network representation at each timepoint. This confirms the point made above: the input data is linearly separable, and this property is retained by the network dynamics. The interesting properties of the network computation lie elsewhere, in that the learned dynamics and resulting dimensionality compression lead to better generalization properties of the representation. In Fig. 2g, we measure generalization by first training an LR classifier on the network response to inputs drawn from a fixed 80% of the input clusters, and then measuring the accuracy of this classifier on the network response to samples drawn from the remaining 20% of the clusters. The dashed lines indicate that, while the representations in the untrained networks are linearly separable, a linear classifier trained on these representations does not generalize well to held-out clusters. In contrast, after training, the network representations become increasingly generalizable through time t, eventually allowing for perfect classification accuracy on

held out clusters and maintaining this for future times.

We emphasize that in this case linear separability – the property needed to solve the task with perfect accuracy – requires neither dimensionality expansion nor compression of inputs. Nevertheless, we find that networks **do** learn to strongly compress their inputs, at the same time achieving representations that lend themselves to good generalization.

Chaos drives dimensionality expansion, enabling SGD to find a balance between expansion and compression

We next turn our attention to inputs embedded in a two-dimensional ambient space, $d = 2$ (see Fig. 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 separate 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.

Fig. 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 remains on the edge-of-chaos through training (Fig. 3b). The strongly chaotic network remains strongly chaotic during training. In comparing the representations in Fig. 3c, we find indications that the strongly chaotic network (bottom) is more successful at achieving class separation at time $t_{\text{eval}} = 10$ than the edge-of-chaos network (top). Fig. 3d confirms that this is indeed the case: the strongly chaotic network learns to achieves near-perfect classification accuracy, while the edge-of-chaos network is not as successful (Fig. 3d).

In looking at the dimensionality of the trained networks through time (Fig. 3e), we find that both initially expand dimensionality until about $t = 7$, with the expansion being much more dramatic for the strongly chaotic network (solid lines). 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, in contrast to the edge-of-chaos network (dashed lines). The compression from $t = 7$ to $t = 10$ is then learned through training.

In Fig. 3f we see that the strongly chaotic network creates a linearly separable representation both before and after training, while the representations for the edge-of-chaos network are far from linearly separable before training. This is consistent with the expansion of dimensionality that occurs even before training in the strongly chaotic network. The generalization of both networks improves with training (Fig. 3g), with the strongly chaotic network achieving better generalization performance.

We next turn our attention to a more challenging task, where input is only shown to a subset of neurons (Fig. 4a); here, two. In this case, dimensionality expansion can have the added benefit of enlisting more neurons to aid in the computation (e.g. [31]). We find that the strongly chaotic network is still able to solve the task near-perfectly, while the edge-of-chaos network's median performance remains at about 75% (Fig. 4c). Looking at ED, we find that the edge-of-chaos network does not learn to significantly expand ED, while the strongly chaotic network is higher-dimensional before training and learns to expand its dimensionality even further (Fig. 4e). This expansion again results in good linear separability by the evaluation time $t = 10$, even before training (Fig. 4f).

The generalization properties of the strongly chaotic network's representation improve after training, even though dimensionality compression does not occur (Fig. 4g, solid red line). However, comparing this network at the evaluation time with the strongly chaotic

network trained on distributed inputs (Fig. 3g, solid red line), we find that here the generalization score is less reliable, with larger error bars. This suggests that the dimensionality compression exhibited in Fig. 3e, solid red line, plays a role in improving generalization properties, while constrained dimensionality expansion as in Fig. 4e, solid red line, can still allow for relatively good (albeit not as good) generalization. Indeed, while the strongly chaotic network expands dimensionality without a clear compression phase, the dimensionality is still small at the evaluation time relative to the size of the network.

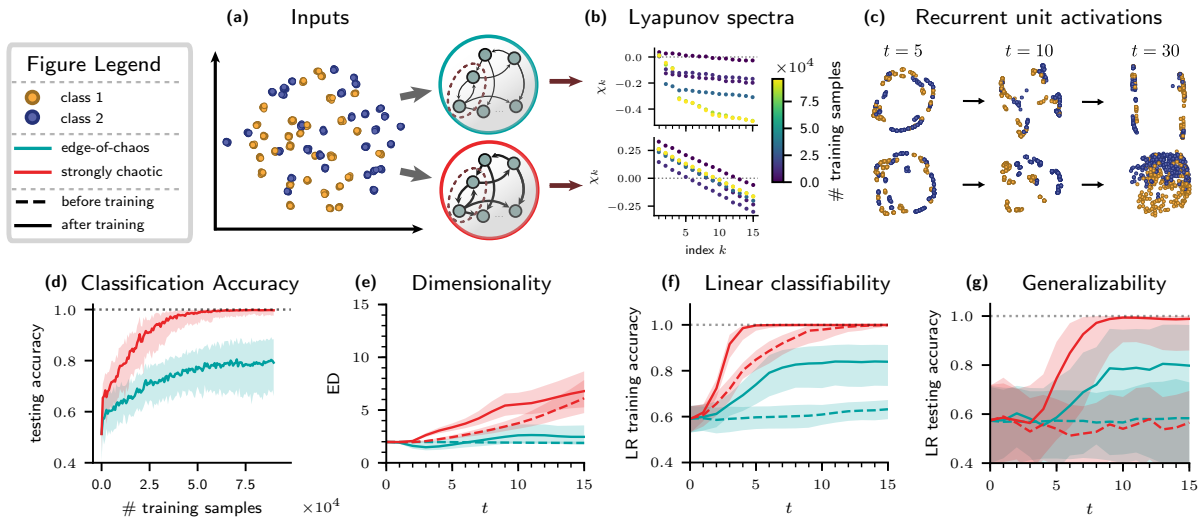

FIG. 4. Comparison of edge-of-chaos and strongly chaotic networks classifying two-dimensional inputs fed to two neurons. Details are similar to Figs. 2 and 3. "After training" designates networks that have been trained on 96,000 input samples. (a) Schematic of the task. Two-dimensional input is delivered to two randomly selected neurons in the network. (b-g) Descriptions as in Fig. 2.

Mechanistic explanations for compression

Our networks exhibit dimensionality compression when tasked with classifying high-dimensional, linearly separable inputs. Above we have discussed the benefits of this compression from the perspective of generalization; in this section, we give mechanistic explanations for why this compression occurs. More specifically, for gradient descent parameter updates, we show how noise can can lead to weight changes that compress the dimensionality of internal unit representations, and then apply this analysis to understand how SGD drives compression. See [25] for a more thorough analysis in the case of networks trained by a particular unsupervised learning rule, [28] for compression induced by the "Pseudo-Inverse" learning rule, and [49] for a recently developed alternative approach to analyzing compression induced by SGD. Our result is related to the pioneering idea of information compression which is studied in, e.g.([53, 54]), which implicates the same driving mechanism (gradient noise) and establishes the resulting impact on generalization.

To shed light on the compression behavior illustrated in Figs. 2d, 2e, 3c and 3e – 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 a linear, single-hidden-layer feedforward network. Our analysis proceeds in two steps. First, we show how gradient updates with isotropic noise injected in the output weights leads to compression in all directions orthogonal to the readouts. This suggests a crucial role for variability in driving compression. Second, we show how the noisy variability of the network weights generated by SGD can also result in compression orthogonal to the readout direction.

In a single-layer linear network, the internal neural activations responding to a single input sample $\boldsymbol{x}$ are $\boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$, and the scalar output is $\hat{o}(\boldsymbol{x}) = \boldsymbol{r}^\top \boldsymbol{h}(\boldsymbol{x})$. Since significant compression occurs for both cross entropy as well as squared error loss in our simulations, for ease of analysis we consider squared error loss $\ell(\boldsymbol{x}) = \frac{1}{2}e(\boldsymbol{x})^2$, where $e(\boldsymbol{x}) = \hat{o}(\boldsymbol{x}) - o(\boldsymbol{x})$ and o maps the input $\boldsymbol{x}$ to its corresponding class label, either $+1$ or -1. The gradient descent learning updates $\boldsymbol{W} \leftarrow \boldsymbol{W} + \delta\boldsymbol{W}$ and $\boldsymbol{b} \leftarrow \boldsymbol{b} + \delta\boldsymbol{b}$ over a batch of input samples B are:

$$\delta\boldsymbol{W} = -\frac{\gamma}{|B|}\sum_{\boldsymbol{x}\in B} e(\boldsymbol{x})\boldsymbol{r}\boldsymbol{x}^T, \tag{2}$$

$$\delta\boldsymbol{b} = -\frac{\gamma}{|B|}\sum_{\boldsymbol{x}\in B} e(\boldsymbol{x})\boldsymbol{r} \tag{3}$$

Here γ is the *learning rate* and $\boldsymbol{r}\boldsymbol{x}^T$ is the outer product of $\boldsymbol{r}$ and $\boldsymbol{x}$. The update rule results in a corresponding update $\boldsymbol{h} \leftarrow \boldsymbol{h} + \delta\boldsymbol{h}$ where $\delta\boldsymbol{h}(\boldsymbol{x}') = \delta\boldsymbol{W}\boldsymbol{x}' + \delta\boldsymbol{b}$.

Note that the change of hidden representation $\delta\boldsymbol{h}$ points in the direction of the readout vector: $\delta\boldsymbol{h}(\boldsymbol{x}') \propto \boldsymbol{r}$. If output weights $\boldsymbol{r}$ are unchanged by the update rule, then gradient descent parameter updates can modulate the response only along this single dimension. This modulation can be expected to cause compression in this single dimension when gradient descent reduces the loss, as the average loss is minimized when $\boldsymbol{r}^\top\boldsymbol{h}(\boldsymbol{x}') = o(\boldsymbol{x}')$ for all $\boldsymbol{x}'$. Satisfying this equation implies that $\boldsymbol{h}(\boldsymbol{x}')$ for $o(\boldsymbol{x}') = +1$ has minimized its variation along $\boldsymbol{r}$, and similarly for $o(\boldsymbol{x}') = -1$.

To see reduction of variability in all dimensions like that exhibited by Fig. 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. For all subsequent calculations, we assume that inputs have zero mean and covariance $\left\langle \boldsymbol{x}\boldsymbol{x}^\top \right\rangle = \sigma_x^2\boldsymbol{I}$ (note that this is approximately satisfied by the inputs used for the simulations above).

Let $\boldsymbol{h}_k$ denote the hidden representation after k steps of gradient descent, and similarly for parameters $\boldsymbol{W}$, $\boldsymbol{b}$, and $\boldsymbol{r}$. A single step of gradient descent results in the representation $\boldsymbol{h}_1(\boldsymbol{x}') = \boldsymbol{h}_0(\boldsymbol{x}') + \delta\boldsymbol{W}_0\boldsymbol{x}' + \delta\boldsymbol{b}_0$. Averaging over batches B, this becomes

$$\langle \boldsymbol{h}_1(\boldsymbol{x}')\rangle = \boldsymbol{h}_0(\boldsymbol{x}') - \gamma\boldsymbol{r}_0\left\langle e(\boldsymbol{x})\boldsymbol{x}^\top\right\rangle \boldsymbol{x}' \tag{4}$$

$$= \boldsymbol{h}_0(\boldsymbol{x}') - \gamma\left(\sigma_x^2\boldsymbol{r}_0\boldsymbol{r}_0^\top\boldsymbol{W}_0 - \boldsymbol{r}_0\left\langle o(\boldsymbol{x})\boldsymbol{x}^\top\right\rangle\right)\boldsymbol{x}' \tag{5}$$

We will first consider variability in the initial readout direction $\boldsymbol{r}_0$ by modeling it as isotropic additive noise: $\boldsymbol{r}_0 = \overline{\boldsymbol{r}} + \boldsymbol{\xi}$ where $\boldsymbol{\xi}$ is a random variable with zero mean and covariance $\langle\boldsymbol{\xi}\boldsymbol{\xi}^\top\rangle = \sigma_\xi^2\boldsymbol{I}$ and $\overline{\boldsymbol{r}}$ is fixed. This noise is a simple model of variability that could arise, for instance, from previous steps of SGD. Let $\mathcal{P}(\overline{\boldsymbol{r}})$ denote any vector proportional to $\overline{\boldsymbol{r}}$ (this notation functions similarly to big O notation). These terms contribute only along a single dimension, and we can expect compression in this direction as long as the loss is decreasing (see discussion above), so we don't analyze these terms further here and instead

focus on the components orthogonal to $\overline{\boldsymbol{r}}$. We then obtain

$$\langle \boldsymbol{h}_1(\boldsymbol{x}')\rangle = \boldsymbol{h}_0(\boldsymbol{x}') - \gamma\sigma_x^2 \left\langle \boldsymbol{\xi}\boldsymbol{\xi}^\top \right\rangle \boldsymbol{W}_0\boldsymbol{x}' + \mathcal{P}(\overline{\boldsymbol{r}}) \tag{6}$$

$$= \boldsymbol{h}_0(\boldsymbol{x}') - \gamma\sigma_x^2\sigma_\xi^2 \boldsymbol{W}_0\boldsymbol{x}' + \mathcal{P}(\overline{\boldsymbol{r}}). \tag{7}$$

If we further assume that the bias is aligned with the readout, $\boldsymbol{b}_0 \propto \overline{\boldsymbol{r}}$, then

$$\langle \boldsymbol{h}_1(\boldsymbol{x}')\rangle = (1 - \gamma\sigma_x^2\sigma_\xi^2)\boldsymbol{h}_0(\boldsymbol{x}') + \mathcal{P}(\overline{\boldsymbol{r}}). \tag{8}$$

This shows that in our simple linear network model, isotropic variability of the readout weights $\boldsymbol{r}$ drives compression in all directions within the subspace orthogonal to $\overline{\boldsymbol{r}}$. See Fig. S2 for a visualization.

In the above simplified analysis, we assumed a form of variability for $\boldsymbol{r}$ and directly "imposed" it. In the following, we explore the impact of true variability in $\boldsymbol{r}$ as driven by SGD (using a batch size of 1 for simplicity). The gradient update for $\boldsymbol{r}$ in this case is $\delta\boldsymbol{r} = -\gamma e(\boldsymbol{x})\boldsymbol{h}(\boldsymbol{x})$. This variability affects the hidden representation only after at least two gradient steps. Assuming that the readouts are at equilibrium $\langle\delta\boldsymbol{r}\rangle = \boldsymbol{0}$, the equation for the updated hidden representation becomes

$$\langle \boldsymbol{h}_2(\boldsymbol{x}')\rangle = \boldsymbol{h}_0(\boldsymbol{x}') - \gamma\sigma_x^2 \left\langle (\delta\boldsymbol{r}_0)(\delta\boldsymbol{r}_0)^\top \right\rangle \boldsymbol{W}_0\boldsymbol{x}' - \gamma\sigma_x^2 \left\langle (\delta\boldsymbol{r}_0)\boldsymbol{r}_0^\top(\delta\boldsymbol{W}_0)\right\rangle \boldsymbol{x}' + \mathcal{P}(\boldsymbol{r}_0) + \mathcal{O}(\gamma^4). \tag{9}$$

Here δ is the update at the first step of SGD, as before. The second term resembles Eq. (6), with $\left\langle (\delta\boldsymbol{r}_0)(\delta\boldsymbol{r}_0)^\top \right\rangle$ replacing $\left\langle \boldsymbol{\xi}\boldsymbol{\xi}^\top \right\rangle$. The third term is an additional cross term that appears due to dependence between the updates to the initial output weights $\boldsymbol{r}_0$ and the initial input weights $\boldsymbol{W}_0$. For this equation to indicate compression, the norm of the right hand side must be less than the norm of $\boldsymbol{h}_0$. While the exact conditions for this to occur are not immediate from the equation, we here outline a set of assumptions that allow it to be analyzed easily, and show that compression occurs in this case. Namely, we assume that $\boldsymbol{W}_0$ is a diagonal matrix, $\boldsymbol{r}_0$ is proportional to a standard basis vector $\boldsymbol{r}_0 \propto \boldsymbol{e}_k$ for some $k \in \{1, 2, ..., N\}$, and $\boldsymbol{b} \propto \boldsymbol{r}_0$. With these assumptions, Eq. (9) decouples into scalar equations

$$(\langle \boldsymbol{h}_2(\boldsymbol{x}')\rangle)_j = \alpha_j(\boldsymbol{h}_0(\boldsymbol{x}'))_j + \mathcal{P}(\boldsymbol{r}_0) + \mathcal{O}(\gamma^4) \tag{10}$$

for $j \neq k$, where $\alpha_j = \left(1 - \gamma^3\sigma_x^4 \left\langle e(\boldsymbol{x})^2\right\rangle \left((\boldsymbol{W}_0)_{jj}^2 + \|\boldsymbol{r}_0\|^2\right)\right)$ is smaller than 1 for γ small enough. Here the w_{jj}^2 and $\|\boldsymbol{r}_0\|^2$ factors come from the second and third terms of Eq. (9), respectively. This reveals that, for $\gamma > 0$ small enough and under the simplifying assumptions above, $\boldsymbol{h}_2$ is on average reduced in all directions orthogonal to the readout by the noise generated by SGD weight updates.

This equation yields some insights. For one, the effect of compression is of order γ^3, which indicates that the effect may be relatively weak for single-layer networks. For another, the compression depends directly on $e(\boldsymbol{x})^2$, so that a network that reduces the error very quickly may not compress as much. The term $\left((\boldsymbol{W}_0)_{jj}^2 + \|\boldsymbol{r}_0\|^2\right)$ shows dependence of compression on the magnitude of the initial parameters.

This simplified analysis gives intuition for how SGD 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 close to the number of classes – a dimension that is typically dramatically compressed when compared with the input data itself (see Fig. S3 for simulations showing dependence of dimensionality on the number of classes). To further support our hypothesis of SGD as a mechanism driving compression, in Fig. S5

we compare the dimensionality compression of our RNN model across two cases: using full batches and using a batch size of one. We find that for mean squared error loss the RNN model only compresses dimensionality with minibatching the minibatching. However, both batch sizes cause dimensionality compression in the case of categorical cross entropy loss combined with tanh nonlinearity, indicating that additional mechanisms could be at play.

CONCLUSIONS AND DISCUSSION

RNNs learn to perform rich operations on their inputs, including expanding and compressing their dimensionality in order solve classification tasks with efficient, compact representations. This behavior relies on the interplay of learning rules with network dynamics. We explore this in two regimes: edge-of-chaos and strongly chaotic dynamics.

Networks in both regimes strongly compress high-dimensional inputs – i.e., those that are initially linearly separable – into distinct sets, one for each class. This behavior also occurs when classifying MNIST (Fig. S4), and has recently been observed in deep networks [12, 49]. We develop mathematical reasoning for why this compression occurs: a linear approximation reveals how SGD itself naturally promotes 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 relatively low-dimensional chaotic attractors (for networks in the strongly chaotic regime). This has implications for coding: networks that are stabilized by training learn to solve the task for all time while networks that remain chaotic partially "reset" after the evaluation period. The former may lend itself to long-term memory; the latter could be useful in flexibly learning new tasks.

In the case of low-dimensional inputs, we observe differences depending on degree of chaos. Networks initialized on the "edge of chaos" lack an effective mechanism for forming high-dimensional representations, and do not achieve linear separability as a result [13]. Highly chaotic networks, on the other hand, naturally expand dimensionality. The beneficial attributes of dimensionality expansion has been explored in feed-forward models such as kernel learning machines [52] and models of olfactory, cerebellar, gustatory, and visual circuits [3, 5, 8, 39, 43, 46, 57], as well as reservoir computing models where recurrent weights are random and untrained [35].

On the other hand, studies have also pointed out the need to constrain dimensionality to enable generalization [18, 35, 57]. We find that SGD balances the expansion induced by chaos with compression through training, resulting in representations that are both linearly separable as well as compact at the readout time, coinciding with good generalization properties. In the most striking cases (Fig. 2e), strongly chaotic networks initially expand, and then compress dimensionality. In the extreme case of spatially localized inputs constrained to two neurons, the strongly chaotic network typically only expands dimensionality, without subsequently compressing it, seemingly with some cost to generalization; still, the ED remains small relative to the size of the network.

In all of these cases, strongly chaotic networks produce ongoing neural variability, a property shared by biological models of neural circuits [24, 30–32, 38, 44, 45, 51, 59] as well as experimental recordings [40, 45, 56]. Our findings suggest that internally generated variability plays a functional role in neural circuits and can add both biological realism as well as computational power to artificial network models.

A clear need for the future is the consideration of a wider range of tasks (c.f. [14]). Also needed is a more complete study a range of input dimensions as well as higher-dimensional

outputs specified by more than two class labels. For a start, we have found that in the case of high-dimensional inputs, the number of class labels strongly modulates the readout dimensionality of the trained RNN, while the number of input clusters does not (Fig. S3). In addition, it remains to extend theoretical arguments for compression induced by SGD 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 inputs. Lastly our results should be generalized to different learning frameworks and models with learning rules other than SGD (see [25] for the case of deep networks trained by an unsupervised learning rule).

Taken together, we find that RNNs learn to balance compression with the natural expansion induced by chaos in a way suitable to the task at hand. These findings invite the further exploration of learning strategies through the lens of modulating dimensionality. The observation that SGD naturally promotes dimensionality compression sheds light on its surprising success in generalizing well (see also [2, 29, 36, 53, 54]). Our observations of compressed *geometry* of learned representations are closely related to prior work on their compressed *information* and allied generalization bounds [53, 54]. Our 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 synergistically coexist (cf. [24, 30, 31, 44]).

METHODS

(See Supplemental Information for more details.)

Model

We consider a recurrent neural network (RNN) model with one hidden layer trained to perform a delayed classification task (Fig. 1). The equation for the hidden unit activations $\boldsymbol{h}_t$ is

$$\boldsymbol{h}_t = \tanh\left(\boldsymbol{W}\boldsymbol{h}_{t-1} + \boldsymbol{x}_t + \boldsymbol{b}^{in}\right) \tag{11}$$

where $\boldsymbol{W}$ is the matrix of recurrent connection weights, $\boldsymbol{x}_t$ is the input, and $\boldsymbol{b}^{in}$ is a bias term. The network is initialized to have zero activation, $\boldsymbol{h}_{-1} = \boldsymbol{0}$. 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 $t_{\text{eval}} = 10$ is converted to a scalar error signal via a categorical cross-entropy loss function. This loss is used to update the recurrent weights $\boldsymbol{W}$ and the output weights $\boldsymbol{W}^{out}$ via SGD.

Inducing Chaos

We initialize $\boldsymbol{W}$ as $\boldsymbol{W} = (1-\varepsilon)\boldsymbol{I} + \varepsilon\boldsymbol{J}$ where $\varepsilon = .01$ ensures smooth dynamics before training. The matrix $\boldsymbol{J}$ has normally distributed entries that scale in magnitude with a coupling strength parameter γ, $J_{ij} \sim \mathcal{N}(0, \gamma^2/N)$. We investigate two dynamical regimes, "edge of chaos" and "strongly chaotic", with gain strength $\gamma = 20$ and $\gamma = 250$, respectively.

ACKNOWLEDGEMENTS

MF is funded by 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, Ali Weber, Nick Steinmetz and Luca Mazzucato for their insights and suggestions. MF would also like to thank Haim Sompolinsky and SueYeon Chung for their mentorship and inspiration.

Appendix: Theoretical arguments for compression

Here we show the full steps of the derivation of the compression caused by stochastic gradient descent. The model again is that of a single-layer linear network, with internal neural activations responding to a single input sample $\boldsymbol{x}$ given by $\boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$, and a scalar output by $\hat{o}(\boldsymbol{x}) = \boldsymbol{r}^\top \boldsymbol{h}(\boldsymbol{x})$. In the following, we assume that $\boldsymbol{x}$ has zero mean and covariance $\sigma_x^2 \boldsymbol{I}$. We consider a squared error loss function $\ell(\boldsymbol{x}) = \frac{1}{2}e(\boldsymbol{x})^2$, where $e(\boldsymbol{x}) = \hat{o}(\boldsymbol{x}) - o(\boldsymbol{x})$ and o maps the input $\boldsymbol{x}$ to its corresponding class label. The gradient descent learning updates over a batch of input samples B are:

$$\delta \boldsymbol{W} = -\frac{\gamma}{|B|} \sum_{\boldsymbol{x} \in B} \frac{\partial \ell(\boldsymbol{x})}{\partial \boldsymbol{W}} = -\frac{\gamma}{|B|} \sum_{\boldsymbol{x} \in B} e(\boldsymbol{x}) \boldsymbol{r} \boldsymbol{x}^\top, \tag{A.1}$$

$$\delta \boldsymbol{b} = -\frac{\gamma}{|B|} \sum_{\boldsymbol{x} \in B} \frac{\partial \ell(\boldsymbol{x})}{\partial \boldsymbol{b}} = -\frac{\gamma}{|B|} \sum_{\boldsymbol{x} \in B} e(\boldsymbol{x}) \boldsymbol{r} \tag{A.2}$$

$$\delta \boldsymbol{r} = -\frac{\gamma}{|B|} \sum_{\boldsymbol{x} \in B} \frac{\partial \ell(\boldsymbol{x})}{\partial \boldsymbol{r}} = -\frac{\gamma}{|B|} \sum_{\boldsymbol{x} \in B} e(\boldsymbol{x}) \boldsymbol{h} \tag{A.3}$$

Here γ is the *learning rate* for the gradient descent routine and $\boldsymbol{r}\boldsymbol{x}^\top$ is the outer product of $\boldsymbol{r}$ and $\boldsymbol{x}$.

Let $\boldsymbol{h}_k$, and e_k denote the hidden representation and error after k steps of gradient descent, respectively, and similarly for parameters $\boldsymbol{W}_k$, $\boldsymbol{b}_k$, and $\boldsymbol{r}_k$. The goal is to compute the hidden representation $\boldsymbol{h}_2(\boldsymbol{x})$ after two steps of SGD and show that it is compressed when compared with the original hidden representation $\boldsymbol{h}_0(\boldsymbol{x})$. For SGD with batch size one, on each step of the gradient descent routine an input sample is chosen randomly to use for backpropagating error and updating parameters. Let $\boldsymbol{x}_0$ and $\boldsymbol{x}_1$ denote the input chosen on the first and second steps of SGD, respectively. As is the usual assumption, we choose $\boldsymbol{x}_0$ and $\boldsymbol{x}_1$ independently. Let $\nabla^k_{\boldsymbol{\theta}} \ell(\boldsymbol{x})$ denote the gradient of ℓ with respect to parameter $\boldsymbol{\theta}$ evaluated at input $\boldsymbol{x}$ and at the parameter values at step k of SGD (that is, parameters $\boldsymbol{W}_k$, $\boldsymbol{b}_k$, and $\boldsymbol{r}_k$). The parameter updates at the first step are then given by $-\gamma \nabla^0_{\boldsymbol{W}} \ell(\boldsymbol{x}_0) = -\gamma e(\boldsymbol{x}_0) \boldsymbol{r}_0 \boldsymbol{x}_0^\top$, $-\gamma \nabla^0_{\boldsymbol{b}} \ell(\boldsymbol{x}_0) = -\gamma e(\boldsymbol{x}_0) \boldsymbol{r}_0$, $-\gamma \nabla^0_{\boldsymbol{r}} \ell(\boldsymbol{x}_0) = -\gamma e(\boldsymbol{x}_0) \boldsymbol{h}_0(\boldsymbol{x}_0)$ and at the second step by $-\gamma \nabla^1_{\boldsymbol{W}} \ell(\boldsymbol{x}_1) = -\gamma e(\boldsymbol{x}_1) \boldsymbol{r}_1 \boldsymbol{x}_1^\top$, $-\gamma \nabla^1_{\boldsymbol{b}} \ell(\boldsymbol{x}_1) = -\gamma e(\boldsymbol{x}_1) \boldsymbol{r}_1$, $-\gamma \nabla^1_{\boldsymbol{r}} \ell(\boldsymbol{x}_1) = -\gamma e(\boldsymbol{x}_1) \boldsymbol{h}_1(\boldsymbol{x}_1)$. Note that the gradient on the second step depends on the value of the parameters after the first step of gradient descent.

The hidden representation on the second step is $\boldsymbol{h}_2(\boldsymbol{x}) = \boldsymbol{W}_2 \boldsymbol{x} + \boldsymbol{b}_2$. The parameters $\boldsymbol{W}_2$ and $\boldsymbol{b}_2$ are related to their initial values via

$$\boldsymbol{W}_2 = \boldsymbol{W}_1 - \gamma \nabla^1_{\boldsymbol{W}} \ell(\boldsymbol{x}_1) = \boldsymbol{W}_0 - \gamma \nabla^0_{\boldsymbol{W}} \ell(\boldsymbol{x}_0) - \gamma \nabla^1_{\boldsymbol{W}} \ell(\boldsymbol{x}_1) \tag{A.4}$$

and

$$\boldsymbol{b}_2 = \boldsymbol{b}_1 - \gamma \nabla^1_{\boldsymbol{b}} \ell(\boldsymbol{x}_1) = \boldsymbol{b}_0 - \gamma \nabla^0_{\boldsymbol{b}} \ell(\boldsymbol{x}_0) - \gamma \nabla^1_{\boldsymbol{b}} \ell(\boldsymbol{x}_1). \tag{A.5}$$

The relationship between $\boldsymbol{h}_2(\boldsymbol{x})$ and $\boldsymbol{h}_0(\boldsymbol{x})$ is then given by

$$\begin{aligned}
\boldsymbol{h}_2(\boldsymbol{x}) &= \boldsymbol{W}_2 \boldsymbol{x} + \boldsymbol{b}_2 \\
&= (\boldsymbol{W}_0 - \gamma \nabla^0_{\boldsymbol{W}} \ell(\boldsymbol{x}_0) - \gamma \nabla^1_{\boldsymbol{W}} \ell(\boldsymbol{x}_1))\boldsymbol{x} + \boldsymbol{b}_0 - \gamma \nabla^0_{\boldsymbol{b}} \ell(\boldsymbol{x}_0) - \gamma \nabla^1_{\boldsymbol{b}} \ell(\boldsymbol{x}_1) \\
&= \boldsymbol{h}_0(\boldsymbol{x}) - \gamma \nabla^0_{\boldsymbol{W}} \ell(\boldsymbol{x}_0)\boldsymbol{x} - \gamma \nabla^1_{\boldsymbol{W}} \ell(\boldsymbol{x}_1)\boldsymbol{x} - \gamma \nabla^0_{\boldsymbol{b}} \ell(\boldsymbol{x}_0) - \gamma \nabla^1_{\boldsymbol{b}} \ell(\boldsymbol{x}_1)
\end{aligned} \tag{A.6}$$

where in the last equation we grouped $\boldsymbol{h}_0(\boldsymbol{x}) = \boldsymbol{W}_0 \boldsymbol{x} + \boldsymbol{b}_0$. Our goal is to relate everything in Eq. (A.6) to initial parameter values. This is straightforward for the gradients $\nabla^0_{\boldsymbol{W}} \ell(\boldsymbol{x}_0)$ and $\nabla^0_{\boldsymbol{b}} \ell(\boldsymbol{x}_0)$: in particular, note that these terms are aligned with $\boldsymbol{r}_0$. We will not be needing to consider terms proportional to $\boldsymbol{r}_0$ for this analysis, so we collect these terms together. For convenience, suppose that P is the orthogonal projector onto the nullspace of $\boldsymbol{r}_0$ and let $\mathcal{P}(\boldsymbol{r}_0)$ denote any quantity that is mapped to zero by this projector (this notation functions similarly to "big O" notation). Examples of objects that are $\mathcal{P}(\boldsymbol{r}_0)$ are vectors proportional to $\boldsymbol{r}_0$ and matrices of the form $\boldsymbol{r}_0 \boldsymbol{v}^\top$. Then $\boldsymbol{h}_2(\boldsymbol{x})$ can be written

$$\boldsymbol{h}_2(\boldsymbol{x}) = \boldsymbol{h}_0(\boldsymbol{x}) - \gamma \nabla^1_{\boldsymbol{W}} \ell(\boldsymbol{x}_1)\boldsymbol{x} - \gamma \nabla^1_{\boldsymbol{b}} \ell(\boldsymbol{x}_1) + \mathcal{P}(\boldsymbol{r}_0) \tag{A.7}$$

Our next step is to average $\boldsymbol{h}_2(\boldsymbol{x})$ over choice of $\boldsymbol{x}_0$ and $\boldsymbol{x}_1$. This results in the expression

$$\langle \boldsymbol{h}_2(\boldsymbol{x}) \rangle = \boldsymbol{h}_0(\boldsymbol{x}) - \gamma \left\langle \nabla^1_{\boldsymbol{W}} \ell(\boldsymbol{x}_1) \right\rangle \boldsymbol{x} - \gamma \left\langle \nabla^1_{\boldsymbol{b}} \ell(\boldsymbol{x}_1) \right\rangle + \mathcal{P}(\boldsymbol{r}_0).$$

We first assume that the bias is at equilibrium, i.e. $\left\langle \nabla^k_{\boldsymbol{b}} \ell(\boldsymbol{x}) \right\rangle_{\boldsymbol{x}} = \boldsymbol{0}$. This leaves us with the expression

$$\langle \boldsymbol{h}_2(\boldsymbol{x}) \rangle = \boldsymbol{h}_0(\boldsymbol{x}) - \gamma \left\langle \nabla^1_{\boldsymbol{W}} \ell(\boldsymbol{x}_1) \right\rangle \boldsymbol{x} + \mathcal{P}(\boldsymbol{r}_0). \tag{A.8}$$

We next focus our attention on computing $\nabla^1_{\boldsymbol{W}} \ell(\boldsymbol{x}_1)$. This update is

$$\nabla^1_{\boldsymbol{W}} \ell(\boldsymbol{x}_1) = -\gamma e_1(\boldsymbol{x}_1) \boldsymbol{r}_1 \boldsymbol{x}_1^\top = -\gamma \left(\boldsymbol{r}_1^\top \left(\boldsymbol{W}_1 \boldsymbol{x}_1 + \boldsymbol{b}_1 \right) - o(\boldsymbol{x}_1) \right) \boldsymbol{r}_1 \boldsymbol{x}_1^\top.$$

Substituting $\boldsymbol{W}_1 = \boldsymbol{W}_0 - \gamma \nabla^0_{\boldsymbol{W}} \ell(\boldsymbol{x}_0)$, $\boldsymbol{r}_1 = \boldsymbol{r}_0 - \gamma \nabla^0_{\boldsymbol{r}} \ell(\boldsymbol{x}_0)$ and $\boldsymbol{b}_1 = \boldsymbol{b}_0 - \gamma \nabla^0_{\boldsymbol{b}} \ell(\boldsymbol{x}_0)$ and simplifying results in the expression

$$\begin{aligned}
\nabla^1_{\boldsymbol{W}} \ell(\boldsymbol{x}_1) = &-\gamma e_1(\boldsymbol{x}_1) \nabla^0_{\boldsymbol{r}} \ell(\boldsymbol{x}_0) \boldsymbol{x}_1^\top \\
&+ \gamma^2 \boldsymbol{r}_0^\top \nabla^0_{\boldsymbol{b}} \ell(\boldsymbol{x}_0) \nabla^0_{\boldsymbol{r}} \ell(\boldsymbol{x}_0) \boldsymbol{x}_1^\top \\
&+ \gamma^2 \boldsymbol{r}_0^\top \nabla^0_{\boldsymbol{W}} \ell(\boldsymbol{x}_0) \boldsymbol{x}_1 \nabla^0_{\boldsymbol{r}} \ell(\boldsymbol{x}_0) \boldsymbol{x}_1^\top \\
&+ \gamma^2 \nabla^0_{\boldsymbol{r}} \ell(\boldsymbol{x}_0)^\top \boldsymbol{h}_0(\boldsymbol{x}_1) \nabla^0_{\boldsymbol{r}} \ell(\boldsymbol{x}_0) \boldsymbol{x}_1^\top) \\
&+ \mathcal{P}(\boldsymbol{r}_0) + \mathcal{O}(\gamma^3)
\end{aligned}$$

To simplify this expression, we collect terms of order $\mathcal{O}(\gamma^3)$ and again collect terms aligned with $\boldsymbol{r}_0$. This simplification results in

$$\begin{aligned}
\nabla^1_{\boldsymbol{W}} \ell(\boldsymbol{x}_1) = &-\gamma e_1(\boldsymbol{x}_1) \nabla^0_{\boldsymbol{r}} \ell(\boldsymbol{x}_0) \boldsymbol{x}_1^\top \\
&+ \gamma^2 \boldsymbol{r}_0^\top \nabla^0_{\boldsymbol{b}} \ell(\boldsymbol{x}_0) \nabla^0_{\boldsymbol{r}} \ell(\boldsymbol{x}_0) \boldsymbol{x}_1^\top \\
&+ \gamma^2 \boldsymbol{r}_0^\top \nabla^0_{\boldsymbol{W}} \ell(\boldsymbol{x}_0) \boldsymbol{x}_1 \nabla^0_{\boldsymbol{r}} \ell(\boldsymbol{x}_0) \boldsymbol{x}_1^\top \\
&+ \gamma^2 \nabla^0_{\boldsymbol{r}} \ell(\boldsymbol{x}_0)^\top \boldsymbol{h}_0(\boldsymbol{x}_1) \nabla^0_{\boldsymbol{r}} \ell(\boldsymbol{x}_0) \boldsymbol{x}_1^\top \\
&+ \mathcal{P}(\boldsymbol{r}_0) \boldsymbol{x}_1^\top + \mathcal{O}(\gamma^3)
\end{aligned}$$

Now we compute the average of this update, $\langle \nabla^1_{\boldsymbol{W}} \ell(\boldsymbol{x}_1) \rangle$. This is

$$\begin{aligned}
\left\langle \nabla^1_{\boldsymbol{W}} \ell(\boldsymbol{x}_1) \right\rangle = &-\gamma \left\langle e_1(\boldsymbol{x}_1) \nabla^0_{\boldsymbol{r}} \ell(\boldsymbol{x}_0) \boldsymbol{x}_1^\top \right\rangle \\
&+ \gamma^2 \left\langle \boldsymbol{r}_0^\top \nabla^0_{\boldsymbol{b}} \ell(\boldsymbol{x}_0) \nabla^0_{\boldsymbol{r}} \ell(\boldsymbol{x}_0) \boldsymbol{x}_1^\top \right\rangle \\
&+ \gamma^2 \left\langle \boldsymbol{r}_0^\top \nabla^0_{\boldsymbol{W}} \ell(\boldsymbol{x}_0) \boldsymbol{x}_1 \nabla^0_{\boldsymbol{r}} \ell(\boldsymbol{x}_0) \boldsymbol{x}_1^\top \right\rangle \\
&+ \gamma^2 \left\langle \nabla^0_{\boldsymbol{r}} \ell(\boldsymbol{x}_0)^\top \boldsymbol{h}_0(\boldsymbol{x}_1) \nabla^0_{\boldsymbol{r}} \ell(\boldsymbol{x}_0) \boldsymbol{x}_1^\top \right\rangle \\
&+ \left\langle \mathcal{P}(\boldsymbol{r}_0) \boldsymbol{x}_1^\top \right\rangle + \mathcal{O}(\gamma^3)
\end{aligned}$$

To proceed, we assume that the output weights are at equilibrium, i.e. $\langle \nabla^0_{\boldsymbol{r}} \ell(\boldsymbol{x}) \rangle_{\boldsymbol{x}} = \boldsymbol{0}$. This removes the first term from the expression for $\langle \nabla^1_{\boldsymbol{W}} \ell(\boldsymbol{x}_1) \rangle$. Furthermore, we assume that the input is mean zero and isotropic with variance σ_x^2, so $\langle \boldsymbol{x} \rangle = 0$ and $\left\langle \boldsymbol{x}\boldsymbol{x}^\top \right\rangle = \sigma_x^2 \boldsymbol{I}$. Using these assumptions below, along with some rearranging, we find that the input weight updates satisfy

$$\begin{aligned}
\left\langle \nabla^1_{\boldsymbol{W}} \ell(\boldsymbol{x}_1) \right\rangle = &-\gamma \left\langle e_1(\boldsymbol{x}_1) \left\langle \nabla^0_{\boldsymbol{r}} \ell(\boldsymbol{x}_0) \right\rangle \boldsymbol{x}_1^\top \right\rangle \\
&+ \gamma^2 \left\langle \boldsymbol{r}_0^\top \nabla^0_{\boldsymbol{b}} \ell(\boldsymbol{x}_0) \nabla^0_{\boldsymbol{r}} \ell(\boldsymbol{x}_0) \right\rangle \left\langle \boldsymbol{x}_1^\top \right\rangle \\
&+ \gamma^2 \left\langle \boldsymbol{r}_0^\top \nabla^0_{\boldsymbol{W}} \ell(\boldsymbol{x}_0) \boldsymbol{x}_1 \nabla^0_{\boldsymbol{r}} \ell(\boldsymbol{x}_0) \boldsymbol{x}_1^\top \right\rangle \\
&+ \gamma^2 \left\langle \nabla^0_{\boldsymbol{r}} \ell(\boldsymbol{x}_0)^\top \boldsymbol{h}_0(\boldsymbol{x}_1) \nabla^0_{\boldsymbol{r}} \ell(\boldsymbol{x}_0) \boldsymbol{x}_1^\top \right\rangle \\
&+ \left\langle \mathcal{P}(\boldsymbol{r}_0) \boldsymbol{x}_1^\top \right\rangle + \mathcal{O}(\gamma^3) \\
= &\gamma^2 \left\langle \boldsymbol{r}_0^\top \nabla^0_{\boldsymbol{W}} \ell(\boldsymbol{x}_0) \boldsymbol{x}_1 \nabla^0_{\boldsymbol{r}} \ell(\boldsymbol{x}_0) \boldsymbol{x}_1^\top \right\rangle \\
&+ \gamma^2 \left\langle \nabla^0_{\boldsymbol{r}} \ell(\boldsymbol{x}_0)^\top \boldsymbol{h}_0(\boldsymbol{x}_1) \nabla^0_{\boldsymbol{r}} \ell(\boldsymbol{x}_0) \boldsymbol{x}_1^\top \right\rangle \\
&+ \left\langle \mathcal{P}(\boldsymbol{r}_0) \boldsymbol{x}_1^\top \right\rangle + \mathcal{O}(\gamma^3) \\
= &\gamma^2 \left\langle \nabla^0_{\boldsymbol{r}} \ell(\boldsymbol{x}_0) \boldsymbol{r}_0^\top \nabla^0_{\boldsymbol{W}} \ell(\boldsymbol{x}_0) \right\rangle \left\langle \boldsymbol{x}_1 \boldsymbol{x}_1^\top \right\rangle \\
&+ \gamma^2 \left\langle \nabla^0_{\boldsymbol{r}} \ell(\boldsymbol{x}_0) \nabla^0_{\boldsymbol{r}} \ell(\boldsymbol{x}_0)^\top \right\rangle \left\langle \boldsymbol{h}_0(\boldsymbol{x}_1) \boldsymbol{x}_1^\top \right\rangle \\
&+ \left\langle \mathcal{P}(\boldsymbol{r}_0) \boldsymbol{x}_1^\top \right\rangle + \mathcal{O}(\gamma^3) \\
= &\gamma^2 \sigma_x^2 \left\langle \nabla^0_{\boldsymbol{r}} \ell(\boldsymbol{x}_0) \boldsymbol{r}_0^\top (\nabla^0_{\boldsymbol{W}} \ell(\boldsymbol{x}_0)) \right\rangle \\
&+ \gamma^2 \left\langle \nabla^0_{\boldsymbol{r}} \ell(\boldsymbol{x}_0) \left(\nabla^0_{\boldsymbol{r}} \ell(\boldsymbol{x}_0) \right)^\top \right\rangle \left\langle \boldsymbol{h}_0(\boldsymbol{x}_1) \boldsymbol{x}_1^\top \right\rangle \\
&+ \left\langle \mathcal{P}(\boldsymbol{r}_0) \boldsymbol{x}_1^\top \right\rangle + \mathcal{O}(\gamma^3)
\end{aligned}$$

Regarding the term with $\left\langle \boldsymbol{h}_0(\boldsymbol{x}_1) \boldsymbol{x}_1^\top \right\rangle$, we compute that

$$\left\langle \boldsymbol{h}_0(\boldsymbol{x}_1) \boldsymbol{x}_1^\top \right\rangle = \left\langle (\boldsymbol{W}\boldsymbol{x}_1 + \boldsymbol{b}_0) \boldsymbol{x}_1^\top \right\rangle = \boldsymbol{W} \left\langle \boldsymbol{x}_1 \boldsymbol{x}_1^\top \right\rangle = \sigma_x^2 \boldsymbol{W}. \tag{A.9}$$

This leaves us with the expression

$$\begin{aligned}
\left\langle \nabla^1_{\boldsymbol{W}} \ell(\boldsymbol{x}_1) \right\rangle = &\gamma^2 \sigma_x^2 \left\langle \nabla^0_{\boldsymbol{r}} \ell(\boldsymbol{x}_0) \boldsymbol{r}_0^\top (\nabla^0_{\boldsymbol{W}} \ell(\boldsymbol{x}_0)) \right\rangle \\
&+ \gamma^2 \sigma_x^2 \left\langle \nabla^0_{\boldsymbol{r}} \ell(\boldsymbol{x}_0) \nabla^0_{\boldsymbol{r}} \ell(\boldsymbol{x}_0)^\top \right\rangle \boldsymbol{W} \\
&+ \left\langle \mathcal{P}(\boldsymbol{r}_0) \boldsymbol{x}_1^\top \right\rangle + \mathcal{O}(\gamma^3)
\end{aligned} \tag{A.10}$$

Substituting this expression for $\langle \nabla^1_{\boldsymbol{W}} \ell(\boldsymbol{x}_1) \rangle$ into Eq. (A.8) and simplifying results in

$$\begin{aligned}
\langle \boldsymbol{h}_2(\boldsymbol{x}) \rangle = &\boldsymbol{h}_0(\boldsymbol{x}) \\
&- \gamma^3 \sigma_x^2 \left\langle \nabla^0_{\boldsymbol{r}} \ell(\boldsymbol{x}_0) \nabla^0_{\boldsymbol{r}} \ell(\boldsymbol{x}_0)^\top \right\rangle \boldsymbol{W}_0 \boldsymbol{x} \\
&- \gamma^3 \sigma_x^2 \left\langle \nabla^0_{\boldsymbol{r}} \ell(\boldsymbol{x}_0) \boldsymbol{r}_0^\top \nabla^0_{\boldsymbol{W}} \ell(\boldsymbol{x}_0) \right\rangle \boldsymbol{x} + \mathcal{P}(\boldsymbol{r}_0) + \mathcal{O}(\gamma^4).
\end{aligned} \tag{A.11}$$

This is a full reduction of the behavior of the representation after two SGD steps in task-irrelevant directions orthogonal to the readouts, up to order $\mathcal{O}(\gamma^4)$, under mild assumptions on the input (mean zero and isotropic) and on the gradients of the output and bias weights (that they are in equilibrium). While a systematic analysis of Eq. (A.11) is desirable, in this work we instead make strong simplifying assumptions that make the equation easy to analyze. The assumptions we make are: (1) $\boldsymbol{W}_0$ is diagonal, (2) $\boldsymbol{r}_0 \propto \boldsymbol{e}_k$ is proportional to a standard unit vector $\boldsymbol{e}_k$, the vector of all zeros except for a one in the kth entry, for some $k \in \{1, 2, ..., N\}$, and (3) $\boldsymbol{b}_0 \propto \boldsymbol{r}_0$. Let's assume without loss of generality that $\boldsymbol{r}_0 \propto \boldsymbol{e}_1$.

We first address the second term in Eq. (A.11) by computing

$$\begin{aligned}\left\langle \nabla^0_{\boldsymbol{r}} \ell(\boldsymbol{x}_0)(\nabla^0_{\boldsymbol{r}} \ell(\boldsymbol{x}_0))^\top \right\rangle = &\boldsymbol{W}_0 \left\langle e(\boldsymbol{x}_0)^2 \boldsymbol{x}_0 \boldsymbol{x}_0^\top \right\rangle \boldsymbol{W}_0^\top \\ &+ \left\langle e(\boldsymbol{x}_0)^2 \boldsymbol{W}_0 \boldsymbol{x}_0 \right\rangle \boldsymbol{b}_0^\top \boldsymbol{x}. \\ &+ \left\langle e(\boldsymbol{x}_0)^2 \boldsymbol{b}_0 \boldsymbol{x}_0^\top \right\rangle \boldsymbol{W}_0^\top \\ &+ \left\langle e(\boldsymbol{x}_0)^2 \right\rangle \boldsymbol{b}_0 \boldsymbol{b}_0^\top \\ &+ \mathcal{P}(\boldsymbol{r}_0).\end{aligned}$$

We first deal with the second term of the above expression. Note that by assumptions (1)-(3), $e(\boldsymbol{x}_0) = \boldsymbol{r}^\top(\boldsymbol{W}_0\boldsymbol{x}_0 + \boldsymbol{b}_0) - o(\boldsymbol{x}_0)$ depends only on the first coordinate of $\boldsymbol{x}_0$. Since $\langle \boldsymbol{x}_0 \rangle = \boldsymbol{0}$, it follows that $\langle e(\boldsymbol{x}_0)^2 \boldsymbol{W}_0 \boldsymbol{x}_0 \rangle \propto \boldsymbol{e}_1 \propto \boldsymbol{r}_0$. The third and fourth terms are also proportional to $\boldsymbol{r}_0$ by assumption (3). Hence

$$\left\langle \nabla^0_{\boldsymbol{r}} \ell(\boldsymbol{x}_0)(\nabla^0_{\boldsymbol{r}} \ell(\boldsymbol{x}_0))^\top \right\rangle = \boldsymbol{W}_0 \left\langle e(\boldsymbol{x}_0)^2 \boldsymbol{x}_0 \boldsymbol{x}_0^\top \right\rangle \boldsymbol{W}_0^\top + \mathcal{P}(\boldsymbol{r}_0).$$

We now finish computing the second term of Eq. (A.11) under assumptions (1)-(3). Using again that $e(\boldsymbol{x}_0)$ depends only on the first coordinate of $\boldsymbol{x}_0$, as well as the fact that the distinct coordinates of $\boldsymbol{x}_0$ are independent, it's straightforward to show that

$$\boldsymbol{W}_0 \left\langle e(\boldsymbol{x}_0)^2 \boldsymbol{x}_0 \boldsymbol{x}_0^\top \right\rangle \boldsymbol{W}_0^\top \boldsymbol{W}_0 \boldsymbol{x} = \left\langle e(\boldsymbol{x}_0)^2 \right\rangle \sigma_x^2 \boldsymbol{W}_0 \boldsymbol{W}_0^\top \boldsymbol{W}_0 \boldsymbol{x} + \mathcal{P}(\boldsymbol{r}_0).$$

Hence

$$\left\langle \nabla^0_{\boldsymbol{r}} \ell(\boldsymbol{x}_0)(\nabla^0_{\boldsymbol{r}} \ell(\boldsymbol{x}_0))^\top \right\rangle \boldsymbol{W}_0 \boldsymbol{x} = \boldsymbol{W}_0 \left\langle e(\boldsymbol{x}_0)^2 \boldsymbol{x}_0 \boldsymbol{x}_0^\top \right\rangle \boldsymbol{W}_0^\top \boldsymbol{W}_0 \boldsymbol{x} + \mathcal{P}(\boldsymbol{r}_0) = \left\langle e(\boldsymbol{x}_0)^2 \right\rangle \sigma_x^2 \boldsymbol{W}_0 \boldsymbol{W}_0^\top \boldsymbol{W}_0 \boldsymbol{x} + \mathcal{P}(\boldsymbol{r}_0). \tag{A.12}$$

Calculations for the third term of Eq. (A.11) follow a similar flow, resulting in

$$\left\langle (\nabla^0_{\boldsymbol{r}} \ell(\boldsymbol{x}_0)) \boldsymbol{r}_0^\top (\nabla^0_{\boldsymbol{W}} \ell(\boldsymbol{x}_0) \right\rangle \boldsymbol{x} = \sigma_x^2 \left\| \boldsymbol{r}_0 \right\|^2 \left\langle e(\boldsymbol{x}_0)^2 \right\rangle \boldsymbol{W}_0 \boldsymbol{x} + \mathcal{P}(\boldsymbol{r}_0). \tag{A.13}$$

Substituting Eq. (A.13) and Eq. (A.12) into Eq. (A.11) results in

$$\begin{aligned}\langle \boldsymbol{h}_2(\boldsymbol{x}) \rangle = &\boldsymbol{h}_0(\boldsymbol{x}) \\ &- \gamma^3 \sigma_x^4 \left\langle e(\boldsymbol{x}_0)^2 \right\rangle \boldsymbol{W}_0 \boldsymbol{W}_0^\top \boldsymbol{W}_0 \boldsymbol{x} \\ &- \gamma^3 \sigma_x^4 \|\boldsymbol{r}_0\|^2 \left\langle e(\boldsymbol{x}_0)^2 \right\rangle \boldsymbol{W}_0 \boldsymbol{x} + \mathcal{P}(\boldsymbol{r}_0) + \mathcal{O}(\gamma^4).\end{aligned}$$

Note that our assumption (3) that $\boldsymbol{b}_0 \propto \boldsymbol{r}_0$ implies that $\boldsymbol{W}_0 \boldsymbol{x} = \boldsymbol{h}_0(\boldsymbol{x}) + \mathcal{P}(\boldsymbol{r}_0)$. Furthermore, since $\boldsymbol{r}_0 \propto \boldsymbol{e}_1$ by assumption (2) and $\boldsymbol{W}_0$ is diagonal by assumption (1), it follows that

$\boldsymbol{W}_0\boldsymbol{W}_0^\top \boldsymbol{b}_0 \propto \boldsymbol{e}_1$ so that $\boldsymbol{W}_0\boldsymbol{W}_0^\top(\boldsymbol{W}_0\boldsymbol{x}+\boldsymbol{b}_0) = \boldsymbol{W}_0\boldsymbol{W}_0^\top\boldsymbol{W}_0\boldsymbol{x} + \mathcal{P}(\boldsymbol{r}_0)$. Hence

$$\begin{aligned}
\langle \boldsymbol{h}_2(\boldsymbol{x})\rangle &= \boldsymbol{h}_0(\boldsymbol{x}) \\
&\quad - \gamma^3\sigma_x^4 \left\langle e(\boldsymbol{x}_0)^2\right\rangle \boldsymbol{W}_0\boldsymbol{W}_0^\top\boldsymbol{W}_0\boldsymbol{x} \\
&\quad - \gamma^3\sigma_x^4\|\boldsymbol{r}_0\|^2 \left\langle e(\boldsymbol{x}_0)^2\right\rangle \boldsymbol{W}_0\boldsymbol{x} + \mathcal{P}(\boldsymbol{r}_0) + \mathcal{O}(\gamma^4) \\
&= \boldsymbol{h}_0(\boldsymbol{x}) \\
&\quad - \gamma^3\sigma_x^4 \left\langle e(\boldsymbol{x}_0)^2\right\rangle \boldsymbol{W}_0\boldsymbol{W}_0^\top\boldsymbol{h}_0(\boldsymbol{x}) \\
&\quad - \gamma^3\sigma_x^4\|\boldsymbol{r}_0\|^2 \left\langle e(\boldsymbol{x}_0)^2\right\rangle \boldsymbol{h}_0(\boldsymbol{x}) + \mathcal{P}(\boldsymbol{r}_0) + \mathcal{O}(\gamma^4) \\
&= \left(\boldsymbol{I} - \gamma^3\sigma_x^4 \left\langle e(\boldsymbol{x}_0)^2\right\rangle \left(\boldsymbol{W}_0\boldsymbol{W}_0^\top + \|\boldsymbol{r}_0\|^2\boldsymbol{I}\right)\right)\boldsymbol{h}_0(\boldsymbol{x}) + \mathcal{P}(\boldsymbol{r}_0) + \mathcal{O}(\gamma^4)
\end{aligned}$$

This shows that for γ small enough, the hidden representation is scaled by a positive constant less than one in the directions orthogonal to $\boldsymbol{r}_0$. This shows that the representation is compressed.

---

[1] Hierarchical models of object recognition in cortex. *Nature Neuroscience*, 2(11):1019–1025, 1999.

[2] 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.

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

[4] Sanjeev Arora, Rong Ge, Behnam Neyshabur, and Yi Zhang. Stronger generalization bounds for deep nets via a compression approach. *arXiv:1802.05296 [cs]*, February 2018. arXiv: 1802.05296.

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

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

[7] 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.

[8] 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.

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

[10] SueYeon Chung, Daniel D. Lee, and Haim Sompolinsky. Linear readout of object manifolds. *Phys. Rev. E*, 93:060301, Jun 2016.

[11] SueYeon Chung, Daniel D. Lee, and Haim Sompolinsky. Classification and Geometry of General Perceptual Manifolds. *Physical Review X*, 8(3):031003, July 2018.

[12] Uri Cohen, SueYeon Chung, Daniel D. Lee, and Haim Sompolinsky. Separability and Geometry of Object Manifolds in Deep Neural Networks. *bioRxiv*, page 644658, May 2019.

[13] 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.

[14] 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.

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

[16] 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.

[17] 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.

[18] 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.

[19] 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.

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

[21] Ian Goodfellow, Honglak Lee, Quoc V. Le, Andrew Saxe, and Andrew Y. Ng. Measuring Invariances in Deep Networks. *Advances in Neural Information Processing Systems 22*, pages 646–654, 2009.

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

[23] G. E. Hinton and R. R. Salakhutdinov. Reducing the Dimensionality of Data with Neural Networks. *Science*, 313(5786):504–507, 2006.

[24] Chengcheng Huang, Douglas A. Ruff, Ryan Pyle, Robert Rosenbaum, Marlene R. Cohen, and Brent Doiron. Circuit Models of Low-Dimensional Shared Variability in Cortical Networks. *Neuron*, 101(2):337–348.e4, January 2019.

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

[26] 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.

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

[28] 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.

[29] 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.

[30] 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.

[31] 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.

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

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

[34] Yann LeCun, Léon Bottou, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, September 1998.

[35] 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.

[36] 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.

[37] 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.

[38] 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.

[39] 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.

[40] M. London, A. Roth, L. Beeren, M. Häusser, 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.

[41] Wolfgang Maass, Thomas Natschläger, 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.

[42] 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.

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

[44] Francesca Mastrogiuseppe and Srdjan Ostojic. Linking Connectivity, Dynamics, and Computations in Low-Rank Recurrent Neural Networks. *Neuron*, 99(3):609–623.e29, August 2018.

[45] Luca Mazzucato, Alfredo Fontanini, and Giancarlo La Camera. Dynamics of multistable states during ongoing and evoked cortical activity. *Journal of Neuroscience*, 35(21):8214–8231, 2015.

[46] Luca Mazzucato, Alfredo Fontanini, and Giancarlo La Camera. Stimuli reduce the dimensionality of cortical activity. *Frontiers in Systems Neuroscience*, 10:11, 2016.

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

[48] Kanaka Rajan, L. F. Abbott, and H. Sompolinsky. Stimulus-dependent suppression of chaos in recurrent neural networks. *Physical Review E - Statistical, Nonlinear, and Soft Matter Physics*, 82(1):1–5, 2010.

[49] Stefano Recanatesi, Matthew Farrell, Madhu Advani, Timothy Moore, Guillaume Lajoie, and Eric Shea-Brown. Dimensionality compression and expansion in Deep Neural Networks.

arXiv:1906.00443 [cs, stat], June 2019. arXiv: 1906.00443.

[50] 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, May 2013.

[51] 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).

[52] B. Schölkopf 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.

[53] Ravid Shwartz-Ziv, Amichai Painsky, and Naftali Tishby. Representation Compression and Generalization in Deep Neural Networks, 2019.

[54] Ravid Shwartz-Ziv and Naftali Tishby. Opening the Black Box of Deep Neural Networks via Information. March 2017.

[55] 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).

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

[57] 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.

[58] David Sussillo and L.F. Abbott. Generating Coherent Patterns of Activity from Chaotic Neural Networks. *Neuron*, 63(4):544–557, 2009.

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

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

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

[62] 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.

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