Revealing structure in trained neural networks through dimensionality-based methods

Matthew S Farrell

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2020

Reading Committee:

Eric Shea-Brown, Chair

Stefan Mihalas

Adrienne Fairhall

Program Authorized to Offer Degree:
Applied Mathematics

University of Washington

Abstract

Revealing structure in trained neural networks through dimensionality-based methods

Matthew S Farrell

Chair of the Supervisory Committee:
Professor Eric Shea-Brown
Applied Mathematics

Neural networks trained by machine learning optimization methods are currently being analyzed to shed light on brain function. While exciting progress is being made, the complicated nature of the network models typically considered has made "opening the black box" a significant challenge. In this thesis I approach the problem by starting with network models and tasks that can understood more easily, but that capture fundamental elements of more complex models. I reveal new aspects of the behavior of these models through the lens of effective dimensionality, which quantifies the number of axes needed to describe data. Through this investigation a new idea of "dimensionality balance" emerges, where neural networks trained with stochastic gradient descent automatically strike a balance between increasing dimensionality (to more easily distinguish between different objects) and decreasing dimensionality (to build invariance to different examples of the same object). Mathematical analysis reveals the core mechanisms that may underlie the effects, and experiments with the image classification network VGG indicate that this balance is a general phenomenon. Finally, I demonstrate how and why dimensionality reduction methods can be used to extract information from network weights in a simple model, laying some guiding principles for ways of extracting insights from the recent explosion of brain connectomics data.

TABLE OF CONTENTS

ACKNOWLEDGMENTS

I want to thank my family for being a constant source of love, support and encouragement. My journey to scientific and mathematical inquiry begins here, as a young child, in conversations with my uncle about physics and the wonders of the universe. I remember these late-night talks fondly. Thank you to my father for doing so much to make sure the spark of curiosity in me always had room to grow. Your support, encouragement, and appreciation for my path in life means a great deal. Thank you to my mother, for your care; for seeing and encouraging good character all through my life; and serving as a role model for how to treat others with kindness and integrity. Thank you to my sister Elizabeth, for providing a fun place to get away during my PhD years and for all the fun conversations about life. Thank you to my brother Nathan for being a friend that I could always go to for fun and time to unwind. Our online connection was an extremely welcome tether to something positive that I could always turn to. Your artistic creations are always a source of joy and awe. Thank you to my sister Nicole for all the love you send my way. You brighten my life with your amazing exploits.

Thank you to my coworkers, my desk mates, my group, my cohort. Whether in Lewis Hall or Health Sciences Center, the community of such good-natured, thoughtful, kind, and all-around awe-inspiring peers has set the bar for me. Thanks especially to Jithin for the ice cream runs and for helping me recalibrate my perspective in a positive direction.

Thank you to my friends living afar – whether we are in close contact or not, the memory of our time together and the knowledge that you are out there ready to take it up again is a constant comfort for me. Thank you especially to Rowdy for hosting my occasional weekend escapes. Thank you also to Sally for a fun final year full of adventures.

Thank you to Lionel Levine, for your amazing guidance during my final undergraduate years. Our research project was a hugely positive experience for me. Thank you to Haim Sompolinsky and SueYeon Chung for guiding me in a summer project that helped pave a foundation for much of the work here.

Thank you to my team – my collaborator Guillaume Lajoie for yanking me into the field

of neural networks with such inspiring work and for the wonderful guidance and help; my collaborator Stefano Recanatesi, for working closely with me on my projects and always pushing things toward the next level; and my unofficial co-mentor Stefan Mihalas, for being a constant source of enthusiasm and clear and thoughtful guidance.

And thank you to my mentor Eric Shea-Brown, for being such an encouraging caterer to my appetite for exploration. You gave me the greatest gift a PhD student could ask for – a space to find and pick projects because they are the most interesting things I could imagine working on. Thank you for tirelessly working with me on this journey, for always encouraging me, and for always being in my corner.

DEDICATION

To my family

Chapter 0
INTRODUCTION

The human brain is widely appreciated as being the most sophisticated computational engine we know of. It is an exciting object to study, as theories describing the basic computational principles by which the brain operates are still nascent. History indicates that a richer description of these principles is likely to drive revolutions in solving problems of real-world significance. For example, artificial neural networks, which shamelessly borrow inspiration from their biological analogues, are revolutionizing the applications of computer science to solving real-world problems.

0.0.1 The brain

The brain hosts a multitude of important biological processes, many of which coordinate with the rest of the body. The brain is both a focal point for incoming sensory information, which collects and travels via nerves and nervous tissue throughout the body, as well as a "command center" that sends signals to exert control of organs and muscle tissue. The complete process by which incoming sensory information is transformed into outgoing command signals is still largely a mystery and the focus of intense study. The allure is clear: understanding this process will likely unlock the brain's computational power for use in artificial intelligence systems, and will also help in the discovery of treatments for subtle disorders of the brain such as depression.

The primary processing unit of the brain is the neuron. Neurons are connected together in an intricate network, and the passing of information signals between these neurons is apparently the main vessel for computation. Fig 1a shows a depiction of Purkinje cell neurons and the large number of incoming connections that each neuron receives, due to Ramón y Cajal [1]. The information passing between neurons takes the form of spikes – when a neuron receives a strong enough signal from its incoming connections, it spikes, sending signal pulses along outgoing connections to other neurons. It should be stressed that incoming signal pulses are not simply summed, but combined in a complex and nonlinear fashion. While a great deal of structure is apparent in animal brains, there is also a large

degree of decentralization and disorder, unlike the circuits found in central processing units. This decentralization is evinced by the brain's incredible robustness to damage. Large areas of the human brain are sometimes destroyed while leaving the basic ability of the person to think and function intact. The effects of the damage are often specific, linked to the alleged role of the area that was destroyed, and killing any one neuron in a mammalian brain is unlikely to have a noticeable effect.

The primary way animals learn is thought to be via the modification of the strength of connections between neurons.[1] A stronger connection means a larger signal is sent over that connection during a spiking event. A multitude of rules have been discovered in the brain that govern how the connections are modulated. The most well-known is the basic Hebbian learning rule [2], where the connection between two neurons is strengthened when both neurons spike sufficiently close together in time. It is still an open question as to how the brain adjusts connections in order to learn (for a review see [3]).

0.0.2 Artificial neural networks

Our (admittedly vague) understanding of the brain's approach to computation has given rise to a breed of brain-inspired artificial learning machines that are having a tremendous impact on the world: artificial neural networks (ANNs).[2] This began with the invention of the perceptron in 1958 by Frank Rosenblatt [4], depicted in Fig 1b.[3] This simple model already employs many ideas fundamental to the field of ANNs, and so it is worth describing in some detail. The perceptron network consists of a single output "neuron" (referred to as a *neural unit*, or simply *unit*) that receives an array of inputs. These inputs could be, for instance, the pixel intensity values of greyscale pictures of cats and dogs. Each input is labeled with either a 0 or 1; for instance, 0 for a picture of a cat, and 1 for a picture of a dog. The input is "shown" to the network: the output unit multiplies each input unit's value by the weight incident to that input unit, and then sums the result over the input units. A *bias*

[1] Here when I say "learn" I mean to learn something in a relatively permanent way, as opposed to short-term memory

[2] As is standard I use ANN to mean an artificial network whose connections are adjusted by some training process. There is also an extensive history of using untrained artificial networks to model brain circuits that falls outside the scope of this narrative.

[3] The first step could also be considered the McCulloch-Pitts neuron conceptualized by Warren McCulloch and Walter Pitts in 1943. The perceptron is a particular McCulloch-Pitts neuron with the addition of learning rules.

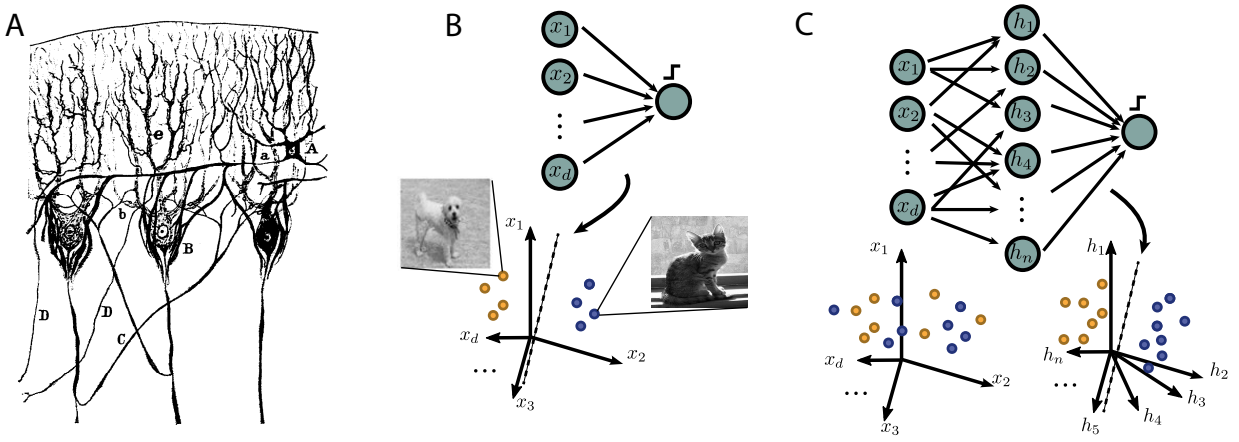

Figure 1: **Neural networks, biological and artificial.** A) Depiction of Purkinje cell neurons (center) and the incoming (top) and outgoing (bottom) connections, due to Ramón y Cajal [1]. Note that the outgoing connections branch out at some point if the connection is followed further, which isn't depicted here. B) Schematic of a perceptron network (top) and inputs being classified by the network (bottom). The weights of the network are incrementally adjusted until the data are correctly classified. This classification corresponds with finding a hyperplane that divides the two classes (dashed line). C) Schematic of a kernel perceptron (top). An intermediate layer is added to the perceptron in (B). The inputs (bottom left) are now depicted such that they may not be linearly separable. The intermediate layer transforms the input into a representation that is linearly separable (bottom right). The output weights are then learned as in a standard perceptron, but using the intermediate layer representation as the input.

term is then added to this sum. Finally, the resulting value is thresholded: 1 if the sum is positive, and 0 otherwise. This thresholded value is then the output of the network. This output is compared with the label for the input, and the weights are adjusted by a particular learning rule based on the discrepancy. This learning rule eventually results in the network producing outputs that match the labels, provided the inputs are *linearly separable*; that is, the points belonging to the different classes can be separated by a hyperplane.

The need for linear separability can be overcome by adding an additional layer to the network. The simplest implementation of this idea is the kernel perceptron [5], which adds an extra set of fixed input weights and "hidden" units with some nonlinear activation function. With enough hidden units, this generically results in a transformed *representation* of the inputs that is linearly separable, a result known as Cover's theorem [6]. This linearly separa-

ble representation can then be classified by adjusting the output weights via the perceptron algorithm, using the transformed representation as inputs. The kernel perceptron set the foundation for the development of the highly influential Support Vector Machine [7, 8, 9, 10], which in its simplest form is a perceptron with a different output unit nonlinearity and training algorithm. However, it was some time before the kernel perspective was also extended to the Support Vector Machine [10]. The fact that the input weights of kernel perceptrons and kernel support vector machines need not be fixed, but can also be efficiently learned, was not appreciated for another decade or so. This insight is the foundation for *deep* neural networks which stack many layers together for increased computational power.

Typically in ANNs, as in the perceptron, complicated details about the biophysics of real neurons and the transmission of signals is ignored, and the remaining richness and power of the models comes from the large number of connections (or "weights") between the units. This emphasis on connectivity patterns as being the essential elements of brain computation is known as *connectionism* [11].

0.0.3 ANNs as models of the brain

A natural conclusion of the connectionist hypothesis is that ANNs capture the essential elements of brain computation. This provides hope that by developing and studying ANNs, we may gain deeper insights into brain function (for reviews of this approach see [12, 13, 14]).

To start, it is helpful to briefly consider the type of artificial neural networks we need to model the different parts of the brain. In many places the brain appears to be structurally organized in a rough hierarchy. The visual system is a good example of this. The seminal work of Felleman and Van Essen [15] used connectivity data to lay out an organization of areas of the brain involved in the processing of visual information. This organization reveals that visual information passes through stages, from one area or set of areas to the next. Using deep neural networks with multiple layers as models is one possible way to gain insight into to the brain's hierarchical processing of visual information.

However, this organization is far from strictly feedforward. It also includes a large number of *feedback* connections, which are connections from higher order areas to lower ones. Neurons within the same area also send many connections to each other. In network models such as ANNs, connections between neural units on the same level of hierarchy are referred to as *recurrent* connections. Networks that have recurrent connections are known as *recurrent neural networks*, or RNNs. Feedback and recurrent connections are likely important elements

to include in ANN models of most areas of the brain. The addition of recurrent connections endows the networks with a natural way of using temporal structure in the inputs, which is useful both in modeling and in real-world applications of ANNs such as language translation. While it can be useful to consider hierarchy and feedback/recurrence in models separately thanks to the much increased simplicity, more recent endeavors have sought to include both.

One of the early successes of using ANNs to model the brain is found in [16]. Here the authors recorded from neurons in monkey prefrontal cortex after the monkey learned a task. The activity of the recorded neurons during performance of the task was visualized by using dimensionality reduction methods. To gain insight on how these neurons might be involved in solving the task, the authors then trained a recurrent neural network to solve the same task, and plotted the trajectories of these networks (after dimensionality reduction). The trajectories of the recorded and artificial neurons looked remarkably similar. Furthermore, the dynamics of the recurrent network were analyzed, which revealed an interesting way in which the task could be solved in the brain.

Deep neural networks have also helped to extend and partially verify foundational ideas of processing in the visual system. Hubel and Wiesel [17] showed that neural responses in cat striatal cortex (an early visual system area) can be built from the summed combinations of the responses of retinal ganglion cell neurons. The hypothesis is that the next level in the visual system hierarchy then combines the responses of the striatal cortex, and so on. Each layer in the visual system hierarchy generates a library of simpler responses, which are then picked from and combined to form the response in the next layer [18]. A deep neural network model that performs this operation was first described in [19]. Experiments with visual-system-inspired *convolutional* neural networks [20] reveal responses across layers that match this behavior: once trained to classify images, the initial layers are found to perform simple functions such as edge detection, and deeper layers progressively combine these simpler responses to form more complex responses, such as layers with neurons that respond selectively to parts of the human face. As an example, the authors of [21] found that neurons in intermediate layers of deep neural networks exhibit similar responses to moderately complex curved boundaries as neurons in the intermediate visual area of monkey V4. For reviews of the development of convolutional networks and the comparisons made with visual circuits, see [22, 23].

This provides evidence that deep neural networks develop complex features in a way similar to the visual system. That this behavior emerges through the process of training a

network to classify inputs, and that these networks are the only artificial systems that can approach human performance on these difficult tasks, is strong evidence that this approach is a good general strategy that is also employed by the brain.

Other work indicates the appropriateness of deep neural networks for modeling the auditory system of the brain [24]. In this study deep neural networks were found to have similar patterns of error-making as human participants on the study. In addition, the neural representations were similar enough to brain FMRI data that the latter could be predicted from the former with relatively high accuracy.

Neural networks have also been seen to reproduce properties of navigation systems in animal brains. In [25], a recurrent neural network trained to predict the location of an agent based on velocity inputs generated neural units whose responses were similar to special cells in the brain known as grid cells. A deep network was then trained on top of this representation to navigate the agent to goal locations. Such networks exhibited animal-like behaviors, such as taking shortcuts despite not having traveled over these shorter paths before.

0.0.4 Training networks

We briefly introduce the basic ideas behind the training of neural networks. For this, we can view a neural network as an abstract function f that is parametrized by a vector of parameters $\boldsymbol{\theta}$ and that receives an input vector $\boldsymbol{x}$, so that the output of the network is $\hat{\boldsymbol{o}} = f(\boldsymbol{x}; \boldsymbol{\theta})$. Note that in general f can be a vector-valued function. In supervised training, we are given a set of training datapoints $\{\boldsymbol{x}_i\}$ and a corresponding set of targets $\{\boldsymbol{o}_i\}$, and the goal is to adjust $\boldsymbol{\theta}$ so that f outputs $\{\boldsymbol{o}_i\}$ when it receives the input $\boldsymbol{x}_i$. To train the network, we first need to define a notion of "nearness" between $\{\boldsymbol{o}_i\}$ and $\{\hat{\boldsymbol{o}}_i\}$. To do so we define a loss function $l(\hat{\boldsymbol{o}}, \boldsymbol{o})$. This function should have the characteristic that it is minimized when $\hat{\boldsymbol{o}}$ becomes close to $\boldsymbol{o}$.[4] One example is the "squared error" distance function $l_{SE}(\hat{\boldsymbol{o}}, \boldsymbol{o}) = \|\hat{\boldsymbol{o}} - \boldsymbol{o}\|^2$. Another that is typically used in classification problems is the categorical cross-entropy loss $l_{CCE}(\hat{\boldsymbol{o}}, \boldsymbol{o}) = \log(p_k)$ where $p_k = (\exp(\hat{o}_k)/\sum_j \exp(\hat{o}_j))$ represents the probability that $\hat{\boldsymbol{o}}$ indicates class k (k is determined by $\boldsymbol{o}$). While this is the objective of the network, usually our true goal is not to just make the network perform well on the training data $\{\boldsymbol{x}_i\}$, but also any other input-target pairs from the same distribution

[4]This loss function is often a surrogate for what we actually want the network to do. For instance, we may be interested in minimizing the classification error. This error is not differentiable and therefore difficult to minimize, so we use a surrogate such as mean-squared error or categorical cross-entropy. Achieving zero loss with these loss functions results in zero classification error.

that may come along in the future. This is known as *generalization*. The hope is that by using what we know (the training data), if the system is designed well and we are careful with the training process, the system should also do well on data that it hasn't seen (testing data).

Modifying $\boldsymbol{\theta}$ is an optimization problem that is amenable to standard approaches such as Gradient Descent. However, writing the gradient $\nabla_{\boldsymbol{\theta}} \sum_i l(\hat{\boldsymbol{o}}_i, \boldsymbol{o}_i)$ requires using the chain rule to compute the gradient with respect to each layer in terms of the deeper layers in the hierarchy. The algorithm implementing this procedure is known as *backpropagation*. The parameter update is then $\boldsymbol{\theta}_{n+1} = \boldsymbol{\theta}_n - \gamma \left(\nabla_{\boldsymbol{\theta}} \sum_i l(\hat{\boldsymbol{o}}_i, \boldsymbol{o}_i)\right)(\boldsymbol{\theta}_n)$ where γ is a (small) positive scalar, the *learning rate*. It turns out that it is not necessary to take the sum in this expression over the entire training set, and indeed training can be faster and more successful when the sum is taken over a subset of the training set that is chosen randomly for every gradient update [26]. This is called *Stochastic Gradient Descent* (SGD), and is the foundation for the majority of optimization algorithms used to train neural networks. Frequently SGD is augmented by incorporating a factor called *momentum*, which helps SGD move faster down the loss surface and avoid getting stuck in local minima. Modern optimization approaches often incorporate some form of learning rate adaptation, which adjusts the effective learning rate based on information about previous updates. When training our networks, we use a variant of SGD, either RMSprop or Adam, unless otherwise stated. While the loss surface is highly nonconvex, in practice this doesn't usually stand in the way of networks learning good solutions provided some simple strategies are employed. A complete pass of the network optimization procedure through the set of training data is referred to as an *epoch*.

Backpropagation is not a biologically plausible learning rule. This casts some doubt on whether networks trained in this way can truly be a good model for the brain. This is an active area of research, and the discovery of backpropagation-like learning rules employed by the brain is hotly sought-after (for a review, see [3]).

0.0.5 "Opening the black box"

Compared to analysis of brain data, analysis of artificial neural networks is much easier. In ANNs every connection weight and neural response is available, while in the brain we must make do with a severe undersample. While ANNs are growing very large and complex, the brain still dwarfs them in terms of number of neural units and connections. Despite these advantages, "opening the black box" of ANN models has been a considerable challenge, and

many basic questions about them are only beginning to be answered. For instance, it is still not fully clear why it is that deep neural networks are so effective and why they do not suffer from overfitting despite being very overparametrized (though see [27, 28, 29, 30, 31] for examples of the rapid developments that are being made in this area).

A major thrust in the effort to understand deep neural networks is by considering the geometry of the representations formed through the layers of these networks [32]. Since these geometries are complicated, the authors of [32] develop sophisticated tools to measure useful characteristics of these geometries. Similar to the kernel perceptron, the network can resolve nonlinearly tangled classes gradually through the layers until they are linearly separable.

A parallel line of work seeks to shed light on the inner workings of recurrent neural network models (for a review see [12]). The early foundational work [33] established the usefulness of investigating the fixed points of RNNs and the local dynamics around these fixed points. RNNs have also been trained to produce sequences, and the solutions used to generate hypotheses about how sequences are generated in the brain [34]. General properties of prefrontal cortical neurons are compared to the coding properties of neurons in RNNs in [35]. Task-solving dynamics can also be implemented in RNNs through means other than training: see [12] for examples and [36] for more recent work.

The approach of this thesis is to start with artificial network models that can be understood more easily. While this is not a new approach, such simple network models currently receive much less attention than networks designed to perform well on difficult tasks such as the ImageNet image classification database [37]. Once we have identified interesting phenomena in these simpler models, we can then see if they hold in more complex models.

What sort of interesting phenomena can we look for in neural networks? In Chapters 1 and 2 we focus on summary statistics regarding the geometry of the neural representations formed in intermediate stages of the computation, such as the intermediate layers of a deep neural network in Chapter 1. We extend the simple geometric reasoning introduced with the kernel perceptron of Fig 1c to a network with more layers. They key idea of the kernel perceptron is that creating a high-dimensional representation of the data makes the data linearly separable and so allows the network to learn a solution. The question we are interested in is: given a network with many layers, all of which are trained by standard learning rules, do these networks also learn to do this trick? What happens to the representation after linear separability has been achieved: do subsequent layers further condition the representation into a more useful form? If so, what form does that representation have, and what

aspects of the learning rule or the architecture drives this formation? We in fact find that feedforward networks learn to expand dimensionality, and furthermore that they then reduce the dimensionality in subsequent layers in a dramatic way after this expansion. Through mathematical reasoning and experiments we provide evidence that randomness in parameter updates through the learning process drives this dimensionality compression. While our goal in characterizing the geometry of representations through the layers of the network is the same as [32], the different measures we use result in the identification of novel phenomena and conclusions.

In Chapter 2 we next move to consider recurrent neural networks and the interactions between the network's dynamics and the geometry of the representations. We find that these networks learn to solve our task with different dynamics depending on initialization. In particular, they do not always learn to initially expand dimensionality, and may fail to solve the task as a result. However, increasing a quantity known as *chaos* enables the networks to expand dimensionality and so find a solution. Furthermore, we find that chaos can aid in learning a new instantiation of the task, suggesting that chaotic solutions can be more flexible. These results suggest possible new benefits of variability in biology.

In Chapter 3 we shift our perspective to an analysis of the weights of a trained neural network. While recordings of neural activity continue to yield insights, there are fundamental barriers associated with recording from a living animal. In contrast, information about the connections between neurons can be collected in nonliving brain tissue, allowing for the possibility of a very thorough mapping via electron microscopy. Moreover, weights provide information complementary to neural recordings as to how networks compute [38]. There is hope that with enough connectivity data, much information about the functioning of a neural circuit can be *inferred* from these data, even with little or no activity recording data from the same neurons. One approach that has recently been proposed is to characterize the geometry of the weight matrices making up the connectivity (such as is seen by plotting the rows or columns of this matrix) [39]. The hope is that with this approach, information about the underlying computation can be extracted by dimensionality reduction methods applied to the weights, both in artificial neural networks and network models for biological circuits. However, it is not yet clear if and when such dimensionality reduction methods will reveal useful information about the functioning of neural circuits. We approach this by training a simple neural network to do a task, and then seeing if information about the task can be inferred from the weights. In this simple model, we are able to mathematically derive

relationships between the geometry of the weights and the nature of the task.

Chapter 1

DIMENSIONALITY BALANCE IN FEEDFORWARD NEURAL NETWORKS

1.1 Introduction

Successful learning in neural networks rests on forming the right representations of data. It is widely appreciated that the role of initial layers of deep networks is to learn features of the data that constitute useful building blocks for later layers to piece back together. While the attributes that make for "good" representations is a complex question in general, it is useful to simplify matters via the perspective of dimensionality. As the seminal work of Cover [6] showed, high-dimensional representations can enable successful classification of data with simple linear readouts. In general, high dimensionality subserves complex and general computations that nonlinearly combine many features of inputs [6, 40, 41, 42, 43, 44]. On the other hand, 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 [45, 40, 9, 46], and see Section 1.2 below for a more detailed discussion). It is likely in balancing these two desiderata that learning is most successful.

Here we investigate the dimensionality of the representations learned by neural networks in several contexts. We begin with a simple task where the dimensionality of the inputs can be controlled, and investigate the effects of training a dense feedforward network on these data. We next shed light on the learning dynamics by investigating a linear network with a single hidden layer, deriving relationships between noise in parameter updates – such as that generated by stochastic gradient descent – and the compression of the dimensionality of the hidden unit representation. We test our findings in simulations comparing stochastic gradient descent with "full-batch" gradient descent. Finally, we evaluate the ideas laid out in this chapter in the context of the deep convolutional network model VGG. In this case the representation manifolds are more complicated, nonlinearly curving through space. To address this, we additionally investigate other measures of dimensionality that take this curvature into account.

Publications and preprints. The contents of Section 1.7.1 and the long-time analysis in Section 1.6.2 can be found in [47]. The two-step analysis of stochastic gradient descent in Section 1.6.3 can be found in [48].

1.2 Effective dimensionality

To begin, we need a useful way to measure the dimensionality of data. The most basic measure is the *rank*, which is the dimension of the span[1] of the mean-centered datapoints when considered as a vector subspace. This is simply the dimension of the smallest vector space that contains the mean-centered data. However, for many kinds of data this definition is not very useful. Data are often in *general position*. One example is the case where the data are corrupted by a miniscule amount of white noise. The rank of such data will typically be either the number of datapoints, or the number of coordinates in the data vectors, whichever is smaller.

A useful way to think about the rank of the data is through the lens of the covariance. The rank is equal to the number of nonzero eigenvalues of the covariance matrix of the data. The covariance matrix $\boldsymbol{C}$ of a set of datapoints $\{\boldsymbol{x}_k : k \in [m]\}$ is $\boldsymbol{C} = \frac{1}{m}\sum_{k=1}^{m} \boldsymbol{x}_k \boldsymbol{x}_k^T - \boldsymbol{\mu}\boldsymbol{\mu}^T$ where $\boldsymbol{\mu} = \frac{1}{m}\sum_{k=1}^{m} \boldsymbol{x}_k$. Consider again the case where data are corrupted by a miniscule amount of white noise, and suppose that the uncorrupted data have rank d but the corrupted data has rank n. Sorting the eigenvalues λ_k of $\boldsymbol{C}$ by size, the distribution of these eigenvalues will have a gap between the first d eigenvalues and the last $n - d$, since the size of the first d eigenvalues is determined by the actual structure of the data while the size of the last $n - d$ is determined by the size of the noise. In this situation, a useful way to recover the "true" dimensionality of the data is to threshold the eigenvalues, setting those below an appropriately small value to zero. One possible way to set this threshold is to search for the (relative) largest gap in the eigenvalue distribution.

While this approach works well for data with a clear gap in the eigenvalue distribution, this kind of gap may not exist in many datasets of interest. An alternative approach is to use the participation ratio of the eigenvalues of the covariance matrix,

$$\mathrm{ED} = \frac{\left(\sum_{k=1}^{n} \lambda_k\right)^2}{\sum_{k=1}^{n} \lambda_k^2}, \tag{1.1}$$

[1]For a set of points $\boldsymbol{x}_1, \boldsymbol{x}_2, ...\boldsymbol{x}_m$, the span of these points is defined as the set of all linear combinations of these points, which can be written $\mathrm{span}(\{\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_m\}) = \{\sum_{k=1}^{m} a_k x_k : a_k \in \mathbb{R}\}$

sometimes referred to as the *effective dimensionality* [49, 50]. The primary advantages of this measure is that (1) it smoothly varies with the eigenvalues of $\boldsymbol{C}$ and (2) it doesn't require a choice of threshold. Fig 1.1 illustrates ED. In many cases the ED gives a good indication of the number of principal components needed to capture most of the variance of the data [51]. This is the primary measure of dimensionality that we use in Chapters 1 and 2.

This "fuzzy" notion of dimension provided by ED is also useful in characterizing how successful we might expect to be in building models on the data. It is well-known that expanding dimension can be useful. This point is made with the (rank) dimension in Cover's seminal paper [6], where it is shown that points become more likely to be linearly separable as they are nonlinearly embedded into higher-dimensional spaces. This idea is used as justification for kernel-based classification methods such as kernel perceptrons and support vector machines. Once the data are transformed to be linearly separable, simple linear readout weights can be trained to classify the data. However, for these linear readouts to find a solution that will generalize well to unseen data, it is generally important to find a solution with a substantial *margin*, which is the minimal distance between the separating hyperplane and any datapoint. In this way dimensionality expansion, even when enabling linear separability, may not afford a very robust solution if the margins are small. An schematic example of such a case is shown in Fig 1.2a. Here the test data are generated by adding finite variance noise to the training points (indicated by shaded regions). When the margin is larger than the extent of the noise, both the training and testing samples are classified successfully (Fig 1.2b). One of the primary applications of the celebrated statistical learning theory was in bounding generalization error in terms of this margin [9]. This picture implicitly uses the intuition that proximity should carry information about class identity (the distance between a point and the closest point of the same class will tend to be smaller than the distance between this point and the closest point in a different class). Note that effective dimensionality in some sense captures this notion of a margin, since only an expansion with a substantial margin causes an increase in the ED. To the best of our knowledge, the explicit connection between increasing the *effective* dimensionality and increasing the probability of allowing for a *robust* separating hyperplane – a slight variation of Cover's perspective – is novel.

It is perhaps a less appreciated fact that it can also be helpful when points of the same class are close together, Fig 1.2c. This confers increased robustness to perturbations in the output weights. The dashed lines indicate the extent to which the angle of the output weights can be changed while still performing perfectly on the test set. This is related to

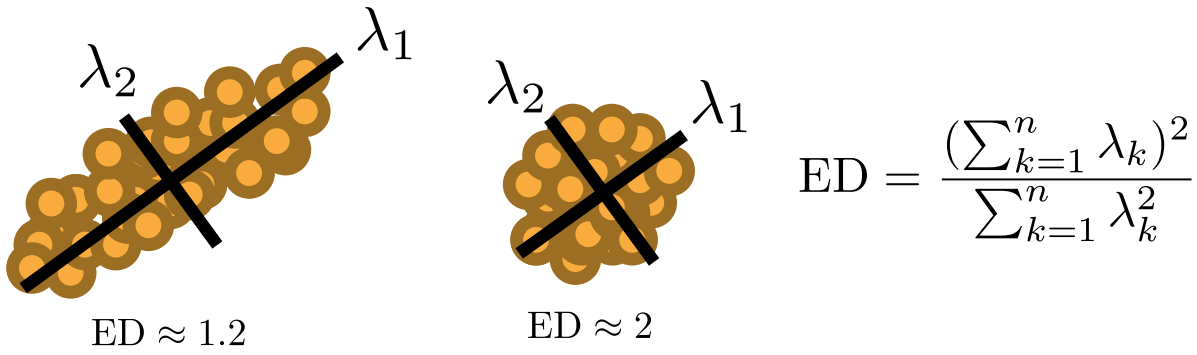

Figure 1.1: **Visualizataion of effective dimensionality (ED). The eigenvalues λ_k of the covariance matrix of the data are used to compute effective dimensionality. The data on the left has an effective dimensionality of a little over 1, while the data in the center of the figure has ED approximately equal to 2.**

the idea of minimum description length: the weights can be written with fewer bits and still do a good job on the test data. This theory allows for explicit bounds on generalization error similar to those derived for margins [52, 53]. (See also [41] Section 7.2 for a general discussion of margins and minimum description link, and more indirectly related references. See also [45, 9, 46] for less formal discussions of these points and [40] for a discussion in a neuroscience context.) Low-dimensional representations may also be related to standard model regularization approaches such as L2 regularization – see Section 1.6 for evidence supporting this.

Note that achieving a representation where the training and test data from the same classes are moved together is made much simpler by first increasing the margin (and thus the effective dimensionality) of Fig 1.2a. This means that in order to achieve a low-dimensional representation with good properties as in Fig 1.2c, it may first be necessary to create a higher-dimensional representation as in Fig 1.2b.

"Dimension" vs "Dimensionality". In describing a set of points we typically use the word "dimension" to refer to the number of objects needed to in some sense fully reconstruct this set of points. An example is the number of basis vectors needed to reconstruct the set of points with linear combinations (the standard definition of the dimension of a linear space). Another example is seen in Section 1.7.1, where dimension can refer to the number

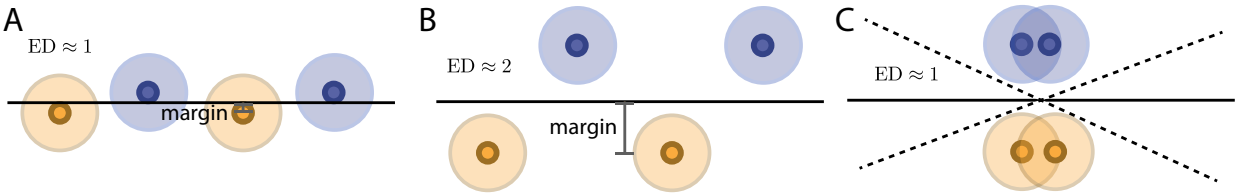

Figure 1.2: **Example schematics of possible data representations.** Training data are visualized as points, and testing data as shaded disks around these points. (a) The training data are two-dimensional, but the variation in the vertical axis is very small, resulting in an effective dimensionality of about 1. As a result, the margin is small, and the separating hyperplane (black line) does a poor job of classifying the testing data. (b) With increased variation in the vertical direction, the effective dimensionality (ED) of the training data and the margin increase. This allows the margin-maximizing separating hyperplane to correctly classify the testing data. (c) By brining points belonging to the same class together, the margin remains large and in addition the separating hyperplane has more freedom to be perturbed while still classifying the testing data correctly. Dashed lines depict the maximum extent that the margin-maximizing hyerplane can be rotated while still classifying the testing data correctly.

of variables that parametrize a nonlinear manifold (*intrinsic dimension*). We typically use the word "dimensionality" when referring to measures that are meant as estimates of a dimension of a supposed underlying "true" structure of the distribution from which the set of points is drawn. This supposed structure will depend on the assumptions made, either implicitly or explicitly, by the method. The examples considered here are ED, which takes the view that the measure of linear space dimension is inflated by the effects of noise and so discounts the contribution of lower-variance modes; and intrinsic dimensionality, which estimates the intrinsic dimension of a supposed manifold underlying the set of points by making assumptions about this manifold.

Inputs used to measure dimensionality. We would like to be able to measure the dimensionality of the network representation over the entire training dataset. However, this set of points may be impractically large and require a very large amount of computer memory in order to compute. To circumvent this issue, we employ two subsampling strategies: (1) We sample the same number of points from every class, and (2) we increase the number of points used in the subsample until the dimensionality has effectively plateaued.

1.3 Deep feedforward networks

The many layers of a deep feedforward network provide a powerful substrate over which the representation of data can be gradually transformed into a desired output. The equations for a network of depth D can be written recursively as

$$\begin{cases} \boldsymbol{h}^0(\boldsymbol{x}) = \boldsymbol{x} \\ \boldsymbol{h}^\ell(\boldsymbol{x}) = \phi^\ell(\boldsymbol{W}^\ell \boldsymbol{h}^{\ell-1}(\boldsymbol{x})) + \boldsymbol{b}^\ell, \quad 0 < \ell < D \\ \boldsymbol{h}^D(\boldsymbol{x}) = \boldsymbol{W}^D \boldsymbol{h}^{D-1}(\boldsymbol{x}) + \boldsymbol{b}^D. \end{cases} \tag{1.2}$$

Here the vector $\boldsymbol{h}^\ell(\boldsymbol{x})$ is the hidden unit response to input vector x at layer ℓ, and $\boldsymbol{h}^D(\boldsymbol{x})$ is the output of the network, which can be scalar or vector valued. The matrix $\boldsymbol{W}^\ell$ specifies the strengths of the connections between hidden units in layers $\ell - 1$ and ℓ. The vector $\boldsymbol{b}^\ell$ is a bias term at layer ℓ. The *activation function* ϕ^ℓ operates on vectors elementwise, and is usually chosen to be a simple function. Standard choices are $\phi^\ell(x) = \tanh(x)$ and $\phi^\ell(x) = \max(x, 0)$. It is useful to think of each unit j as having its own activation function ϕ^ℓ_j, and generally ϕ^ℓ_j is taken to be the same for each j. A unit that has $\phi^\ell_j(x) = \max(x, 0)$ is often referred to as a Rectified Linear Unit, or ReLU for short. A simple illustration of a feedforward network can be found in Fig 1.3.

The parameters $\boldsymbol{W}^\ell$ and $\boldsymbol{b}^\ell$ are modified through training via SGD (see Section 0.0.4) in order to satisfy an optimization criterion on the outputs $\boldsymbol{h}^D(\boldsymbol{x})$. Often in the case of classification problems, the categorical cross entropy loss between the network output and targets is minimized (see Chapter 1 Methods). This is the method used here, except where otherwise noted.

If we consider a dataset $\mathcal{D} = \{\boldsymbol{x}_k : k \in [m]\}$, we can think of the *representation* of these data at a particular layer ℓ, $\mathcal{H}^\ell = \{\boldsymbol{h}^\ell(\boldsymbol{x}_k) : k \in [m]\}$. Just as we may ask questions about $\mathcal{D}$ – what is the dimensionality of $\mathcal{D}$? how many clusters exist in $\mathcal{D}$? is $\mathcal{D}$ linearly separable with respect to a particular labeling of the data? – we may also ask similar questions about $\mathcal{H}^\ell$. The answers may similarly be enlightening.

1.3.1 Dense networks trained on clustered data

To begin, we consider the case of simple dense feedforward networks trained on a simple dataset consiting of Gaussian clusters. We set the number of hidden units of each hidden

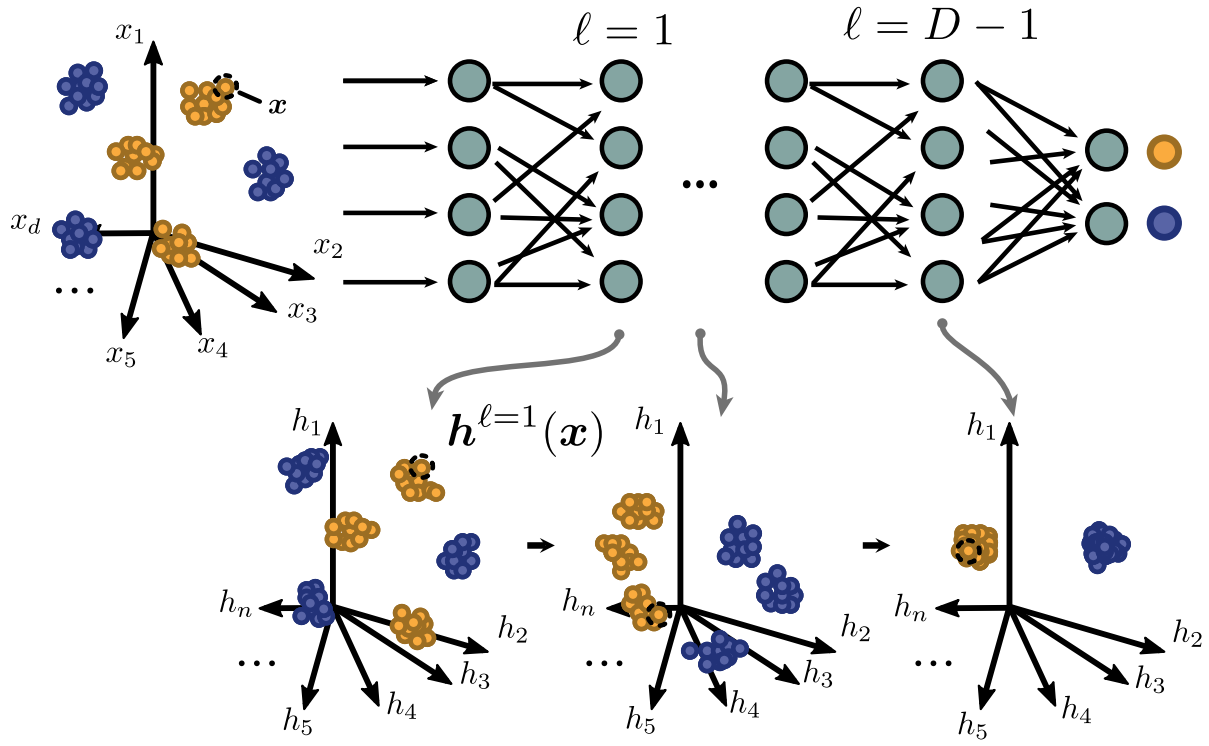

Figure 1.3: **Schematic illustration of a deep feedforward neural network transforming data.** Inputs (left) are gradually transformed through the layers of a deep neural network (bottom). In labeling the axes of the network representation (bottom), the dependence of the axes on the layer ℓ is suppressed for readability, and the number of hidden units in each layer is set to n. The dashed circle tracks the evolution of a single input sample $\boldsymbol{x}$. The representation changes to allow for a separating hyperplane (bottom-middle) and may continue to evolve in other ways, such as bringing points of the same class closer together (bottom-right).

layer to a fixed number n, and constrain the nonlinearity to be the same for every hidden unit: $\phi_j^\ell(x) \equiv \phi(x)$. We use networks of depth 11, so that $\ell = 10$ is the final hidden layer.

The inputs are d-dimensional vectors. They are selected from Gaussian-distributed clusters whose means are distributed randomly within the d-dimensional input space. We consider two scenarios for the input space dimension: $d = 2$ and $d = n$. Each cluster is assigned one of two class labels, at random. Thirty clusters are assigned label 1, and thirty label 2. A schematic of six clusters can be seen in Fig 1.3. Additional details of the training procedure can be found in Chapter 1 Methods.

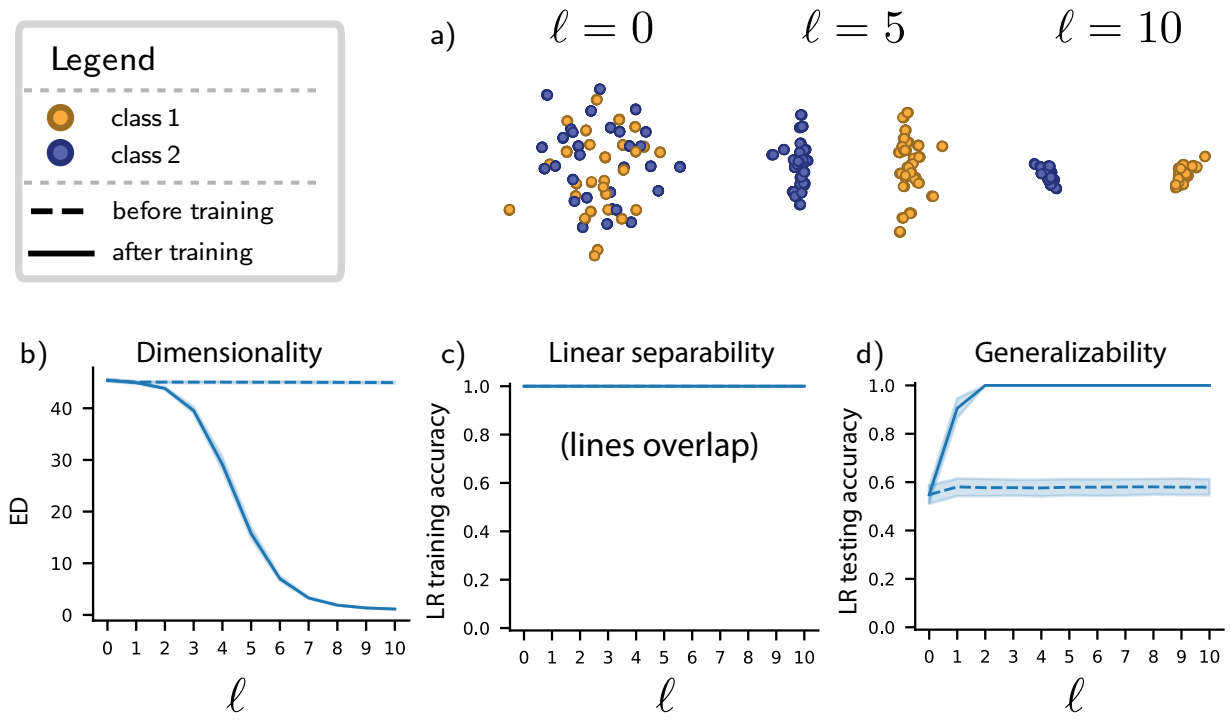

Figure 1.4: **Geometry of the representation of a deep feedforward neural network trained on high-dimensional clustered data.** (a) Top two principal components of hidden representation at layers $\ell = 0$, $\ell = 5$, and $\ell = 10$. (b) Effective dimensionality (ED) of the representation through layers ℓ. Multiple network and task instantiations were generated, creating variability as indicated by shaded regions (see Chapter 1 Methods). (c) Mean accuracy of a logistic regression (LR) linear classifier trained on the representation through the layers ℓ. (d) Mean testing accuracy of an LR classifier. The LR 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).

1.4 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 hidden units. 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 producing linear separability. Weights are initialized to identity matrices.

The network easily achieves perfect testing accuracy (not shown). What is then of interest is how properties of the network's *internal representation* change over the course of training. Fig 1.4a shows the top two principal components of the network responses to 600 input samples at snapshots through the layers. Here we see that points belonging to different classes are pulled apart while points within the same class are compressed together in all dimensions.

This compression phenomenon is partly captured by measuring the ED of the representation. In Fig 1.4b the ED is plotted through the layers ℓ. The EDs of the trained networks are approximately equal to that of the input at $\ell = 1$, since the initial states only differ from the inputs by the application of a single layer. The dimensionality drops with increasing ℓ and is highly compressed at the final hidden layer $\ell = 10$. This compression results both from increasing distances between different classes as well as decreasing distances within classes (see Fig 1.4a). The degree of these trends depends somewhat on the learning procedure, with more aggressive weight updates resulting in faster and sometimes more dramatic 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 [54, 55, 56, 57], in this case invariance to input cluster identity. When the number of layers is increased, the shape of the curve remains similar (see Fig 1.12).

We next study the coding properties of these representations. Fig 1.4c shows the mean accuracy of a logistic regression (LR) linear classifier trained to classify the network representation at each layer. This confirms the point made above: the input data are linearly separable, and this property is retained through the layers. The interesting properties of the network computation lie elsewhere, in that the learned transformations and resulting dimensionality compression correspond with better generalization properties of the representation. In Fig 1.4d, 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 the layers ℓ, eventually allowing for perfect classification accuracy on held-out clusters.

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.

1.5 Classification of low-dimensional inputs

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

As shown in Fig 1.5b, the dimensionality in this case initially expands in early layers, and then reverses course to compress in the later layers. The specifics of this behavior is variable and sensitive to choice of hyperparameters – sometimes the peak of the curve occurs at earlier or later layers, and may be more or less pronounced (data not shown, see Chapter 1 Methods). However, the existence of such a peak is fairly consistent. The classification accuracy of the network through training iterations is also sensitive to hyperparameters (not shown) – the final accuracy on the test data seems to range from about 90% to nearly 100%. Correspondingly, the top principal components of the representation for a particular realization (Fig 1.5a) indicate that the representation is transformed to be very nearly, but not quite, linearly separable by $\ell = 10$

Fig 1.5c shows that indeed, the network very nearly forms a linearly separable representation. It should be stressed that this is not a robust result – for small changes in optimization, the network doesn't come as close to achieving linear separability (not shown). What is consistent is that the network learns to improve linear separability a substantial amount. We observe that the generalizability of the representation, while increased due to training and through layers, is less than perfect (Fig 1.5d).

Taken together, this indicates that the network tries to learn to expand dimensionality in a similar spirit to hand-built kernel machines, but has mixed success at doing so.

1.6 Mathematical analysis of dimensionality compression

To shed light on the compression illustrated in Figs 1.4a and 1.4b – where points that belong to the same class are brought close together through the layers of the trained network – we

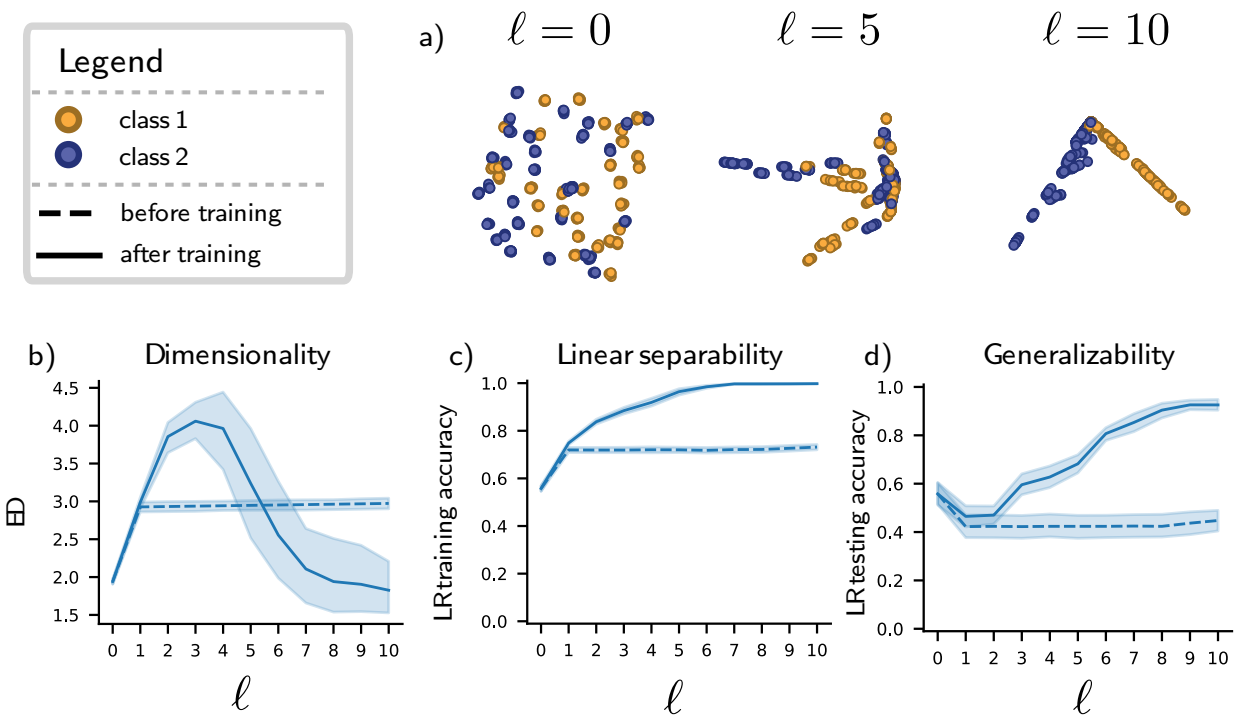

Figure 1.5: **Geometry of the representation of a deep feedforward neural network trained on two-dimensional clustered data.** (a) - (d) As in Fig 1.4, but for a network trained on two-dimensional data.

show how compression can occur in a simplified scenario: 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 output weights, on average. Second, we show how the noisy variability of the network weights generated by SGD can also cause compression orthogonal to the readout direction by calculating the average effect of two SGD updates. Analysis of dimensionality compression can also be found in [58, 59, 60] and of compression of mutual information in [61, 62], but this is the first work that we are aware of showing an explicit connection between minibatching SGD and dimensionality compression. Implicit regularization of gradient descent, particularly with exponential type losses, may also contribute to compression, see [63]. See also [64] for an analysis of linear networks under full batch gradient descent learning dynamics.

For simplicity, first consider the case where the output of the network is a scalar and

the output weights $\boldsymbol{r}$ is a vector. In such 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 $\hat{o}(\boldsymbol{x}) = \boldsymbol{r}^\top \boldsymbol{h}(\boldsymbol{x})$, where $\boldsymbol{W}$ are the input weights. 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:

$$\delta\boldsymbol{W} = -\frac{\gamma}{|B|}\sum_{\boldsymbol{x}\in B} e(\boldsymbol{x})\boldsymbol{r}\boldsymbol{x}^T, \tag{1.3}$$

$$\delta\boldsymbol{b} = -\frac{\gamma}{|B|}\sum_{\boldsymbol{x}\in B} e(\boldsymbol{x})\boldsymbol{r} \tag{1.4}$$

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: $\delta\boldsymbol{h}(\boldsymbol{x}') \propto \boldsymbol{r}$. It follows that in order to see compression of points in all directions (such as in Fig 1.4a), we need to consider sources of variability of $\boldsymbol{r}$ through learning.

1.6.1 Simple one-step example

Let $\boldsymbol{h}_k$ denote the hidden representation after k steps of gradient descent, and similarly for parameters $\boldsymbol{W}$, $\boldsymbol{b}$, and $\boldsymbol{r}$. For the calculations in this section, we assume that inputs have zero mean and covariance $\left\langle \boldsymbol{x}\boldsymbol{x}^\top \right\rangle = \sigma_x^2\boldsymbol{I}$. A single step of gradient descent results in the representation $\boldsymbol{h}_1(\boldsymbol{x}') = \boldsymbol{h}_0(\boldsymbol{x}') + \delta\boldsymbol{W}_0\boldsymbol{x}' + \delta\boldsymbol{b}_0$. 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. Averaging $\boldsymbol{h}_1$ over batches B and the noise $\boldsymbol{\xi}$ (see Chapter 1 Appendix), we obtain

$$\langle \boldsymbol{h}_1(\boldsymbol{x}') \rangle = \boldsymbol{h}_0(\boldsymbol{x}') - \gamma\sigma_x^2 \left\langle \boldsymbol{\xi}\boldsymbol{\xi}^\top \right\rangle \boldsymbol{W}_0\boldsymbol{x}' + \alpha\overline{\boldsymbol{r}} \tag{1.5}$$

$$= \boldsymbol{h}_0(\boldsymbol{x}') - \gamma\sigma_x^2\sigma_\xi^2\boldsymbol{W}_0\boldsymbol{x}' + \alpha\overline{\boldsymbol{r}}. \tag{1.6}$$

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

$$\langle \boldsymbol{h}_1(\boldsymbol{x}') \rangle = (1 - \gamma\sigma_x^2\sigma_\xi^2)\boldsymbol{h}_0(\boldsymbol{x}') + \alpha\overline{\boldsymbol{r}}. \tag{1.7}$$

This shows that in our simple linear network model and with the assumptions described above, isotropic variability of the readout weights $\boldsymbol{r}$ drives compression in all directions within the subspace orthogonal to $\overline{\boldsymbol{r}}$. See Fig 1.10 for a visualization.

1.6.2 Long-time learning analysis with simple noise

We first consider a more thorough analysis of the learning dynamics under the assumption of isotropic noise being added to the parameters. Note that this is a certain simplified model of minibatching SGD.

In this case we analyze a two-layer neural network trained to classify inputs according to c classes. The equations for the network are:

$$\begin{aligned} h_i &= \sum_j \phi\left(W^1_{ij} x_j\right) \\ \hat{o}_i &= \sum_j W^2_{ij} h_j, \end{aligned} \tag{1.8}$$

where $\boldsymbol{x}, \boldsymbol{h}$ and $\hat{\boldsymbol{o}}$ are the input, the hidden representation, and the output of the network, respectively. Additionally, $\boldsymbol{W}^1, \boldsymbol{W}^2, \phi$ are respectively the input weights, readout weights, and a (possibly nonlinear) activation function. We consider the cost function $L(\Theta) = \sum_{\mu=1}^{P} \|\boldsymbol{o}^\mu - \hat{\boldsymbol{o}}^\mu\|_2^2$ where $\Theta = [\boldsymbol{W}^1, \boldsymbol{W}^2]$ and μ indexes the training set of size P. Here the output $\hat{\boldsymbol{o}}^\mu$ is a length c vector, and the targets $\boldsymbol{o}^\mu$ one-hot encode the labels.

Over training, SGD generates effective noise in the parameter updates (see e.g. [65, 27]) due to the fact that each update is performed on a subset of the training data. In general SGD leads to noisy gradient updates of the form:

$$\Delta \boldsymbol{\Theta} = -\eta \nabla_{\boldsymbol{\Theta}} L(\boldsymbol{\Theta}) + \boldsymbol{z}_t \tag{1.9}$$

where η is the learning rate and z_t is the noise generated from each mini-batch, or the difference between the gradient of the full batch and mini-batch t. This noise in the gradient updates is correlated. Here we simplify the analysis by modeling this noise in parameter updates by adding noise of variance σ^2 directly to the weights $\boldsymbol{W}^1$ and $\boldsymbol{W}^2$, and by assuming this noise is isotropic Gaussian with zero mean.

Our analysis leads to the effective cost function:

$$L(\Theta, \sigma) = \sum_{\mu=1}^{P} \sum_{i} (o_i^\mu - \sum_j (\boldsymbol{W}_{ij}^2 + \sigma \xi_{ij}^\mu) h_j^\mu)^2, \tag{1.10}$$

where $\boldsymbol{\xi}$ is Gaussian noise with unit variance independently sampled across μ. Taking an average over ξ, for a learning rate small enough, we can rewrite Equation (1.10) in the form:

$$L(\Theta, \sigma) \approx \langle L(\Theta, \sigma) \rangle_\xi = L(\Theta, 0) + R(\sigma) = L(\Theta, 0) + \sigma^2 \mathrm{Tr}(\boldsymbol{C}), \tag{1.11}$$

with $C_{jj'} = \sum_\mu h_j^\mu h_{j'}^\mu$ and where we view $R(\sigma) = \sigma^2 \mathrm{Tr}(C)$ as a regularization term. Similar effective regularization terms have been shown to arise from dropout [66, 55], and the effects of regularization on generalization have been heavily studied (for a review see [67]). Here we focus on the effects of such regularization in shaping the dimensionality of the representation. While the compressive effects of an initial step of SGD have been previously noted [48], here we consider the limit of many steps.

Geometrically, all the directions of the representation $\boldsymbol{h}^\mu$ in the span of the readouts $\boldsymbol{W}^2$ contribute to the cost both in $L(\Theta, 0)$ and $R(\sigma)$, while the directions orthogonal to this span contribute only to the regularization penalty $R(\sigma)$. By penalizing the norm of the representation $\|\boldsymbol{h}^\mu\|_2^2$, the regularizer $R(\sigma)$ encourages the reduction of all $\boldsymbol{h}^\mu$ components, including the orthogonal components $\boldsymbol{h}_\perp^\mu$ of the representation with respect to the readout weights $\boldsymbol{W}^2$. We refer to this action as compression of task-irrelevant directions (directions orthogonal to the readout $\boldsymbol{W}^2$). For $\sigma > 0$ there is indeed a unique solution $(\boldsymbol{h}^\mu)^*$ with null orthogonal components $(\boldsymbol{h}_\perp^\mu)^* = 0$ that minimizes the cost Eq. (1.11): $(\boldsymbol{h}^\mu)^* = ((\boldsymbol{W}^2)^T \boldsymbol{W}^2 + \sigma^2 \boldsymbol{I})^\dagger (\boldsymbol{W}^2)^T \boldsymbol{o}^\mu$. Here $\dagger$ denotes the Moore-Penrose pseudo-inverse. The uniqueness is a consequence of the strict convexity of $R(\sigma)$ and the convexity of $L(\Theta, 0)$ as functions of $\boldsymbol{h}^\mu$ when $\boldsymbol{h}^\mu$ is unconstrained. The cost increase of straying from this solution is quadratic.

A network that learns the task balances minimizing the task cost $L(\Theta, 0)$ with encouraging the compression of task-irrelevant directions due to $R(\sigma)$. For instance, learning a task that is not linearly separable with large amounts of training data requires a higher-dimensional hidden representation, which may come at the expense of increasing $R(\sigma)$, since $R(\sigma)$ increases isotropically as $\boldsymbol{h}^\mu$ diverges from $(\boldsymbol{h}^\mu)^*$. In other words, balancing the two terms $L(\Theta, 0)$ and $R(\sigma)$ in the loss shapes the representation h^μ so that the manifold dimension of $\boldsymbol{h}^\mu$ is expanded only when aiding the reduction of the task loss $L(\Theta, 0)$ by separating

the classes (see [6] for formal connections between dimensionality expansion and class separation).

To provide further intuition regarding this balance, we consider the case of linear activations ($\phi(z) = z$). In this setting we can write a closed form expression for the first layer weights. When $\sigma > 0$ the cost Equation (1.11) is strictly convex with respect to the weights, and the unique minimizer $\hat{\boldsymbol{W}}^1$ is:

$$\hat{\boldsymbol{W}}^1 = ((\boldsymbol{W}^2)^T \boldsymbol{W}^2 + \sigma^2 \boldsymbol{I})^\dagger (\boldsymbol{W}^2)^T \boldsymbol{Y}^T X (\boldsymbol{X}^T \boldsymbol{X})^\dagger, \tag{1.12}$$

where $\boldsymbol{X}$ is a $P \times d$ matrix of input samples and $\boldsymbol{Y}$ is a $P \times c$ matrix of labels. Here d is the embedding space dimension for the inputs. This equation reveals that the range of $\hat{\boldsymbol{W}}^1$ lies within the span of the output weights, which implies that $\hat{\boldsymbol{h}}^\mu = \hat{\boldsymbol{W}}^1 \boldsymbol{x}^\mu$ lies in the span of the output weights as well. Equivalently, $\hat{\boldsymbol{h}}^\mu_\perp = 0$ for all μ. Strict convexity assures that for an appropriate learning rate scheme, SGD will converge to this solution. The linear network is nearly always able to achieve $(\boldsymbol{h}^\mu)^*$ when $P \leq d$, but not necessarily for larger numbers of samples. However, in either setting it will still remove all task-irrelevant directions from the representation.

We can consider how well these results generalize to the commonly applied ReLU nonlinearity ($\phi(x) = \max\{x, 0\}$). In the limit as $\sigma \to 0$, the hidden activity minimizing the loss will have the form $h^* + h_\perp$, where $w^{(2)} \boldsymbol{h}_\perp = 0$. The addition of $\boldsymbol{h}_\perp$ will not impact the ability to fit the training data, but is required to satisfy the nonnegativity constraint imposed by the ReLU activation. Thus, the dimensionality of the representation is larger in this nonlinear setting, and as the regularization strength is increased it leads to a trade-off between reducing dimensionality and fitting the training data.

1.6.3 Two-step analysis of stochastic gradient descent

Can the true variability induced by SGD drive compression? Here we show that two steps of SGD can cause compression under certain assumptions, and return to the case where the output of the network is scalar valued, as in Section 1.6.1. For simplicity, we consider the case of batch size 1. The gradient update for the readout vector $\boldsymbol{r}$ 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 (see Chapter 1 Appendix)

$$\langle \boldsymbol{h}_2(\boldsymbol{x}')\rangle = \boldsymbol{h}_0(\boldsymbol{x}') - \gamma\sigma_x^2 \left\langle (\delta \boldsymbol{r}_0)(\delta \boldsymbol{r}_0)^\top \right\rangle \boldsymbol{W}_0 \boldsymbol{x}' - \gamma\sigma_x^2 \left\langle (\delta \boldsymbol{r}_0)\boldsymbol{r}_0^\top (\delta \boldsymbol{W}_0)\right\rangle \boldsymbol{x}' + \alpha' \boldsymbol{r}_0 + \mathcal{O}(\gamma^4). \quad (1.13)$$

Where α' is some constant. Here δ is the update at the first step of SGD, as before. The second term resembles Eq. (1.5), 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. (1.13) decouples into scalar equations (see Chapter 1 Appendix)

$$(\langle \boldsymbol{h}_2(\boldsymbol{x}')\rangle)_j = \alpha_j (\boldsymbol{h}_0(\boldsymbol{x}'))_j + \mathcal{O}(\gamma^4) \quad (1.14)$$

for $j \neq k$, where $\alpha_j = \left(1 - \gamma^3\sigma_x^4 \langle e(\boldsymbol{x})^2\rangle \left((\boldsymbol{W}_0)_{jj}^2 + \|\boldsymbol{r}_0\|^2\right)\right)$ 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. (1.13), 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.

The effect of compression is of order γ^3, which indicates that the effect is weak for a single time step; however, this effect may compound over additional layers and over additional training steps. In addition, the compression depends directly on $e(\boldsymbol{x})^2$, so that a network that reduces the error very quickly may not compress as much.

1.6.4 Simulations

We test the finding that SGD drives dimensionality compression in simulations of dense feedforward networks by comparing networks trained with (full batch) gradient descent and those trained with stochastic gradient descent with batch size 1. In both cases, the RMSprop optimization scheme is used, and results are similar with Adam. Significant dimensionality compression does not occur with vanilla SGD (not shown). Fig 1.6 shows the evolution of the dimension of the final hidden layer of a 10 layer network over training. These figures

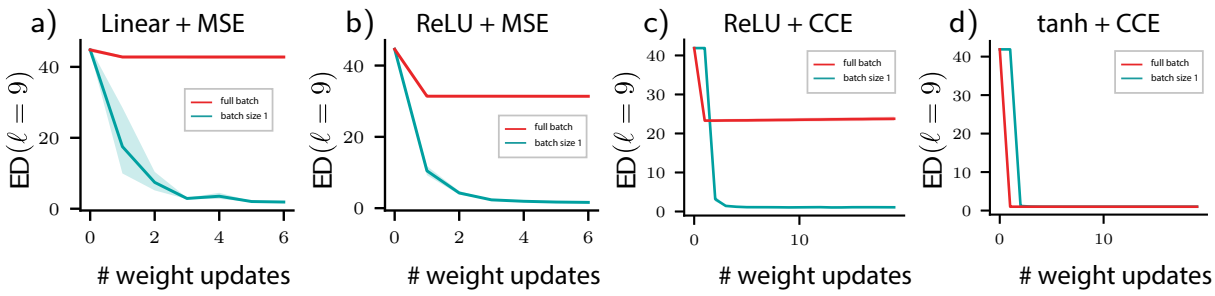

Figure 1.6: **Minibatching can drive dimensionality compression.** Comparison of the dimensionality of networks trained with a full batch vs networks trained with batch size 1 on the gaussian cluster data. In all plots, the dimensionality of the final hidden layer ($\ell = 9$) is reported through training (# weight updates). Red denotes the network trained with (full batch) gradient descent, and green the network trained with stochastic gradient descent with batch size 1. (a) Dimensionality of linear networks trained with mean squared error (MSE). (b) Dimensionality of networks with ReLU hidden units trained with MSE. (c) Dimensionality of networks with ReLU hidden units trained with categorical cross entropy (CCE). (d) Dimensionality of networks with hyperbolic tangent hidden units trained with CCE.

illustrate that in many cases, the network trained with gradient descent does not converge to a low-dimensional solution, while the network trained with SGD does. This is in spite of the fact that at each point of the x axis (number of weight updates), the GD network has lower loss than the SGD network (not shown). One exception to this rule is the case of hyperbolic tangent nonlinearities combined with using categorical cross entropy loss, in which case minibatching is not needed for compression to occur (Fig 1.6d).

1.7 Dimensionality of VGG-11

In this section we consider the deep feedforward neural network known as VGG-11 [68]. VGG-X is a set of neural network architectures of varying depth, where the number of weight layers is specified by X. These networks achieved state-of-the-art performance on the ImageNet Large Scale Visual Recognition Challenge [37] and are one of a handful of standard convolutional network models used for image classification. A schematic for this architecture is in Fig 1.7a.

We consider three datasets to use for training the network. The first two are CIFAR-10

and CIFAR-100 [69]. CIFAR-10 is a collection of 60000 32×32 color images, where each image depicts one of ten different objects. CIFAR-100 is the same, but with 100 different objects being depicted. The third is a subset of ImageNet. ImageNet is a large dataset of 224×224 color images, with 1000 categories. For our subset, we take the first 10 training labels and the images associated with these. We here call this subset ImageNet-10 for convenience. The reason for taking this subset is that the time required to train a network on the full ImageNet dataset for even a single epoch takes many hours on a single GPU.

Fig 1.7b shows the results of training the network on CIFAR-10. This figure reveals a theme: after training, the dimensionality increases in initial layers and decreases in the last layers. This behavior emerges quickly through training, from a relatively flat profile before training to a dramatic hump by epoch 5, and converging to a stationary profile by epoch 35. Figs 1.7c and 1.7d show similar behavior in the case of CIFAR-100 and ImageNet-10, respectively. These trends correspond to generally decreasing distances between points in the same class Fig 1.11.

Is this dimensionality expansion in some sense optimal? Is it limited by the architecture, or by the training dynamics? One interesting feature illustrated by these plots is that the dimensionality of the network trained on CIFAR-100 is not significantly higher than that of the network trained on CIFAR-10. This is perhaps surprising, since more class labels would suggest the need to generate more features and embed the images in a higher-dimensional space in order to unravel them. To probe these questions, we pretrain networks to expand dimensionality (in this pretraining stage, the network is not trained on the class labels). Since the effective dimensionality measure is differentiable with respect to the model parameters, we can use the standard automatic gradient computation engines to push the model to increase it. In Figs 1.7e to 1.7g, the model is trained to increase dimensionality of the representation for layers 1 through 24 on datasets CIFAR-10, CIFAR-100, and ImageNet-10, respectively. This pretraining is successful in prompting the network to lift its dimensionality significantly above the final dimensionality in Figs 1.7b to 1.7d. However, during training the dimensionality immediately drops after a single epoch, and slowly grows back to the levels reached in Figs 1.7e to 1.7g. This suggests that the the extra dimensionality expansion is not useful to the network, so that the final dimensionality may in some sense be optimal. Alternatively, the learning dynamics may not be able to make use of this expansion, or this expansion learned by the pretraining may not actually be a useful operation in this case. This story is reinforced by Fig 1.7h, which compares the accuracy of a network pretrained to

expand dimensionality with one that is not. Pretraining in this way confers no meaningful benefits to task performance. Results are similar for CIFAR-100 and ImageNet-10 (not shown). Results are also similar if the loss used during the pretrain stage combines expanding dimensionality with the loss on the training set (not shown).

As in the case for the dense feedforward network, we next probe the coding properties of the representation in Fig 1.8. Fig 1.8a shows the same linear separability metric as in Fig 1.4c, applied to the response of VGG-11 to the entire CIFAR-10 training dataset (using CIFAR-100 or ImageNet was prohibitively computationally expensive). This reveals that the linear separability of the representation in the untrained network increases over the layers where dimensionality increases, and then decreases after the dimensionality falls off in later layers. Training gradually lifts the separability, until the data are perfectly linearly separable by about layer 13. Fig 1.8c measures the same quantity as Fig 1.8a, but this time the network is pretrained to expand dimensionality. While this pretraining results in a more linearly separable representation, this is discarded in subsequent training epochs. Ultimately pretraining doesn't do much to impact the linear separability of the trained network. Fig 1.8c illustrates the test accuracy of the linear classifier, where 80% of the points are used to train the classifier, and 20% are used to test the accuracy. This shows that test accuracy gradually emerges through training, and builds up through the layers. Notably, the linear separability achieved by layer 12 in the trained network is not sufficient to allow the linear classifier to generalize. As the dimension is compressed, such generalization becomes more successful. Fig 1.8d shows the same quantity as in Fig 1.8c, but with a pretraining stage. This pretraining again doesn't play a big role in the network's behavior.

1.7.1 Intrinsic dimensionality

Representations in image classification networks such as VGG-11 are sometimes hypothesized to be described by nonlinear, curved manifolds, with a single manifold typically existing for each object class. Effective dimensionality is sensitive to this nonlinear curvature. There is a meaningful notion of dimension that instead ignores the curvature of the manifold: the number of variables needed to parametrize the manifold. A common example is given by the "swiss-roll", where a two-dimensional sheet is curled up and embedded in a three dimensional space. While the manifold from the perspective of effective dimensionality is essentially 3d, the *intrinsic*, nonlinear dimension of the manifold is two, since two variables are sufficient to parametrize the manifold. Many approaches have been taken to estimate this intrinsic

dimension, resulting in multiple intrinsic dimensionality measures. One approach that has been successfully applied to the representations of deep neural networks is described in [70], and is similar to the approach simultaneously used in our paper [60]. We use the approach of [70] here, for which there is excellent Python code available.

In measuring intrinsic dimensionality (ID), we measure the ID of each class manifold individually, and report the mean and standard deviation. The results are similar to the case of ED. Figs 1.9a to 1.9c all show the signature humpback shape that is learned through training for CIFAR-10, CIFAR-100, and ImageNet-10, respectively.

1.8 Discussion

In this chapter we probed properties of the representations of deep feedforward neural networks as these representations are transformed through the layers. Starting with simple dense feedforward networks, we found that the networks learn to compress high dimensional inputs to a low-dimensional representation gradually through the layers. On the other hand, when faced with two-dimensional inputs the networks learn to expand the representation slightly, but this expansion is not sufficient to linearly separate the class labels and the networks do not learn to classify the data perfectly. This points to stochastic gradient descent being predisposed to compressing representations when this doesn't interfere with decreasing the loss, at least in dense layers.

To probe this point, we analyzed a simple model of a single hidden layer linear network and found that, under certain assumptions, SGD does indeed cause compression. These arguments revealed that in fact variation of the output weights, or some other mechanism, is necessary to prompt gradient descent to compress the hidden unit representations in all dimensions. We probed whether or not the variability generated by SGD makes a meaningful difference, and found that when looking at deep dense feedforward networks, minibatching over learning makes the difference between the networks learning a compressed representation and not learning such a representation in most scenarios, with the caveat that accelerated learning optimizers such as RMSprop or Adam must be used during training.

We next tested these ideas on the deep convolutional neural network VGG-11 trained on several image classification datasets. We found that the networks learn to lift the dimensionality of their representation in the first layers, and compress this dimensionality in the last layers. By manipulating the dimensionality of the network in a pretraining stage, we found that the ultimate dimensionality of the trained network isn't greatly affected by this

pretraining. In addition, this pretraining had little effect on performance. This suggests that the training dynamics are sufficient for learning to expand dimensionality. However, more studies are needed, such as applying the dimensionality regularizer throughout training. We next considered a nonlinear measure of intrinsic dimensionality, and found similar patterns of dimensionality expansion followed by compression.

Intriguingly, we found that in the network trained on CIFAR-10, the representation on the training data is linearly separable by layer 12. This means that a complex transformation in the remaining 20 layers is unnecessary for achieving perfect classification accuracy of the dataset – a linear readout would suffice. However, these layers do appear to be important to allow a linear classifier trained on a subset of the data to generalize. Layer 12 also appears to be the peak point of dimensionality expansion, and after this point the trend is dimensionality compression. Taken together this indicates that the network does more than simply separate class manifolds to be linearly separable – the later layers engage in transformations that help the learned solutions generalize.

1.9 Methods

See Section 3.2 for a description of the network architecture. The equation for the categorical cross entropy loss is $\mathrm{CCE}(\hat{\boldsymbol{o}}, k) = \log(p_k)$ where $p_k = \exp(\hat{o}_k) \,/\, \left(\sum_j \exp(\hat{o}_j)\right)$ represents the probability that $\hat{\boldsymbol{o}}$ indicates class k. This loss is used to update the parameters via backpropagation using PyTorch 1.5.1 optimization schemes. In Section 1.3.1 we use the RMSprop optimization scheme with a batch size of 10 (loss is summed over the batch). The plots in Fig 1.4 used a learning rate of .0001 and the plots in Fig 1.5 used a rate of .001. These were chosen to roughly maximize performance, particularly in the case of low-dimensional inputs. Similar results are found when using the Adam optimization scheme, although the network performance in the case of low-dimensional inputs is diminished. For VGG-11 in Section 1.7 we use the PyTorch implementation of SGD with a momentum factor of 0.9. In both cases, 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 "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 parameter with the most significant impact on our

results is the learning rate. Decreasing the learning rate can decrease the degree to which the network trained on low-dimensional inputs expands dimensionality, as well as the degree to which this expansion is followed by a compression. Increasing the learning rate can make the dimensionality compression of high-dimensional inputs occur more rapidly through the layers.

The clustered 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 $l = 4$ 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). Our results are robust to the exact specifics of this arrangement; for instance, cluster centers can be drawn from within a d-dimensional ball. 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, where each column has norm $1/\sqrt{d}$ (this transformation preserves the geometry of the inputs). 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. In the dense network, weights $\boldsymbol{W}^\ell$ are initialized as the identity matrix (with extra zeros in the case of the final weight layer).

Throughout the chapter, shaded regions on plots indicate 75% probability mass of a shifted gamma distributions fit to values of the dependent variables over a sample of 10 network and input realizations, with solid lines indicating the median of the distribution. The standard gamma distribution has $x \geq 0$ as its support. For the dimensionality plots,

the distribution is shifted so that its support is at $x \geq 1$, reflecting the fact that $ED \geq 1$. For the accuracy plots, the distribution is reflected around $x = 0$, and then shifted so that its support is $x \leq 1$, reflecting the fact that accuracy is bounded from above by 1. The gamma distribution is chosen over the normal distribution (as is done when reporting standard deviation) because it is better at capturing the skewness of the data, which is often substantial.

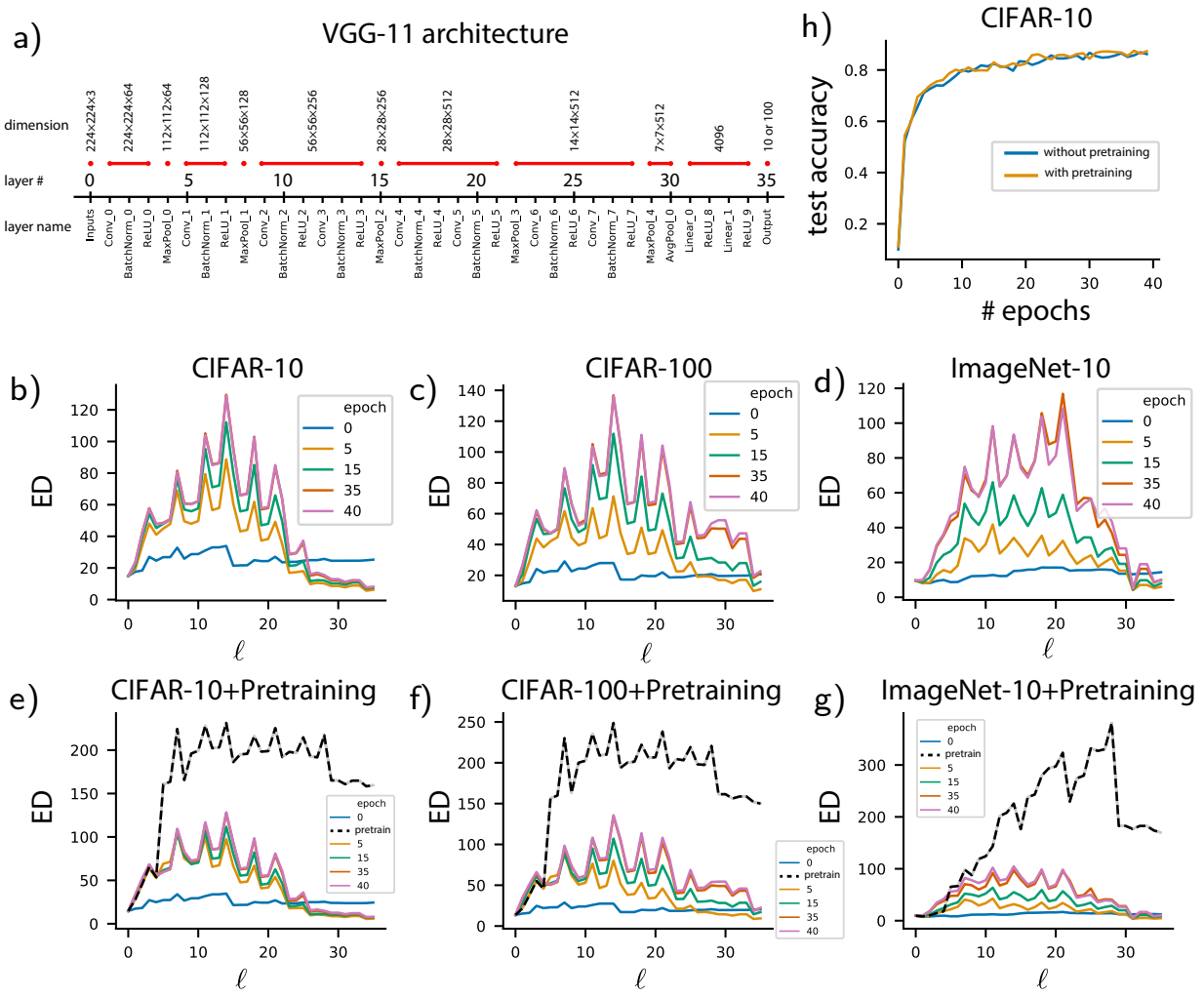

Figure 1.7: **Effective dimensionality of VGG-11.** (a) Schematic of VGG-11 architecture. The dimension of the layer is indicated above, with red lines to indicate layers where the dimension remains constant. The layer indices are indicated in increments of 5. The layer names (type and number) are below. (b) Effective dimensionality of VGG-11 trained on CIFAR-10, with layers ℓ on the x axis. Colors correspond to epochs (see legend). (c) As in b, but for CIFAR-100. (d) As in c, but for ImageNet-10. (e) As in b, but this time the network is first pretrained to increase dimensionality (the network is not trained on the actual data during this pretraining phase). (f) As in e, but for CIFAR-100. (g) As in f, but for ImageNet-10. (h) Accuracy of networks through training. Blue line depicts a network trained as usual. Orange line depicts a network that is first pretrained to expand dimensionality, as in (e).

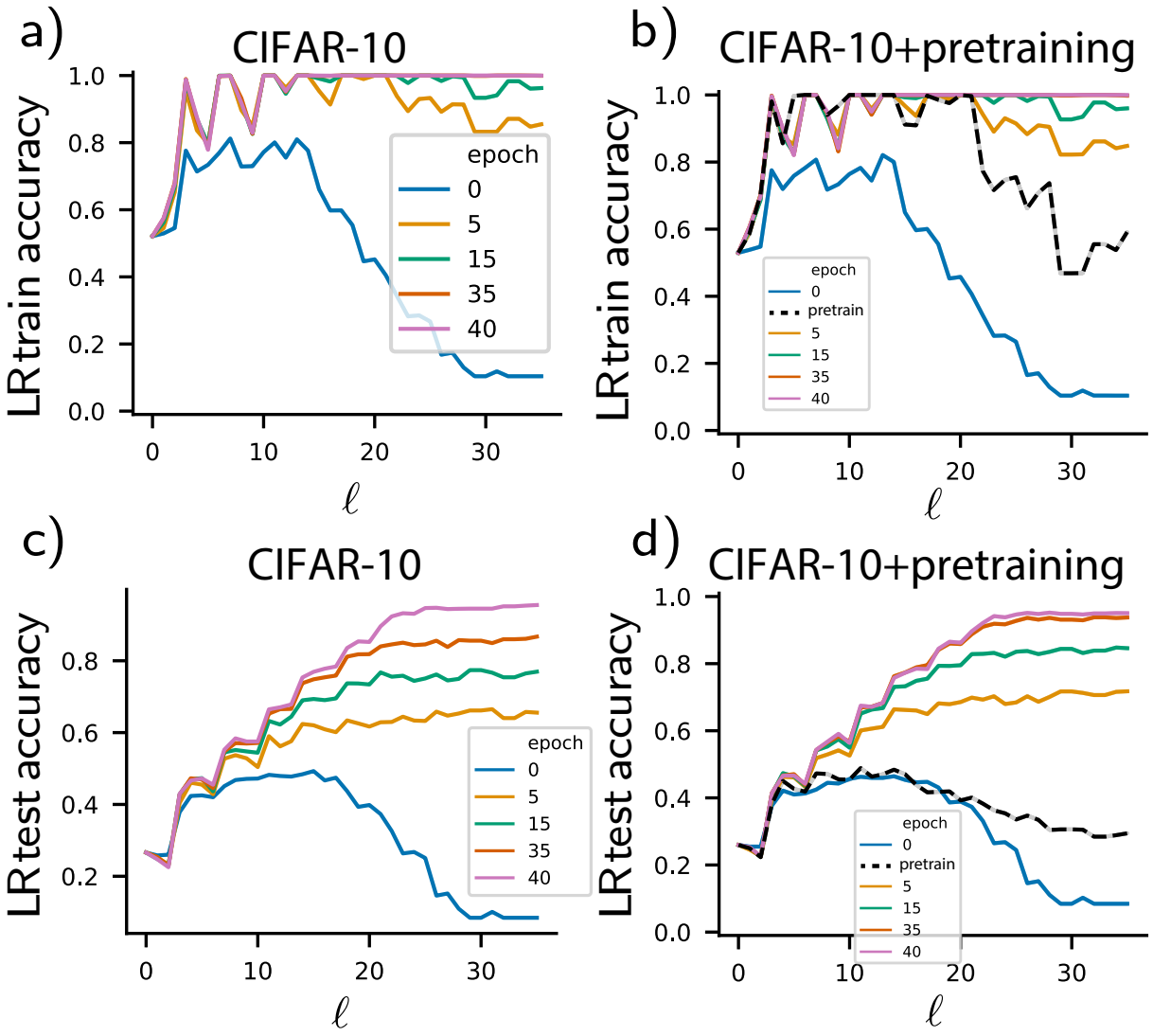

Figure 1.8: **Linear separability of the class manifolds through the layers.** (a) The accuracy of a linear classifier trained to predict true class labels on a subset of the data, measured through the layers. (b) As in (a), but with epoch 1 being a used to pretrain the network to expand dimensionality. (c) Accuracy of the classifier trained in (a) on a held-out set of data. (d) Accuracy of the classifier trained in (b) on a held-out set of data.

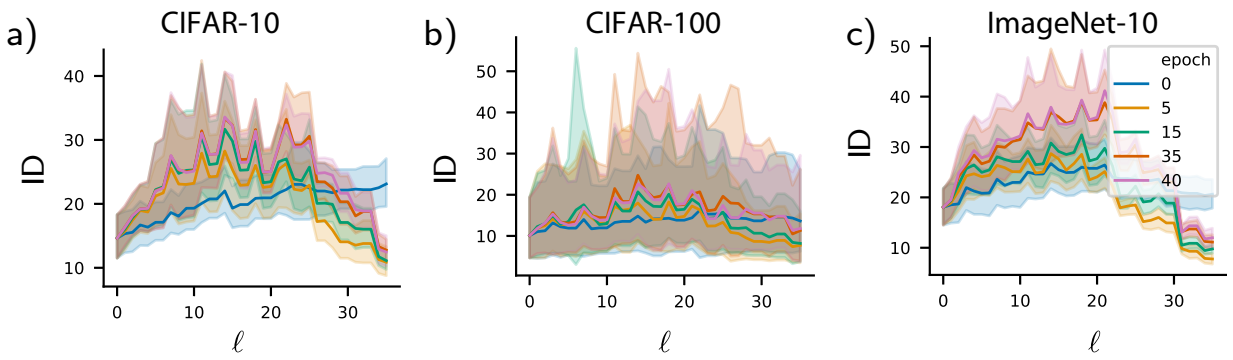

Figure 1.9: **Intrinsic dimensionality of VGG-11.** (a) Intrinsic dimensionality (ID) of class manifolds measured through the layers, where the network is trained on CIFAR-10. Each class manifold is a datapoint, whose variation is depicted by the shaded regions. The shaded regions are defined as in Fig 1.7 and Chapter 1 Methods. Solid lines indicate medians of these fitted distributions. (b) Same as (a), but for CIFAR-100. (c) Same as (a), but for ImageNet-10.

1.10 Appendix

1.10.1 Details of the calculations for Section 1.6.3

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:

$$\delta \boldsymbol{W} = -\frac{\gamma}{|B|} \sum_{\boldsymbol{x} \in B} \frac{\partial \ell(\boldsymbol{x})}{\partial \boldsymbol{W}} = -\frac{\gamma}{|B|} \sum_{\boldsymbol{x} \in B} e(\boldsymbol{x}) \boldsymbol{r} \boldsymbol{x}^\top, \tag{1.15}$$

$$\delta \boldsymbol{b} = -\frac{\gamma}{|B|} \sum_{\boldsymbol{x} \in B} \frac{\partial \ell(\boldsymbol{x})}{\partial \boldsymbol{b}} = -\frac{\gamma}{|B|} \sum_{\boldsymbol{x} \in B} e(\boldsymbol{x}) \boldsymbol{r} \tag{1.16}$$

$$\delta \boldsymbol{r} = -\frac{\gamma}{|B|} \sum_{\boldsymbol{x} \in B} \frac{\partial \ell(\boldsymbol{x})}{\partial \boldsymbol{r}} = -\frac{\gamma}{|B|} \sum_{\boldsymbol{x} \in B} e(\boldsymbol{x}) \boldsymbol{h} \tag{1.17}$$

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 $\boldsymbol{r}_k$.

We start by computing Equation 5 of the main text, under the assumption that the readout weights take the form $\boldsymbol{r}_0 = \overline{\boldsymbol{r}} + \boldsymbol{\xi}$ 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. It is helpful to introduce some notation. Suppose that P is the orthogonal projector onto the nullspace of $\boldsymbol{r}$. Let $\mathcal{P}(\boldsymbol{r})$ denote any quantity that is mapped to zero by this projector (this notation functions similarly to "big O" notation). Examples of objects that are $\mathcal{P}(\boldsymbol{r})$ are vectors proportional to $\boldsymbol{r}$ and matrices of the form $\boldsymbol{r}\boldsymbol{v}^\top$.

A single step of gradient descent results in the representation $\boldsymbol{h}_1(\boldsymbol{x}') = \boldsymbol{h}_0(\boldsymbol{x}') + \delta \boldsymbol{W}_0 \boldsymbol{x}' + \delta \boldsymbol{b}_0$. This is

$$\boldsymbol{h}_1(\boldsymbol{x}') = \boldsymbol{h}_0(\boldsymbol{x}') - \frac{\gamma}{|B|} \sum_{x \in B} e(x) \boldsymbol{r}_0 \boldsymbol{x}^\top \boldsymbol{x}' - \frac{\gamma}{|B|} \sum_{x \in B} e(\boldsymbol{x}) \boldsymbol{r}_0.$$

Substituting $\boldsymbol{r}_0 = \overline{\boldsymbol{r}} + \boldsymbol{\xi}$ and $e(\boldsymbol{x}) = \boldsymbol{r}_0^\top(\boldsymbol{W}_0\boldsymbol{x} + \boldsymbol{b}) - o(\boldsymbol{x})$, this becomes

$$\boldsymbol{h}_1(\boldsymbol{x}') = \boldsymbol{h}_0(\boldsymbol{x}') - \frac{\gamma}{|B|}\sum_{x\in B}(\boldsymbol{r}_0\boldsymbol{r}_0^\top(\boldsymbol{W}_0\boldsymbol{x} + \boldsymbol{b}) - \boldsymbol{r}_0 o(\boldsymbol{x}))\boldsymbol{x}^\top\boldsymbol{x}' - \frac{\gamma}{|B|}\sum_{x\in B} e(\boldsymbol{x})\boldsymbol{r}_0.$$

Averaging over $\boldsymbol{\xi}$ and batches B, this becomes

$$\langle \boldsymbol{h}_1(\boldsymbol{x}')\rangle = \boldsymbol{h}_0(\boldsymbol{x}') - \gamma\langle\boldsymbol{\xi}\boldsymbol{\xi}^\top\rangle\boldsymbol{W}_0\langle\boldsymbol{x}\boldsymbol{x}^\top\rangle\boldsymbol{x}' + \mathcal{P}(\overline{\boldsymbol{r}}) \tag{1.18}$$

$$= \boldsymbol{h}_0(\boldsymbol{x}') - \gamma\sigma_\xi^2\sigma_x^2\boldsymbol{W}_0\boldsymbol{x}' + \mathcal{P}(\overline{\boldsymbol{r}}). \tag{1.19}$$

This concludes the derivation of Equation 5.

We next derive Equation 7, which gives the hidden representation $\boldsymbol{h}_2(\boldsymbol{x})$ after two steps of SGD and shows 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

$$\boldsymbol{W}_2 = \boldsymbol{W}_1 - \gamma\nabla^1_{\boldsymbol{W}}\ell(\boldsymbol{x}_1) = \boldsymbol{W}_0 - \gamma\nabla^0_{\boldsymbol{W}}\ell(\boldsymbol{x}_0) - \gamma\nabla^1_{\boldsymbol{W}}\ell(\boldsymbol{x}_1) \tag{1.20}$$

and

$$\boldsymbol{b}_2 = \boldsymbol{b}_1 - \gamma\nabla^1_{\boldsymbol{b}}\ell(\boldsymbol{x}_1) = \boldsymbol{b}_0 - \gamma\nabla^0_{\boldsymbol{b}}\ell(\boldsymbol{x}_0) - \gamma\nabla^1_{\boldsymbol{b}}\ell(\boldsymbol{x}_1). \tag{1.21}$$

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

$$\begin{aligned}\boldsymbol{h}_2(\boldsymbol{x}) &= \boldsymbol{W}_2\boldsymbol{x} + \boldsymbol{b}_2 \\ &= (\boldsymbol{W}_0 - \gamma\nabla^0_{\boldsymbol{W}}\ell(\boldsymbol{x}_0) - \gamma\nabla^1_{\boldsymbol{W}}\ell(\boldsymbol{x}_1))\boldsymbol{x} + \boldsymbol{b}_0 - \gamma\nabla^0_{\boldsymbol{b}}\ell(\boldsymbol{x}_0) - \gamma\nabla^1_{\boldsymbol{b}}\ell(\boldsymbol{x}_1) \\ &= \boldsymbol{h}_0(\boldsymbol{x}) - \gamma\nabla^0_{\boldsymbol{W}}\ell(\boldsymbol{x}_0)\boldsymbol{x} - \gamma\nabla^1_{\boldsymbol{W}}\ell(\boldsymbol{x}_1)\boldsymbol{x} - \gamma\nabla^0_{\boldsymbol{b}}\ell(\boldsymbol{x}_0) - \gamma\nabla^1_{\boldsymbol{b}}\ell(\boldsymbol{x}_1) \qquad (1.22)\end{aligned}$$

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. (1.22) 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. Then $\boldsymbol{h}_2(\boldsymbol{x})$ can be written

$$\boldsymbol{h}_2(\boldsymbol{x}) = \boldsymbol{h}_0(\boldsymbol{x}) - \gamma\nabla^1_{\boldsymbol{W}}\ell(\boldsymbol{x}_1)\boldsymbol{x} - \gamma\nabla^1_{\boldsymbol{b}}\ell(\boldsymbol{x}_1) + \mathcal{P}(\boldsymbol{r}_0) \qquad (1.23)$$

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

$$\langle \boldsymbol{h}_2(\boldsymbol{x})\rangle = \boldsymbol{h}_0(\boldsymbol{x}) - \gamma\left\langle\nabla^1_{\boldsymbol{W}}\ell(\boldsymbol{x}_1)\right\rangle\boldsymbol{x} + \mathcal{P}(\boldsymbol{r}_0). \qquad (1.24)$$

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

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

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 \\
&\quad + \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 \\
&\quad + \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 \\
&\quad + \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 \\
&\quad + \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 \\
&\quad + \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 \\
&\quad + \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 \\
&\quad + \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 \\
&\quad + \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 \\
&\quad + \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 \\
&\quad + \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

$$\left\langle \boldsymbol{h}_0(\boldsymbol{x}_1) \boldsymbol{x}_1^\top \right\rangle = \left\langle (\boldsymbol{W} \boldsymbol{x}_1 + \boldsymbol{b}_0) \boldsymbol{x}_1^\top \right\rangle = \boldsymbol{W} \left\langle \boldsymbol{x}_1 \boldsymbol{x}_1^\top \right\rangle = \sigma_x^2 \boldsymbol{W}. \tag{1.25}$$

This leaves us with the expression

$$\begin{aligned}
\left\langle \nabla^1_{\boldsymbol{W}} \ell(\boldsymbol{x}_1) \right\rangle &= \gamma^2 \sigma_x^2 \left\langle \nabla^0_{\boldsymbol{r}} \ell(\boldsymbol{x}_0) \boldsymbol{r}_0^\top (\nabla^0_{\boldsymbol{W}} \ell(\boldsymbol{x}_0)) \right\rangle \\
&\quad + \gamma^2 \sigma_x^2 \left\langle \nabla^0_{\boldsymbol{r}} \ell(\boldsymbol{x}_0) \nabla^0_{\boldsymbol{r}} \ell(\boldsymbol{x}_0)^\top \right\rangle \boldsymbol{W} \\
&\quad + \left\langle \mathcal{P}(\boldsymbol{r}_0) \boldsymbol{x}_1^\top \right\rangle + \mathcal{O}(\gamma^3)
\end{aligned} \tag{1.26}$$

Substituting this expression for $\langle \nabla^1_{\boldsymbol{W}} \ell(\boldsymbol{x}_1) \rangle$ into Eq. (1.24) and simplifying results in

$$\begin{aligned} \langle \boldsymbol{h}_2(\boldsymbol{x}) \rangle = \boldsymbol{h}_0(\boldsymbol{x}) & \\ - \gamma^3 \sigma_x^2 \left\langle \nabla^0_{\boldsymbol{r}} \ell(\boldsymbol{x}_0) \nabla^0_{\boldsymbol{r}} \ell(\boldsymbol{x}_0)^\top \right\rangle \boldsymbol{W}_0 \boldsymbol{x} & \\ - \gamma^3 \sigma_x^2 \left\langle \nabla^0_{\boldsymbol{r}} \ell(\boldsymbol{x}_0) \boldsymbol{r}_0^\top \nabla^0_{\boldsymbol{W}} \ell(\boldsymbol{x}_0) \right\rangle \boldsymbol{x} + \mathcal{P}(\boldsymbol{r}_0) + \mathcal{O}(\gamma^4). & \end{aligned} \tag{1.27}$$

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. (1.27) 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. (1.27) 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. (1.27) 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

$$\left\langle \nabla^0_{\boldsymbol{r}} \ell(\boldsymbol{x}_0)(\nabla^0_{\boldsymbol{r}} \ell(\boldsymbol{x}_0))^\top \right\rangle \boldsymbol{W}_0 \boldsymbol{x} = \boldsymbol{W}_0 \left\langle e(\boldsymbol{x}_0)^2 \boldsymbol{x}_0 \boldsymbol{x}_0^\top \right\rangle \boldsymbol{W}_0^\top \boldsymbol{W}_0 \boldsymbol{x} + \mathcal{P}(\boldsymbol{r}_0) \tag{1.28}$$

$$= \left\langle e(\boldsymbol{x}_0)^2 \right\rangle \sigma_x^2 \boldsymbol{W}_0 \boldsymbol{W}_0^\top \boldsymbol{W}_0 \boldsymbol{x} + \mathcal{P}(\boldsymbol{r}_0). \tag{1.29}$$

Calculations for the third term of Eq. (1.27) follow a similar flow, resulting in

$$\left\langle (\nabla^0_{\boldsymbol{r}} \ell(\boldsymbol{x}_0)) \boldsymbol{r}_0^\top (\nabla^0_{\boldsymbol{W}} \ell(\boldsymbol{x}_0)) \right\rangle \boldsymbol{x} = \sigma_x^2 \left\| \boldsymbol{r}_0 \right\|^2 \left\langle e(\boldsymbol{x}_0)^2 \right\rangle \boldsymbol{W}_0 \boldsymbol{x} + \mathcal{P}(\boldsymbol{r}_0). \tag{1.30}$$

Substituting Eq. (1.30) and Eq. (1.28) into Eq. (1.27) 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

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

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 in these directions.

1.10.2 Additional figures

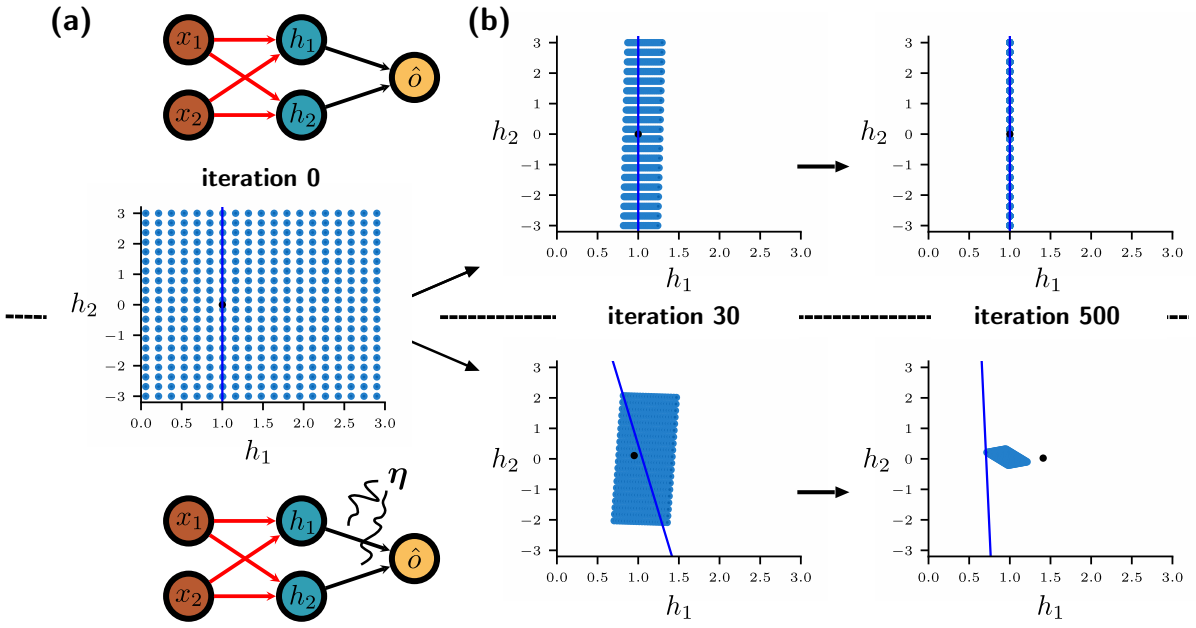

Figure 1.10: **Example of noise in output weights driving compression of the hidden representation in a linear network with two hidden layer units.** The equation for the network is $\boldsymbol{h} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$ with output $\hat{o} = \boldsymbol{r}^T\boldsymbol{h}$. 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 $\hat{o}$ is trained to minimize the squared error loss $0.5(\hat{o} - 1)^2$. 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). Black dot denotes the output weight vector, and the blue line is the affine subspace of points that $\overline{\boldsymbol{r}}$ 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 $\overline{\boldsymbol{r}}$, 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 $\overline{\boldsymbol{r}}$.

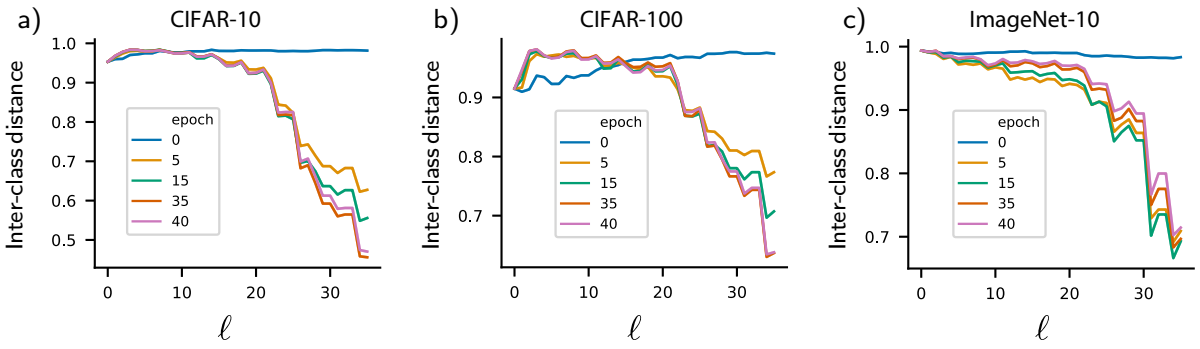

Figure 1.11: **Average inter-class distances between points in class 1, normalized by average distance between points in class 1 and class 2.** Color indicates training epoch. (a) Inter-class distance measured through the layers of the network trained on CIFAR-10. (b) Inter-class distance measured through the layers of the network trained on CIFAR-100. (c) Inter-class distance measured through the layers of the network trained on ImageNet-10.

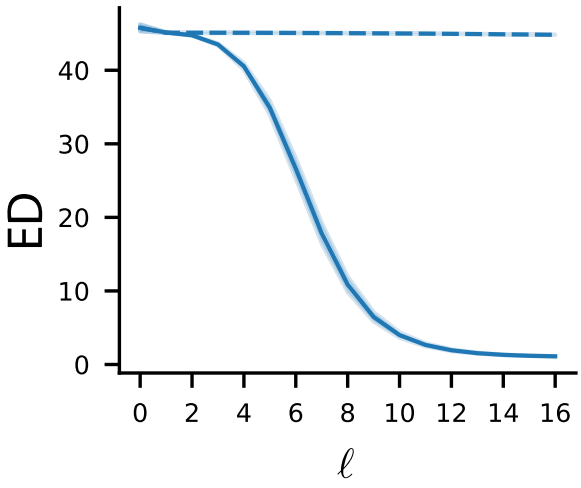

Figure 1.12: **Dimensionality compression in 16 layer feedforward networks.** Details are as in Fig 1.1b, but using a feedforward network with 16 hidden layers.

Chapter 2

DIMENSIONALITY BALANCE IN RECURRENT NEURAL NETWORKS

2.1 Introduction

Feedforward neural networks, which were the focus of Chapter 1, are key to the fields of machine learning and computational neuroscience. However, it is also of high interest to understand the behavior of recurrent neural networks (RNNs). Besides possessing useful computational attributes for machine learning applications, recurrent networks capture essential elements of biological brains. Recurrent network models reflect the abundance of feedback and lateral connections between neurons in the brain. This architecture supports ongoing temporal dynamics, which is both observed biologically and useful for machine learning applications in the domain of processing inputs with temporal structure. These networks form representations that change through time, enabling complex computations analogous to those performed through the layers of deep feedforward networks. In this context, it is essential to understand how the dynamical properties of the network influence the formation of representations through learning.

Studies in this vein have been successfully applied to recurrent networks with fixed recurrent weights. These findings have revealed the important role of dynamical chaos (whereby small changes in internal states are amplified by unstable, but deterministic, dynamics) in determining the computational properties of RNN models [71, 72, 73, 74]. Frequently encountered in models of brain networks, 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 [75, 76, 77, 78, 79, 80, 81, 82]. Furthermore, chaos has been linked to dimensionality; mathematically in [74], and in [73] through simulations showing that chaos can drive an expansion of dimensionality which is useful for discriminating between inputs. However, the attributes of this phenomenon as recurrent weights evolve through training are less understood (but see [83, 84]).

The goal of this work is to reveal how the dynamics of recurrent networks and the dynamics of learning interact to influence the geometry of the learned representations. We

furthermore probe the computational consequences of these representations. As in Chapter 1, we find experimental evidence that stochastic gradient descent (SGD) is predisposed to find solutions that compress dimensionality. Whereas in Chapter 1 we found that the dense networks were able to expand dimensionality (but not necessarily enough to classify the data perfectly), in the case of these recurrent networks, dimensionality expansion may not occur. In light of this, chaos in this 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 compressing opportunistically. This results in learned representations that balance expansion and compression and that consequently lend themselves to good generalization.

Our results indicate new computational roles for variability – both in neural responses as well as in synaptic modifications – in the formation of robust representations.

Publications and preprints. The contents of this chapter can be found in [48].

2.2 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 [85, 86] (see Chapter 2 Methods). Interactions between the $n = 200$ neural units are determined by a randomly initialized recurrent connectivity matrix. The dynamics of random recurrent networks typically become more chaotic as the average magnitude of neural coupling strength increases. 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 a classification task described below. To measure chaos, we numerically compute the *Lyapunov exponents* of the system.

The network is trained to perform a delayed classification task, which proceeds as follows: (1) an input is 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. This delay period is analogous to the hidden layers of a feedforward neural network, when viewing the recurrent weights as being "unrolled" through time. Inputs are the same as the clustered data in Chapter 1. For convenience, we repeat the description here. These inputs are n-dimensional

vectors. They are selected from Gaussian-distributed clusters whose means are distributed randomly within an d-dimensional, random subspace of the n-dimensional neural activity space. Each cluster is assigned one of two class labels, at random.

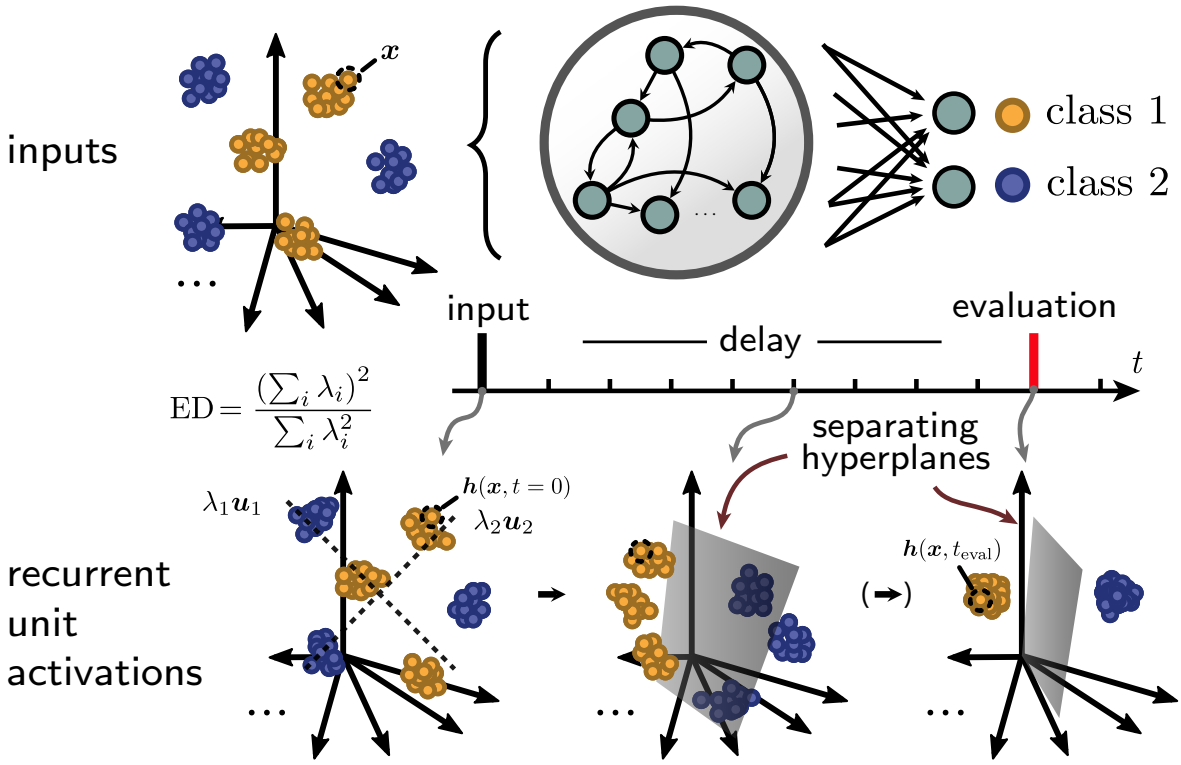

Figure 2.1: **Task and model schematic.** Input clusters are distributed in a n-dimensional subspace of neural space. An input $\boldsymbol{x}$ is shown to the network for one timestep, and after a delay period in which the network evolves undriven, the network state $\boldsymbol{h}(\boldsymbol{x}, t_{\mathrm{eval}})$ 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.

SGD trains networks to compress high-dimensional inputs

We start by considering the classification of high-dimensional inputs, where the input ambient space dimensionality m is equal to the number n of recurrent units (see Fig 2.2a for a visualization). As in Chapter 1, these inputs are linearly separable, so that the question is what recurrent networks do beyond producing linear separability.

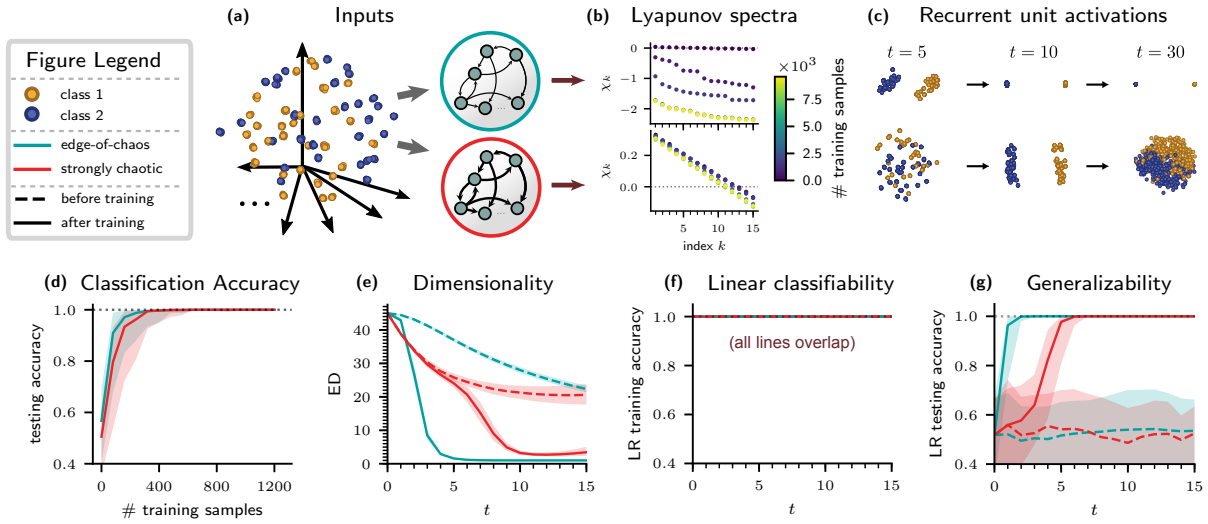

Figure 2.2: **Dynamical and geometric properties of networks learning to classify high-dimensional inputs.** Two dynamical regimes are considered: initialization on the edge-of-chaos (blue-green) and a strongly chaotic (red) initialization. Input color (yellow-orange or purple) denotes true class label. Shaded regions are as described in Chapter 1 Methods. "After training" designates networks that have been trained on 9,600 input samples. (a) Schematic of the inputs. (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).

We study networks in two dynamical regimes: a strongly chaotic network and one that is initialized at the edge of chaos. Fig 2.2b 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 2.2b 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 2.2d, both networks easily achieve perfect testing accuracy on the delayed classification task. As in Chapter 1, we now turn our attention to how properties of the network's internal representation change over the course of training.

In Fig 2.2c 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 2.7 for a quantification of this phenomenon). The case is similar for the 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. The chaotic solution has some benefits when compared to the solution found by the edge-of-chaos network. The edge-of-chaos network has moved points that belong to the same class very close together, so if the data were to change (say, the class labels were to be scrambled), it would be difficult to learn this change. In contrast, the chaotic solution is able to learn a new instantiation of the task more quickly, as shown in Fig 2.10. Here, the input data are regenerated after ten epochs of training, and the strongly chaotic network more quickly regains perfect testing classification accuracy.

As in Chapter 1, this compression phenomenon is partly captured by measuring the effective dimensionality of the representation (see Section 1.2). In Fig 2.2e the ED is plotted through time t. 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 2.2c and Fig 2.7). As in Chapter 1, the degree of these trends depends somewhat on the learning procedure (data not shown). However, the general behavior is robust to moderate changes in the learning algorithm parameters.

We next study the coding properties of these representations as in Chapter 1. Fig 2.2f shows the mean accuracy of a logistic regression (LR) linear classifier trained to classify the network representation at each timepoint. As in Chapter 1, the input data are linearly separable, and this property is retained by the network dynamics. However, the learned dynamics and resulting dimensionality compression correspond with better generalization properties of the representation. In Fig 2.2g, 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.

As in Section 1.6.4, we test if minibatching can be a driving force behind this dimensionality compression by comparing networks trained with full batch gradient descent and stochastic gradient descent with batch size 1 (Fig 2.9). We find similar results: minibatching does make the difference, except in the case where tanh nonlinearity and categorical cross entropy is used – in this case full batch gradient descent also compresses the representation. This may have to do with the saturating behavior of the nonlinearity. As in Chapter 1, these results depend on using accelerated optimizers such as RMSprop or Adam.

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 2.3a 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

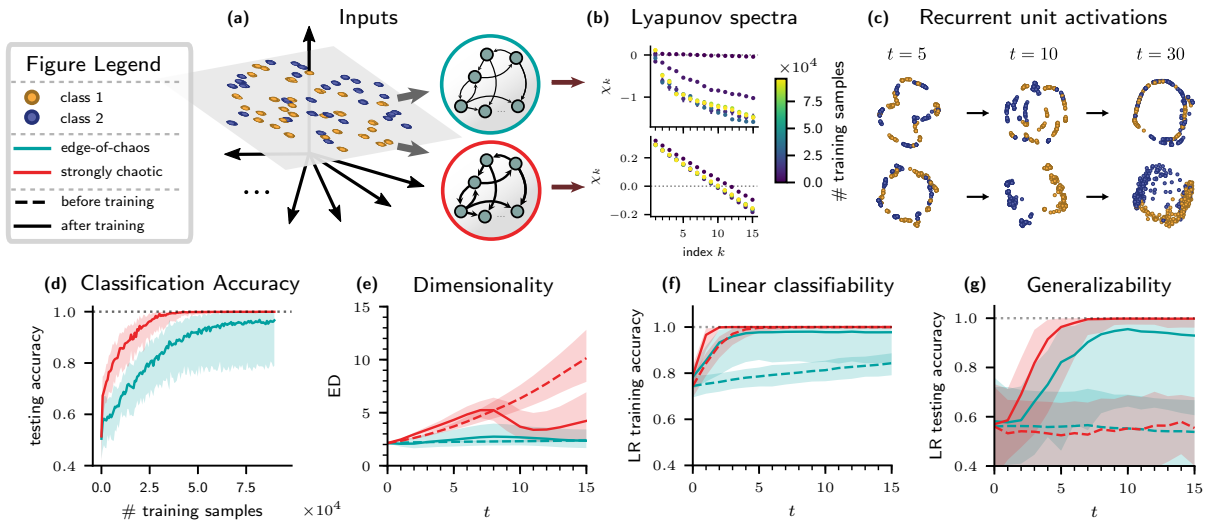

Figure 2.3: **Dynamical and geometric properties of networks learning to classify two-dimensional inputs.** Two dynamical regimes are considered: initialization on the edge-of-chaos (blue-green) and a strongly chaotic (red) initialization. Details are similar to Fig 2.2. "After training" designates networks 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 2.2.

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 2.3 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 2.3b). The strongly chaotic network remains strongly chaotic during training. In comparing the representations in Fig 2.3c, 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 2.3d 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 2.3d).

In looking at the dimensionality of the trained networks through time (Fig 2.3e), 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. This behavior is more consistent than in the case of the dense feedforward networks of Chapter 1. Changing the learning rate can change the behavior – in particular, reducing the learning rate can make both the dimensionality expansion and compression less pronounced.

In Fig 2.3f 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 2.3g), 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 2.4a); here, two. In this case, dimensionality expansion can have the added benefit of enlisting more neurons to aid in the computation (e.g. [87]). 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 2.4c). 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 2.4e). This expansion again results in good linear separability by the evaluation time $t = 10$, even before training (Fig 2.4f).

The generalization properties of the strongly chaotic network's representation improve after training, even though dimensionality compression does not occur (Fig 2.4g, solid red line). However, comparing this network at the evaluation time with the strongly chaotic network trained on distributed inputs (Fig 2.3g, 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 2.3e, solid red line, plays a role in improving generalization properties, while constrained dimensionality expansion as in Fig 2.4e, 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.

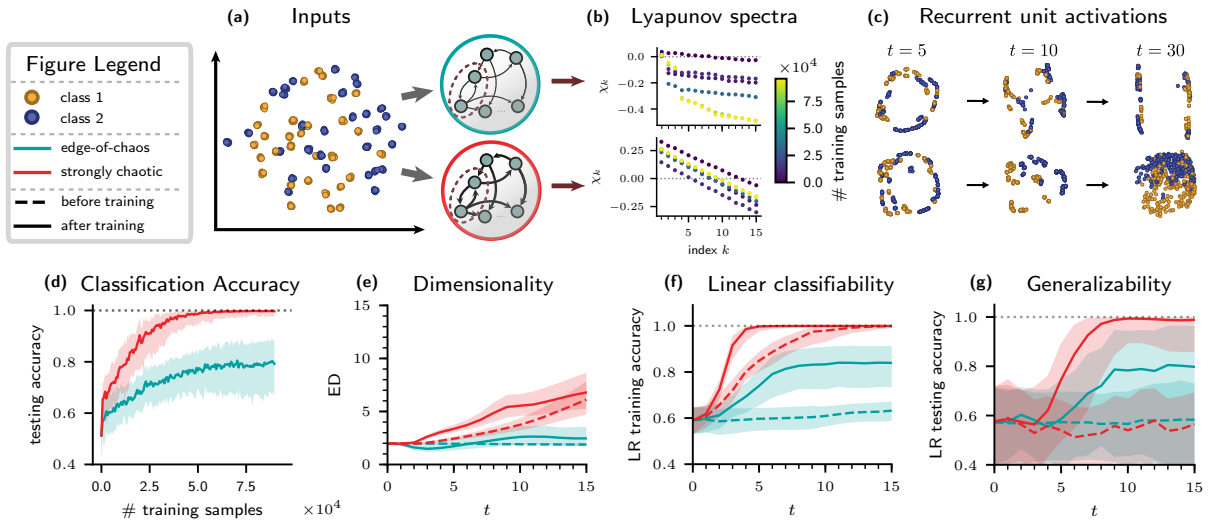

Figure 2.4: **Dynamical and geometric properties of networks learning to classify two-dimensional inputs restricted to two neurons.** Details are similar to Figs 2.2 and 2.3. "After training" designates networks that have been trained on 96,000 input samples. (a) Schematic of the inputs. Two-dimensional input is delivered to two randomly selected neurons in the network. (b-g) Descriptions as in Fig 2.2.

2.3 Discussion

RNNs learn to perform rich operations on their inputs, including expanding and compressing 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 2.6). 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. 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 [41] and models of olfactory, cerebellar, gustatory, and visual circuits [88, 89, 90, 43, 44, 91, 92], as well as reservoir computing models where recurrent weights are random and untrained [73].

On the other hand, studies have also pointed out the need to constrain dimensionality to enable generalization [73, 91, 40]. 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 2.2e), strongly chaotic networks learn to initially expand, and then compress dimensionality.

In all of these cases, strongly chaotic networks produce ongoing neural variability, a property shared by biological models of neural circuits [76, 77, 93, 94, 79, 87, 95, 36, 96] as well as experimental recordings [81, 80, 96]. 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. Assessing the extent to which these findings generalize to a wider range of tasks and network architectures will be important going forward (see Fig 2.6 and Fig 2.7 for first steps in this direction).

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 [97, 98, 99, 61, 62]). Our findings highlight a scenario where chaotic variability and salient low-dimensional structure synergistically coexist (cf. [79, 87, 36, 95]).

2.4 Methods

Model and Task details

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

$$\boldsymbol{h}_t = \tanh\left(\boldsymbol{W}\boldsymbol{h}_{t-1} + \boldsymbol{x}_t + \boldsymbol{b}^{in}\right) \tag{2.1}$$

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 nonlinearity is tanh = 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 equation for the categorical cross entropy loss is $\mathrm{CCE}(\hat{\boldsymbol{o}}, k) = \log(p_k)$ where $p_k = \exp(\hat{o}_k) \,/\, \left(\sum_j \exp(\hat{o}_j)\right)$ represents the probability that $\hat{\boldsymbol{o}}$ indicates class k. This loss is used to update the parameters via backpropagation-through-time using PyTorch 1.5.1 optimization schemes. We use the RMSprop optimization scheme with a batch size of 10 (loss is summed over the batch) and a learning rate of .001. These were chosen to roughly maximize performance, particularly in the case of low-dimensional inputs. The trends and measures analyzed are qualitatively robust to small changes in the learning parameters. Perhaps the parameter with the most significant impact on our results is the learning rate. Increasing the learning rate typically increases the amount of the network's compression after training and the degree to which Lyapunov exponents are decreased. In some cases, such as the case of two-dimensional inputs shown to all neurons, doubling the learning rate can have a strong effect in inhibiting the dimensionality expansion of the strongly chaotic network, while halving it is enough to make the compression that follows this expansion considerably fainter. We chose our initial 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. See Chapter 1 Methods for a description of the clustered data.

The input is transformed by a random $n \times d$ transformation with orthogonal columns, where each column has norm $1/\sqrt{d}$ (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 $(2,2)$ positions, and zero everywhere else. The output weights of the network are initialized as a random matrix $\boldsymbol{J'}$ where $J'_{ij} \sim \mathcal{N}(0, .3^2/n)$.

As in Chapter 1 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.

The shaded regions on plots are calculated as in Chapter 1 (see Chapter 1 Methods), except that here 30 network and input realizations are used.

Inducing Chaos

We initialize $\boldsymbol{W}$ as $\boldsymbol{W} = (1-\varepsilon)\boldsymbol{I} + \varepsilon\boldsymbol{J}$ where $\varepsilon = .01$ ensures relatively smooth dynamics before training (note that the dynamics are never completely smooth by the formal mathematical definition, even as $\varepsilon \to 0$). The matrix $\boldsymbol{J}$ has normally distributed entries that scale in magnitude with a coupling strength parameter γ, $J_{ij} \sim \backslash(0, \gamma^2/n)$. We investigate two dynamical regimes, "edge of chaos" and "strongly chaotic", with gain strength $\gamma = 20$ and $\gamma = 250$, respectively.

2.5 Appendix

2.5.1 Additional figures

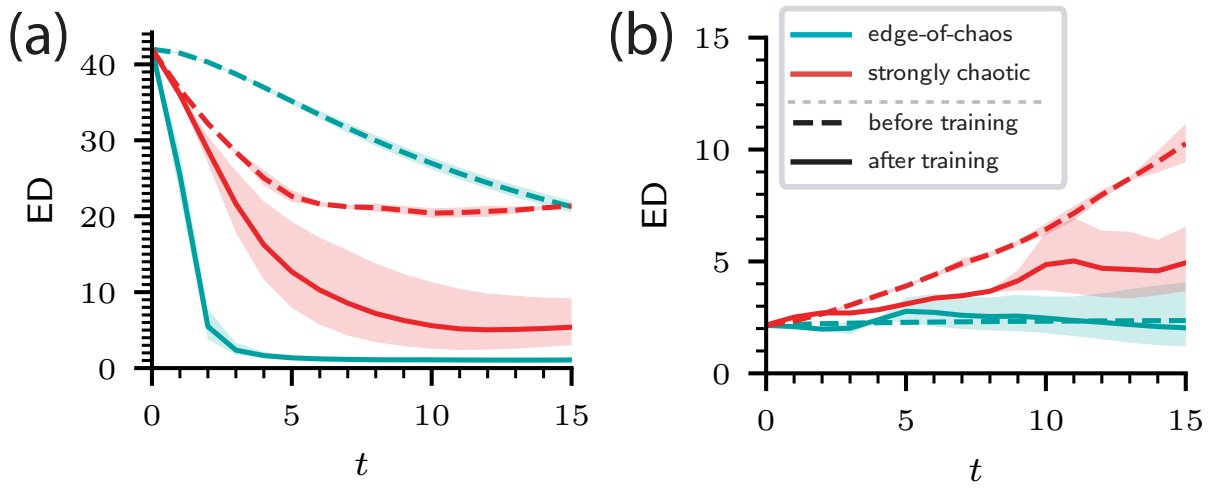

Figure 2.5: **Effective Dimensionality (ED) of networks trained with mean squared error loss resembles that of networks trained with categorical cross entropy.**. Effective Dimensionality (ED) of networks trained with mean squared error loss. Shaded regions indicate 75% probability mass of gamma distributions fit to values of the dependent variables over a sample of 2 network and input realizations, with solid lines indicating medians. (a) As in Fig. 2e of the main text, but using mean squared error loss. (b) As in Fig. 3e of the main text, but using 220 input clusters and mean squared error loss. Using 60 clusters, the edge-of-chaos network is able to learn to expand dimensionality in this case (not shown).

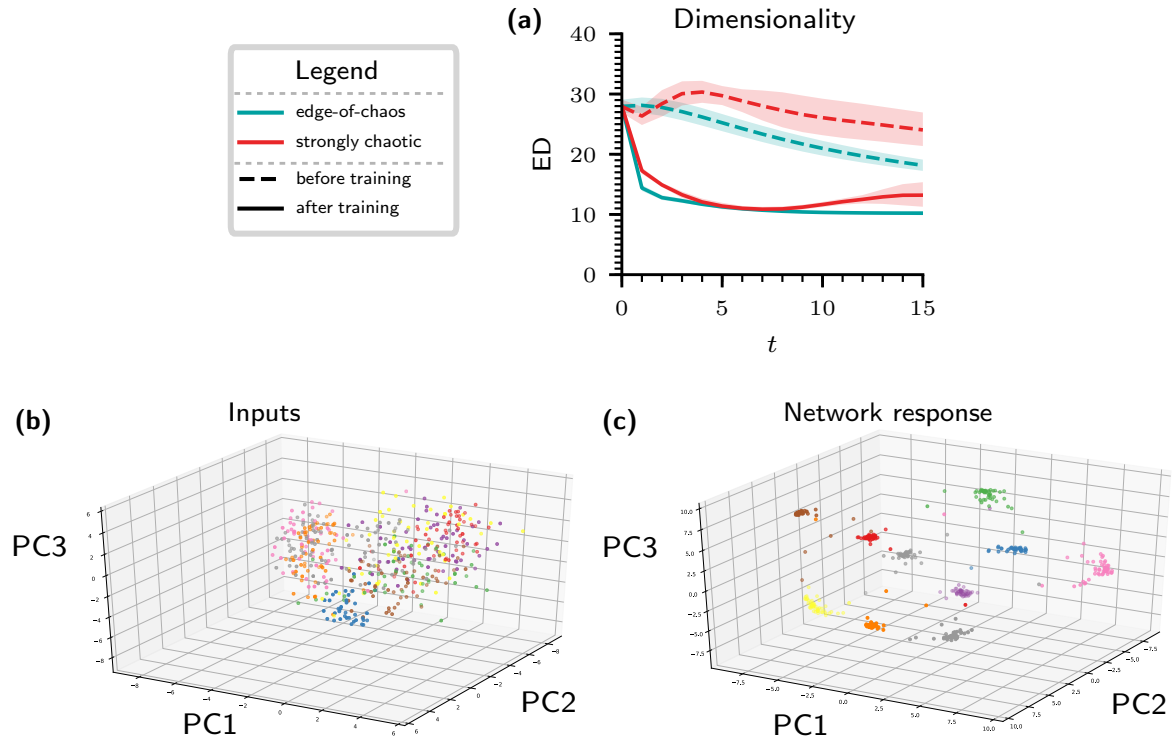

Figure 2.6: **Representations of RNNs trained on the MNIST digit recognition dataset.** (a) Effective dimensionality (ED) of RNNs trained on the MNIST digit recognition dataset. The ED of the network's responses to test inputs is plotted. After training, dimensionality compresses down to a value by $t = 10$ that roughly matches the number of class labels (10). This compression is similar to that seen in Fig. 2 of the main text. (b) Projection onto the top three principal components of MNIST test data. Colors indicate true class label (i.e. digit identity). (c) Projection onto the top three principal components of the edge-of-chaos recurrent network's responses to the inputs in (b) after training, at the evaluation time $t = 10$. Colors indicate true class label as in (b). The network forms a localized cluster for each digit.

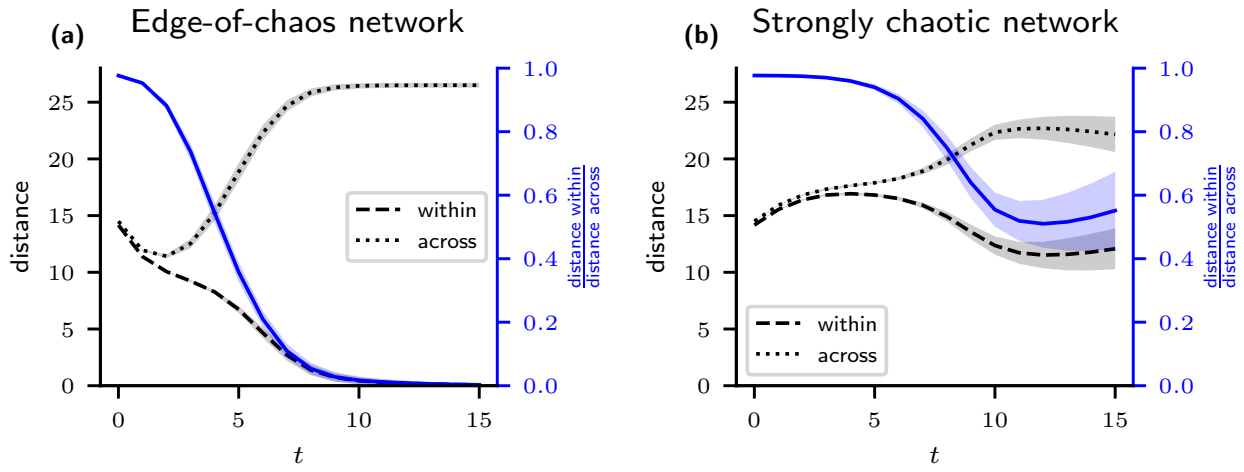

Figure 2.7: **Between class distances are increased while within class distances are diminished by the network dynamics.** Mean pairwise distance between points belonging to the same class (dashed lines), mean pairwise distance between points belonging to different classes (dotted lines), and the ratio of the first to the second (blue lines and axes), for the representations of trained networks over time t. (a) Edge-of-chaos network as defined in the main text. (b) Strongly chaotic network as defined in the main text.

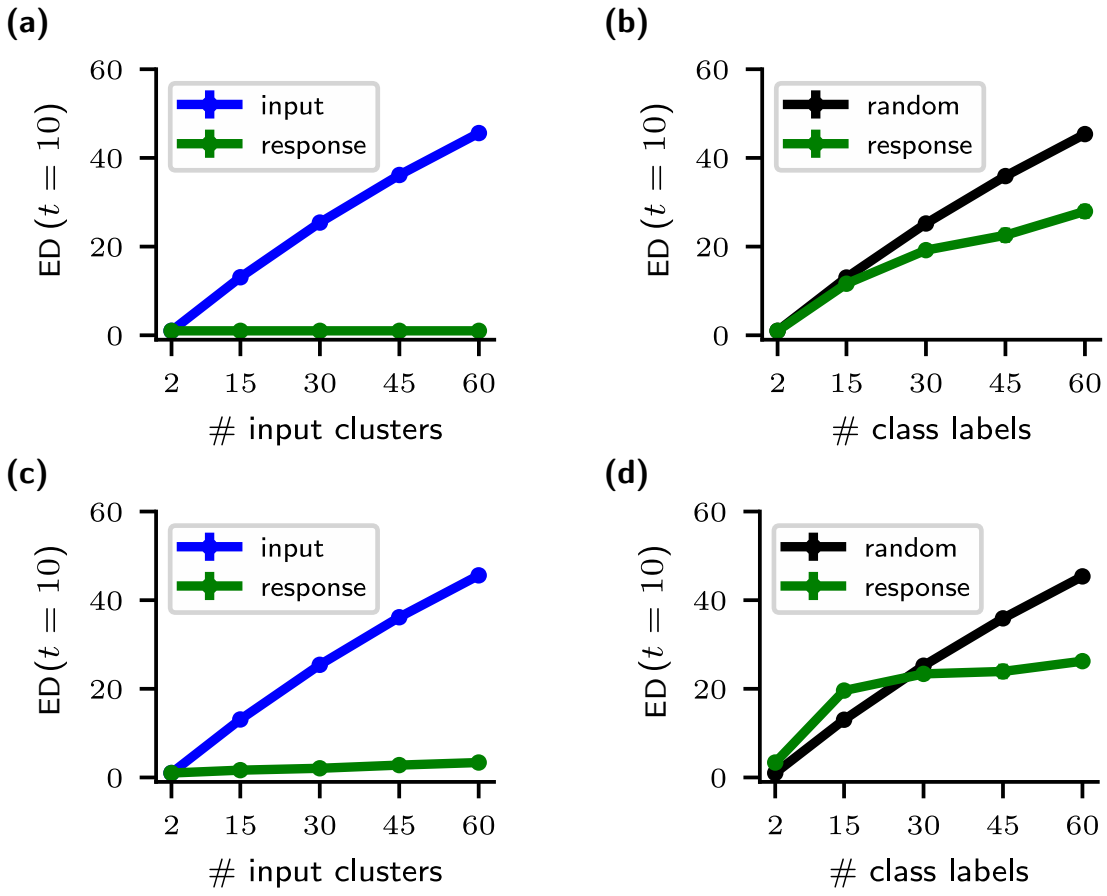

Figure 2.8: **Dimensionality increases with number of class labels, but not number of clusters** Effective dimensionalities (EDs) of the trained network responses to inputs embedded in an N-dimensional ambient space, measured at the evaluation time $t_{eval} = 10$. Error bars denote two standard deviations 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).

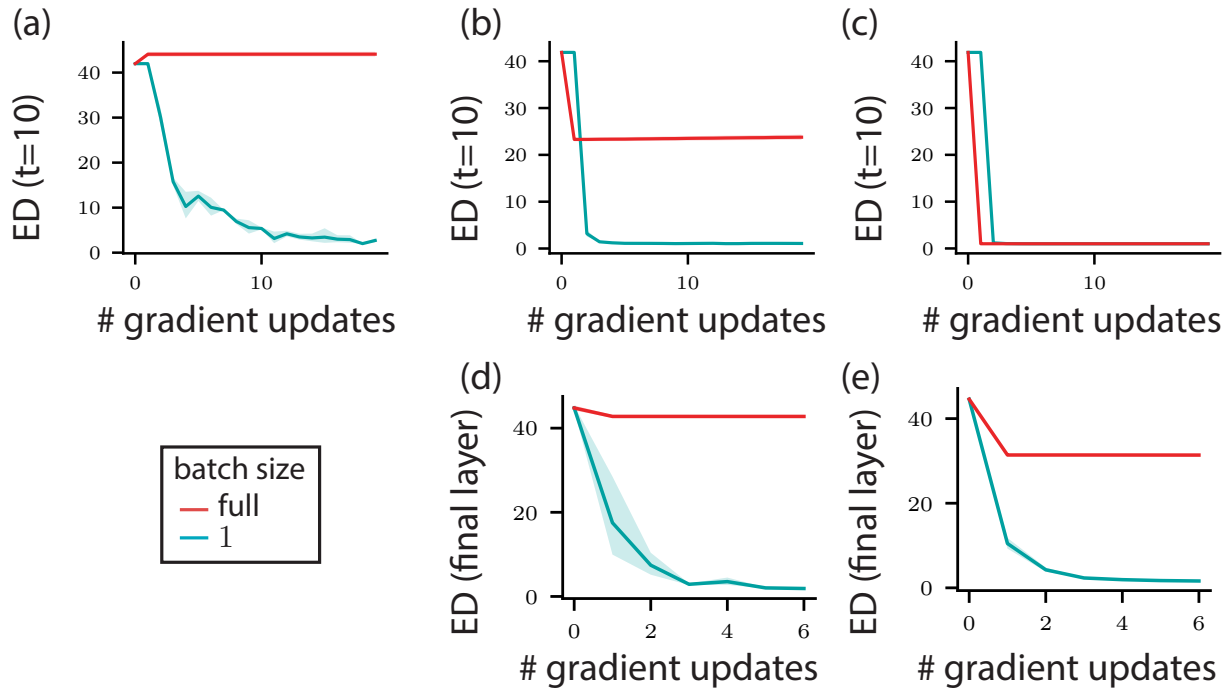

Figure 2.9: **Minibatching drives dimensionality compression** Effective Dimensionality (ED) of the hidden representation at $t = 10$ evaluated through training, comparing the effects of minibatching (blue-green) and full batches (red). In both cases, weight updates use momentum (RMSprop). Note that the # of gradient updates implies different numbers of training samples for different batch sizes: for a batch size of one, each gradient update follows feeding one input sample to the network, while for a full batch size each gradient update follows only after feeding 200 input samples to the network. In both cases the learning rate is the same. (a) Result for a linear recurrent network trained via mean squared error loss. Only the minibatch updates result in dimensionality compression. (b) Result for a recurrent network with a tanh nonlinearity, trained using a mean squared error loss. The full batch network compresses dimensionality, but much less than the minibatch network. (c) Result for a recurrent network with a tanh nonlinearity, trained using a categorical cross entropy loss. In this case both networks compress dimensionality. (d) Dimensionality of the last hidden layer of a ten-hidden-layer feedforward, densely connected network, measured through training. Here the network is linear and trained with mean squared error. (e) Same as (d), but here with ReLU nonlinearities for the hidden units.

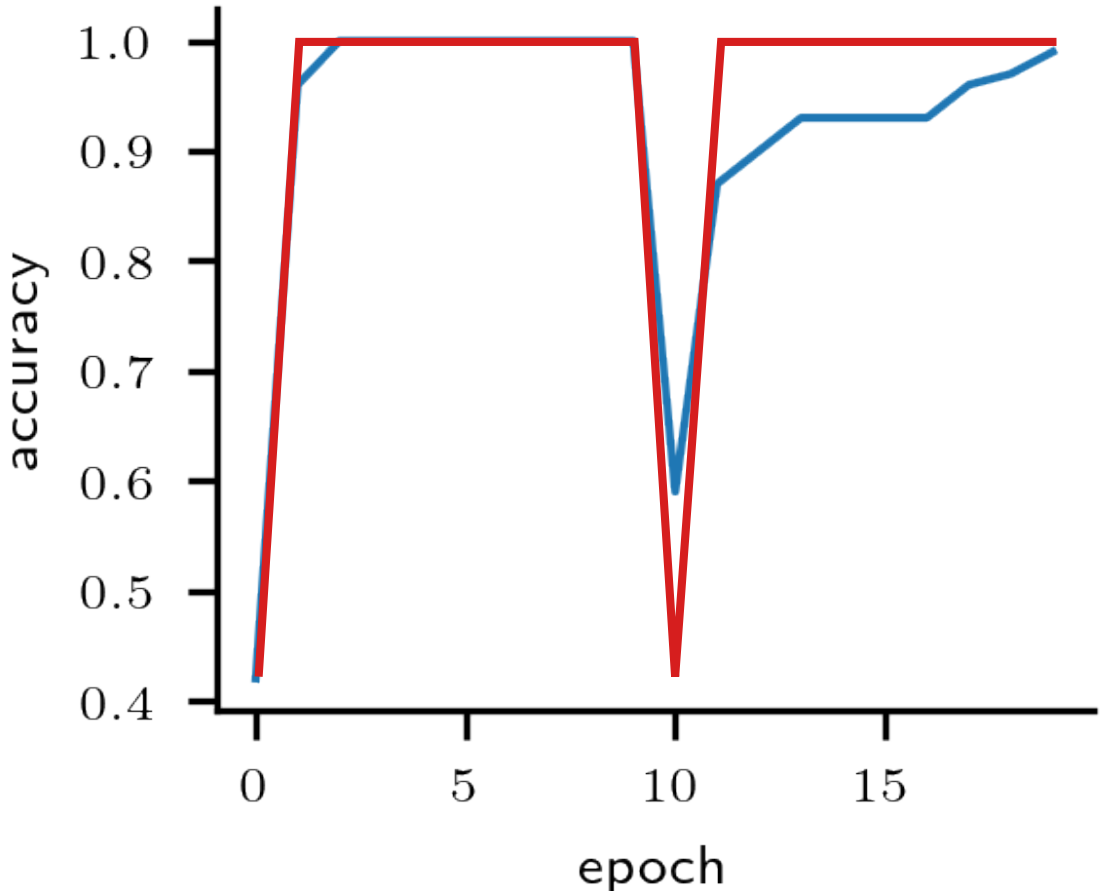

Figure 2.10: **Chaotic solutions are flexible.** A network at the edge-of-chaos (blue) and a strongly chaotic (red) network are trained to classify the high-dimensional cluster data as defined in the main text. At epoch 10, the data are regenerated, with new cluster center positions and new class label assignments (epoch 10 shows the accuracy on these new data before the network is trained on them). The strongly chaotic network learns the new data more quickly.

Chapter 3

AUTOENCODER NETWORKS EXTRACT LATENT VARIABLES AND ENCODE THESE VARIABLES IN THEIR CONNECTOMES

3.1 Introduction

Chapters 1 and 2 primarily focused on the representations of networks formed through learning. In this chapter we focus on the structure of network weights that are formed through learning.

The past years have seen spectacular effort, and spectacular success, in mapping synaptic connections in the brain. For example, the hemi-brain connectome of *Drosophila* was recently released, and imaging of the whole brain connectome is currently underway [100]. The synaptic connections in a cubic millimeter of mouse visual cortex have also recently been imaged [101]. These data build on pioneering efforts to map the connectome of *c. elegans*. From these advances there have emerged stunning new opportunities and new challenges for modelers and theoreticians. Key among these – and the topic of our paper – is to understand which features of these connectomes are informative about the computations that the underlying circuits perform.

Connectivity data has long been leveraged to shed light on circuit function. This includes the discovery of hierarchical organization in mammalian visual systems [15], which is thought to support the assembly of complex and abstract visual information in higher areas out of combinations of simple, local neural responses in lower areas; and the repeated structure across different neocortical regions [102, 103], indicating that neocortex supports general-purpose learning. As connectivity data become more complete, they can be used to more precisely constrain models [104]. For instance, while a variety of mechanisms have been proposed to explain motion processing in the retina, recent connectivity data allowed the authors of [105] to refine this class of models into one that better fits the observed connectivity. There is hope that with enough connectivity data, much information about the functioning of a neural circuit can be *inferred* from these data, even with little or no activity recording data from the same neurons [38].

Additional studies link connectivity to the function of sensory circuits through the lens of increasing or decreasing the dimension of their inputs. For example, connections between mossy fiber and the large number of cerebellar granule cells is thought to support associative learning by expanding dimension [89, 90, 106, 107, 108, 44]. More recently it has been observed that the *geometry* of the weights (as seen by viewing the weight matrix as a set of vectors embedded in a potentially high-dimensional space) holds information about the underlying computation that can be extracted by dimensionality reduction methods, both in artificial neural networks and network models for biological circuits [39].

Other studies point out physical bottlenecks in sensory circuits that strongly suggest a compression of dimension, particularly in early visual pathways (for a review, see [109]). Such a compression forces circuits to select elements of their inputs which are necessary for downstream computations. Often this operation is modeled as extracting a low-dimensional set of *latent variables* that generate the higher-dimensional input signal.

Our work follows on these observations by asking: in compressive circuits, can the actual structure of the selected latent variables be extracted from the connectome? While this question will prove to be a significant challenge to answer in general, here we start with a simple and mathematically tractable model of input compression: an autoencoder with a single layer of hidden units. This work sets out to discover if the weights that optimally compress inputs contain recoverable information about the input stimulus; namely, the latent variables generating the inputs. Here we focus on using dimensionality reduction methods on the weight matrices to recover this information, inspired by [39]. We place the observations of that study on a firmer theoretical footing by showing how they arise in a normative framework.

It is not clear *a priori* how much information about a network's computation can be extracted from weight matrices. In particular, it is frequently the case that network models produce the same input-output mappings with different values for the weights.

Our main finding is that – despite this potentially confounding ambiguity – structure from the latent variables underlying the inputs *can* be extracted from the weights of our network. Successful extraction depends on the Frobenius norm of the learned network weights being sufficiently small, a condition that can be ensured via biologically inspired costs on weight resources applied during training, or possibly by proper initialization of the network or "implicit" regularization caused by the dynamics of stochastic gradient descent. The structure that can be extracted includes the basic topology of the latent space; in particular,

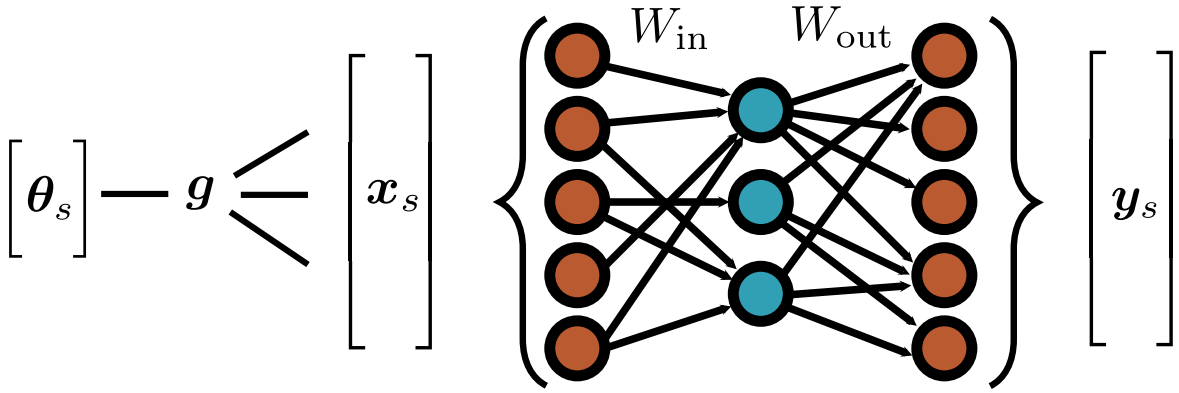

Figure 3.1: Depiction of network model.

it can be inferred if the latent variables live on a space that wraps around periodically (like a circle) or that doesn't wrap around (like a line). We give mathematical arguments in the case of linear autoencoders as well as simulation results in the case of trained autoencoders with hyperbolic tangent nonlinearities. Our results are a proof of concept that meaningful information about network function can be extracted from the optimal weights alone. They also reinforce an emerging narrative in the analysis of neural networks: that regularization is important not only for producing network models that generalize, but also for producing models that are interpretable (see [110, 111]).

Publications and preprints. The contents of this chapter are found in [112].

3.2 Network architecture and task structure

3.2.1 Network architecture and training objective

The model we consider is an autoencoder network with a single layer of hidden units, as illustrated in Fig 3.1. The equation for the hidden unit activations in response to an input $\boldsymbol{x}_s$ is

$$\boldsymbol{h}_s = \phi(\boldsymbol{W}_{\text{in}}\boldsymbol{x}_s) + \boldsymbol{b}_1 \in \mathbb{R}^N$$

where N is the number of hidden units and ϕ is the activation function for the hidden units. The length N vector $\boldsymbol{b}_1$ is a bias term. For our mathematical analysis we take ϕ equal to the identity (i.e. a linear model), but we also show the results of simulations where $\phi = \tanh$.

The activation of the output units is

$$\boldsymbol{y}_s = \boldsymbol{W}_{\text{out}}\boldsymbol{h}_s + \boldsymbol{b}_2 \in \mathbb{R}^m. \tag{3.1}$$

where m is the dimension of the input space. The network is trained via stochastic gradient descent (SGD) with momentum (RMSprop) to minimize the regularized L2 reconstruction error

$$\mathscr{L}(\boldsymbol{W}) = \sum_{s=1}^{T} \|\boldsymbol{x}_s - \boldsymbol{y}_s\|_2^2 + \lambda(\|\boldsymbol{W}_{\text{out}}\|_F^2 + \|\boldsymbol{W}_{\text{in}}\|_F^2) \tag{3.2}$$

where T is the size of the training dataset. Here the network is trained so that outputs $\boldsymbol{y}_s$ are close to $\boldsymbol{x}_s$, regularized by a cost on weights.

3.2.2 Model non-identifiability

This model is generally *non-identifiable*: it is impossible to infer the value of the parameters by observing the outputs of the model to inputs. For instance, consider the case where ϕ is equal to the identity and the bias terms are fixed to zero. In this case

$$\boldsymbol{y} = \boldsymbol{W}_{\text{out}}\boldsymbol{A}\boldsymbol{A}^{-1}\boldsymbol{W}_{\text{in}}\boldsymbol{x} = \boldsymbol{W}_{\text{out}}\boldsymbol{W}_{\text{in}}\boldsymbol{x}.$$

for any $N \times N$ invertible matrix $\boldsymbol{A}$, so that the same input-output mapping is satisfied by $\boldsymbol{W}_{\text{out}}\boldsymbol{A}$ and $\boldsymbol{A}^{-1}\boldsymbol{W}_{\text{in}}$ for any invertible $\boldsymbol{A}$. The model becomes identifiable up to geometry-preserving transformations of the parameters when the regularizer term is introduced in Eq. (3.2), as shown in our analysis below.

We describe the training dataset next.

3.2.3 Constructing inputs generated by latent variables

Suppose that our inputs have the form $\boldsymbol{x}_s = \boldsymbol{g}(\boldsymbol{\theta}_s) \in \mathbb{R}^m$, where $\boldsymbol{\theta}_s \in \mathcal{S}$ is some low-dimensional underlying process in a space $\mathcal{S}$ and $\boldsymbol{g}$ maps this process into a higher-dimensional space embedded in $\mathbb{R}^m$. Here $\boldsymbol{\theta}_s$ is a latent variable that underlies the inputs $\boldsymbol{x}_s$, and the induced process $\boldsymbol{x}_s$ can be thought of as a high-dimensional encoding of $\boldsymbol{\theta}_s$. The subscript s is the index for samples of the corresponding variables, and is sometimes suppressed when context makes it clear. Throughout, mathematical symbols in bold font denote vectors or vector-valued functions, while scalars are denoted by lowercase symbols with normal font.

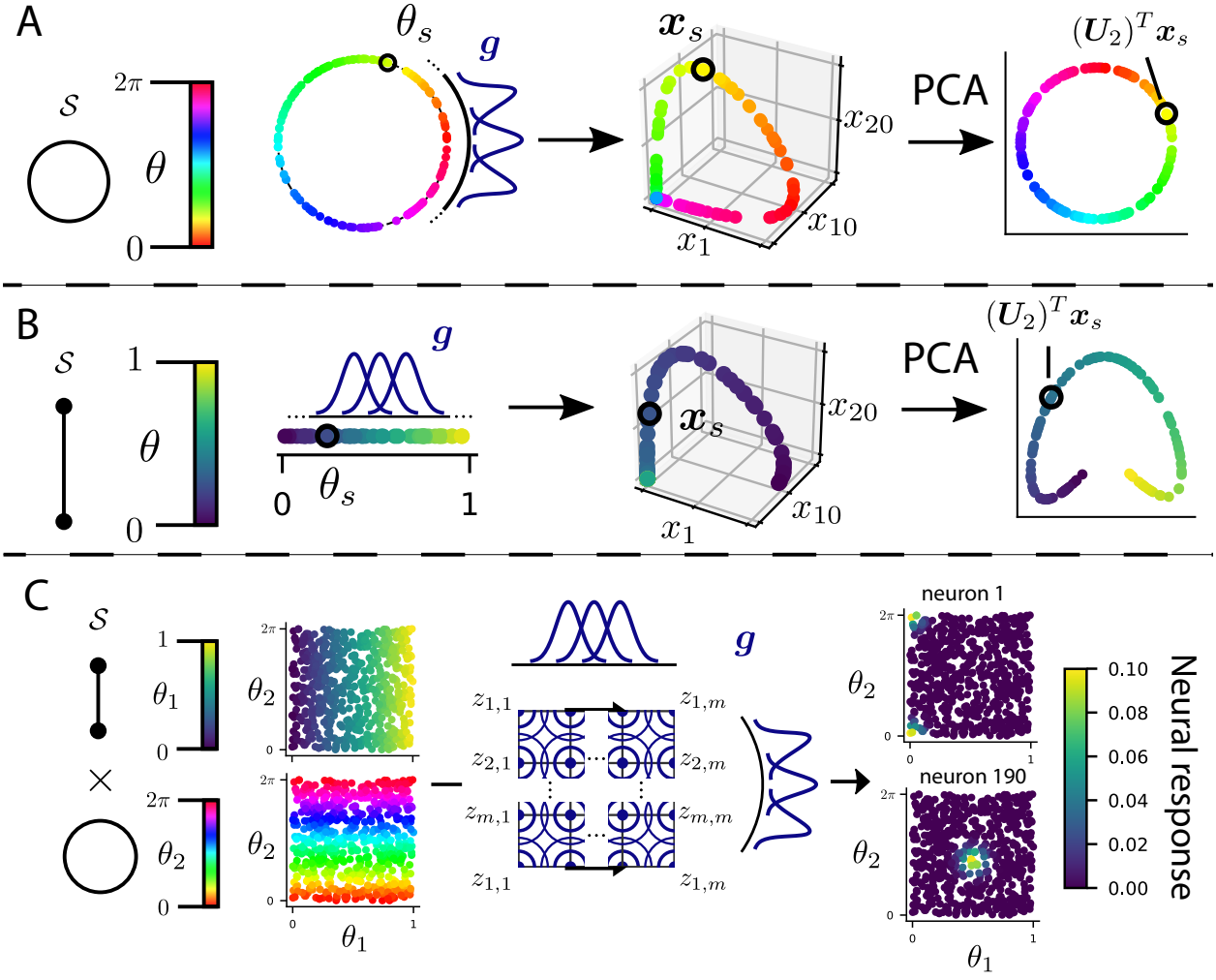

Figure 3.2: Depiction of latent variables on different spaces $\mathcal{S}$. (a) Example where $\mathcal{S}$ is a circle. A periodic scalar latent variable is transformed into a higher-dimensional encoding via the receptive field neural response function $\boldsymbol{g}$. The periodicity of θ is expressed by a periodic colormap for θ. The periodic structure is revealed by PCA. (b) Same as (a), but for a nonperiodic scalar latent variable, so that $\mathcal{S}$ is a line segment. (c) Example where $\mathcal{S}$ is a cylinder. Tensor product of a nonperiodic (θ_1) and a periodic (θ_2) latent variable is transformed into a higher-dimensional encoding via the receptive field neural response function $\boldsymbol{g}$. The scatter plots to the left depict the samples $\boldsymbol{\theta}_s$ with coloration based on θ_1 (top) and coloration based on θ_2 (bottom). Receptive field centers $\boldsymbol{z}_{k,k'}$ tile the latent space $\mathcal{S}$. Specifically, the top and bottom edges of the space are glued together. The responses of the first and 190th out of 400 receptive field neurons are shown on the right.

When indexing the entries of these vectors with an index k, we either use the notation $(x_k)_s$ or suppress the s, simply writing x_k. In our mathematical analysis and simulations we assume that the samples $\boldsymbol{\theta}_s$ are drawn uniformly from $\mathcal{S}$.

The high-dimensional nature of the inputs is important in our framework. While networks that take lower-dimensional inputs and project them into a higher-dimensional hidden representation space are also of general interest, our objective here is to require the network to extract latent variables from a high-dimensional signal. This more constrained scenario affords a greater possibility that the weights will contain information about the latent variables. In particular, when treating weight matrices as geometric objects , the number of input dimensions m is the number of datapoints that we plot, and these points are embedded in an N-dimensional space. From this perspective, m will need to be large enough for meaningful structure to emerge.

One important class of encodings that increase the dimensionality of the encoded variables are the tuned neural response functions. In this case, each component x_k of $\boldsymbol{x}$ is the response of a neuron with tuning curve g_k to the variable $\boldsymbol{\theta}$. As an example, consider the classic case of direction-selective retinal ganglion cells. We suppose each ganglion cell to be tuned as a Gaussian centered at its preferred orientation, $g_k(\theta) \propto \exp(-d(\theta, z_k)^2/\sigma^2)$, where θ is an angle in the interval from 0 to 2π, z_k is the neuron's preferred orientation, and σ captures the width of the tuning. Here d is a distance function in angle space, which can be written $d(\theta, \theta') = \min\{|\theta - \theta'|, 2\pi - |\theta - \theta'|\}$. A visualization of θ and the response $\boldsymbol{g}(\theta)$ is shown in Fig 3.2a.

Fig 3.2b depicts a latent variable θ that is also scalar-valued, but differs from the previous example in that the latent space $\mathcal{S}$ is the closed interval $[0, 1]$ with the standard topology of an interval, as opposed to a circle. In this case the natural encoding is $g_k(\theta) \propto \exp(-d(\theta, z_k)^2/\sigma^2)$ where $d(\theta, z_k) = |\theta - z_k|$.

Another important case is that of "place cell" neurons tuned to (x, y) position on a grid over a two-dimensional flat surface $\mathcal{S}$. The space $\mathcal{S}$ can be thought of as the product space of the product of two line segments. In this case we suppose each place cell to be tuned as a gaussian centered at its location on the grid, $g_{k,\ell}(\boldsymbol{\theta}) \propto \exp(-d(\boldsymbol{\theta}, \boldsymbol{z}_{k,\ell})^2/\sigma^2)$ where $\boldsymbol{z}_{k,\ell}$ is the neuron's tuning center and $d(\boldsymbol{\theta}, \boldsymbol{z}_{k,\ell}) = \|\boldsymbol{\theta} - \boldsymbol{z}_{k,\ell}\|_2$. Note that in this case the indices (k, ℓ) need to be "unrolled" into a vector to form the vector-valued $\boldsymbol{g}(\boldsymbol{\theta}_s)$. The idea can be further extended to other latent variables, such as the joint spatial and orientation tuning seen also in retinal ganglion cells. To illustrate such a joint encoding, consider "place cell" neurons

as before, but instead of tiling a grid, suppose that one edge of the grid wraps around and connects to the opposite edge, so that the neurons tile a cylinder. A realization of a latent variable on a cylinder and the responses of two receptive field neurons is depicted in Fig 3.2c.

3.3 Results

Each of the examples in the preceding section illustrates a different topology of the space $\mathcal{S}$ on which the latent variable $\boldsymbol{\theta}_s$ can live. Our main finding is that, in our network trained to autoencode the inputs $\boldsymbol{x}_s$ generated by $\boldsymbol{\theta}_s$, the weights of the network generally reflect the geometry of $\mathcal{S}$. This is in spite of the problem of non-identifiability as introduced in Section 3.2. In this section we first illustrate this phenomenon in simulations, and then provide mathematical reasoning to shed light on why it occurs. In particular, our mathematical analysis will provide heuristic reasons for why non-identifiability is not overly deleterious to structure in the weights.

3.3.1 The latent variables appear in the weights of the trained autoencoder

An overview of this phenomenon in the case of a periodic latent variable is given in Fig 3.3, where the different parts of the network model (inputs, hidden unit responses, and weights) are "dissected". This illustrates that the periodicity of the inputs translates into a periodicity of both the neural response and of the weight matrices. This information is reiterated in Figs 3.4a to 3.4c. In Fig 3.4a, a trained autoencoder's response to the inputs $\boldsymbol{x}_s$ generated by a periodic latent variable θ_s is plotted in the top two-dimensional principal component space. In Fig 3.4b, the top two principal components of the columns of the output weights are plotted. The resulting structure suggests periodicity, but isn't always clearly seen. Using the nonlinear dimensionality reduction method Isomap [113] to reduce the output weights to a two-dimensional space reveals the circular structure of the latent variable space clearly (Fig 3.4c). A description of Isomap and explanation of its success in recovering the periodic structure is given in Section 3.3.2. This shows that the structure of the latent variable of the trained autoencoder is apparent not only in the hidden unit responses to the periodic inputs, but also in the learned weights of the network.

A similar phenomenon occurs for an autoencoder trained to reconstruct inputs $\boldsymbol{x}_s$ formed by receptive field responses $\boldsymbol{g}(\theta_s)$ to a non-periodic latent variable θ_s. Here we see that the topology of the latent space $\mathcal{S}$ is again reflected in the network weights (Figs 3.4e and 3.4f).

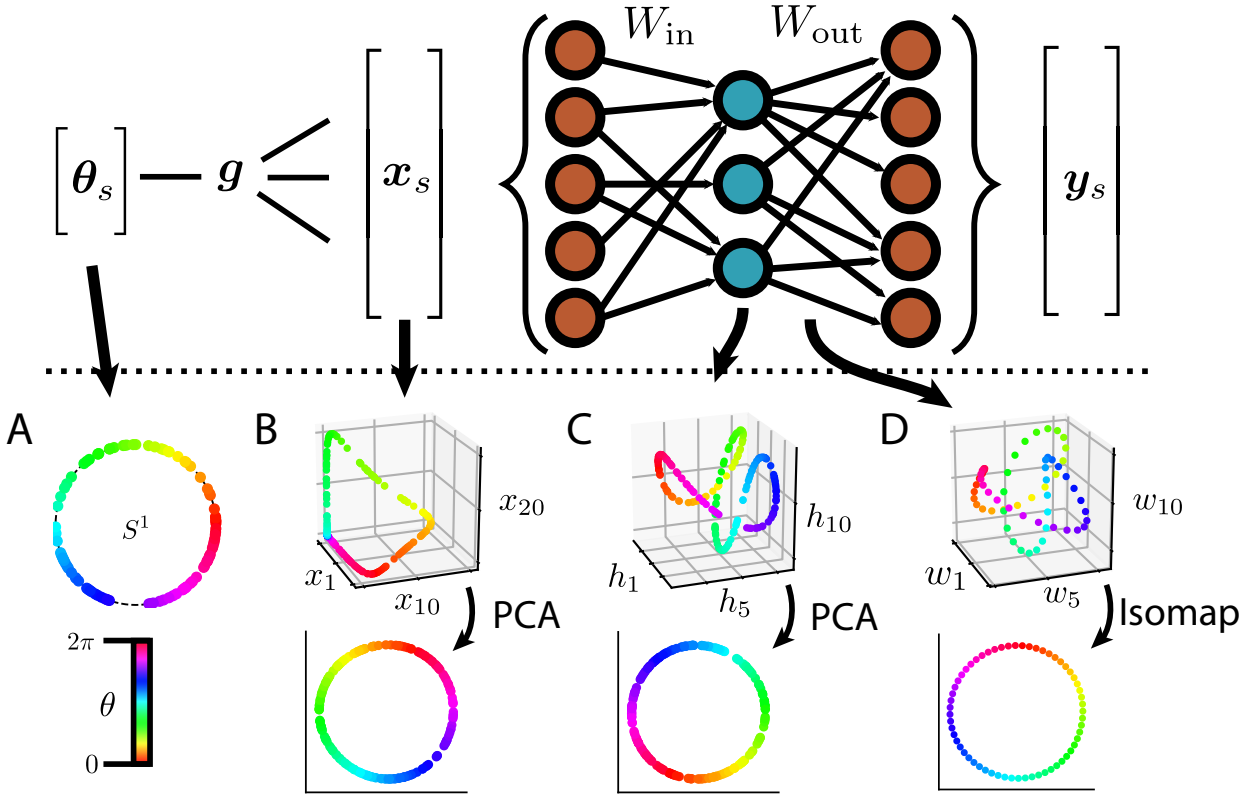

Figure 3.3: Overview of the characteristics of an autoencoder network trained to reproduce inputs that are generated by a periodic latent variable. Top: Network architecture. A low-dimensional set of latent variables is transformed into a high-dimensional input via a function $\boldsymbol{g}$. The network is then trained to reconstruct this input under the constraint of a bottleneck in the number of hidden unit neurons. A hyperbolic tangent nonlinearity is used for the hidden units. Bottom: Example network measurements for a periodic, scalar latent variable θ_s. Network is trained with $\lambda = 4e^{-6}$ and $m = 60$. (a-c) Color denotes value of θ_s. (a) Latent variable θ_s drawn from S^1. Color denotes value of θ_s. (b) Responses of receptive field neurons 1, 10, and 20 to θ_s. The receptive fields are periodic. Bottom: The response ensemble projected down to a two-dimensional space with PCA. (c) Responses of hidden units 1, 5, and 10 to θ_s, out of 10 hidden units. Bottom: The hidden unit response projected down to a two-dimensional space with PCA. (d) Columns 1, 5, and 10 of the output weight matrix, out of 10 columns. Color corresponds to the receptive field neuron index. While the structure is similar to (c), the relative scaling of the axes is different, as evidenced by the grid lines. Bottom: The output weight matrix projected onto a two-dimensional space with Isomap.

This phenomenon also occurs in the case of inputs generated by tensored latent variables as in Fig 3.2c, resulting in the weights reflecting the cylindrical topology of $\mathcal{S}$ (Figs 3.4g to 3.4i). When both tensored variables are periodic, the structure in the weights is that of a torus (Fig 3.4l)

While in the examples above the latent variables are reflected both in the structure of the weights and the structure of the hidden layer activations of the trained network, structure in the activations depends on the choice of inputs given to the network after training. In the example of a periodic random variable, the ring structure in the activations does not appear if white noise inputs are shown to the trained network (data not shown). This is intuitively because a linear transformation of white noise cannot have a ring-like structure. This illustrates how the information provided by the weights is in some ways distinct from that provided by the hidden unit activations.

Note also that the structure of the weights is most clearly extracted by the nonlinear dimensionality reduction method Isomap (Figs 3.4c, 3.4f and 3.4i) as opposed to the linear method of principal component analysis (Figs 3.4b, 3.4e and 3.4h). We shed light on why this is in Section 3.3.2.

3.3.2 Extracting latent variables from network weights

We now explain these observations through mathematical analysis. For ease of analysis, we consider a linear autoencoder model where ϕ is taken to be the identity. Our analysis involves three steps. The first is to relate the minimizers of Eq. (3.2) to the familiar solutions found by principal component analysis (PCA). The second is to resolve the problem of *non-identifiability* of the model, as introduced in Section 3.2. Once the minimizers of Eq. (3.2) have been related to the PCA solutions and the degeneracy of the solution space has been resolved, the third step in our analysis is to look more closely at the PCA solutions and to show that these solutions in fact encode the latent variable information.

Relating autoencoders to PCA

We start by rewriting the loss Eq. (3.2) in matrix form with $\lambda = 0$ and with ϕ taken to be the identity:

$$L(\boldsymbol{W}_{\text{out}}, \boldsymbol{W}_{\text{in}}, \boldsymbol{b}_1, \boldsymbol{b}_2) = \left\| \boldsymbol{X} - \boldsymbol{W}_{\text{out}}\boldsymbol{W}_{\text{in}}\boldsymbol{X} + \boldsymbol{W}_{\text{out}}\boldsymbol{b}_1\mathbf{1}_T^\top + \boldsymbol{b}_2\mathbf{1}_T^\top \right\|_F^2 \tag{3.3}$$

where $\boldsymbol{X}$ is an $m \times T$ matrix with column s holding the sample $\boldsymbol{x}_s$ and $\mathbf{1}_T$ is the length T vector of all ones. Let $\boldsymbol{\mu_x} = \langle \boldsymbol{x}_s \rangle_s$ and $\boldsymbol{\mu_h} = \langle \boldsymbol{h}_s \rangle_s$. The optimal values for the bias terms have the effect of transforming the problem into one that has been mean-centered, i.e. the minimal weights of Eq. (3.3) coincide with those of

$$\|\boldsymbol{X'} - \boldsymbol{W}_{\text{out}}\boldsymbol{H'}\|_F^2 \tag{3.4}$$

where $\boldsymbol{X'} = \boldsymbol{X} - \boldsymbol{\mu}_x\mathbf{1}^\top$ and $\boldsymbol{H'} = \boldsymbol{W}_{\text{in}}\boldsymbol{X} - \boldsymbol{\mu_h}\mathbf{1}^\top$ [114]. Minimizing this loss while enforcing that $\boldsymbol{W}_{\text{in}}$ and $\boldsymbol{W}_{\text{out}}$ have orthonormal rows and columns, respectively, results in the PCA solution. This solution is naturally expressed in terms of the *singular value decomposition* (SVD) of $\boldsymbol{X'}$: $\boldsymbol{X'} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$ where $\boldsymbol{U}$ is an $m \times m$ orthogonal matrix, $\boldsymbol{V}$ is a $T \times m$ matrix with orthonormal columns, and $\boldsymbol{\Sigma}$ is an $m \times m$ diagonal matrix with nonnegative entries $\sigma_1, \sigma_2, ..., \sigma_m$ called the *singular values* of $\boldsymbol{X'}$. The standard PCA solutions are then $\boldsymbol{W}^*_{\text{out}} = \boldsymbol{U}_N$ and $\boldsymbol{W}^*_{\text{in}} = (\boldsymbol{U}_N)^T$ where $\boldsymbol{U}_N$ is the matrix $\boldsymbol{U}$ truncated to the first N columns. However, as discussed above any solution of the form $\boldsymbol{W}^*_{\text{out}} = \boldsymbol{U}_N\boldsymbol{A}$ and $\boldsymbol{W}^*_{\text{in}} = \boldsymbol{A}^{-1}(\boldsymbol{U}_N)^T$ is also a global minimum, where $\boldsymbol{A}$ is an arbitrary invertible $N \times N$ matrix. This is the most general form of optimal solution [115, 116].

Resolving non-uniqueness of the optimal weights

The arbitrary invertible linear transformation $\boldsymbol{A}$ introduced in the previous section can potentially skew the structure of the weights past the point where the latent variables can be extracted. While $\boldsymbol{A}$ preserves topological information, it doesn't necessarily preserve local distances between points. Nonlinear dimensionality reduction methods like Isomap are generally designed to embed points in a lower-dimensional space while preserving local distances, and losing information about these local distances can be destructive. This can become a problem in practice since the number of columns (rows) of $\boldsymbol{W}_{\text{in}}$ ($\boldsymbol{W}_{\text{out}}$) is finite, and the structure of local distances among finitely many points can be lost as skewing becomes large. Here we describe biologically motivated conditions that address this issue.

If we add the regularizer $\lambda(\|\boldsymbol{W}_{\text{out}}\|_F^2 + \|\boldsymbol{W}_{\text{in}}\|_F^2)$ with $\lambda > 0$ to Eq. (3.3), then the solution becomes more constrained. Let $\boldsymbol{\Sigma}_\lambda$ be the diagonal matrix $\boldsymbol{\Sigma}_\lambda = \text{diag}(\lfloor 1 - \lambda/\sigma_1^2 \rfloor_+, ..., \lfloor 1 - \lambda/\sigma_N^2 \rfloor_+)$, where the σ_k are again the singular values of $\boldsymbol{X}$ and $\lfloor \cdot \rfloor_+$ is the threshold function $\max\{\cdot, 0\}$. [115, 116] recently showed that, under the assumption that $\sigma_1 > \sigma_2 > ... > \sigma_N$, the optimal weights in this case have the form $\boldsymbol{W}^*_{\text{in}} = \boldsymbol{Q}^T\boldsymbol{\Sigma}_\lambda^{1/2}(\boldsymbol{U}_N)^T$ and $\boldsymbol{W}^*_{\text{out}} = \boldsymbol{U}_N\boldsymbol{\Sigma}_\lambda^{1/2}\boldsymbol{Q}$

where $\boldsymbol{Q}$ is an arbitrary $N \times N$ orthogonal matrix. In particular, these solutions are unique up to arbitrary orthogonal transformations $\boldsymbol{Q}$, rather than arbitrary invertible transformations $\boldsymbol{A}$. The thresholding of the terms $1 - \lambda/\sigma_k^2$ along the diagonal of $\boldsymbol{\Sigma}_\lambda$ can create an effective bottleneck. Let k^* be the smallest index such that $1 - \lambda/\sigma_k^2 \leq 0$. If $k^* < N$, then this has the effect of setting the last $N - k^*$ columns of $\boldsymbol{U}_N$ to zero in the expression for $\boldsymbol{W}_{\text{in}}^*$ and $\boldsymbol{W}_{\text{out}}^*$, which is equivalent to having a network with bottleneck size k^*. If the terms $1 - \lambda/\sigma_k^2$ are all positive, then the thresholding has no effect. Since our results do not depend closely on the size of the bottleneck (the main requirement is that the bottleneck be large enough to capture the dimensionality of the latent space), we have some liberty in our choice of λ. In our simulations, unless otherwise noted we take $N = 10$ and $\lambda = 4e - 6$, which is small enough so that the $1 - \lambda/\sigma_k^2$ terms are positive for $1 \leq k \leq N$.

It follows that this regularizer can help to preserve the geometric information encoded in the weights found by optimization methods. In particular, for positive but small λ, $\boldsymbol{W}_{\text{in}}^* \approx \boldsymbol{Q}^T(\boldsymbol{U}_N)^T$ and $\boldsymbol{W}_{\text{out}}^* \approx \boldsymbol{U}_N\boldsymbol{Q}$. Since $\boldsymbol{Q}$ preserves distances, analyzing the geometric structure of optimal weights $\boldsymbol{W}_{\text{in}}^*$ and $\boldsymbol{W}_{\text{out}}^*$ reduces to analyzing the geometric structure of $\boldsymbol{U}_N$. Regularization can make the difference: see the insets of Fig 3.6a for an example of failure to extract the periodic latent variable when regularization isn't used.

As we show in Section 3.3.2 below, the matrix $\boldsymbol{U}_N$ contains geometric information about the inputs.

Relating the weights to the latent variables

In the previous section we showed how the optimal weights share the same geometric structure as $\boldsymbol{U}_N$: $\boldsymbol{W}_{\text{in}}^* \approx \boldsymbol{Q}^T(\boldsymbol{U}_N)^T$ and $\boldsymbol{W}_{\text{out}}^* \approx \boldsymbol{U}_N\boldsymbol{Q}$ for λ small. We now show how $\boldsymbol{U}_N$ is related to the latent variables underlying the inputs. To do so, we first note that according to basic properties of the SVD, $\boldsymbol{U}$ is a matrix of normalized eigenvectors of the covariance matrix $\boldsymbol{C} = \left\langle \boldsymbol{x}_s \boldsymbol{x}_s^T \right\rangle_s - \left\langle \boldsymbol{x}_s \right\rangle_s \left\langle \boldsymbol{x}_s \right\rangle_s^T$ of $\boldsymbol{X}$, so $\boldsymbol{C} = \boldsymbol{U}\boldsymbol{\Sigma}^2\boldsymbol{U}^T$ (recall that $\boldsymbol{\Sigma}$ holds the singular values for the mean-centered $\boldsymbol{X}'$). Assuming that $\boldsymbol{x}$ has the form $x_j = g_j(\boldsymbol{\theta}) \propto \exp(-d(\boldsymbol{\theta}, \boldsymbol{z}_j)^2/\sigma^2)$, we can work out the form of the covariance between x_j and x_k. Taking the limit $T \to \infty$ and invoking the law of large numbers, we have that $\langle (x_j)_s \rangle_{s=1,\ldots,\infty} = \langle g_j(\boldsymbol{\theta}) \rangle_{\boldsymbol{\theta}}$ and $\langle (x_j)_s (x_k)_s \rangle_{s=1,\ldots,\infty} = \langle g_j(\boldsymbol{\theta}) g_k(\boldsymbol{\theta}) \rangle_{\boldsymbol{\theta}}$. Letting $\mu_j = \langle g_j(\boldsymbol{\theta}) \rangle_{\boldsymbol{\theta}}$, it follows that in this limit

$$C_{jk} = \langle g_j(\boldsymbol{\theta}) g_k(\boldsymbol{\theta}) \rangle_{\boldsymbol{\theta}} - \mu_j \mu_k.$$

From this form of the covariance matrix, we can now work out the eigenvector structure for different choices of the latent space $\mathcal{S}$ and distance function d.

Periodic latent variables give rise to periodic weight structure

We first consider the case of a periodic latent variable θ. Recall that the inputs are formed by encoding θ via orientation selective receptive fields $g_k(\theta) \propto (\exp(-d(\theta, z_k)^2/\sigma^2)$, with d being a distance function in angle space, $d(\theta, \theta') = \min\{|\theta - \theta'|, 2\pi - |\theta - \theta'|\}$. Assume that the receptive field centers z_k evenly tile the space.

The salient structure of this encoding can be expressed through the idea of *equivariance*. Let $\overline{a+b}$ denote addition of a and b modulo the number, m, of receptive field neurons. In our scenario, equivariance means that shifting the identity of the receptive field neuron is the same as shifting the input to the receptive field neuron: $g_{\overline{k+\ell}}(\theta) = g_k(\theta - z_\ell \mod 1)$. This equivariance implies a special structure of the input covariance matrix $\boldsymbol{C}$: $\boldsymbol{C}$ is a *circulant* matrix. This means that every row in $\boldsymbol{C}$ is a shifted version of the first row, where the shifting operation wraps around at the edges of the matrix. To show this, we show that entry C_{jk} is equal to $C_{\overline{j+\ell},\overline{k+\ell}}$, where ℓ is any integer.

Recall our assumption that θ is uniformly distributed on the circle $\mathcal{S} = S^1$, and suppose without loss of generality that $\mathcal{S}$ has Lebesque measure 2π (so θ varies from 0 to 2π). Then the probability density function of θ is the constant function that returns $1/(2\pi)$ for all θ. This means in particular that the expected value of $f(\theta)$ is $\langle f(\theta) \rangle = \frac{1}{2\pi} \int_0^{2\pi} f(\theta) d\theta$ for any reasonably well-behaved function f.

To show that $\boldsymbol{C}$ is circulant, we first compute that

$$\begin{aligned}
\left\langle g_{\overline{j+\ell}}(\theta) g_{\overline{k+\ell}}(\theta) \right\rangle &= \langle g_j \left(\theta - z_\ell \mod 1\right) g_k \left(\theta - z_\ell \mod 1\right) \rangle \\
&= \frac{1}{2\pi} \int_0^{2\pi} g_j \left(\theta - z_\ell \mod 2\pi\right) g_k \left(\theta - z_\ell \mod 2\pi\right) d\theta \\
&= \frac{1}{2\pi} \int_{0-z_\ell \mod 2\pi}^{2\pi - z_\ell \mod 2\pi} g_j \left(\theta\right) g_k \left(\theta\right) d\theta \\
&= \frac{1}{2\pi} \int_0^{2\pi} g_j \left(\theta\right) g_k \left(\theta\right) d\theta \\
&= \langle g_j(\theta) g_k(\theta) \rangle \,. \qquad (3.5)
\end{aligned}$$

Computing shifts of the mean μ_j has a similar flavor:

$$\begin{aligned}
\mu_{\overline{j+\ell}} &= \frac{1}{2\pi}\int_0^{2\pi} g_j\left(\theta - z_\ell \mod 2\pi\right) d\theta \\
&= \frac{1}{2\pi}\int_{0-z_\ell \mod 2\pi}^{2\pi - z_\ell \mod 2\pi} g_j\left(\theta\right) d\theta \\
&= \frac{1}{2\pi}\int_0^{2\pi} g_j\left(\theta\right) \\
&= \mu_j,
\end{aligned}$$

so that μ_j is independent of its index. Hence we can write $\mu_j\mu_k = \mu^2$. Combining this with Eq. (3.5), we have that $C_{\overline{j+\ell},\overline{k+\ell}} = C_{jk}$. An example of the resulting circulant matrix is shown in Fig 3.5a.

In addition to being circulant, the covariance matrix $\boldsymbol{C}$ is by definition symmetric: $\boldsymbol{C} = \boldsymbol{C}^T$. The eigenvectors of circulant matrices are known and together make up the discrete Fourier transform matrix [117]. In the case where the circulant matrix is also symmetric, the real and imaginary parts of the eigenvectors are themselves eigenvectors. This means that an eigenvector basis for $\boldsymbol{C}$ can be taken to be real, which results in eigenvectors that have one of three forms: cosine transforms, sine transforms, and the all-ones vector $\mathbf{1}_m$ (recall that m is the dimension of the inputs $\boldsymbol{x}_s$). More precisely, the jth cosine transform eigenvector has the form $v_k^{(j)} = \cos(2\pi jk/m)$ for $k \in \{0, 1, ..., m-1\}$ and the jth sine transform eigenvector has the form $w_k^{(j)} = \sin(2\pi jk/m)$. In particular, the eigenvectors are periodic, reflecting the periodicity of the latent variable θ. These eigenvectors together form the columns of the matrix $\boldsymbol{U}$. As an illustration, the eigenvectors $\boldsymbol{v}^{(1)}$ and $\boldsymbol{w}^{(1)}$ are plotted against each other in Fig 3.5b.

Consider the truncation $\boldsymbol{U}_N$ of $\boldsymbol{U}$ to N columns. We're interested in the properties of $\boldsymbol{U}_N$ embedded as a geometric object, with each row constituting a single data point in N-dimensional space. The sine and cosine structure of the eigenvectors ensures that this structure is periodic. In particular, the rows of $\boldsymbol{U}_N$ are m samples from a loop that nonlinearly curves through N-dimensional space.

Since this loop structure of $\boldsymbol{U}_N$ is nonlinearly embedded, nonlinear dimensionality reduction methods are well suited for recovering this structure. Indeed, since the singular values of $\boldsymbol{U}_N$ are all 1, trying to extract structure from it with the linear method of PCA will only return a random set of the columns of $\boldsymbol{U}_N$. In general, the ability of nonlinear dimensionality

reduction methods to successfully extract the structure of interest from a dataset depends on having enough datapoints, and our situation is no exception. In our case, the number of datapoints is m, and this will need to be a large enough number for the dimensionality reduction method to succeed. The precise number of datapoints needed will depend on the specifics of the dimensionality reduction method used. To proceed, we will assume that m is sufficiently large.

For intuition as to why nonlinear methods work, we focus on the approach of the nonlinear method Isomap. The first step of Isomap involves building a graph on the datapoints where points that are sufficiently close are connected by an edge. Let's consider the strategy of connecting every point to its two nearest neighbors. Then in our case this graph will indeed be a loop through high-dimensional space, and embedding this graph in two dimensions in a way that best preserves distance information reveals a ring.

Recall the minimizer $\boldsymbol{W}_{\text{out}}^* \approx \boldsymbol{U}_N\boldsymbol{Q}$, $\boldsymbol{W}_{\text{in}}^* \approx \boldsymbol{Q}^T(\boldsymbol{U}_N)^T$ of the regularized loss for the linear model. In practice we find that the periodicity of $\boldsymbol{U}_N$ is reflected in the weights $\boldsymbol{W}_{\text{out}}$ in the autoencoder trained with stochastic gradient descent. As illustrated by Fig 3.4, this extends to networks with tanh nonlinearity. Here the network is trained with $\lambda = 4e^{-6}$ and $m = 100$. The latent variable structure can be partially seen in the apparent periodicity of points obtained by using PCA to project the columns of $\boldsymbol{W}_{\text{out}}$ onto a three-dimensional space in Fig 3.4b. As discussed above, this periodicity is revealed more clearly by using Isomap to "unravel" the coils caused by the higher frequency modes, as can be seen in Fig 3.4c.

Nonperiodic latent variables give rise to nonperiodic weight structure

The above analysis can be repeated in a similar form for the case of a nonperiodic latent variable θ on a line segment, where this time $g_k(\theta) \propto \exp(-|\theta - z_k|^2/\sigma^2)$. Suppose the receptive field centers z_1 through z_m evenly tile the line segment $[0, 1]$, with $z_1 = 0$ and $z_m = 1$. While we are interested in the case where θ is uniformly distributed on $[0, 1]$, this becomes mathematically challenging to work with due to conditions at the boundary being different than conditions in the center of the interval. Instead, we let $\mathcal{S}$ be the interval $[-s, s+1]$, with the usual interval topology. Taking s sufficiently large will allow us to deal with boundary effects; for instance, this assumption ensures that $\langle g_k(\theta)\rangle$ is approximately independent of k. In this case the covariance matrix, instead of being circulant, is approximately *Toeplitz*, which means that the entries on each descending diagonal from left to right are the same. This can be seen by choosing indices j, k and ℓ constrained such that $j, k \in \{1, ..., m\}$,

$j+\ell \in \{1,...,m\}$, $k+\ell \in \{1,...,m\}$ and computing

$$\begin{aligned}
\langle g_{j+\ell}(\theta) g_{k+\ell}(\theta)\rangle &= \langle g_j\left(\theta - z_\ell\right) g_k\left(\theta - z_\ell\right)\rangle \\
&= \frac{1}{2s+1}\int_{-s}^{s+1} g_j\left(\theta - z_\ell\right) g_k\left(\theta - z_\ell\right) d\theta \\
&= \frac{1}{2s+1}\int_{-s-z_\ell}^{s+1-z_\ell} g_j\left(\theta\right) g_k\left(\theta\right) d\theta \\
&\approx \frac{1}{2s+1}\int_{-s}^{s+1} g_j\left(\theta\right) g_k\left(\theta\right) d\theta \\
&= \langle g_j(\theta) g_k(\theta)\rangle\,.
\end{aligned}$$

The approximation is justified when s is much larger than z_m, since the contribution to the integral near the integration limits is vanishingly small. This approximation becomes exact for large enough s if we clip the receptive field functions g_j to have finite support.

Computing the shifted means has a similar flavor:

$$\begin{aligned}
\langle g_{j+\ell}(\theta)\rangle &= \langle g_j\left(\theta - z_\ell\right)\rangle \\
&= \frac{1}{2s+1}\int_{-s}^{s+1} g_j\left(\theta - z_\ell\right) d\theta \\
&= \frac{1}{2s+1}\int_{-s-z_\ell}^{s+1-z_\ell} g_j\left(\theta\right) d\theta \\
&\approx \frac{1}{2s+1}\int_{-s}^{s+1} g_j\left(\theta\right) d\theta \\
&= \langle g_j(\theta)\rangle\,.
\end{aligned}$$

Taken together, these equations imply that $C_{j+\ell,k+\ell} = C_{j,k}$, so that $\boldsymbol{C}$ is Toeplitz. In our simulations we take $s = 0$ so that $\boldsymbol{C}$ is only approximately Toeplitz, but find that the conclusions below still hold in practice.

While the eigenvectors of Toeplitz matrices are not in general determined as they are for circulant matrices, they are known for tridiagonal Toeplitz matrices. Symmetric tridiagonal Toeplitz matrices all have the same eigenvectors, of the form $u_j^{(k)} = a\sin\left(\frac{j\pi k}{m+1}\right)$ for $k, j = 1,...,m$, where a is an arbitrary nonzero scalar. The odd eigenvectors $\boldsymbol{u}^{(2k+1)}$ are symmetric (which in particular means that $u_1^{(2k+1)} = u_m^{(2k+1)}$) while the even eigenvectors are antisymmetric (in particular, $u_1^{(2k)} = -u_m^{(2k)}$). Recall that the $\boldsymbol{u}^{(k)}$ make up the columns of

$\boldsymbol{U}$.

As before, we consider $\boldsymbol{U}_N$ as a geometric object embedded in N dimensional space, where the rows are datapoints. Under the assumptions of tridiagonal covariance matrix, the eigenvectors $\boldsymbol{u}^{(k)}$ given above reveal a particular structure: in our numerical tests, the rows of $\boldsymbol{U}_N$ lie along a curve with the endpoints disconnected, provided that $N < m$. To show this, we need to show that the distance the first and last row of $\boldsymbol{U}_N$ is larger than the distance between adjacent rows. While we do not prove this here, in practice we have found this to be the case numerically (data not shown). In fact, for $m - N$ sufficiently large we find that the distance between the first and last row is larger than the distance between rows k and $k + 2$, for $k = 1, ..., m - 2$ as well.

With this "gap" between the first and last row of $\boldsymbol{U}_N$, we can use nonlinear dimensionality reduction methods to reveal the structure of a line segment. Consider again the strategy of connecting every point to its two nearest neighbors, as is done in using Isomap. In this case the "middle" sections of the curve will look as in the case of the periodic latent variable, but the ends will be different. If $m - N$ is large enough then the two endpoints of the line will not be connected, and the general structure of the graph will be that of a line.

Now we consider Toeplitz matrices with more than three (but still finitely many) nonzero diagonals. For fixed k, the eigenvector $\boldsymbol{u}^{(k)}$ in this case approaches the form $a \sin\left(\frac{j\pi k}{m+1}\right)$ in the asymptotic limit of large m [118]. It follows that, after truncating to $\boldsymbol{U}_N$ for finite N and taking m to be large, we can use the same reasoning as in the tridiagonal case to infer that the rows of $\boldsymbol{U}_N$ lie along a curve with the endpoints disconnected. The structure of the eigenvectors is illustrated by plotting $\boldsymbol{u}^{(1)}$ and $\boldsymbol{u}^{(2)}$ against each other in Fig 3.5d for $m = 60$.

This topology appears in the weights of the trained autoencoder, as shown by Isomap in Fig 3.4f. Here the network is trained with $\lambda = 4e^{-6}$ and $m = 100$. PCA projections of the weights do not reveal this structure as clearly (Fig 3.4e).

Tensored latent variables give rise to tensored weights

In this section we consider combinations of latent variables found by taking tensor products of other latent variables. Consider the case of "place cell" encoding on a torus, where both boundaries of the grid are periodic. This can be thought of as a tensored combination of two periodic latent variables. Suppose that the first and second coordinates of $\boldsymbol{\theta}$ correspond to the periodic latent variables θ_1 and θ_2, respectively, and that each is i.i.d. uniformly distributed on the circle S^1.

Recall our choice of Gaussian curve response function on the circle: $g_k(\theta) = a\exp(-d_{S^1}(\theta, z_k)^2/\sigma^2)$ where a is a positive scalar and $d_{S^1}(\theta,\theta') = \min\{|\theta-\theta'|, 2\pi - |\theta-\theta'|\}$ is distance on the circle. We abuse notation slightly and use the same name for a Gaussian curve response function on the torus: $g_{i,k}(\boldsymbol{\theta}) = a^2\exp(-d(\boldsymbol{\theta}, \boldsymbol{z}_{i,k})^2/\sigma^2)$ where d is Euclidean distance on the torus, which can be written

$$d(\boldsymbol{\theta},\boldsymbol{\theta}') = \sqrt{d_{S^1}(\theta_1,\theta_1')^2 + d_{S^1}(\theta_2,\theta_2')^2}.$$

This time the tuning curve centers $\boldsymbol{z}_{i,k}$ have two indices and evenly tile the two-dimensional surface of the torus. Our goal is to decompose $\boldsymbol{C}_{i,j,k,\ell} = \langle g_{i,k}(\boldsymbol{\theta})g_{j,\ell}(\boldsymbol{\theta})\rangle - \langle g_{i,k}(\boldsymbol{\theta})\rangle\langle g_{j,\ell}(\boldsymbol{\theta})\rangle$ into contributions from tuning curves $g_k(\theta)$ defined on the circle. We start with the observation that

$$g_{ik}(\boldsymbol{\theta}) = g_i(\theta_1)\, g_k(\theta_2).$$

Using this, along with independence of θ_1 and θ_2,

$$\begin{aligned}\boldsymbol{C}_{i,j,k,\ell} &= \langle g_{i,k}(\boldsymbol{\theta})g_{j,\ell}(\boldsymbol{\theta})\rangle - \langle g_{i,k}(\boldsymbol{\theta})\rangle\langle g_{j,\ell}(\boldsymbol{\theta})\rangle \\ &= \langle g_i(\theta_1)\, g_j(\theta_1)\, g_k(\theta_2)\, g_\ell(\theta_2)\rangle - \langle g_i(\theta_1)\, g_k(\theta_2)\rangle\langle g_j(\theta_1)\, g_\ell(\theta_2)\rangle \\ &= \langle g_i(\theta_1)\, g_j(\theta_1)\rangle\langle g_k(\theta_2)\, g_\ell(\theta_2)\rangle - \langle g_i(\theta_1)\rangle\langle g_k(\theta_2)\rangle\langle g_j(\theta_1)\rangle\langle g_\ell(\theta_2)\rangle.\end{aligned}$$

Recall that $\langle g_j(\theta)\rangle = \mu$ is independent of j. If we let $(\boldsymbol{C}_{S^1})_{ij} = \langle g_i(\theta)\, g_j(\theta)\rangle - \mu^2$ be the covariance matrix for inputs on a circle, then we can write

$$\begin{aligned}C_{i,j,k,\ell} &= ((\boldsymbol{C}_{S^1})_{ij} + \mu^2)((\boldsymbol{C}_{S^1})_{k\ell} + \mu^2) - \mu^4 \\ &= (\boldsymbol{C}_{S^1})_{ij}(\boldsymbol{C}_{S^1})_{k\ell} + \mu^2(\boldsymbol{C}_{S^1})_{ij} + \mu^2(\boldsymbol{C}_{S^1})_{k\ell}.\end{aligned}$$

From this equation, we can see that $\boldsymbol{C}$ can be written as sums of Kronecker tensor products (denoted $\otimes$):

$$\boldsymbol{C} = \boldsymbol{C}_{S^1}\otimes\boldsymbol{C}_{S^1} + \mu^2\boldsymbol{C}_{S^1}\otimes\mathbf{1}_m\mathbf{1}_m^T + \mu^2\mathbf{1}_m\mathbf{1}_m^T\otimes\boldsymbol{C}_{S^1}.$$

Matrix multiplication of Kronecker products satisfies the *mixed-product property*: $(\boldsymbol{A}\otimes\boldsymbol{B})(\boldsymbol{C}\otimes\boldsymbol{D}) = (\boldsymbol{AC})\otimes(\boldsymbol{BD})$. Suppose that $\boldsymbol{u}$ and $\boldsymbol{v}$ are eigenvectors of $\boldsymbol{C}_{S^1}$ with eigenvalues

λ_u and λ_v, respectively. Then

$$\begin{aligned}\boldsymbol{C}(\boldsymbol{u}\otimes\boldsymbol{v}) &= \lambda_u\lambda_v\boldsymbol{u}\otimes\boldsymbol{v} + \mu^2\lambda_u\boldsymbol{u}\otimes\mathbf{1}_m\mathbf{1}_m^T\boldsymbol{v} + \mu^2\lambda_v\mathbf{1}_m\mathbf{1}_m^T\boldsymbol{u}\otimes\boldsymbol{v} \\ &= \lambda_u\lambda_v\boldsymbol{u}\otimes\boldsymbol{v} + \mu^2\lambda_u\|\boldsymbol{v}\|_1\boldsymbol{u}\otimes\mathbf{1}_m + \mu^2\lambda_v\|\boldsymbol{u}\|_1\mathbf{1}_m\otimes\boldsymbol{v}.\end{aligned}$$

This equation reveals that $\boldsymbol{u}\otimes\boldsymbol{v}$ are eigenvectors of $\boldsymbol{C}$ provided either (1) $\mu = 0$ or (2) $\|\boldsymbol{u}\|_1 = 0$ or $\boldsymbol{u} = \mathbf{1}_m$, and $\|\boldsymbol{v}\|_1 = 0$ or $\boldsymbol{v} = \mathbf{1}_m$. The eigenvectors of $\boldsymbol{C}_{S^1}$ satisfy condition (2), as is easy to verify. Hence in the case of inputs formed from tensoring two periodic latent variables, we can find closed form solutions for the eigenvectors of the covariance matrix. These eigenvectors are tensor products of periodic latent variables, so that their structure reflects that of a torus (twisted nonlinearly through N dimensional space). This structure appears in the weights of the trained autoencoder (Fig 3.4l). Here the network is trained with $\lambda = 4e^{-6}$ and $m = 20 \cdot 20 = 400$.

In the case where one or both of the variables being tensored is nonperiodic, we currently lack a general mathematical characterization of the eigenvectors. In the special case when the mean response is zero, $\mu = 0$, the eigenvectors of $\boldsymbol{C}$ are the tensor products of the eigenvectors of the covariance matrices for the two latent variables. This can be shown by similar reasoning as above. Even when μ is nonzero, we see experimentally that the structure of the tensor product of a periodic and nonperiodic latent variable resembles a breaking of the toroidal structure similar to the scalar case. In particular, one end of the torus has a gap, which makes the structure resemble that of a cylinder. In this case the covariance matrix has the form shown in Fig 3.5e if the periodic variable is the first coordinate and Fig 3.5g if the nonperiodic variable is the first coordinate (this relationship may be reversed depending on how the four indices of $\boldsymbol{C}$ are unrolled into two indices). The cylindrical structure can be seen in Figs 3.5f and 3.5h. This cylindrical structure is also reflected in the weights of the autoencoder (Fig 3.4i). Here the network is trained with $\lambda = 4e^{-6}$ and $m = 20 \cdot 20 = 400$.

3.3.3 Effects of regularization

In Section 3.3.2 we found that regularization resulted in the optimal weights being defined up to an orthogonal transformation. Without this regularization, the optimal solution may be skewed by an invertible linear transformation. In this section we explore the effects of regularization in more detail. In particular, we probe to what extent regularization is necessary for recovering structure from the weights.

As an indication for the amount of skewness in the weights, we take the ratio of the first and second singular values, skew $= \sigma_1/\sigma_2$. This skew is 1 for matrices with orthonormal columns, and is always at least 1. In the following analyses, the weights are initialized to have extra skew beyond what would occur with normally distributed weights initialized in the typical fashion. To do this, we scale the top singular value of the weights (for both input and output weights) by a factor of five, while keeping everything else in the singular value decomposition the same. With this initialization, the skewness measure is plotted through training in Fig 3.6a, for a network trained in the case of a periodic scalar latent variable and a bottleneck size of $n = 10$. Here we see that while the skewness in the output weights is rapidly removed, with no regularization skewness in the input weights persists. Regularization removes skewness from both input and output weights. Fig 3.6b shows skewness at the end of 60 epochs across a range of n, showing the same phenomenon as in the case $n = 10$. All of these figures are generated while using the tanh nonlinearity.

This skewness only suggests the degree to which the periodic latent variable is recoverable. To concretely measure the recoverability, we formulate a measure of "circleness" which indicates the degree to which the low-dimensional structure extracted from the weights is a circle. To do this, we first use Isomap to embed the weights in a two-dimensional space. We next build a persistence diagram tracking the birth and death of loops on the two-dimensional data. As a brief description of this process, a graph is built on the datapoints, where points are connected if they come within a radius r of each other. As r is increased from zero, loops in the graph will be created ("birth time") and eventually destroyed ("death time") once all the space within the loop is "filled in". The time between the birth and death of the loop signifies the "size" of the loop. To measure circleness, we first select the loops with the longest and second longest lifespan. We then use the formula circleness $= \frac{\text{longest lifespan} - \text{second longest lifespan}}{\text{death time of loop with longest lifespan}}$. When there is only one loop, we instead use the formula circleness $= \frac{\text{lifespan}}{\text{death time}}$. Note that these formulas return values in the region 0 to 1. In order for the data to be perfectly circular (circleness value 1), points need to be densely arrayed in a loop, so that at radius ≈ 0 a loop appears. In addition, the data will not be reported as perfectly circular if more than one loop with a significant lifespan appears.

Fig 3.6c shows this circleness measure plotted through training (details of the hyperparameters are similar to the skewness plot Fig 3.6a). The results mirror Fig 3.6a: the circleness of output weights quickly approaches a value close to 1, while the circleness of the input weights does not unless regularization is used. This result holds as the bottleneck size

n varies (Fig 3.6d).

These results suggest that, for useful signals to be reliably extracted from weight matrices, some factor that puts pressure on weight norms to be small may need to be in play. However, this may be less important for downstream (higher level) weights. In the Supplemental Figure S2, the skewness measure is plotted for a variety of latent variable combinations; the results are similar to those shown here. These results also suggest that the learning dynamics of stochastic gradient descent supply some, but potentially not sufficient, implicit regularization to preserve the latent variable structure in the weights.

3.4 Discussion

It is important to investigate the ways that connectivity data can be used to help us understand neural circuits. Here we focus on using dimensionality reduction techniques to infer elements of the function of a neural circuit from the structure of the weights. We find that the latent variables that underlie the inputs can be recovered from the weights of an autoencoder with a single layer of hidden units. We find that periodic inputs give rise to periodic weight structure, and nonperiodic inputs to nonperiodic weight structure. The tensor products of such inputs results in an analogous structure in the weights. These results rest on using nonlinear dimensionality reduction techniques to extract the structure. The emergence of this structure is promoted by regularization that penalizes large weights. It also depends on the inputs encoding low-dimensional latent variables in a high-dimensional way.

The approach of focusing on connectivity data to deduce information about the function of a neural circuit complements other very fruitful efforts of probing the activity of neurons in the circuit. The latter includes the seminal work of Hubel and Wiesel [119], which provided strong evidence via recordings in cat striate cortex that neural responses in this area are built from simple combinations of the responses of retinal ganglion cell neurons. Another noteworthy example is the analysis of bump-attractor-like dynamics in the *Drosophila* ellipsoid body [120], which demonstrated through two-photon calcium imaging that the circuit tracks orientation information through integrated sensory information. There are, however, difficulties in using neural activations alone to draw inferences. For instance, it isn't always clear how to satisfactorily explore the space of all possible input stimuli. Often multiple competing models arise to reproduce neural circuit function or neural activity, and connectivity data can be used to select among them [104]. Connectivity data may also be useful for choosing parameters in models that are overparametrized. In the *Drosophila* ellipsoid body

example, fine-grain analysis of connectivity data will probably be necessary to answer once and for all whether the bump attractor dynamics are implemented by a ring attractor network topology, and how this ring attractor is implemented precisely (see [121] for significant recent steps in this direction).

As analysis of connectivity data has its own set of shortcomings – for instance, information about neural modulation, the precise nonlinear responses of neurons to inputs, and many other factors are left out – hybrid methods that take into account both neural activation as well as connectivity data will be important far into the future. There are also other promising avenues for using connectivity data in ways distinct from the methods considered here. One approach is to develop models that fit neural activity data or that satisfy the believed function of a neural circuit while constraining them with connectivity data (reviewed in [104]). Another fruitful approach is to first generate a network model with a connectivity determined through data-driven means, assume a form of neural unit dynamics in the model, and then analyze the resulting network dynamics (for instance, [122, 47]).

While limited by the simplicity of the task and network model, we hope that the techniques laid out here open the door to interesting future studies. These include applying these techniques to brain connectivity data, as well as developing extensions to more complicated tasks and models such as deeper autoencoders or more general feedforward networks trained on more sophisticated tasks. It would be valuable to determine if the structure can be recovered when exact synapse values are not known, when sparsity constraints on the weights are applied, or when different nonlinearities are used in the model. Brain circuits contain both deep hierarchy and recurrent connections, and it remains to be seen if our methods will be successful in networks that have these complexities. In extending to data from the brain, persistent homology techniques could potentially be combined with nonlinear dimensionality reduction techniques to help deal with inaccuracies in the data. Finally, in laying out the importance of regularization in enforcing meaningful structure, we hope to provide a clearer picture of when efforts to understand biological and artificial neural networks can be successful, and why these efforts might fail. It is still an open question as to if L2 regularization or other constraints are sufficient for enforcing interpretability in broader classes of network models (see [123, 124, 111] for works related to these issues).

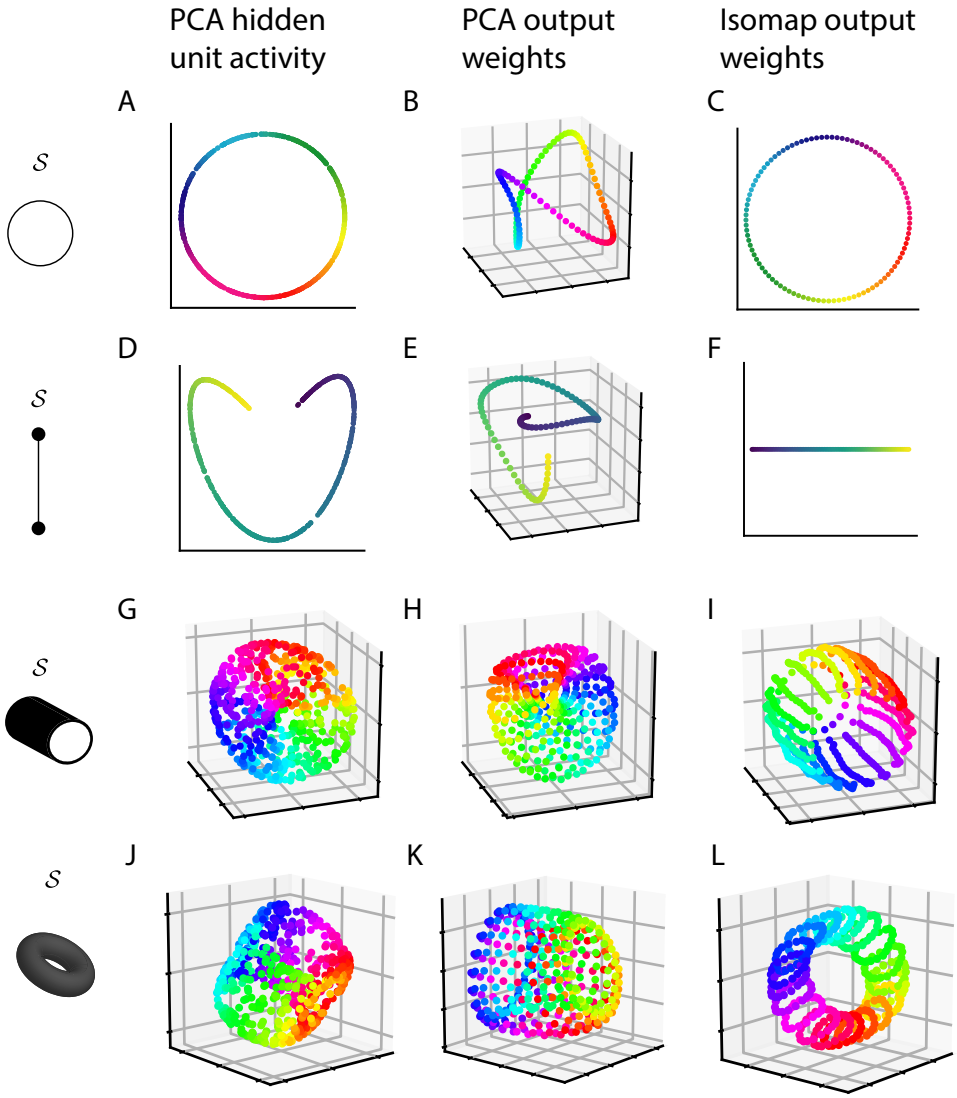

Figure 3.4: The structure of the latent variables can be recovered from the weights of the trained autoencoder by nonlinear dimensionality reduction methods. (a) Hidden unit activations of the autoencoder trained to reconstruct an encoding $\boldsymbol{g}(\theta_s)$ of a periodic latent variable θ_s as in Fig 3.2a and using the same coloration. (b) Principal components of the columns of the output weights of the autoencoder trained on the periodic latent variable. (c) Two-dimensional embedding via Isomap of the columns of the output weights for the network trained on the periodic inputs. (d-f) As in (a-c) but for an encoding $\boldsymbol{g}(\theta_s)$ of a nonperiodic latent variable θ_s as in Fig 3.2b. (g-i) As in (a-c) but for a joint encoding $\boldsymbol{g}(\boldsymbol{\theta}_s)$ of a periodic and non-periodic latent variable, such as that illustrated by Fig 3.2c. In this case the Isomap embedding in (i) is three-dimensional. In (g) color corresponds with the periodic latent variable, while in (h-i) coloration is by the index of the receptive field centers corresponding to the periodic latent variable. (j-l) Same as in (g-i), but for a joint encoding of two periodic latent variables. Color corresponds with the first latent variable. The latent space $\mathcal{S}$ in this case is a torus.

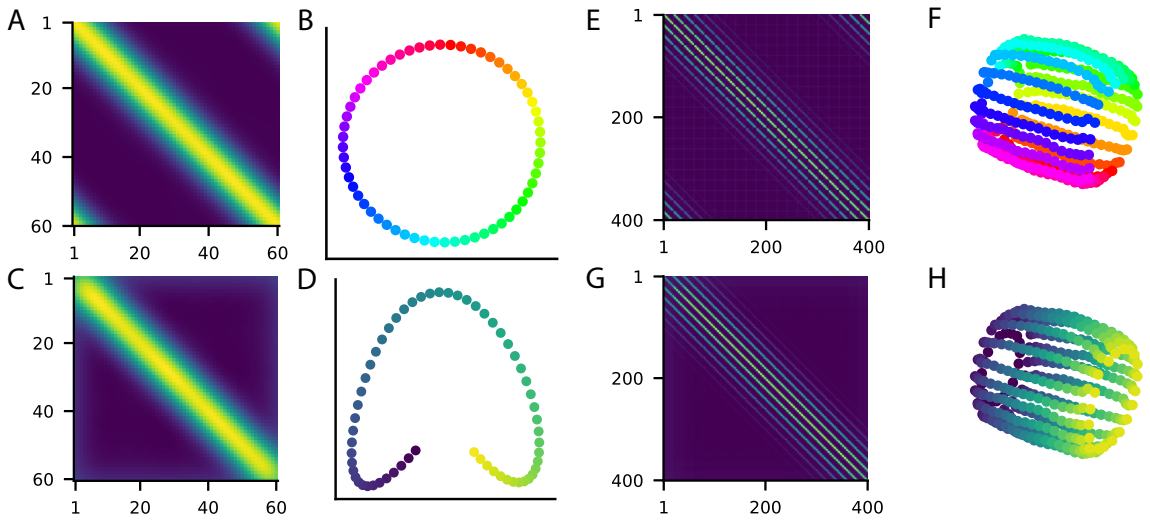

Figure 3.5: The structure of the latent variables can be recovered from the eigenvectors of the covariance matrix of the inputs. (a) Covariance matrix for the responses of $m = 60$ receptive field neurons to a periodic latent variable as in Fig 3.2a. (b) Lowest frequency eigenvectors of the circulant matrix in A, plotted against each other and colored by index. (c) Covariance matrix for the responses of $m = 60$ receptive field neurons to a nonperiodic latent variable as in Fig 3.2b. (d) Lowest frequency eigenvectors of the covariance matrix in (c), plotted against each other and colored by index. (e) Covariance matrix resulting from the tensored responses to a periodic and nonperiodic latent variable as in Fig 3.2c, where the periodic variable is in the first coordinate and the nonperiodic variable is in the second. (f) Eigenvectors of the covariance matrix in (E), reduced to three dimensions by Isomap. Coloration is by the index of the receptive field centers corresponding to the periodic latent variable. The eigenvectors for the covariance matrix in (G) look similar. (g) Covariance matrix as in (E) but with the position of the nonperiodic and periodic variable switched. (h) Same as (f), but colored by the index of the receptive field centers corresponding to the nonperiodic latent variable.

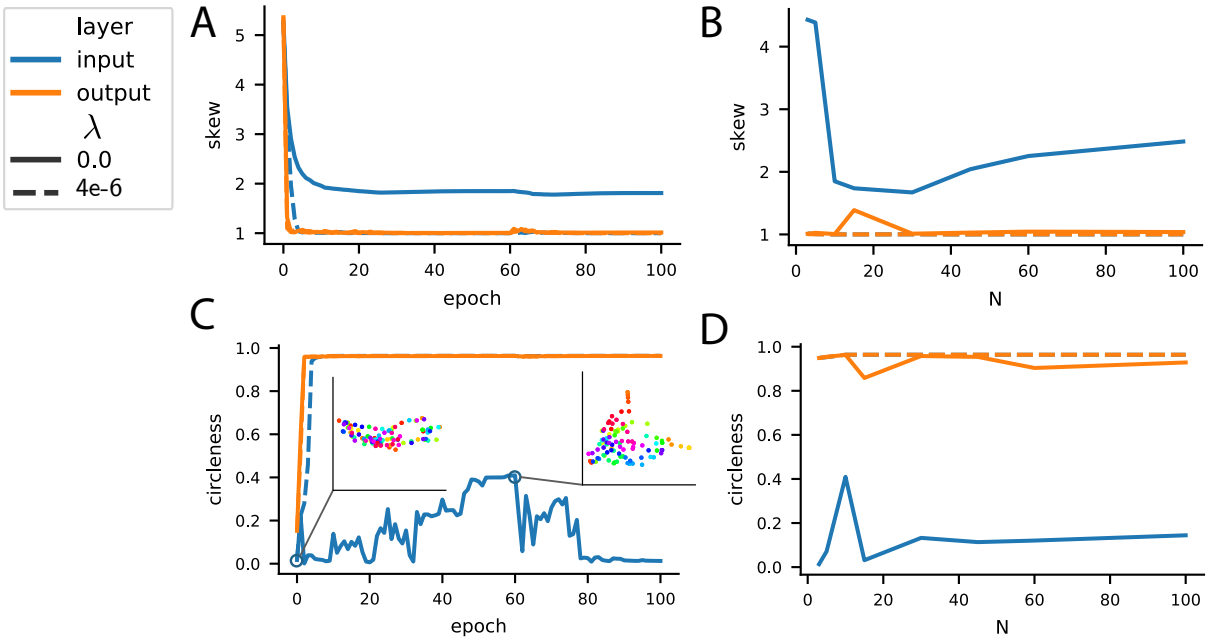

Figure 3.6: L2 regularization ensures the appearance of the latent variable structure in the weights. (a) Skew (ratio of first and second singular values) of the input (blue) and output (red) weight matrices through training. Regularization (dashed line) reduces this skew. (b) Skew measured at the end of training (epoch 60), as a function of increasing bottleneck size. (comment about overlap) (c) Circleness of the weight matrices through training. Regularization ensures that the circle can be recovered. Note that the dashed lines and red solid line overlap. Left inset: Isomap embedding of input weights at initialization. Right inset: Isomap embedding of input weights at epoch 60. (d) Circleness measured at epoch 60, as a function of increasing bottleneck size.

3.5 Appendix

3.5.1 Additional figures

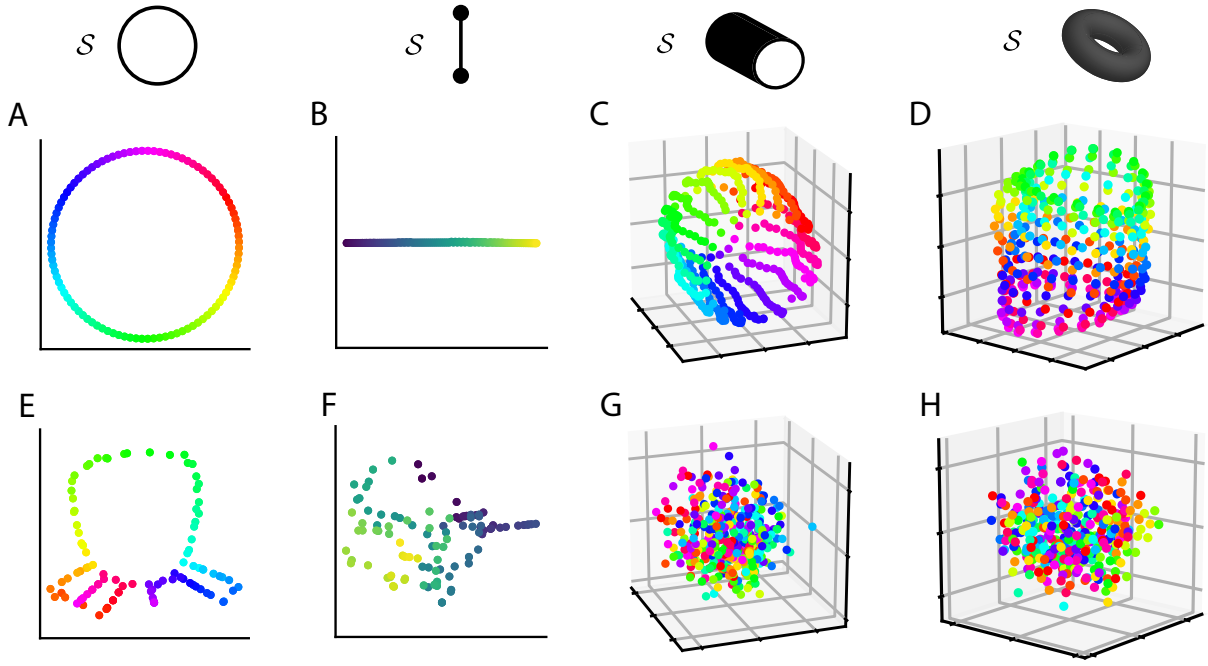

Figure 3.7: Isomap embeddings of weight matrices of trained autoencoder networks without regularization. Here the network is trained to minimize the L2 reconstruction error without the regularization term. In plots (A)-(D), Isomap embedding of the output weights is plotted as in Figure 4 of the main text. Figures (E)-(F) instead plot the input weights. (A) Using a periodic scalar latent variable for the inputs. (B) Using a nonperiodic scalar latent variable. (C) Tensor product of a nonperiodic and periodic latent variable. (D) Tensor product of two latent variables. (E)-(H) Same as (A)-(B), but for input weights.

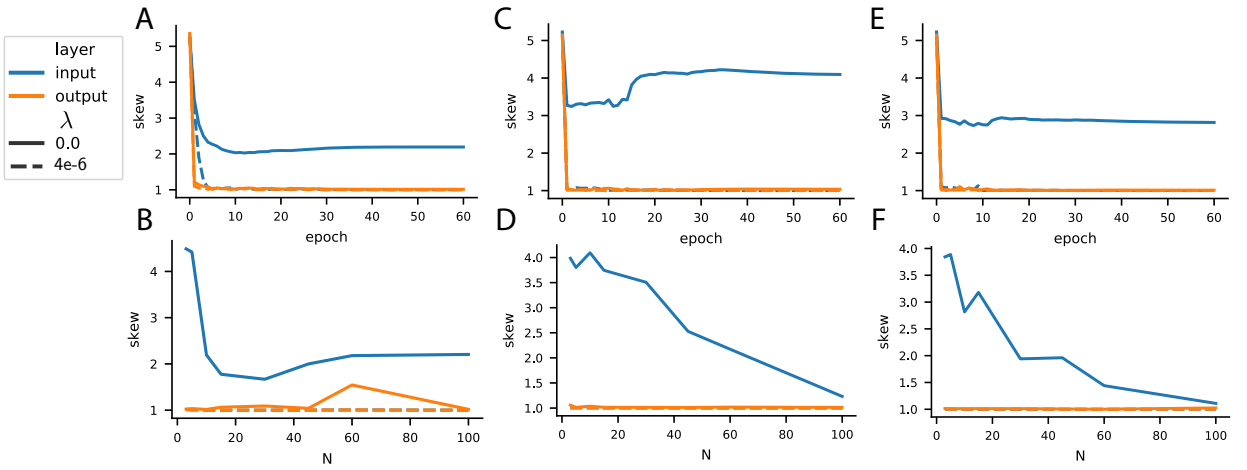

Figure 3.8: Networks trained without regularization remain skewed through training. Skewness is measured by taking the ratio of the largest and second largest singular values, σ_1/σ_2. In the legend, "input" and "output" refers to the skewing of the input and output weights, respectively. The regularizer used during training is $\lambda(||W_{in}||_F^2 + ||W_{out}||_F^2)$. The plots are shown for a variety of latent variables as explored in the main text. (add little schematic from main text) A) Skewing results for inputs generated by a scalar, non-periodic variable, measured through training. B) Skewing results for inputs generated by a scalar, non-periodic variable, measured at epoch 60 over a range of bottleneck sizes n. C) Same as (A) but for a tensoring of a periodic and nonperiodic latent variable. D) Same as (B) but for a tensoring of a periodic and nonperiodic latent variable. E) Same as (A) but for a tensoring of two periodic latent variables. F) Same as (B) but for a tensoring of two periodic latent variables.

Chapter 4

FINAL DISCUSSION

The three chapters in this thesis establish new methods for analyzing artificial neural networks, identify new phenomena that shed light on the success of these networks in solving difficult problems, and provide mechanistic explanations of these phenomena through mathematical analysis. In Chapter 1 we focus on feedforward network models, starting with simple dense networks where we uncover a phenomenon that we coin *dimensionality balance*. This phenomenon is characterized by an initial transformation of input data into a higher-dimensional space until the inputs are near to being linearly separable with significant margins. In subsequent stages (such as in later layers of a deep network), this higher-dimensional representation is compressed as points belonging to the same class are brought closer together. We observe this behavior in simple dense feedforward networks and the more complicated and powerful VGG-11 network model. This shows directly for the first time that networks employ a dimensionality expansion strategy similar to that used in the well-studied kernel learning machines, while also employing the additional strategy of subsequent compression. Through mathematical analysis we argue that the later compression phase is driven by noise in the parameter updates during training, such as that generated by stochastic gradient descent. This noise has a similar effect as putting an L2 cost on the network parameters, which connects this work to a larger body exploring the implicit regularization that deep neural networks appear to enjoy. This sheds light on the incredible success that these network models enjoy in difficult tasks like ImageNet.

In Chapter 2 we extend these investigations to a recurrent neural network model and show how the dynamics of the network influence the dimensionality of the representation through training. We find that chaos can play a valuable role as a dimensionality expansion mechanism, as well as aiding in flexibly learning new tasks. This suggests a beneficial role of the variability found in the response properties of biological networks. This also highlights that dimensionality expansion may not always come for free when training neural networks with SGD, reinforcing the need to make sure that the network architecture and training rules will allow it.

In Chapter 3 we shift our attention away from neural representations and onto the weights of trained networks. We show in a simple network model how and why information about the function of the network can be revealed through dimensionality reduction techniques applied to the weights. We found that regularization is an important ingredient in order for this to occur. This suggests that the interpretability of the data we analyze from the brain may depend to some extent on the brain's drive to conserve resources. It also suggests that regularization of artificial neural networks, whether explicit or implicit, is an important ingredient when interpreting them and comparing them to the brain.

4.1 Future steps

4.1.1 A better understanding of network models

This work offered a look at neural networks through the lens of dimensionality and a particular measure of dynamics (the Lyapunov exponents). This can feel a bit like looking at the networks through a pinhole. Another approach called representational similarity analysis [125, 126] seeks to compare representations of deep neural networks directly with the representations of other deep networks, and with the representations formed in the brain, using their pairwise stimulus correlation matrix.

We suggest an intermediate approach that we hope can be fruitful: to develop a number of salient summary statistics (including measures such as effective and intrinsic dimensionality) and use this ensemble of measures to capture the important aspects of a network's behavior. ANNs can then be compared with brain data, and with each other, within this relatively low-dimensional space of summary statistics. The hope is that eventually the summary statistics will together be able to capture most of what we are interested in. This approach is perhaps analogous to comparing the current weather of Seattle with that of Tokyo by comparing the temperature, chance of precipitation, humidity, and other salient measures. Using representational similarity analysis methods is perhaps a bit more like comparing a pixel map of Seattle's air pressure with a pixel map of Tokyo's air pressure. Clearly both approaches complement each other. The work of inventing interesting summary statistics is well underway, but still needs to be augmented and refined. Some examples are using mutual information [61], using measures of tuning of neural units [21], and using other measures that capture the geometry of response manifolds [32, 127, 128].

In addition, a more developed mathematical theory of neural networks would likely pro-

vide insight. While this work is still nascent, the last several years has seen rapid development in this area, including detailed analyses of deep linear networks [64, 129, 130] and the development of simplified analytical descriptions of deep networks in the limit of infinitely wide hidden layers [131, 132, 133]. These tools and others are poised to unlock new discoveries regarding the behavior of neural networks (for instance [31]). They may also allow for a more sophisticated treatment of dimensionality than the one given here (see [134]).

4.1.2 Dependence on initialization and training hyperparameters

In general the solutions found by training neural networks depends on the parameter initializations as well as the details of the training method. This sensitivity can confound efforts to characterize network models, as it is very challenging if not impossible to fully explore the space of possible initializations and training procedures and hyperparameters. This is one of the reasons why a mathematical explanation of observed phenomena can be so powerful: it makes clearer the assumptions needed for such phenomena to occur. In Chapter 1 we make the case that optimizing with stochastic gradient descent vs gradient descent can have a large impact on the resulting geometry of the representation. In Chapter 2 we highlight how the initialization of the network can have a large impact on both the resulting geometry and dynamics of a recurrent neural network. A more thorough exploration of the changes that can occur in such network models as a result of the model details will be important going forward.

This sensitivity to details is primarily caused by the significant nonconvexity of the loss landscape. It may be that sensible regularization can reduce the complexity of the landscape considerably. In Chapter 3 we highlight such a situation, where a penalty on use of weight resources limits the degree of freedom of the model from arbitrary invertible linear transformations to arbitrary orthogonal transformations. This helps ensure that training the model with different initializations and using different training methods results in networks with the same geometry, both in the weights and in the representation. Such regularization will likely have a similar effect on other network models such as deep neural networks.

In addition to these challenges, there is considerable freedom in choosing how to encode the inputs of the network, as well as the outputs. For instance, in classification tasks the outputs are generally encoded in a one-hot fashion, but this is not necessarily the way that the brain encodes categories. The choice of these encoding will undoubtably affect the representations that are formed in hidden layers. A very interesting path of future

direction is exploring output encodings that are many-hot, so that each class is defined by a set of labels that can be viewed as properties (see [135] for an example of this approach). Related to this, the choice of loss function may have significant effects on the properties of the trained network. The freedom of input and output encoding becomes even larger with spiking artificial neural networks.

4.1.3 Increased model realism and sophistication

Most of the network models considered here are relatively simple. The network considered in Chapter 3 is particularly simple and it is imperative to see if the results extend to other models, such as deep neural networks trained to classify images.

Even the image classification network VGG is an extreme simplification when compared with brain circuits. Brain circuits include a large number of feedback and recurrent connections, details that are only recently being incorporated into image/movie classification networks [136, 137]. In addition, the brain employs an array of distinct kinds of neurons. One of the broadest categories is that of excitatory or inhibitory. When excitatory neurons spike, the signal pulse they emit drives connected neurons to spike as well. Inhibitory neurons are the opposite: their spikes inhibit the spiking of connected neurons. A neuron is either excitatory or inhibitory (Dale's law), unlike in ANNs where a unit can have both positive and negative outgoing connections. The purpose of this division of neurons into excitatory and inhibitory types is still unknown, although it is hypothesized that inhibitory neurons are mainly involved with stabilizing the network while the excitatory neurons carry the true information signal. The purpose of the further division of neurons into myriad types is likewise unknown. While neuron types are sometimes defined primarily by the connectivity patterns into and out of these neurons, the way that a neuron transforms its incoming inputs and converts these into spikes is also a differentiating factor. It is not clear if such a level of realism is necessary to incorporate into ANN models in order to capture the fundamental aspects of brain function.

The ubiquitous backpropagation algorithm is known to not be biologically plausible. Finding the learning rules that enable the brain to learn is currently an area of intense research. It may be that training artificial networks with more biologically realistic learning rules may result in networks that more faithfully model the brain. Such learning rules may or may not be necessary in the quest to use ANNs to uncover basic principles of brain function. As an example, incorporating conservation of resources in our optimization criteria may be

necessary for the learning dynamics to satisfactorily parallel those of the brain.

In order to approach the brain's flexibility and power to perform a range of sophisticated functions, ANNs will probably need to adopt the modular strategy used by the brain. The problem of engineering such a modular network, where individual modules specialize but communicate and ultimately work together to solve tasks, is daunting indeed. If such networks are built to perform well on difficult tasks, it will be an additional challenge to extract insights by analyzing their inner workings. See [14] for a review. In general, the connections between neurons both within and across modules depend on spatial proximity. It may be that incorporating a spatial embedding of neural units will result in ANNs that more faithfully model the brain and reveal new insights. A version of this has already occurred in convolutional neural networks, where the local nature of convolutional filters as inspired by the early visual system turned out to be a remarkable way of biasing networks to finding much more accurate solutions.

ANN modelers will also need to continue to push beyond simple well-defined loss functions. Biological organisms learn through a complex process of positive and negative reinforcement, with a rich set of goals and rewards and punishments that are often simultaneously at play. The field of reinforcement learning has grown around the intention of capturing this complexity in artificial learning agents. Unfortunately, many of the issues that occur with simple but highly nonconvex loss functions are even more pronounced here, where a very large array of different approaches to training agents has been developed, each with hyperparameters that can be quite difficult to tune and with architectures that are not necessarily chosen to emulate biological systems. Nonetheless, this work has made breathtaking progress in the past years and continues to push artificial agents further into the domain of biological intelligence. Even with the challenges involved, very fruitful comparisons with neural recordings have already been made [25, 138]. There is hope that these models will someday consolidate into a more coherent picture, opening the door to many more efforts at modeling brain circuits with ANNs.

4.1.4 Applying findings to data

This work lays out phenomena that appear to have a relatively general character. Dimensionality balance and the typical humpback-shaped curves that result (i.e. Fig 1.7) may also appear in biological neural circuits. Indeed, dimensionality expansion is most often hypothesized to occur in lower-order sensory areas while low-dimensional activity is often observed

in higher-order cognitive areas such as prefrontal cortex. Investigating explicitly where dimensionality increases and decreases along the hierarchy of brain circuit computation could shed light on the computation. For instance, if an area is observed to increase dimensionality significantly, one can then look closer at what properties of the neural representation are causing this, and what computational purpose this may serve. Certain tunings, such as place and grid cell encodings of spatial location, likely have key dimensionality signatures (in this case high effective dimensionality and low intrinsic dimensionality). Analyzing the connectivity data may further shed light on how these transformations in neural circuits occur, such as has already been done in olfactory circuits [43, 44]. As we demonstrate here in a simple model, a geometric consideration of connectivity data can yield additional insights. A testing ground for these techniques could be in neural circuits with a seemingly well-defined and relatively well-understood function, such as early visual system circuits and drosophila ellipsoid body. Other models inspired more directly by the biology may also be fertile ground for geometric considerations of the network weights, as indicated in [39].

Our hope is that the investigations laid out in this thesis establish useful conceptual foundations for further inquiry into the workings of artificial and biological neural networks, and the connections between.

BIBLIOGRAPHY

[1] Santiago Ramón y Cajal. The Structure and Connexions of Neurons. *Nobel Media AB*.

[2] Donald O. Hebb. *Organization of Behavior*. Psychology Press, 1949.

[3] Timothy P. Lillicrap, Adam Santoro, Luke Marris, Colin J. Akerman, and Geoffrey Hinton. Backpropagation and the brain. *Nature Reviews Neuroscience*, 21(6):335–346, June 2020.

[4] Frank Rosenblatt. The Perceptron – a perceiving and recognizing automaton. Technical Report 85-460-1, Cornell Aeronautical Laboratory, Inc, January 1957.

[5] M. A. Aizerman, Emmanuel M. Braverman, and L. I. Rozoner. Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control*, 25:821–837, 1964.

[6] 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.

[7] V. N. Vapnik and A. Ya. Lerner. Pattern recognition using generalized portraits. *Avtomatika i Telemekhanika*, 24:774–780, June 1963.

[8] V. N. Vapnik and A. Ya. Chervonenkis. A class of algorithms for pattern recognition learning. *Avtomatika i Telemekhanika*, 25(6):937–945, 1964.

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

[10] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, pages 144–152, New York, NY, USA, 1992. Association for Computing Machinery.

[11] David E. Rumelhart and James L. McClelland. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations*. MIT Press, 1987.

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

[13] Blake A. Richards, Timothy P. Lillicrap, Philippe Beaudoin, Yoshua Bengio, Rafal Bogacz, Amelia Christensen, Claudia Clopath, Rui Ponte Costa, Archy de Berker, Surya Ganguli, Colleen J. Gillon, Danijar Hafner, Adam Kepecs, Nikolaus Kriegeskorte, Peter Latham, Grace W. Lindsay, Kenneth D. Miller, Richard Naud, Christopher C. Pack, Panayiota Poirazi, Pieter Roelfsema, João Sacramento, Andrew Saxe, Benjamin Scellier, Anna C. Schapiro, Walter Senn, Greg Wayne, Daniel Yamins, Friedemann Zenke, Joel Zylberberg, Denis Therien, and Konrad P. Kording. A deep learning framework for neuroscience. *Nature Neuroscience*, 22(11):1761–1770, November 2019.

[14] Guangyu Robert Yang, Michael W Cole, and Kanaka Rajan. How to study the neural mechanisms of multiple tasks. *Current Opinion in Behavioral Sciences*, 29:134–143, October 2019.

[15] D. J. Felleman and D. C. Van Essen. Distributed hierarchical processing in the primate cerebral cortex. *Cerebral Cortex (New York, N.Y.: 1991)*, 1(1):1–47, 1991 Jan-Feb.

[16] 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.

[17] D. H. Hubel and T. N. Wiesel. Receptive fields of single neurones in the cat's striate cortex. *The Journal of Physiology*, 148(3):574–591, October 1959.

[18] Bruno A. Olshausen and David J. Field. Sparse coding with an overcomplete basis set: A strategy employed by V1? *Vision Research*, 37(23):3311–3325, December 1997.

[19] Maximilian Riesenhuber and Tomaso Poggio. Hierarchical models of object recognition in cortex. *Nature Neuroscience*, 2(11):1019–1025, November 1999.

[20] Yann LeCun and Yoshua Bengio. Convolutional networks for images, speech, and time series. In *The Handbook of Brain Theory and Neural Networks*, pages 255–258. MIT Press, Cambridge, MA, USA, 1998.

[21] Dean A Pospisil, Anitha Pasupathy, and Wyeth Bair. 'Artiphysiology' reveals V4-like shape tuning in a deep network trained for image classification. *eLife*, 7:e38242, December 2018.

[22] Daniel L K Yamins and James J DiCarlo. Using goal-driven deep learning models to understand sensory cortex. *Nature Neuroscience*, 19(3):356–365, March 2016.

[23] Grace Lindsay. Convolutional neural networks as a model of the visual system: Past, present, and future. *Journal of Cognitive Neuroscience*, 0(0):1–15.

[24] Alexander J.E. Kell, Daniel L.K. Yamins, Erica N. Shook, Sam V. Norman-Haignere, and Josh H. McDermott. A Task-Optimized Neural Network Replicates Human Auditory Behavior, Predicts Brain Responses, and Reveals a Cortical Processing Hierarchy. *Neuron*, 98(3):630–644.e16, May 2018.

[25] 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):429–433, 2018.

[26] Yann LeCun, Leon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. Efficient BackProp. In *Neural Networks: Tricks of the Trade*, volume 7700 of *Lecture Notes in Computer Science*. Springer, Berlin, Heidelberg, 2012.

[27] Sam Smith and Quoc V. Le. A bayesian perspective on generalization and stochastic gradient descent. In *ICLR*, 2018.

[28] Andrew K Lampinen, Andrew K Lampinen, and Surya Ganguli. An analytic theory of generalization dynamics and transfer learning in deep linear networks. *arXiv.org*, September 2018.

[29] Sanjeev Arora, Simon S. Du, Wei Hu, Zhiyuan Li, and Ruosong Wang. Fine-Grained Analysis of Optimization and Generalization for Overparameterized Two-Layer Neural Networks. In *36th International Conference on Machine Learning*, volume 2019-June, pages 477–502, January 2019.

[30] Tomaso Poggio, Andrzej Banburski, and Qianli Liao. Theoretical issues in deep networks. *Proceedings of the National Academy of Sciences*, 2020.

[31] Blake Bordelon, Abdulkadir Canatar, and Cengiz Pehlevan. Spectrum Dependent Learning Curves in Kernel Regression and Wide Neural Networks. *arXiv:2002.02561 [cs, stat]*, June 2020.

[32] Uri Cohen, SueYeon Chung, Daniel D. Lee, and Haim Sompolinsky. Separability and geometry of object manifolds in deep neural networks. *Nature Communications*, 11(1):746, February 2020.

[33] David Sussillo and Omri Barak. Opening the black box: Low-dimensional dynamics in high-dimensional recurrent neural networks. *Neural computation*, 25(3):626–49, 2013.

[34] Kanaka Rajan, Christopher D D. Harvey, and David W W. Tank. Recurrent Network Models of Sequence Generation and Memory. *Neuron*, 90(1):128–142, 2016.

[35] Pierre Enel, Emmanuel Procyk, René Quilodran, and Peter Ford Dominey. Reservoir Computing Properties of Neural Dynamics in Prefrontal Cortex. *PLOS Computational Biology*, 12(6):e1004967, June 2016.

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

[37] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A large-scale hierarchical image database. In *CVPR09*, 2009.

[38] H. Sebastian Seung. Reading the Book of Memory: Sparse Sampling versus Dense Mapping of Connectomes. *Neuron*, 62(1):17–29, April 2009.

[39] R. Clay Reid. Computational Geometry of Cortical Networks: Manifold-Embedding Analysis of Multi-Site Recordings, Connectomic Graphs, and Deep Learning Networks — Simons Institute for the Theory of Computing, March 2018.

[40] 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.

[41] Bernhard Schölkopf and Alexander J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Adaptive Computation and Machine Learning.

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

[43] 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.

[44] 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, 2017.

[45] 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.

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

[47] Stefano Recanatesi, Gabriel Koch Ocker, Michael A. Buice, and Eric Shea-Brown. Dimensionality in recurrent spiking networks: Global trends in activity and local origins in connectivity. *PLOS Computational Biology*, 15(7):e1006446, July 2019.

[48] Matthew Farrell, Stefano Recanatesi, Timothy Moore, Guillaume Lajoie, and Eric Shea-Brown. Recurrent neural networks learn robust representations by dynamically balancing compression and expansion. *bioRxiv*, page 564476, December 2019.

[49] Kanaka Rajan, Laurence F. Abbott, and Haim Sompolinsky. Stimulus-dependent suppression of intrinsic variability in recurrent neural networks. *BMC Neuroscience*, 11(1):O17, July 2010.

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

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

[52] C. S. Wallace and D. M. Boulton. An information measure for classification. *The Computer Journal*, 11(2):185–194, August 1968.

[53] J. Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978.

[54] Maximilian Riesenhuber and Tomaso Poggio. Hierarchical models of object recognition in cortex. *Nature Neuroscience*, 2(11):1019, November 1999.

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

[56] 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.

[57] James Dicarlo, Davide Zoccolan, and Nicole Rust. How does the brain solve visual object recognition? *Neuron*, 73:415–34, February 2012.

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

[59] 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.

[60] 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.

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

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

[63] Tomaso Poggio, Qianli Liao, and Andrzej Banburski. Complexity control by gradient descent in deep networks. *Nature Communications*, 11(1):1027, February 2020.

[64] Andrew M. Saxe, James L. McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv:1312.6120 [cond-mat, q-bio, stat]*, February 2014.

[65] Yao Zhang, Andrew M. Saxe, Madhu S. Advani, and Alpha A. Lee. Energy-entropy competition and the effectiveness of stochastic gradient descent in machine learning. *Molecular Physics*, 116(21-22):3214–3223, November 2018.

[66] Stefan Wager, Sida Wang, and Percy S Liang. Dropout Training as Adaptive Regularization. *Advances in Neural Information Processing Systems*, page 9.

[67] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[68] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *International Conference on Learning Representations*, April 2015.

[69] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[70] Alessio Ansuini, Alessandro Laio, Jakob H Macke, and Davide Zoccolan. Intrinsic dimension of data representations in deep neural networks. page 11.

[71] 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*, 148:1–47, January 2001.

[72] 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.

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

[74] Christian Keup, Tobias Kühn, David Dahmen, and Moritz Helias. Transient chaotic dimensionality expansion by recurrent networks. *arXiv:2002.11006 [cond-mat, q-bio]*, February 2020.

[75] H. Sompolinsky, A. Crisanti, and H. J. Sommers. Chaos in random neural networks. *Physical Review Letters*, 61(3):259–262, 1988.

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

[77] 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.

[78] 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.

[79] Guillaume Lajoie, Kevin Lin, and Eric Shea-Brown. Chaos and reliability in balanced spiking networks with temporal drive. *Physical review. E, Statistical, nonlinear, and soft matter physics*, 87(5):2432–2437, 2013.

[80] 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.

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

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

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

[84] 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.

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

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

[87] 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.

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

[89] David Marr. A theory of cerebellar cortex. *The Journal of Physiology*, 202(2):437–470, 1969.

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

[91] Carsen Stringer, Marius Pachitariu, Nicholas Steinmetz, Matteo Carandini, and Kenneth D. Harris. High-dimensional geometry of population responses in visual cortex. *Nature*, 571(7765):361–365, July 2019.

[92] Luca Mazzucato, Alfredo Fontanini, and Giancarlo La Camera. Stimuli Reduce the Dimensionality of Cortical Activity. *Frontiers in Systems Neuroscience*, 2016.

[93] 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.

[94] Itamar Daniel Landau and Haim Sompolinsky. Coherent chaos in a recurrent neural network with structured connectivity. *PLOS Computational Biology*, 14(12):e1006309, December 2018.

[95] 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.

[96] 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.

[97] 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.

[98] 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.

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

[100] C. Shan Xu, Michal Januszewski, Zhiyuan Lu, Shin-ya Takemura, Kenneth J. Hayworth, Gary Huang, Kazunori Shinomiya, Jeremy Maitin-Shepard, David Ackerman, Stuart Berg, Tim Blakely, John Bogovic, Jody Clements, Tom Dolafi, Philip Hubbard, Dagmar Kainmueller, William Katz, Takashi Kawase, Khaled A. Khairy, Laramie Leavitt, Peter H. Li, Larry Lindsey, Nicole Neubarth, Donald J. Olbris, Hideo Otsuna, Eric T. Troutman, Lowell Umayam, Ting Zhao, Masayoshi Ito, Jens Goldammer, Tanya Wolff, Robert Svirskas, Philipp Schlegel, Erika R. Neace, Christopher J. Knecht, Chelsea X. Alvarado, Dennis A. Bailey, Samantha Ballinger, Jolanta A Borycz, Brandon S. Canino, Natasha Cheatham, Michael Cook, Marisa Dreher, Octave Duclos, Bryon Eubanks, Kelli Fairbanks, Samantha Finley, Nora Forknall, Audrey Francis, Gary Patrick Hopkins, Emily M. Joyce, SungJin Kim, Nicole A. Kirk, Julie Kovalyak, Shirley A. Lauchie, Alanna Lohff, Charli Maldonado, Emily A. Manley, Sari McLin, Caroline Mooney, Miatta Ndama, Omotara Ogundeyi, Nneoma Okeoma, Christopher Ordish, Nicholas Padilla, Christopher Patrick, Tyler Paterson, Elliott E. Phillips, Emily M. Phillips, Neha Rampally, Caitlin Ribeiro, Madelaine K Robertson, Jon Thomson Rymer, Sean M. Ryan, Megan Sammons, Anne K. Scott, Ashley L. Scott, Aya Shinomiya, Claire Smith, Kelsey Smith, Natalie L. Smith, Margaret A. Sobeski, Alia Suleiman, Jackie Swift, Satoko Takemura, Iris Talebi, Dorota Tarnogorska, Emily Tenshaw, Temour Tokhi, John J. Walsh, Tansy Yang, Jane Anne Horne, Feng Li, Ruchi

Parekh, Patricia K. Rivlin, Vivek Jayaraman, Kei Ito, Stephan Saalfeld, Reed George, Ian Meinertzhagen, Gerald M. Rubin, Harald F. Hess, Louis K. Scheffer, Viren Jain, and Stephen M. Plaza. A connectome of the adult drosophila central brain. *bioRxiv*, 2020.

[101] The Quest to Unravel The Connectome. https://alleninstitute.org/what-we-do/brain-science/news-press/articles/quest-unravel-connectome, 2018.

[102] Rodney J. Douglas, Kevan A. C. Martin, and David Whitteridge. A canonical microcircuit for neocortex. *Neural Comput.*, 1(4):480–488, December 1989.

[103] Rodney J. Douglas and Kevan A. C. Martin. Neuronal circuits of the neocortex. *Annual Review of Neuroscience*, 27:419–451, 2004.

[104] Ashok Litwin-Kumar and Srinivas C Turaga. Constraining computational models using electron microscopy wiring diagrams. *Current Opinion in Neurobiology*, 58:94–100, October 2019.

[105] Jinseop S. Kim, Matthew J. Greene, Aleksandar Zlateski, Kisuk Lee, Mark Richardson, Srinivas C. Turaga, Michael Purcaro, Matthew Balkam, Amy Robinson, Bardia F. Behabadi, Michael Campos, Winfried Denk, and H. Sebastian Seung. Space–time wiring specificity supports direction selectivity in the retina. *Nature*, 509(7500):331–336, May 2014.

[106] Sophie J. C. Caron, Vanessa Ruta, L. F. Abbott, and Richard Axel. Random convergence of olfactory inputs in the Drosophila mushroom body. *Nature*, 497(7447):113–117, May 2013.

[107] Mala Murthy, Ila Fiete, and Gilles Laurent. Testing Odor Response Stereotypy in the Drosophila Mushroom Body. *Neuron*, 59(6):1009 – 1023, 2008.

[108] 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.

[109] Li Zhaoping. Theoretical understanding of the early visual processes by data compression and data selection. *Network: Computation in Neural Systems*, 17(4):301–334, January 2006.

[110] Arya A. Pourzanjani, Richard M. Jiang, and Linda R. Petzold. Improving the identifiability of neural networks for bayesian inference. 2017.

[111] Andrzej Banburski, Qianli Liao, Brando Miranda, Lorenzo Rosasco, Jack Hidary, and Tomaso Poggio. Theory III: Dynamics and Generalization in Deep Networks – a simple solution. *arXiv:1903.04991 [cs, stat]*, July 2019. arXiv: 1903.04991.

[112] Matthew Farrell, Stefano Recanatesi, R. Clay Reid, Stefan Mihalas, and Eric Shea-Brown. Autoencoder networks extract latent variables and encode these variables in their connectomes. *bioRxiv*, page 2020.03.04.977702, March 2020.

[113] Joshua B. Tenenbaum, Vin de Silva, and John C. Langford. A Global Geometric Framework for Nonlinear Dimensionality Reduction. *Science*, 290(5500):2319–2323, December 2000.

[114] H. Bourlard and Y. Kamp. Auto-association by multilayer perceptrons and singular value decomposition. *Biological Cybernetics*, 59(4):291–294, September 1988.

[115] Daniel Kunin, Jonathan M. Bloom, Aleksandrina Goeva, and Cotton Seed. Loss Landscapes of Regularized Linear Autoencoders. *arXiv:1901.08168 [cs, stat]*, May 2019.

[116] Madeleine Udell, Corinne Horn, Reza Zadeh, and Stephen Boyd. Generalized Low Rank Models. *Foundations and Trends® in Machine Learning*, 9(1):1–118, 2016.

[117] Philip Davis. *Circulant Matrices*. Pure and Applied Mathematics. New York : Wiley, 1979.

[118] Albrecht Böttcher, Sergei M. Grudsky, and Egor A. Maksimenko. *On the Structure of the Eigenvectors of Large Hermitian Toeplitz Band Matrices*, pages 15–36. Springer Basel, Basel, 2010.

[119] D. H. Hubel and T. N. Wiesel. Receptive fields of single neurones in the cat's striate cortex. *The Journal of Physiology*, 148(3):574–591, October 1959.

[120] Johannes D. Seelig and Vivek Jayaraman. Neural dynamics for landmark orientation and angular path integration. *Nature*, 521(7551):186–191, May 2015.

[121] Daniel B. Turner-Evans, Kristopher T. Jensen, Saba Ali, Tyler Paterson, Arlo Sheridan, Robert P. Ray, Tanya Wolff, Scott Lauritzen, Gerald M. Rubin, Davi Bock, and Vivek Jayaraman. The neuroanatomical ultrastructure and function of a biological ring attractor. *bioRxiv*, 2020.

[122] Hannah Choi and Stefan Mihalas. Synchronization dependent on spatial structures of a mesoscopic whole-brain network. *PLOS Computational Biology*, 15(4):e1006978, April 2019.

[123] Arya A Pourzanjani, Richard M Jiang, and Linda R Petzold. Improving the Identifiability of Neural Networks for Bayesian Inference. *NeurIPS*, Second workshop on Bayesian Deep Learning, 2017.

[124] Ilyes Khemakhem, Diederik P. Kingma, Ricardo Pio Monti, and Aapo Hyvärinen. Variational Autoencoders and Nonlinear ICA: A Unifying Framework. *arXiv:1907.04809 [cs, stat]*, October 2019.

[125] Seyed-Mahdi Khaligh-Razavi and Nikolaus Kriegeskorte. Deep Supervised, but Not Unsupervised, Models May Explain IT Cortical Representation. *PLOS Computational Biology*, 10(11):e1003915, November 2014.

[126] Nikolaus Kriegeskorte. Relating population-code representations between man, monkey, and computational models. *Frontiers in Neuroscience*, 3, 2009.

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

[128] SueYeon Chung, Daniel D. Lee, and Haim Sompolinsky. Linear readout of object manifolds. *Physical Review E*, 93(6):060301, June 2016.

[129] Ben Poole, Subhaneil Lahiri, Maithra Raghu, Jascha Sohl-Dickstein, and Surya Ganguli. Exponential expressivity in deep neural networks through transient chaos. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 3360–3368. Curran Associates, Inc., 2016.

[130] Andrew K Lampinen and Surya Ganguli. An analytic theory of generalization dynamics and transfer learning in deep linear networks. In *International Conference on Learning Representations*, page 20, 2019.

[131] Arthur Jacot, Franck Gabriel, and Clement Hongler. Neural Tangent Kernel: Convergence and Generalization in Neural Networks. *Advances in Neural Information Processing Systems*, page 10, 2018.

[132] Sanjeev Arora, Simon S Du, Wei Hu, Zhiyuan Li, Russ R Salakhutdinov, and Ruosong Wang. On exact computation with an infinitely wide neural net. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8141–8150. Curran Associates, Inc., 2019.

[133] Jaehoon Lee, Lechao Xiao, Samuel S. Schoenholz, Yasaman Bahri, Roman Novak, Jascha Sohl-Dickstein, and Jeffrey Pennington. Wide Neural Networks of Any Depth Evolve as Linear Models Under Gradient Descent. *arXiv:1902.06720 [cs, stat]*, December 2019.

[134] Jonas Paccolat, Leonardo Petrini, Mario Geiger, Kevin Tyloo, and Matthieu Wyart. Compressing invariant manifolds in neural nets. *arXiv:2007.11471 [cs, stat]*, July 2020.

[135] Andrew M. Saxe, James L. McClelland, and Surya Ganguli. A mathematical theory of semantic development in deep neural networks. *Proceedings of the National Academy of Sciences*, 116(23):11537–11546, June 2019.

[136] Hanlin Tang, Martin Schrimpf, William Lotter, Charlotte Moerman, Ana Paredes, Josue Ortega Caro, Walter Hardesty, David Cox, and Gabriel Kreiman. Recurrent computations for visual pattern completion. *Proceedings of the National Academy of Sciences*, 115(35):8835–8840, 2018.

[137] Hosein Hasani, Mahdieh Soleymani, and Hamid Aghajan. Surround modulation: A bio-inspired connectivity structure for convolutional neural networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 15903–15914. Curran Associates, Inc., 2019.

[138] Kimberly L. Stachenfeld, Matthew M. Botvinick, and Samuel J. Gershman. The hippocampus as a predictive map. *Nature Neuroscience*, 20(11):1643–1653, November 2017.