THE HIDDEN GEOMETRY OF LEARNING

Jialin Mao

A DISSERTATION

in

Graduate Group of Applied Mathematics and Computational Science

Presented to the Faculties of the University of Pennsylvania

in

Partial Fulfillment of the Requirements for the

Degree of Doctor of Philosophy

2025

Supervisor of Dissertation

Pratik Chaudhari, Assistant Professor, Electrical and Systems Engineering

Graduate Group Chairperson

Yoichiro Mori, Calabi-Simons Chair in Mathematics and Biology

Dissertation Committee

Kostas Daniilidis, Ruth Yalom Stone Professor, Computer and Information Science
James Sethna, James Gilbert White Professor of Physical Sciences (Cornell University)
René Vidal, Rachleff University Professor, Electrical and Systems Engineering

ABSTRACT

THE HIDDEN GEOMETRY OF LEARNING

Jialin Mao

Pratik Chaudhari

Deep learning has achieved remarkable success in recent years, yet it is still mysterious how they achieved this success. The optimization problem underlying deep learning is usually highly non-convex, and the search space consists of functions with high expressive power. However, simple local search algorithms like stochastic gradient descent can often successfully find solutions that generalize well to unseen data.

This thesis explores the hypothesis that neural networks tend to explore a low-dimensional region of the function space when trained on typical data. This phenomenon may explain both the success of optimization and the generalizability. This perspective emphasizes the interplay between data and optimization dynamics and provides a unifying framework that can be used to study different architectures and tasks.

In Chapter 2, we develop information-geometric techniques to properly define and analyze the model manifold explored by training trajectories of deep networks. We construct a training manifold by training networks with different configurations from different initializations and characterize its geometry using the tools we developed. We reveal that the training process explores an effectively low-dimensional manifold. We associate this low-dimensional manifold with the hyper-ribbon structure found in statistical physics.

In Chapter 3, we extend our techniques to the broader case of multitask learning and allow comparison of networks initially trained on different tasks. We show that the same low dimensionality can be observed in those scenarios as well. We also demonstrate how our methods could reveal structure in the space of multitask learning.

In Chapter 4, we connect the hyper-ribbon structure to sloppiness in data covariance structure and the Fisher Information Matrix of trained networks. We show that the input correlation matrix of typical classification datasets has a "sloppy" eigenspectrum where, after a sharp initial drop, a large number of small eigenvalues

are distributed uniformly over an exponentially large range. This structure is mirrored in the Hessian of a trained network, which allows us to compute non-vacuous generalization bounds via PAC-Bayes.

In Chapter 5, we provide theoretical insights into training manifolds by analyzing the gradient descent trajectories of linear models. We identify the key factors determining the dimensionality of the training manifold and characterize conditions under which they are provably low-dimensional. We show how our analysis can be extended to variants like stochastic gradient descent and kernel methods.

We conclude the thesis with a discussion on possible future directions in Chapter 6, focusing on what our observations could imply for practical trainingg.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

Deep neural networks have revolutionized various domains by their remarkable ability to learn complex functions efficiently. The underlying mechanisms driving this efficiency, however, remain largely elusive. The classical theory suggests the opposite: the highly non-convex optimization objective should be difficult to solve, and the huge capacity of the hypothesis class should lead to severe overfitting on the training data. However, neither of the above happens in practice; on the contrary, basic optimizers like stochastic gradient descent (SGD) typically solve the training problem efficiently, and the solutions generalize well to test data.

There have been numerous prior studies devoted to answering these questions, proposing theories like the loss landscape is "nice" (e.g., contains no spurious local minima), the optimization landscape is "nice" (e.g., almost convex and smooth) in a neighborhood near initialization, the optimization algorithms are biased towards "simpler" solutions that generalize better (e.g., min-norm solutions, flatter minima). A more detailed review of previous works can be found in Section 1.1.

In this thesis, we would like to focus on understanding the learning process of different networks by looking at the "model manifold" explored during training. The concept of "model manifold" is adapted from the statistical manifold in Information Geometry (Amari, 2016), where models are viewed as parameterized probability distributions living on a manifold $\mathcal{M} = \{p(\cdot|\theta), \theta \in \Theta\}$, where Θ is a set of parameters. The learning process can then be viewed as searching on the manifold to find a minimizer of a functional, typically a distance to a true data distribution. Information geometry has been useful in understanding the geometry of probabilistic models (Amari, 2016), improving optimization (Amari, 1998b) and providing insights into generalization (Dherin et al., 2021; Sun and Nielsen, 2023).

This function space perspective is attractive because it allows us to circumvent the complex nonlinear map from the parameter space to the function space and the pathological singular geometry in the parameter space. Moreover, this view allows us to compare very different models (in particular, models with different architectures) as long as they are trained to perform the same set of tasks, and they allow us to emphasize the influence of tasks on shaping the training dynamics of different networks.

The model manifold we study in this thesis is inspired by the formulation in (Transtrum et al., 2011b) and can be viewed as a subset of the classical statistical manifold. We restrict the study to predictions on a finite set of data points and the models explored during training. This finite-dimensional ambient space allows us to conduct more global analysis of the model manifold, which is impossible to do in the infinite-dimensional space of the classical case.

The central theme of the thesis is to reveal that the learning process of neural networks on typical tasks possesses a low-dimensional subspace of the full function space. We provide empirical evidence and theoretical insights demonstrating this low-dimensional structure and propose that it plays a key role in the success of deep learning. We further hypothesize that the inherent structure of the tasks they are trained on facilitates this low-dimensionality and hope a deeper understanding of this low-dimensionality could lead to guiding principles for more efficient training algorithms and enhanced generalization performance.

1.1. Related Work

1.1.1. Early work on optimizing deep networks

Understanding why deep neural networks can be optimized efficiently despite their non-convex nature has been a central question in deep learning theory. One early line of thought was to study the loss landscape and identify conditions under which the non-convex objectives can be solved by local search algorithms like gradient descent. For example, for the set of functions such that all local minima are also global, and the Hessian of every saddle point has a negative eigenvalue, gradient descent can find a global minimum (Jin et al., 2017; Du et al., 2017). Many such objectives exist in non-convex optimization, e.g. matrix factorization, tensor decomposition, also certain forms of ReLU-activated neural networks (Ge et al., 2015, 2018; Haeffele and Vidal, 2015; Kawaguchi and Kaelbling, 2020; Hardt and Ma, 2018). Unfortunately, in the more general case of learning the landscape could have spurious local minima (Kawaguchi, 2016; Safran and Shamir, 2018; Liu et al., 2019). Also, loss landscape is a rather static perspective that does not take into consideration the effect of initialization, optimizer (Sutskever et al., 2013).

The limitation of the loss landscape perspective inspired researchers to shift to a more focused view of the optimization problem: the trajectory-based analysis. This has been successful in revealing interesting dynamics in deep linear networks (Saxe et al.; Arora et al., 2019a,b) but extending it to nonlinear networks

has proven challenging, with major difficulties lying in the analysis of the complex nonlinear map from parameter space to the function space. Moreover, since different networks have totally different weight spaces, networks with distinct architectures need separate treatment and cannot be compared with each other under this framework. While dynamics in the weight space seem to be a more intuitive approach since the gradient descent algorithm directly modifies the weights, these difficulties suggest that it may be beneficial to bypass the parameterization map and study the trajectories directly in the function space One of the foundational works in this direction is the Neural Tangent Kernel (NTK) theory (Jacot et al., 2018b; Soltanolkotabi et al., 2019). The NTK approach decomposes the parameterization map and the functional cost (which is often convex, e.g., for mean squared error or cross-entropy) and shows that under suitable initialization and parameterization, dynamics of wide neural networks under gradient descent follow a kernel gradient descent in the function space. This allows further results on the convergence of gradient descent for wide neural networks in the NTK regime (Du and Zhai, 2019; Li and Liang, 2019; Arora et al., 2019c).

However, convergence results under the NTK regime require the dynamics to stay in a local region around initialization so that the features do not change significantly (so-called lazy training) (Woodworth et al., 2020; Chizat et al., 2019b). The lack of feature learning in the NTK regime contradicts the prevailing belief that the success of deep learning comes from its ability to extract useful features from data, and there has been empirical and theoretical evidence of gaps in generalizability or sample complexity between neural networks and their corresponding kernels (Arora et al.; Wei et al., 2019; Ghorbani et al., 2019b, 2020; Damian et al., 2022; Mei et al., 2022; Allen-Zhu et al., 2019).

There have been some recent works trying to go beyond NTK and understand the convergence in the feature learning regime. Most of those analysis modifies the optimizer to explicitly separate the feature learning and convergence stages in the learning process (e.g.training in multiple stages (Damian et al., 2022; Abbe et al., 2022) or adding NTK-related regularizers (Nichani et al., 2022)) and so do not fully explain the success in practice. The fact that NTK finds a global minimizer of the loss function and yet still lacks generalizability also leads to the study of the implicit bias of optimization algorithm, showing that GD or SGD are biased towards "simpler" solutions, characterized by minimum norm (Woodworth et al., 2020; Gunasekar et al., 2018), lower-order polynomial (Abbe et al., 2023; Kalimeris et al., 2019), flatter minima (Liu et al., 2023;

Damian et al., 2023), maximum margin solutions (Ji and Telgarsky, 2020; Lyu and Li, 2019).

1.1.2. Model Manifold

Our work draws significant inspiration from the function space perspective used in the NTK approach. We would like to characterize the space explored by typical training dynamics in the function space, going beyond the neighborhood-around initialization.

We achieve this by looking at the model manifold introduced in (Transtrum et al., 2011b), where they studied a nonlinear least squares problem given a dataset $\{(x_i, y_i)\}_{i=1}^N$. A real-valued function f_θ can be identified with its outputs on these data points, corresponding to a point in $\mathbb{R}^N$: $\boldsymbol{f}_\theta = [f_\theta(x_1), \ldots, f_\theta(x_N)]$. As θ is varied over a region in $\mathbb{R}^d$, $\boldsymbol{f}_\theta$ spans a region $\mathbb{R}^N$ that is at most d-dimensional, and the process of finding the best fit of the least squares problem corresponds to finding a point within this region that is closest (in Euclidean distance) to the target $\boldsymbol{y} = [y_1, \ldots, y_N] \in \mathbb{R}^N$.

For function classes that are universal approximators, the predictions on a set of distinct inputs can, in principle, fill the entire output space. However, when we impose constraints on the parameters or inputs, the resulting model manifold could exhibit a more interesting geometry—referred to as a hyper-ribbon by (Transtrum et al., 2011b), where the cross-sectional widths decay exponentially across dimensions. As an example, the model manifold of an analytic function, when evaluated on data from a bounded interval, lies within a hyperellipsoid whose widths decay geometrically (Quinn et al., 2019b). This kind of sloppiness is frequently seen in many multi-parameter models in biology, physics, and other domains (Transtrum et al., 2009, 2011a). In our work, we focus on the subset of neural networks explored during the standard training process, which naturally introduces such constraints.

Notice the model manifold is closely related to Information Geometry (Amari, 2016). For a set $\mathcal{X}$, consider a family of probability distributions on $\mathcal{X}$ that can be parameterized by θ: $S = \{p_\theta, \theta \in \Theta \subset \mathbb{R}^d\}$, where

$$p_\theta \in \mathcal{P}(\mathcal{X}) \coloneqq \{p : \mathcal{X} \to \mathbb{R} : p(x) \geq 0, \forall x \in \mathcal{X}, \int_{\mathcal{X}} p(x)dx = 1\},$$

When the map $\theta \mapsto p_\theta$ is smooth and injective, S is a smooth manifold with local coordinates given by θ. It can be shown (Amari, 2016) that there is a unique (up to scaling) Riemannian metric on this manifold (i.e.,

Fisher Information Metric) given by

$$g_\theta = \mathbb{E}_x \mathbb{E}_{y \sim p_\theta(\cdot|x)} \left[\left(\partial_\theta \log p_\theta(y|x) \right) \left(\partial_\theta \log p_\theta(y|x) \right)^\top \right].$$

Notice this manifold lives in the space of all probability distributions on $\mathcal{X}$, which is typically infinite-dimensional, even though the manifold is finite-dimensional. The model manifold we study is locally isometric to the statistical model manifold equipped with the Fisher information metric (see Section 2.5.3.1), but it lives in a finite-dimensional space because we restrict our investigations to the predictions on a finite set of samples. We will see in Chapter 2 that the finite-dimensional nature of the ambient space fundamentally enables us to perform complicated computations such as embeddings of high-dimensional models, geodesics in these spaces, projections of a model onto the geodesic, distances between trajectories in the prediction space, etc.

1.2. Statement of Contributions

In Chapter 2, we introduce the concept of model manifold and tools from information geometry to analyze the manifold. By examining the underlying high-dimensional probabilistic models, we reveal that the training process explores an effectively low-dimensional manifold. Networks with a wide range of architectures and sizes, trained using different optimization methods, regularization techniques, data augmentation techniques, and weight initializations, lie on the same manifold in the prediction space. We study the details of this manifold to find that networks with different architectures follow distinguishable trajectories, but other factors have a minimal influence; larger networks train along a similar manifold as that of smaller networks, just faster; and networks initialized at very different parts of the prediction space converge to the solution along a similar manifold.

In Chapter 3, we explore a similar phenomenon happening in other learning systems like transfer, meta, semi-, and self-supervised learning as they learn different sets of tasks. We found that the model manifold is, again, effectively low-dimensional. This observation suggests that different tasks may also have a very strong shared structure, and our method is capable of uncovering it. We also study the behavior of different representation learning algorithms and find that (a) episodic meta-learning algorithms and supervised learning traverse different trajectories during training, but they fit similar models eventually, (b) contrastive and

semi-supervised learning methods traverse trajectories similar to those of supervised learning, etc.

In Chapter 4, we investigate possible explanations for the low dimensionality observed in Chapter 2. Inspired by prior studies on "hyperribbons" (Transtrum et al., 2011b), we study the eigenspectrum of the Fisher Information Matrix of trained networks and find they have a "sloppy" eigenspectrum, where, after a sharp initial drop, a large number of small eigenvalues are distributed uniformly over an exponentially large range. We show this sloppiness may be coming from the data covariance structure, which tends to be sloppy across common tasks in vision, language, and audio. We also show that this sloppy structure allows us to compute non-vacuous generalization bounds analytically using PAC-Bayesian theory, providing supporting evidence to the hypothesis that the sloppiness of inputs aids generalization in deep networks.

In Chapter 5, we study the model manifold explored by running gradient descent on a linear regression problem. We identified three factors determining the effective dimensionality of this model manifold: the training time, data sloppiness, and the ratio of initialization and the truth. These findings corroborate the synthetic data experiments and the conjectures proposed in the appendix of Chapter 2. We then extend the analysis to some variants of gradient descent on linear models, including stochastic gradient descent and kernel methods. We end the chapter with a discussion on the implications of this analysis for practice.

In Chapter 6, we discuss directions for future work.

The material presented in this thesis comes from the following papers:

1. Jialin Mao, Itay Griniasty, Han Kheng Teoh, Rahul Ramesh, Rubing Yang, Mark K. Transtrum, James P. Sethna, and Pratik Chaudhari. The training process of many deep networks explores the same low-dimensional manifold. Proceedings of the National Academy of Sciences, 121(12):e2310002121, March 2024. doi: 10.1073/pnas.2310002121.

2. Rubing Yang, Jialin Mao, and Pratik Chaudhari. Does the data induce capacity control in deep learning? In Proc. of International Conference of Machine Learning (ICML), 2022.

3. Rahul Ramesh, Jialin Mao, Itay Griniasty, Rubing Yang, Han Kheng Teoh, Mark K. Transtrum, James P. Sethna, and Pratik Chaudhari. A picture of the space of typical learnable tasks. In Proceedings of

the 40th International Conference on Machine Learning, volume 202 of ICML'23, pages 28680–28700. JMLR.org, July 2023.

4. Jialin Mao, Itay Griniasty, Yan Sun, Edgar Dobriban, Mark K. Transtrum, James P. Sethna, and Pratik Chaudhari. An Analytical Characterization of Sloppiness in Neural Networks: Insights from Linear Models. *In preparation*.

CHAPTER 2

THE MANIFOLD INDUCED BY THE TRAINING PROCESS OF NEURAL NETWORKS

2.1. Introduction

In this chapter, we introduce tools for analyzing and visualizing the trajectories of neural network training in their prediction space. The key idea is to analyze the probabilistic model underlying deep neural networks via their representation as probabilistic models as they are trained to classify images. In Section 2.2, we develop techniques to analyze such high-dimensional probabilistic models and embed these models into lower-dimensional spaces for visualization.

In Section 2.3, we first show, using experimental data (with $NC \sim 10^6 - 10^8$), that the training process explores an effectively low-dimensional manifold in the prediction space. The top three dimensions in our embedding explain 76% of the "stress" (which is a quantity used to characterize how well the embedding preserves pairwise distances) between probability distributions of about 150,000 different models with many different architectures, sizes, optimization methods, regularization mechanisms, data augmentation techniques, and weight initializations. In spite of this huge diversity in configurations, the probabilistic models underlying these networks lie on the same manifold in the prediction space. This sheds new light upon a key open question in deep learning, namely, how can training a deep network, with many millions of weights, on datasets with millions of samples, using a non-convex objective, be feasible?

Next, we study the details of the structure of this manifold. We find that networks with different architectures have distinguishable trajectories in the prediction space; in contrast, details of the optimization method and regularization technique do not change the trajectories in the prediction space much. We find that a larger network trains along a similar manifold as that of a smaller network with a similar architecture but it makes more progress for the same number of gradient updates.

In Section 2.4, we discuss our findings in greater detail and propose possible explanations for the observed low dimensionality. We continue in Section 2.5 to provide additional experimental details and discuss variations of methods presented earlier (e.g., different intensive distances, embeddings using subsets of samples) and

include further experiments exploring the possible explanations proposed in Section 2.4.

2.2. Methods

To aid the reader, Section 2.5.1 collects all the notation in one place.

Models as probability distributions Consider a dataset $\{(x_n, y_n^*)\}_{n=1}^N$ of N samples, each of which consists of an input x_n and its corresponding ground-truth label $y_n^* \in \{1, \dots, C\}$ where C is the number of classes. Let $\boldsymbol{y} = (y_1, \dots, y_N) \in \{1, \dots, C\}^N$ denote any sequence of outputs. If samples in the dataset are independent and identically distributed, then the joint probability of the predictions can be modeled as

$$P_w(\boldsymbol{y}) = \prod_{n=1}^{N} p_w^n(y_n) \tag{2.2.1}$$

where w are the parameters of the network and we have used the shorthand $p_w^n(y_n) \equiv p_w(y_n \,|\, x_n)$. The quantity in Equation (2.2.1) is the joint likelihood of all the labels given the weights. Observe that

$$\begin{aligned}
P_w(\boldsymbol{y}) &\equiv p(\{(x_n, y_n)\}_{n=1}^N ; w) \\
&= p(x_1, \dots, x_N)\, p_w(y_1, \dots, y_N \,|\, x_1, \dots, x_N) \\
&\overset{\text{(a)}}{=} p(x_1, \dots, x_N) \prod_{n=1}^{N} p_w(y_n \,|\, x_1, \dots, x_N) \\
&\overset{\text{(b)}}{=} p(x_1, \dots, x_N) \prod_{n=1}^{N} p_w(y_n \,|\, x_n) \\
&\overset{\text{(c)}}{=} \left(\frac{1}{N} \sum_{n=1}^{N} \delta_{x_n}(x_n) \right) \prod_{n=1}^{N} p_w(y_n \,|\, x_n) \\
&= \prod_{n=1}^{N} p_w(y_n \,|\, x_n).
\end{aligned}$$

In this calculation, we have used the assumption that (a) predictions on two samples are independent of each other *given the weights and the input samples* (if we marginalize on the weights, they are certainly dependent), (b) we are performing inductive inference, i.e., $p(y_n \,|\, x_1, \dots, x_n) = p(y_n \,|\, x_n)$, and (c) the samples are frozen to the ones in the training set for the analysis, i.e., the distribution $p(x_1, \dots, x_n) \equiv \left(\frac{1}{N} \sum_{n=1}^{N} \delta_{x_n}(x_n) \right) = 1$. So we actually do not need to use the assumption that the training samples $x_1, \dots, x_N$ are independent of each other to write down the joint likelihood that factorizes over the samples in the training set. Certainly, if

the training samples are independent, then this derivation also holds. Let us note that training samples being independent of each other is one of the most common assumptions in machine learning. This assumption is used to derive, for instance, the maximum likelihood estimator in (Bishop et al., 1995, Equation 1.61).

The probability distribution in Equation (2.2.1) is $N(C-1)$-dimensional object. Any network that makes predictions on the same set of samples—irrespective of its architecture, the optimization algorithm and regularization techniques that were used to train it—can be analyzed as a probabilistic model in this same $N(C-1)$-dimensional space; we will refer to this space as the "prediction space".

Measuring distances in the prediction space We first mark two special points in the prediction space that we will refer to frequently. The true probabilistic model of the data which corresponds to ground-truth labels is denoted by $P_* = \delta_{\boldsymbol{y}^*}(\boldsymbol{y})$ where $\boldsymbol{y}^*$ are ground-truth labels and δ is the Kronecker delta function. We will call this the "truth". Similarly, we will mark a point called "ignorance": it is a probability distribution P_0 that predicts $p_0^n(c) = 1/C$ for all samples n and classes c. Given two probabilistic models P_u and P_v with weights u and v respectively, the Bhattacharyya distance per sample between them can be derived as follows (note that $\boldsymbol{y}$ can take a total of C^N distinct values, and each $y_n \in \{1, \ldots, C\}$):

$$
\begin{aligned}
\mathrm{d_B}(P_u, P_v) &\doteq -\frac{1}{N} \log \sum_{\boldsymbol{y}} \sqrt{P_u(\boldsymbol{y})} \sqrt{P_v(\boldsymbol{y})} \\
&= -\frac{1}{N} \log \sum_{\boldsymbol{y}} \prod_{n=1}^{N} \sqrt{p_u^n(y_n)} \sqrt{p_v^n(y_n)} \\
&= -\frac{1}{N} \log \sum_{y_1} \cdots \sum_{y_{N-1}} \prod_{n=1}^{N-1} \sqrt{p_u^n(y_n)} \sqrt{p_v^n(y_n)} \left(\sum_{y_N} \sqrt{p_u^N(y_n)} \sqrt{p_v^N(y_n)} \right) \\
&\vdots \\
&= -\frac{1}{N} \log \prod_{n=1}^{N} \sum_{c} \sqrt{p_u^n(c)} \sqrt{p_v^n(c)} \\
&= -\frac{1}{N} \sum_{n} \log \sum_{c} \sqrt{p_u^n(c)} \sqrt{p_v^n(c)}.
\end{aligned}
\tag{2.2.2}
$$

Calculations like the one above hold in general, the joint entropy of two independent random variables is the sum of their individual entropy. Just like the familiar cross-entropy loss used for training deep networks is an average over the samples, the Bhattacharyya distance is also an average over the training samples.

In other words, the Bhattacharyya distance between two probabilistic models can be written as the average of the Bhattacharyya distances of their predictive distributions p_u^n and p_v^n on each input x_n. We can also use other distances to measure the discrepancy between P_u and P_v, such as the symmetrized Kullback-Leibler divergence (Teoh et al., 2020), or the geodesic distance on the product space. But many other distances (e.g., the Hellinger distance $\mathrm{d_H}(P_w, P_*) = 2\left(1 - \prod_n \sum_c \sqrt{p_w^n(c)}\sqrt{p_*^n(c)}\right)$) saturate quickly as the number of dimensions of the probability distribution grows, obscuring the intrinsic low-dimensional structures we seek. This is because two high-dimensional random vectors are orthogonal with high probability. When the number of samples N is large, distances such as the Bhattacharyya distance are better behaved due to their logarithms.

Measuring distances between trajectories in the prediction space Consider a trajectory $(u(k))_{k=0,\dots,T}$ in the weight space that is initialized at $u(0)$ and records the weights after each update made by the optimization method during training. This corresponds to a trajectory $\tilde{\tau}_u = (P_{u(k)})_{k=0,\dots,T}$ in the prediction space. We are interested in distances between trajectories in the prediction space. Different networks (depending upon the initialization, architecture, and the training procedure) train at different speeds and make different amounts of progress towards P^* after each epoch. This makes it problematic to simply use a distance like $\sum_k \mathrm{d_B}(P_{u(k)}, P_{v(k)})$ which sums up the distances between models at each instant k. To see why, observe that such a distance between $\tilde{\tau}_u$ and $\tilde{\tau}_v := (u(0), u(2), u(4), \dots, u(2k), u(2k+2), \dots)$ which progresses twice as fast as $\tilde{\tau}_u$, is non-zero even if the two trajectories are intrinsically the same.

To better compare trajectories, we need a notion of time that allows us to index any trajectory in prediction space. We shall measure progress along the trajectory by the projection onto the geodesic between ignorance and truth. Geodesics are locally length-minimizing curves in a metric space. Our trajectories evolve on the product manifold of the individual probability distributions in Equation (2.2.1). Geodesics in this space using the Fisher Information Metric (FIM) (Amari, 2016) are a good candidate for constructing our index. The FIM is realized by a simple embedding. For each n, consider a vector consisting of the square-root of the probabilities $(\sqrt{p_u^n(c)})_{c=1,\dots,C}$ as a point on a $(C-1)$-dimensional sphere. Therefore the geodesic connecting two probability distributions P_u and P_v is the great circle on the sphere. A point along it with interpolation parameter $\alpha \in [0,1]$ denoted by $P_{u,v}^\alpha(\boldsymbol{y}) = \prod_n p_{u,v}^{n,\alpha}(y_n)$ satisfies (Ito and Dechant, 2020,

Eq. 47)

$$\sqrt{p^{n,\alpha}_{u,v}} = \frac{\sin\left((1-\alpha)\mathrm{d}^n_{\mathrm{G}}\right)}{\sin\left(\mathrm{d}^n_{\mathrm{G}}\right)}\sqrt{p^n_u} + \frac{\sin\left(\alpha\mathrm{d}^n_{\mathrm{G}}\right)}{\sin\left(\mathrm{d}^n_{\mathrm{G}}\right)}\sqrt{p^n_v}; \tag{2.2.3}$$

where $\mathrm{d}^n_{\mathrm{G}} = \cos^{-1}\left(\sum_c \sqrt{p^n_u(c)}\sqrt{p^n_v(c)}\right)$ is one half of the great circle distance between $p^n_u(\cdot)$ and $p^n_v(\cdot)$. Any point P_w along a trajectory can be reindexed using "progress" that is defined as

$$s_w = \operatorname*{argmin}_{\alpha\in[0,1]} \mathrm{d}_{\mathrm{G}}(P_w, P^\alpha_{0,*}), \tag{2.2.4}$$

where

$$\mathrm{d}_{\mathrm{G}}(P_u, P_v) = N^{-1}\sum_n \cos^{-1}\sum_c \sqrt{p^n_u(c)}\sqrt{p^n_v(c)}$$

is the geodesic distance on the product manifold. Note that progress $s_w \in [0,1]$ and it intuitively quantifies the motion along the trajectory by projecting onto the geodesic connecting ignorance and truth.

Remark 2.2.1. (Relationship between progress and error) Progress is related to the error but they are not the same. Suppose we have a model P that predicts very confidently, i.e., $p^n(c) \in \{0,1\}$ for all $c \in \{1,\ldots,C\}$ and all samples n. The progress of this model is given by

$$\begin{aligned}\alpha^* &= \operatorname*{argmin}_{\alpha\in[0,1]} \mathrm{d}_{\mathrm{G}}(P, P^\alpha_{0,*}) \\ &= (1-\epsilon)\cos^{-1}\left(\frac{\sin((1-\alpha)\mathrm{d}^n_{\mathrm{G}})}{\sin(\mathrm{d}^n_{\mathrm{G}})}\cos(\mathrm{d}^n_{\mathrm{G}}) + \frac{\sin(\alpha\mathrm{d}^n_{\mathrm{G}})}{\sin(\mathrm{d}^n_{\mathrm{G}})}\right) + \epsilon\cos^{-1}\left(\frac{\sin((1-\alpha)\mathrm{d}^n_{\mathrm{G}})}{\sin(\mathrm{d}^n_{\mathrm{G}})}\cos(\mathrm{d}^n_{\mathrm{G}})\right)\end{aligned}$$

where $\epsilon = N^{-1}\sum_n \mathbf{1}\{\mathrm{argmax}_c\, p^n_w(c) \neq y^*_n\}$ is the fraction of errors made by the model on the N samples and $\mathrm{d}^n_{\mathrm{G}} = \cos^{-1}(1/\sqrt{C})$ if there are C classes. We can show that if $\epsilon < 1 - 1/\sqrt{C}$, then the progress $\alpha^* = 1$. This suggests that progress and error are not directly analogous to each other: models with high progress do not necessarily have small errors. In practice, if the number of samples N is small and the number of classes is large, then we will find instances of models with high progress and high error. This is not often the case in our experiments for the training data, but we do see very high progress for some models on the test data (see Figure 2.7).

To find a point's progress we solve Equation (2.2.4) using a bisection search (Brent, 1971).

We would now like to convert each trajectory $\tilde{\tau}_u = (P_{u(k)})_{k=0,\ldots,T}$ into a continuous curve $\tau_u = (P_{u(s)})_{s\in[0,1]}$ and uniformly sample them for values of s between $[0, 1]$. To do this, we first calculate the progress $s_{u(k)}$ of all checkpoints along the trajectory $\tilde{\tau}_u$ using Equation (2.2.4). For any $s \in [s_{u(k)}, s_{u(k+1)}]$, we can now define $\alpha = (s - s_{u(k)})/(s_{u(k+1)} - s_{u(k)})$ and calculate (using Equation (2.2.3)) the geodesically-interpolated probability distribution $P^{\alpha}_{u(k),u(k+1)}$ that corresponds to this progress s on the trajectory of interest $\tilde{\tau}_u$. Finally, we define the distance between trajectories τ_u and τ_v as

$$\mathrm{d}_{\mathrm{traj}}(\tau_u, \tau_v) = \int_0^1 \mathrm{d}_{\mathrm{B}}(P_{u(s)}, P_{v(s)})\mathrm{d}s, \tag{2.2.5}$$

which compares points on the trajectories at equal progress.

Embedding predictions into a lower-dimensional space for visualization We use a technique called intensive principal component analysis (InPCA) (Quinn et al., 2019a; Teoh et al., 2020) which is closely related to multi-dimensional scaling (MDS (Cox and Cox, 2008)) to project the predictions of the network into a lower-dimensional space to visually inspect their training trajectories. For m probability distributions, consider a matrix $D \in \mathbb{R}^{m\times m}$ with entries $D_{uv} = \mathrm{d}_{\mathrm{B}}(P_u, P_v)$ and

$$W = -LDL/2 \tag{2.2.6}$$

where $L_{uv} = \delta_{uv} - 1/m$, and W is the centered version of D. An eigen-decomposition of $W = U\Lambda U^{\top}$ where the eigenvalues are sorted in descending order of their magnitudes $|\Lambda_{00}| \geq |\Lambda_{11}| \geq \ldots$ allows us to compute the embedding of the m probability distributions into an m-dimensional Minkowski space with metric signature $(p, m - p)$ derived from the p positive eigenvalues of W as $\mathbb{R}^{p,m-p} \ni X = U\sqrt{\Lambda}$[1]. In standard PCA, the embedding is always Euclidean since the eigenvalues of W are guaranteed to be non-negative. However, InPCA can have both positive and negative eigenvalues. Coordinates corresponding to positive eigenvalues are analogous to "space-like" components in special relativity that have a positive-squared contribution to the distance between two points. Coordinates corresponding to negative eigenvalues

[1]In special relativity, the axes corresponding to negative eigenvalues are often referred to as imaginary coordinates, and the metric signature is replaced by $(x, it)\cdot(x, it) = x^2 + i^2t^2 = x^2 - t^2$. However, this is not the inner product $\|(x, it)\| = x^2 + t^2$ over the complex numbers. We define a space where the distance between "$(1, i)$" and the origin vanishes and therefore its embedding is $\mathbb{R}^{p,m-p}$ and not $\mathbb{C}^m$.

are "time-like" components in that they have a negative contribution to the distance between two points. One can think of the coordinates with negative eigenvalues as being imaginary axes in the embedding. Space-like and time-like coordinates can give rise to "light-like" directions along which the distance between two visually different points is zero.

The key property of InPCA that we exploit in this paper is that its embedding is isometric, i.e.,

$$\|X_u - X_v\|^2 = \mathrm{d_B}(P_u, P_v) \geq 0 \tag{2.2.7}$$

for embeddings $X_u, X_v \in \mathbb{R}^{p,m-p}$ of two probability distributions P_u and P_v and the norm in Minkowski space is

$$\|X_u - X_v\|^2 = \sum_{k=1}^{m} \mathrm{sign}(\Lambda_{kk}) |X_{uk} - X_{vk}|^2,$$

see Section 2.5.3.1 for a proof. Like PCA, InPCA generates an optimal embedding of a geometrical object with a fixed number of points, preserving long-distance structures. Such an isometric embedding is different from the one created by methods like t-SNE (Van der Maaten and Hinton, 2008) or UMAP (McInnes et al., 2018) which approximately preserves local pairwise distances but distorts the global geometry. All the analysis in this paper is conducted using the full pairwise Bhattacharyya distance matrix D. In contrast with t-SNE or UMAP, the isometric embedding in InPCA ensures that the visualization is consistent with our conclusions (up to the fact that we only visualize the top few dimensions). For a $d < m$ dimensional InPCA embedding, the fraction of the centered pairwise distance matrix W that is preserved is

$$1 - \sqrt{\frac{\sum_{ij}\left(W_{ij} - \sum_{k=1}^{d} \sqrt{\Lambda_{kk}} U_{ik} \sqrt{\Lambda_{kk}} U_{kj}\right)^2}{\sum_{ij} W_{ij}^2}} = 1 - \sqrt{\frac{\sum_{k=d+1}^{m} \Lambda_{kk}^2}{\sum_i \Lambda_{ii}^2}}; \tag{2.2.8}$$

which is similar to the explained variance for standard PCA. Following the MDS literature, we call this quantity "explained stress". In this paper, we embed predictions of $m \sim 10^3$–10^5 models with $NC \sim 10^6$–10^8 using InPCA. This is very challenging computationally. Implementing InPCA—or even PCA—for such large matrices requires a large amount of memory. We reduced the severity of this issue using Numpy's memmap functionality. Note that calculating only the top few eigenvectors of Equation (2.2.6) by magnitude suffices for the purpose of visualization.

Adding new networks into an existing embedding Given the embedding of predictions of m networks, we can project the prediction of a new network into the same space. Observe that we can rewrite Equation (2.2.6) to be

$$W_{uv} = -\frac{\mathrm{d_B}(P_u, P_v)}{2} + \frac{1}{2m}\sum_{u'}\left(\mathrm{d_B}(P_u, P_{u'}) + \mathrm{d_B}(P_v, P_{u'}) - \frac{1}{m}\sum_{v'}\mathrm{d_B}(P_{u'}, P_{v'})\right); \tag{2.2.9}$$

where $u', v' \in \{1, \ldots, m\}$. The embedding of a new probability distribution P_w into this space is $X_w = \sum_{u=1}^{n} W_{w,u} U_u |\Lambda_{uu}|^{-1/2}$; where U_u denotes the u^{th} column of U. This is equivalent to a triangulation of the position of the added points, such that distances and the overall geometry are preserved. Although we do not do so in this paper, this procedure can also be used to embed a large set of points by computing the eigen-decomposition for only a subset, e.g., as done in De Silva and Tenenbaum (2004).

Computing averages in the prediction space For our analysis, we will need to compute averages of the predictions of probabilistic models, e.g., of the same architecture but trained from different initializations. Depending upon what distance we use in the prediction space, there can be different ways to compute such an average. The most natural candidate is the Bhattacharyya centroid of a set of m probability distributions $\{P_i\}_{i=1}^{m}$ given by $\operatorname{argmin}_{P_w} m^{-1}\sum_i \mathrm{d_B}(P_i, P_w)$ (Nielsen and Boltz, 2011). In this paper, we will need to compute such averages thousands of times. For computational convenience, we will instead use the arithmetic mean of the probabilities $m^{-1}\sum_u p_u^n(c)$ for all n, c as our average, which we have found to produce similar results in preliminary experiments, which discusses the effect of different kinds of averaging. We have found that the harmonic mean of an ensemble of probabilistic models performs slightly better on the test data in comparison to their arithmetic mean, which is commonly used in machine learning.

2.3. Results

[1]For CIFAR-10, some configurations had models that did not get to zero train error, and in very few cases, models had 90% train error. For ImageNet, all networks were trained with standard data augmentation techniques and they do not reach zero training error.

Table 2.1: Median (and 25–75 percentile on the second row) train and test error (%) of different architectures (with number of parameters in the brackets) used in our analysis, averaged over different optimization methods, regularization techniques and weight initializations.

	CIFAR-10					
	Fully-Connected (3.8M)	AllCNN (0.4M)	Small ResNet (0.3M)	Large ResNet (43.9M)	ConvMixer (0.6M)	ViT (9.5M)
Train Error	1.5	0.1	0.6	0.0	0.0	0.3
	(0.0, 4.4)	(0.0, 0.5)	(0.0, 2.3)	(0.0, 0.0)	(0.0, 0.0)	(0.0, 18.6)
Test Error	39.7	15.4	17.6	9.6	11.7	32.7
	(38.1, 41.9)	(11.7, 20.3)	(12.5, 21.5)	(6.5, 11.2)	(9.9, 16.8)	(21.7, 36.2)

	ImageNet		
	ResNet-18 (11.6M)	ResNet-50 (25.6M)	ViT-S (22M)
Train Error	22.7	15.8	16.6
	(22.5, 22.7)	(15.8, 15.8)	(15.1, 16.9)
Test Error	31.9	25.2	41.5
	(31.8, 31.9)	(25.1, 25.3)	(41.3, 42.2)

Experimental Data [2]

We trained 2,296 different configurations on the CIFAR-10 dataset (Krizhevsky, 2009) corresponding to networks [3] with different (a) network architectures (fully-connected, convolutional: AllCNN (Springenberg et al., 2015), residual: Wide ResNet (Zagoruyko and Komodakis, 2016), and ConvMixer (Trockman and Kolter, 2022), self-attention-based: ViT (Dosovitskiy et al., 2020)), (b) network sizes (a small residual network and a large residual network), (c) optimization methods (SGD, SGD with Nesterov's acceleration and Adam (Kingma and Ba, 2015)), (d) hyper-parameters (learning rate and batch-size), (e) regularization mechanisms (with and without weight-decay (Ioffe and Szegedy, 2015)), (f) data augmentation (mean-standard deviation-based normalization, and another one where we add horizontal flips and random crops) and (g) random initializations of weights (using 10 different random seeds). We recorded the training trajectories at about 70 different points during training (more frequently at the beginning of training when the models train quickly). This gave us 151,407 different models, after removing some models that did not train correctly due to numerical overflows/underflows during gradient updates.

[2] Data, pre-processing scripts, and code are available at https://github.com/grasp-lyrl/low-dimensional-deepnets

[3] In the sequel, "network" denotes a particular configuration with a specific architecture, optimization method, regularization technique, hyper-parameter choice, data-augmentation, and weight initialization. "Model" denotes a probability distribution along the training trajectory of such a network.

We also performed a smaller scale experiment on ImageNet using (a) three different architectures (a small residual network: ResNet-18 (He et al., 2016), a larger residual network ResNet-50, and a self-attention-based network: ViT), (b) different optimization algorithms (SGD with Nesterov's acceleration for the residual networks, and a variant of Adam for ViT (Heo et al., 2021)), (c) 5 random weight initializations for the residual networks and 3 for the ViT. We recorded each training trajectory at 61 different points to obtain a total of 792 different models for ImageNet.

Table 2.1 summarizes the train and test errors of models used in our analysis. Section 2.5.2 gives more details of the training procedure. About 60,000 GPU hours were used to obtain and analyze the data in this paper.

The training process explores an effectively low-dimensional manifold in the prediction space

Figure 2.1a shows the first three dimensions of the InPCA embedding of the probabilistic model in Equation (2.2.1) computed over samples in the training set. Each point corresponds to one model (i.e., one architecture, optimization algorithm, hyper-parameters, regularization, weight initialization and a particular checkpoint along the training trajectory) and is colored by the architecture. The explained stress Equation (2.2.8) of the first three dimensions is 76% as shown in Figure 2.1b; it increases to 98% within the first 50 dimensions. The prediction space for CIFAR-10 has 4.5×10^5 dimensions ($N = 5 \times 10^4$ and $C = 10$); the rank of the distance matrix in InPCA is at most 151,407. For ImageNet, all networks are trained on the entire training set ($N = 1.28 \times 10^6$) but we use a subset of the training samples ($N = 50,000$) across $C = 10^3$ classes to calculate the embedding (i.e., the prediction space has 4.995×10^7 dimensions). For ImageNet, nearly 84% of the explained stress is captured by the top three components of the InPCA embedding Figure 2.1d; this increases to 96% in the top 50 dimensions. The fact that so few dimensions capture such a large fraction of the stress suggests that in spite of the huge diversity in the configurations of these networks, they all explore an effectively low-dimensional manifold in the prediction space during training.

Ignorance is marked by P_0. The truth P_* is off the edge of the plot (see Figure 2.2b). The black curve denotes the embedding of the geodesic between P_0 and P_* calculated using Equation (2.2.3). Typical weight initialization schemes initialize models near P_0 irrespective of the configuration. Towards the end of training,

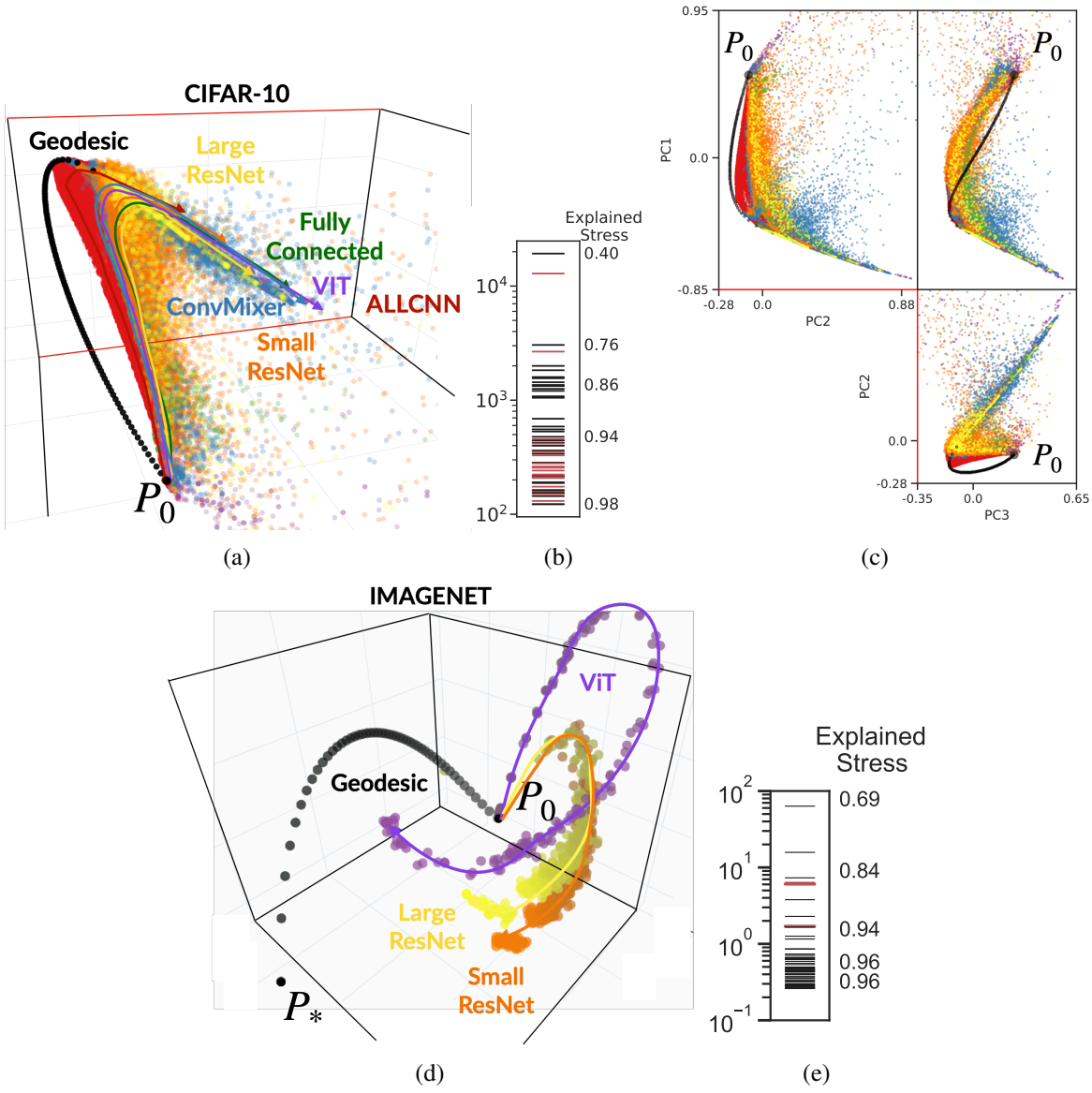


Figure 2.1: The manifold of models along training trajectories of networks with different configurations (architectures denoted by different colors, optimization algorithms, hyper-parameters, and regularization mechanisms) is effectively low-dimensional for **(a)** CIFAR-10, and **(d)** ImageNet. Different configurations train along similar trajectories but are quite different from the geodesic between ignorance P_0 and truth P_* (not seen here). The manifold is hyper-ribbon-like (Transtrum et al., 2011b): eigenvalues of the InPCA distance matrix Equation (2.2.6) for CIFAR-10 **(b)** and ImageNet **(e)** are spread over a large range with the top few dimensions capturing a large fraction of the stress Equation (2.2.8) (numbers indicate explained stress in the top 1, 3, 10, 25 and 50 dimensions). Time-like coordinates corresponding to negative InPCA eigenvalues are red. **(c)**: a pairwise comparison for the first three principal components, note that PC2 is time-like (same data as **(a)**). In **(a,d)**, we have drawn smooth curves denoting trajectories by hand to guide the reader.

models that trained well are close to the truth P_* in terms of the Bhattacharyya distance. Note that if the truth P_* has probabilities that are either zero or one (which is the case in our experiments), then the Bhattacharyya distance is one half of the cross-entropy loss used for classification. In this large prediction space, training trajectories of different configurations could be very diverse; on the contrary, not only do they all lie on an effectively low-dimensional manifold but trajectories of different configurations appear remarkably similar to each other. Sub-manifolds corresponding to each configuration seem to be rather similar; we will analyze this quantitatively in Figure 2.6a. For now, we note that probabilistic models learned by different architectures, training, and regularization methods, are very similar to each other—not only at the end of training when they fit the data but also along the entire training trajectory.

All trajectories seem to take a different path than the geodesic (shortest distance) path between P_0 and P_*. However, the geodesic is also largely captured by the top few dimensions of InPCA. Along the geodesic, all samples are trained towards the truth at the same rate, and so all models on it have zero training error. The deviation of paths away from the geodesic may reflect the learning of easy images early and confusing ones late, perhaps due to first-order gradient-based methods. We explore this further in Figures 2.17a and 2.17b. The geodesic corresponds to the trajectory of natural gradient descent (Amari, 1998a), which is not a first-order method. That the geodesic is faithfully represented in the low-dimensional embedding suggests that the low dimensionality observed in Figure 2.1a is not a direct consequence of using gradient-based algorithms.

All these observations also hold for networks trained on ImageNet. Note that in this case, the top three eigenvalues of InPCA are all positive; we have noticed this to be the case when the number of models embedded is small. The manifold of all trajectories is still effectively low-dimensional. Sub-manifolds spanned by ViTs and ResNets appear different from each other while sub-manifolds of the smaller and larger ResNet are quite similar; we will see in Figure 2.6a that architectures are the primary distinguishing factors of different training trajectories. In this case, all three architectures are quite different from the geodesic. Training trajectories do not end as close to truth P_* as those of CIFAR-10; for ImageNet, the trajectories end at a progress Equation (2.2.4) close to 0.9. This should not be surprising because typically networks trained on ImageNet do not achieve zero training error (zero training error can be achieved but they perform very poorly on the test data).

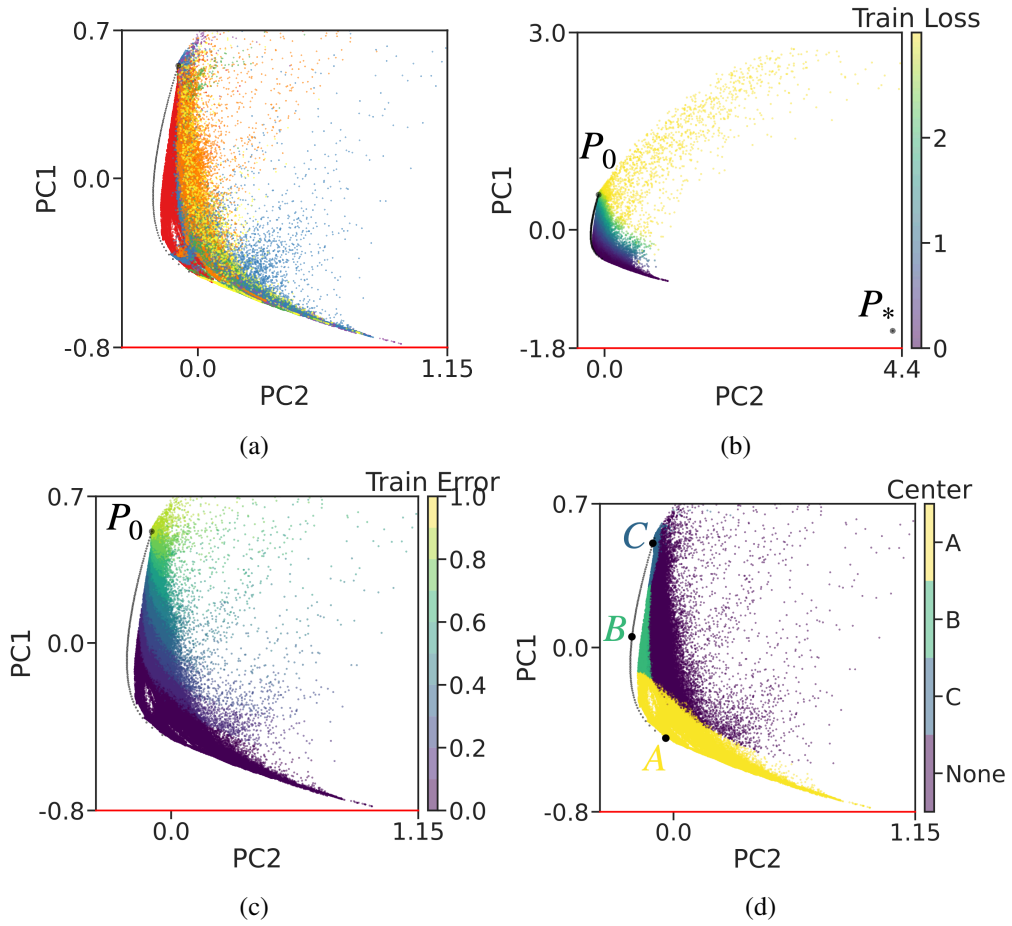
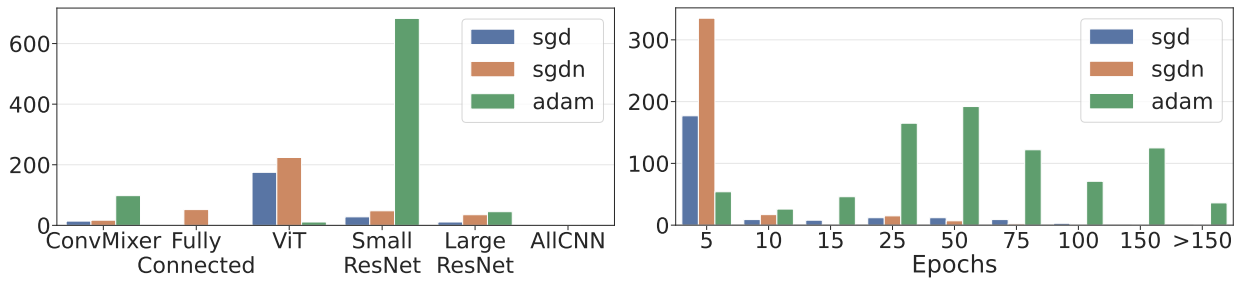


Figure 2.2: Comparison of the top two principal components of an InPCA embedding of all models on CIFAR-10 colored by the architectures **(a)** (same as Figure 2.1c), train loss **(b)**, which is two times the Bhattacharyya distance $\mathrm{d_B}(P, P_*)$ for classification tasks like ours, train error in **(c)**, and by whether they are within a Bhattacharyya distance < 0.15 from models marked A, B, and C on the geodesic in **(d)**. These figures are discussed in the narrative and should be studied together with Figure 2.1c.

Characterizing the details of the train manifold Figure 2.2a shows a pairwise comparison for the first three principal components of InPCA (same data as that of Figure 2.1a). Qualitatively, the first principal component, which is space-like, distinguishes models according to their distance to the truth P_* (i.e., half of the cross-entropy loss). The second principal component, however, is time-like because the second eigenvalue of InPCA is negative; shown in red in Figure 2.1. The third principal component is again space-like. All models that train well have small Bhattacharyya distances to the truth P_* towards the end of training; they also have small errors (zero in almost all cases). But these probabilistic models are different from each other, and they are also different from the truth P_*. Our visualization technique is emphasizing these subtle differences using all coordinates, including the imaginary coordinate corresponding to the negative eigenvalue. Figure 2.2b shows the train loss of all models (colored by purple for small, yellow for large). Even if the truth looks far away from them visually (> 4 in a Euclidean sense), models colored purple in Figure 2.2b have small distances from the truth $\mathrm{d_B}(P_w, P_*) < 0.2$; incidentally their Minkowski distance to the truth in the top three coordinates is negative.

In Figure 2.2b, the spread of points (yellow) near P_0 consists of some models that have 90% error (same as that of ignorance). There are 1500 such points, coming from 370 different trajectories (over 85% of points are from 145 trajectories). Over half of these high error deviating networks (see Figure 2.3b) eventually trained to zero error. These models have the same error as that of ignorance P_0 but the visualization method distinguishes them from ignorance because their probabilities are not uniform. The spread of the points in the visualization in this case is therefore coming from differences in the probabilities. These models can be brought back to the manifold of good training trajectories simply by training them further. Now notice the points colored purple in Figure 2.2d. These models have a large Bhattacharyya distance (> 0.15) from points marked A, B or C on the geodesic (which corresponds to progress of 0.01, 0.5 and 0.99 respectively). Figure 2.2c shows that these models also have very different errors from each other. This spread of points away from the manifold is therefore also coming from large differences in the probabilities.

Now notice the blue cluster of models (ConvMixer) in Figure 2.2a; as Figure 2.2d shows, the distance of a bulk of these ConvMixer models to point A is small (< 0.1). And Figure 2.2c suggests that these models have error < 10% (some also have larger errors). In this region, the spread of the points in the visualization is



(a) (b)

Figure 2.3: Number of models P with $\mathrm{d_B}(P, P_*) > 2$ (that are away from the main manifold) stratified by **(a)** architectures and **(b)** the number of epochs.

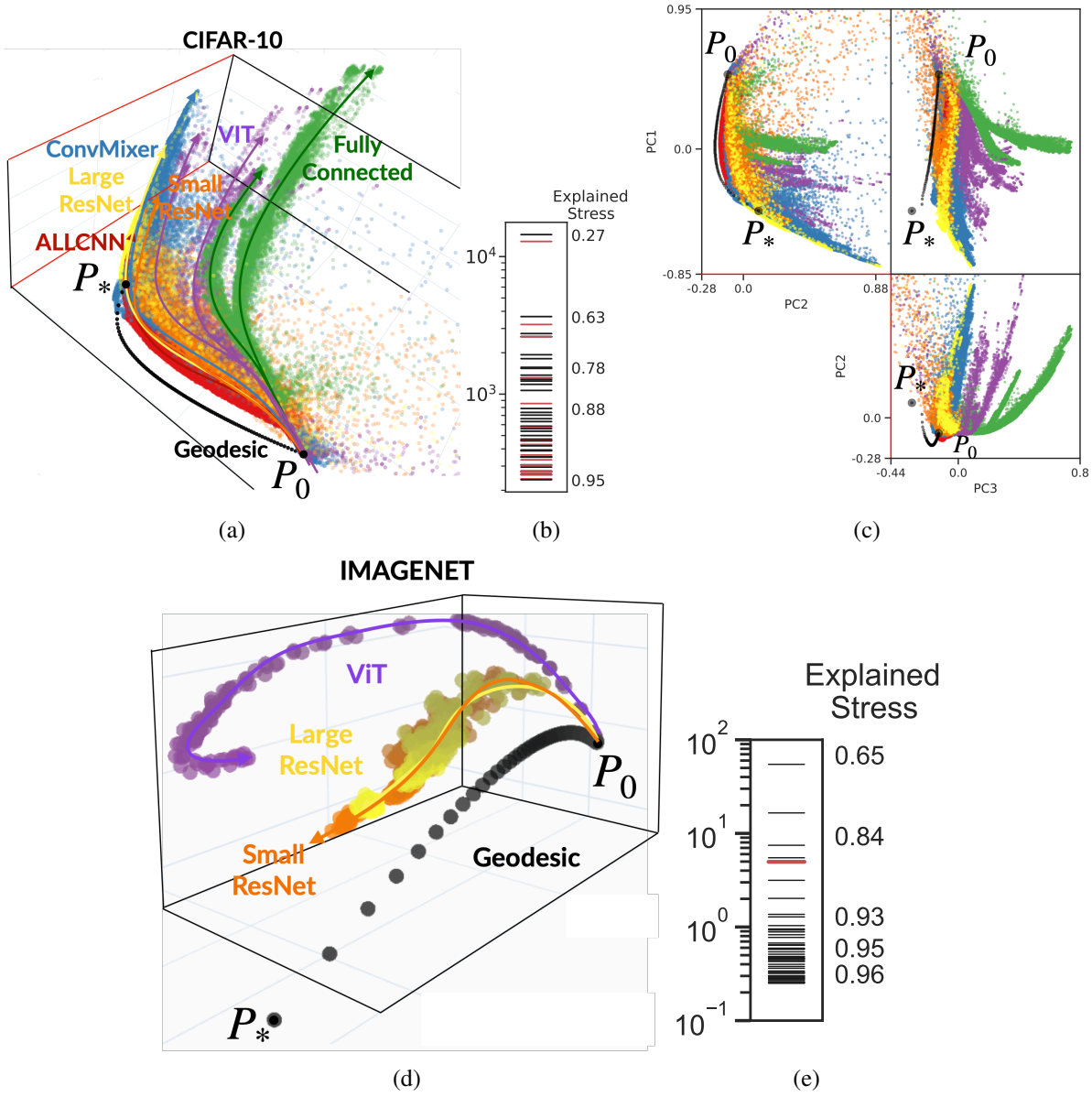
coming predominantly from the small differences in the probabilities.

Figure 2.3a studies models that are away from the manifold, with $\mathrm{d_B}(P, P_*) > 2$ (yellow in Figure 2.2b). For ConvMixer and the two residual networks, a majority of these models were trained by Adam. No AllCNN models were away from the manifold. Figure 2.3b stratifies these models by the optimization algorithm. In early stages of training, these are networks trained with SGD or SGD with Nesterov's acceleration with large batch-sizes (more than 500); this accounts for about 35% of the models. Adam is primarily responsible for models that are away from the manifold at later stages of training (about 55% of the points). We speculate that this could be related to poorer test errors of Adam than SGD for image classification tasks.

The manifold of predictions on the test data is also effectively low-dimensional, with more significant differences among architectures

Figure 2.4a shows the first three dimensions of the InPCA embedding of predictions on the test data using the same networks as that of Figure 2.1a. The explained stress of the first three dimensions is still high (63%) and it increases to 95% within the first 50 dimensions; these numbers are smaller than those for the training data. For CIFAR-10, the prediction space has 9×10^4 dimensions ($N = 10^4$ and $C = 10$) and for ImageNet the prediction space has 4.995×10^7 dimensions ($N = 50,000$ and $C = 1000$). This suggests that in spite of the vast diversity in configurations of these networks, their trajectories in the prediction space of the test samples also lie on an effectively low-dimensional manifold.

The test manifold is broadly similar to the train manifold in Figure 2.1a. Trajectories begin near ignorance



(a) (b) (c)

(d) (e)

Figure 2.4: Predictions on the test data of networks with different configurations (architectures denoted by different colors, different optimization algorithms and regularization mechanisms) on CIFAR-10 in **(a)** and on ImageNet in **(d)** is also effectively low-dimensional. Trajectories of different architectures are distinctive on the test data. Test manifold is also hyper-ribbon-like: eigenvalues of the InPCA distance matrix Equation (2.2.6) for CIFAR-10 **(b)** and ImageNet **(e)** are spread over a large range and the top few dimensions capture a large fraction of the stress Equation (2.2.8) (numbers indicate explained stress in the top 1, 3, 10, 25 and 50 dimensions. **(c)** shows a pairwise comparison for the first three principal components for CIFAR-10 models. PC1-PC2 of Figure 2.1c look quite similar to those of **(c)**. In **(a,d)**, we have drawn smooth curves denoting trajectories by hand to guide the reader.

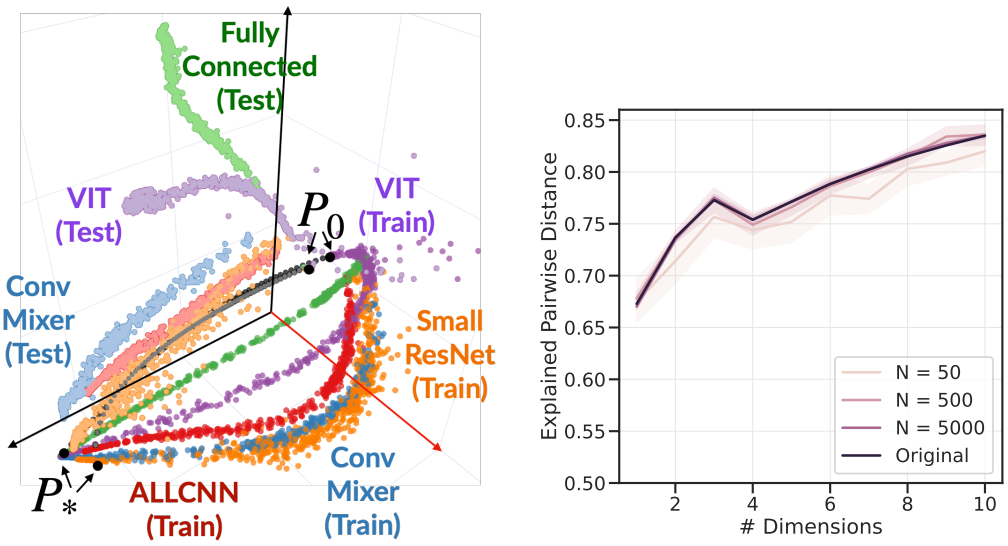
($\mathrm{d_B}(P, P_0) < 0.6$ at the start of training) but they do not always end near P_*. This is expected because different architectures have different test loss/errors at the end of training. The Bhattacharyya distance to the truth is one half of the test cross-entropy loss; models with poor test loss should be farther from P_* than those with a small test loss. Bhattacharyya distances of the end points of trajectories are as large as 0.58 for the test manifold compared to 0.02 for the train manifold after excluding models with train error > 10%.

Trajectories of different configurations seem to be more dissimilar in Figure 2.4a than those in Figure 2.1a; networks of different architectures have more distinctive test trajectories. We have analyzed these differences quantitatively in Figure 2.19a. But it is remarkable that even if different architectures have quite different trajectories, different models with the same architecture predict similarly on the test data. In other words, all fully-connected networks make the same kind of mistakes, and all convolutional networks are correct on generally the same samples. For fully-connected networks and ViTs, we see two different test trajectories corresponding to the two kinds of data augmentation techniques. For convolutional architectures, there are minor differences in test trajectories due to augmentation. This could be because we used randomly cropped images for augmentation: convolutional networks are relatively insensitive to random crops because their features have translational equivariance.

Section 2.5.4.2 provides a detailed analysis of the test trajectories.

Embedding probabilistic models along train and test trajectories into the same space So far, we have analyzed train and test manifolds independently of each other. Indeed, probabilistic models Equation (2.2.1) corresponding to train and test data belong to different sample spaces, even if the two were created from the same underlying weights. It is however useful to visualize the two manifolds in the same space to understand how progress towards the truth in the train space results in progress towards the truth in the test space.

We first computed InPCA coordinates using probabilistic models on train data, let us denote one such model with weights u as P_u. We then used the procedure developed in Equation (2.2.9) to embed test models into



(a) (b)

Figure 2.5: **(a)**: A joint embedding of a subset of networks on CIFAR-10 using their predictions on samples from both the train (bold) and test (translucent) sets. **(b)**: the explained pairwise Bhattacharyya distances computed using Equation (2.4.2) is quite high for models on the train data after embedding them into an InPCA embedding computed using a small number of samples ($N = 5000$, $N = 500$, and $N = 50$) in the train data. Section 2.5.3.3 discusses this further.

these coordinates as follows. Let us denote by P'_u the model on the test data for the same weights u. Calculate

$$W'_{uv} = -\frac{\mathrm{d_B}(P'_u, P'_v)}{2} + \frac{1}{2m}\left(\sum_{u'} \mathrm{d_B}(P_u, P_{u'}) + \mathrm{d_B}(P_v, P_{u'}) - \frac{1}{m}\sum_{v'} \mathrm{d_B}(P_{u'}, P_{v'})\right); \tag{2.3.1}$$

for all models P_u and P_v. The first term is the distance between two test models but the second term is computed using only train data and is the same as that of Equation (2.2.9). The embedding of a test model P'_w is set to be $X'_w = \sum_{u=1}^{n} W'_{w,u} U_u |\Lambda_{uu}|^{-1/2}$ using the eigenvectors and eigenvalues of the train embedding. The procedure in Equation (2.2.9) was intended to embed new models of the same set of samples into an existing embedding. This present, somewhat peculiar, trick works when the number of train models and the number of test models are the same (which is the case for us), and when the second term in Equation (2.3.1) is close to its counterpart in Equation (2.2.9) (which is expected if there is self-averaging).

We first built an InPCA embedding using the train models and then used the procedure in Equation (2.3.1) to calculate the coordinates of the test models and obtained Figure 2.5a. Observations drawn from this procedure

are qualitatively the same as those from Figures 2.1 and 2.4, e.g., train and test trajectories of different architectures still lie on similar manifolds, test trajectories of AllCNN, ConvMixer and Small ResNet are close to each other, and test trajectories of Fully-Connected and ViT architectures are far from the others. The explained pairwise distances for the test models using the InPCA coordinates computed from the train models are also consistent with those obtained from embedding the test models independently like Figure 2.4a; 0.52 versus 0.56 in the top 10 dimensions, respectively. This indicates that pairwise distances in the test data are well-preserved by the InPCA coordinates constructed using pairwise distances on the train data. When two models differ on the train data, they also differ in a similar way on the test data.

We also built a new InPCA embedding using pairwise Bhattacharyya distances in Equation (2.2.2) calculated using only a subset of the samples. Figures 2.5b, 2.13 and 2.14 show the result of using the procedure in Equation (2.3.1) to project the original distance matrix into the coordinates of this new InPCA. The explained pairwise distance of the original checkpoints is consistently quite high, even when as few as $N = 50$ or $N = 10$ samples are used to calculate the embedding out of the 50,000 and 10,000 samples for train and test sets respectively. This suggests that our techniques for analysis of high-dimensional models can also be used on very large datasets. For ImageNet, where $C = 1000$, we have also noticed that the InPCA embedding looks similar if we first project the output probabilities into a smaller space by multiplying by a random matrix (with columns that sum up to 1).

Architectures—not training or regularization schemes—primarily distinguish training trajectories in the prediction space

For all networks that trained to zero error, we interpolated the checkpoints from their trajectories to get models along the training trajectory that are equidistant in terms of their progress (Equation (2.2.4)) towards the truth P_*. Using these interpolations, we calculated the distance between trajectories corresponding to different configurations using Equation (2.2.5), averaged over the weight initializations. Figure 2.6a shows a dendrogram obtained from a hierarchical clustering of these distances. Clusters identified from this analysis primarily correspond to different architectures (row colors match those in Figures 2.1a and 2.4a). The cluster of trajectories of networks with convolutional architectures has a diameter that is about as large as the cluster of trajectories of fully-connected and self-attention-based networks (about 0.1 pairwise Bhattacharyya

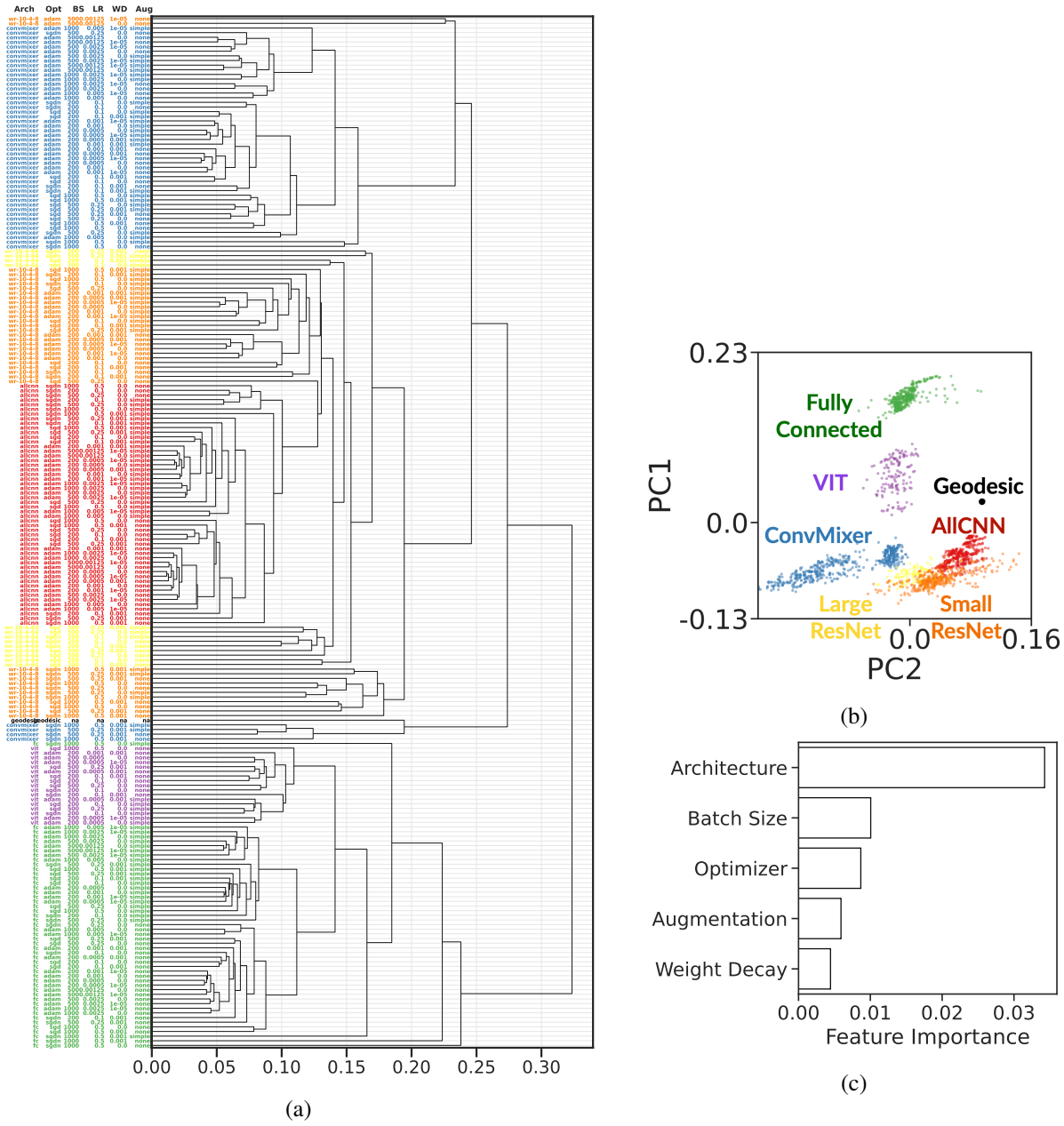


Figure 2.6: **(a)**: dendrogram obtained from hierarchical clustering of pairwise distances (averaged over weight initializations) between training trajectories (computed using Equation (2.2.5)) of networks with different configurations (X-labels correspond to architecture, optimization algorithm, batch-size, learning rate, weight-decay coefficient and augmentation strategy). There are strong similarities in how networks with different architectures, optimization algorithms and regularization mechanisms learn. **(b)**: the first two components of an InPCA embedding (without averaging over weight initializations) of train trajectories, each point is one trajectory; explained stress of top two dimensions is 63.6%. **(c)**: variable importance from a permutation test ($p < 10^{-6}$) using a random forest to predict pairwise distances. These three plots suggest that architecture is the primary distinguishing factor of trajectories in the prediction space. Test trajectories exhibit similar patterns (see Figure 2.19).

distance on average between models on these trajectories that have the same progress). This points to a strong similarity in how networks with different architectures, optimization algorithms, hyper-parameters, regularization and data augmentation techniques learn. Fully-connected and self-attention-based networks train along different trajectories than networks with convolutional architectures. The geodesic is far from all trajectories.

Within a cluster, say fully-connected networks (green), there are only marginal differences between different configurations, e.g., different optimization methods, different batch-sizes, weight-decay vs. no weight decay, augmentation vs. no augmentation. The dendrogram is created using distances between entire trajectories. So this analysis suggests that training trajectories of most fully-connected networks are similar. This pattern largely holds for the other architectures also. Small vs. large residual networks (orange vs. yellow respectively) have similar training trajectories; Figure 2.8 shows that the larger network progresses faster towards P_*.

Optimization (i.e., the algorithm and the batch-size) is the second prominent distinguishing factor. Within clusters of different architectures, networks trained with the same optimization algorithm have similar trajectories. In particular, for convolutional architectures, trajectories of Adam are more similar to each other than those of SGD or SGD with Nesterov's acceleration. We do not see such a separation for non-convolutional architectures where different optimization algorithms lead to similar trajectories (for them, differences come from data augmentation techniques). The details of different optimization algorithms matter little, e.g., trajectories of networks trained with different learning rate and batch-sizes are quite similar to each other. In general, networks that use weight-decay and networks that do not use weight-decay have similar trajectories. In general, for all architectures, networks trained with augmentation and without augmentation have only marginally different trajectories in the prediction space.

In Figure 2.6b, we computed an InPCA embedding of the pairwise distances between trajectories corresponding to different configurations (without averaging across weight initializations). This gives a qualitative understanding of the dendrogram: clusters of InPCA are consistent with the clusters in the dendrogram. While an InPCA embedding of the pairwise distances between models in Figure 2.1c depicts a low-dimensional manifold, Figure 2.6b illustrates differences in how different configurations train, in particular architectures. This is also evidence that our techniques can also be used to understand entire trajectories in the prediction

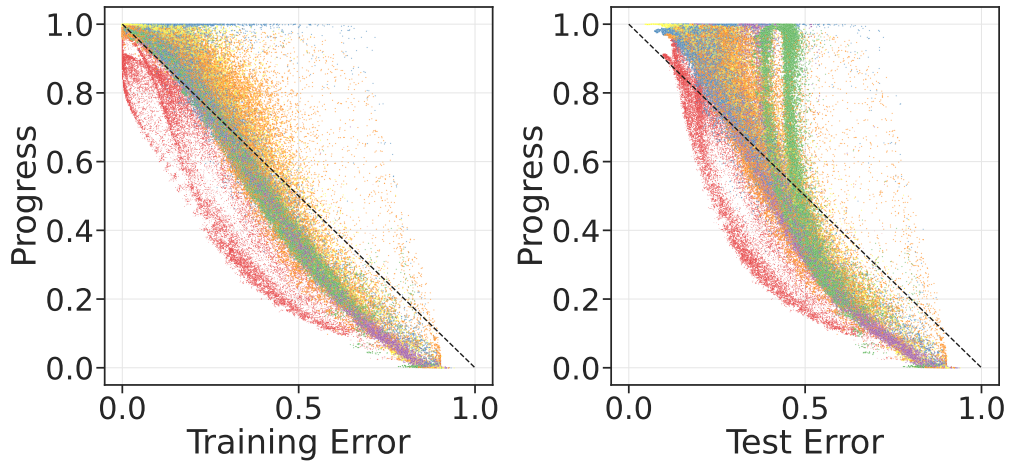


Figure 2.7: Progress of models with different configurations (color scheme is same as that of Figure 2.1a) is strongly correlated with **(a)** train error ($R^2 = 0.95$), and **(b)** test error ($R^2 = 0.88$).

space. We built a random forest-based predictor of the distance between trajectories of two configurations using their distance to the geodesic (real-valued covariate) and their configuration (categorical covariate) as inputs. A permutation-test performed using the random forest to estimate variable importance in Figure 2.6c confirms our discussion above: architecture is the most important distinguishing factor of these trajectories and optimization (batch-size, training algorithm) is the next important factor.

Section 2.5.4.1 provides a more detailed analysis of the train trajectories. For all architectures, optimization algorithms and regularization mechanisms, networks with different weight initializations train along very similar trajectories in the prediction space. We quantify this phenomenon using "tube widths" which capture the differences between models corresponding to different weight initializations at the same progress. Train trajectories are close to the geodesic at early (because they begin near P_0) and late parts (because they end near P_*) of the training process. While test trajectories also begin near ignorance P_0, their distance to the geodesic is larger, and towards the end of training all test models are quite far from truth. As Section 2.5.4.2 and Figure 2.19a show, test trajectories exhibit largely consistent patterns.

A larger network trains along a similar manifold as that of a smaller network with a similar architecture but makes more progress towards the truth for the same number of gradient updates

Networks with different configurations make progress towards the truth P_* at different rates. As Figure 2.7 shows, progress is strongly correlated with both train error ($R^2 = 0.95$) and test error ($R^2 = 0.88$). Progress towards the train truth and towards the test truth are also highly correlated with each other ($R^2 = 0.99$). This suggests that progress, which can be calculated easily using Equation (2.2.4), is a good way to judge how close models are to both train and test truths. Note that models may not have a progress of 1 even if they have zero training error (AllCNN trained with Adam in our case). In our work, we have used progress, which is a geometrically natural quantity in probability space, to measure and interpolate trajectories. Figure 2.7 also suggests that we could have used training error to interpolate checkpoints and would have obtained similar conclusions.

On both train and test manifold, at low error, AllCNN in red and Large ResNet in yellow have markedly different progress than other architectures (too low and too high respectively). Recall from Figure 2.1a and Figure 2.17a that trajectories of AllCNNs are also closest to the geodesic and those of Large ResNet are farthest. At high errors, which are typically seen at early training times, all architectures exhibit similar progress. Different weight initializations do not result in different rates of progress. For the same batch-size, SGD with Nesterov's acceleration makes faster progress than SGD or Adam at very early training times but this difference vanishes at later stages of training. In general, models trained with weight decay achieve a lower final progress on both train and test manifolds.

We saw in Figure 2.6a that trajectories of the Large ResNet lie on the same sub-manifold as that of the Small ResNet; see Figure 2.16 for the tube widths. The trend for the test manifold in Figure 2.19a is similar. After the same number of gradient updates, the Large ResNet makes more progress towards the truth than the Small ResNet on CIFAR-10 (Figure 2.8a). Figure 2.8b shows the training progress against epochs averaged over different weight initializations for models trained on ImageNet. Again, the larger network (ResNet-50) makes more progress compared to the smaller network (ResNet-18) when trained using an identical optimization algorithm, learning rate schedule, batch-size and data augmentation.

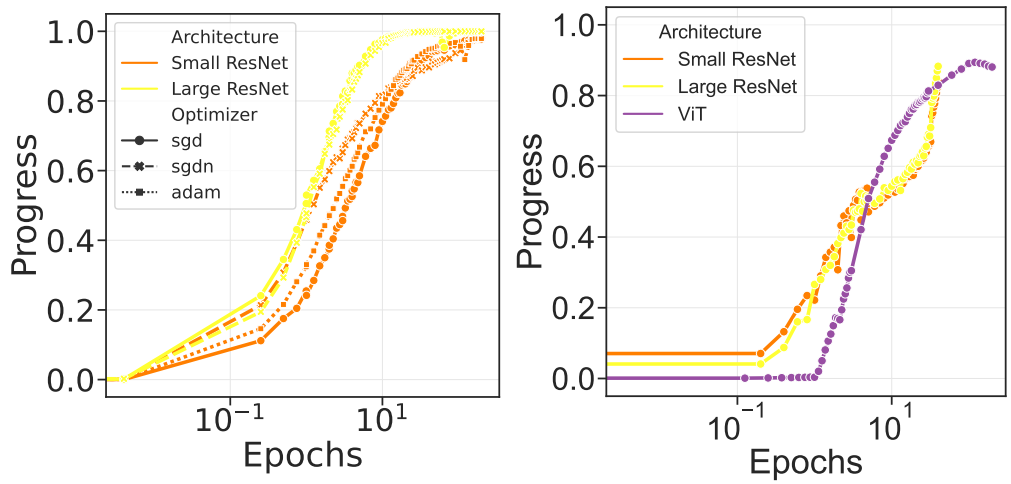


Figure 2.8: A Large ResNet makes more progress towards the truth than a Small ResNet for the same number of gradient updates on CIFAR-10 **(a)** and ImageNet **(b)**, irrespective of the optimization algorithms. Since the manifold of train and test trajectories for the two architectures are very similar (see Figures 2.1a and 2.6a), this suggests that larger networks and smaller networks make the same kind of predictions but the larger ones simply learn faster.

2.4. Discussion

A new insight into optimization in deep learning The central challenge in understanding why we can train deep networks effectively stems from the fact that the likelihood $p_w(y \mid x)$ of an output y given an input x is a complicated function of the parameters w. There is a large body of work that tackles this issue, e.g., optimization and generalization in function spaces for simpler architectures (Baldi and Hornik, 1989; Liang and Rakhlin, 2020) or analytical models (Mei et al., 2019; Chizat and Bach, 2020; Jacot et al., 2018a), analyzing representations of different layers (Shwartz-Ziv and Tishby, 2017; Achille and Soatto, 2018), properties of stochastic optimization methods (Chaudhari and Soatto, 2018) etc. This has led to some successes, e.g., a characterization of the training dynamics and generalization for two-layer neural networks. But there is a vast diversity of different architectures, optimization methods and regularization mechanisms in deep learning, and it is difficult to draw general conclusions from these analyses.

We have taken a different approach in this experimental paper. We studied many different network configurations to discover surprising phenomena that are not predicted by existing theory. We give two examples here. First, the optimization process explores an effectively low-dimensional manifold in the space of predictions

on the train and test data, in spite of the enormous dimensionality of both the embedding space and the weight space. This suggests that the optimization problem in deep learning might have a much smaller computational complexity than what is suggested by existing theory. Second, there is overwhelming empirical evidence that large networks with more parameters generalize better than smaller networks with fewer parameters (Brown et al., 2020; Vaswani et al., 2017; Dosovitskiy et al., 2021). A large body of work has sought to analyze this phenomenon (Belkin, 2021; Belkin et al., 2019; Bartlett et al., 2021b) and it has also been argued that we need to rethink our understanding of generalization in machine learning (Zhang et al., 2017a). We have found that a Large ResNet trains along the same manifold as that of a Small ResNet. It proceeds further towards the truth in the later parts of the trajectory. In view of the effectiveness of pruning and knowledge distillation (Frankle and Carbin, 2019; Hinton et al., 2015), this could mean that the superior test error of large networks could be matched by smaller networks using better training methods.

There is some previous work that has argued that weight configurations along a particular training trajectory lie on low-dimensional manifolds, e.g., using PCA (Feng and Tu, 2021), or by arguing that the mini-batch gradient has a large overlap with the subspace spanned by the top few eigenvectors of the Hessian during training for networks without batch-normalization (Gur-Ari et al., 2018; Ghorbani et al., 2019a; Sagun et al., 2017). These analyses that study the low-dimensionality of trajectories in the weight space provide important insights into the dynamics of training and foreshadow our work. But their findings are not related to the ones we discussed here. To wit, weights of different architectures lie in totally different vector spaces. We also checked that weights along trajectories of the same network configuration but different weight initialization cannot be explained using few principal components, i.e., they do not lie in a low-dimensional linear subspace, and in fact the explained variance of the top few dimensions decreases proportionally with the number of distinct weight initializations. The mapping between the weight space and the prediction space is quite complicated, and phenomena that occur in the former do not imply that they occur in the latter space in general. Even if the set of models explored by the training process were to lie in a low-dimensional linear subspace, the set of predictions of these models need not lie in a low-dimensional linear subspace. This is because the singular vectors of the Jacobian between the prediction space and the weight space can rotate. Conversely, if the predictions of a set of models lie on low-dimensional manifolds, this does not imply that weights do so as well, because, for instance, there are symmetries in the the parameterization of deep

networks.

Computational Information Geometry Information Geometry (Amari, 2016) is a rich body of sophisticated ideas, but it has been difficult to wield it computationally, especially for high-dimensional probabilistic models like deep networks. The construction in Equation (2.2.1) is a finite-dimensional probability distribution, in contrast to the standard object in information geometry which is an infinite-dimensional probability distribution defined over the entire domain of input data. It is this construction fundamentally that enables us to perform complicated computations such as, embeddings of high-dimensional models, geodesics in these spaces, projections of a model onto the geodesic, distances between trajectories in the prediction space, etc. Analysis of high-dimensional probabilistic models is challenging due to the curse of dimensionality: most points are orthogonal to each other in such spaces (Antognini and Sohl-Dickstein, 2018). Our visualization techniques, that build upon InPCA and IsKL (Quinn et al., 2019a; Teoh et al., 2020), work around this issue using multi-dimensional scaling (Cox and Cox, 2008; Saxe et al., 2019) and distances between probability distributions that violate the triangle inequality, e.g., the Bhattacharya distance. This has some mysterious benefits, e.g., our visualization technique can distinguish between small differences in high-dimensional probability distributions as they approach the truth in Minkowski space (Laub and Müller, 2004). Together with these visualization techniques, the theory developed in this paper gives new tools for the analysis of high-dimensional probabilistic models.

Interpretation of the top three principal coordinates It is surprising that just three-dimensions can capture 76% of the stress (for CIFAR-10) of such a large set of diverse training trajectories in Figure 2.1a. We next offer an interpretation of this phenomenon. Our probabilistic models are an N-product of probability distributions corresponding to points $(\sqrt{p_u^n(1)}, \ldots, \sqrt{p_u^n(C)})$ which lie on a $(C-1)$-dimensional sphere. Training trajectories begin near ignorance P_0 and end near P_*, so let us consider the straight line that joins ignorance and truth as one basis. Tangents to a training trajectory at ignorance (e.g., when networks are presumably learning "easy" images) and at truth (e.g., when networks are learning the most challenging images) can be two more basis vectors. This defines a three-dimensional subspace of the 450,000-dimensional prediction space. To represent this three-dimensional space, we can choose four probability distributions: P_0, P_*, and P_{s_1}, P_{s_2} computed by weighted averages of models with progress close to s_1 and s_2, respectively.

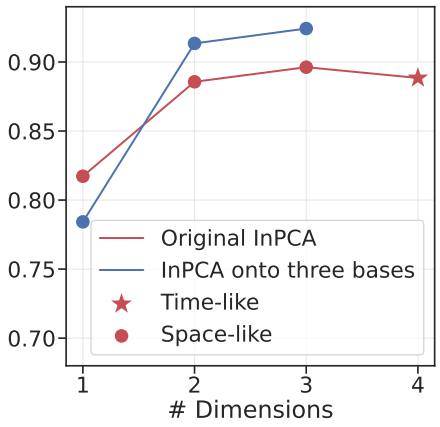


Figure 2.9: The procedure in Equation (2.2.9) was used to add original models used for Figure 2.1a into an InPCA embedding created using 4 points corresponding to three "bases" (straight line from ignorance to truth, and tangents to the training trajectories at ignorance and truth) for three configurations, all with AllCNN architecture. This new embedding preserves pairwise Bhattacharyya distances between the original models to a similar degree as that of the original InPCA embedding. The two embeddings also assign the same signs to the top few eigenvalues; for the embedding using 4 points, only the first 3 dimensions are non-trivial.

The latter two are stand-ins for the tangents to the trajectories at P_0 and P_* and they are calculated using

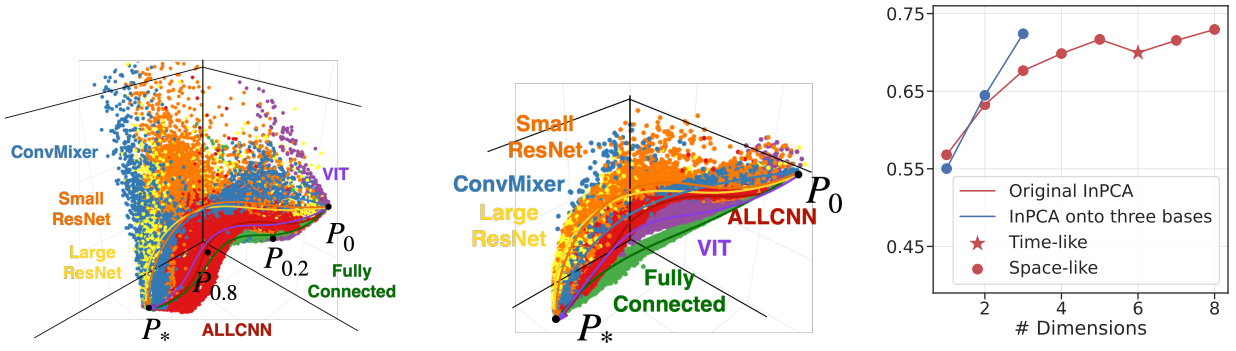
$$P_s = \frac{1}{Z} \sum_{P'} \exp\left(\frac{-(s_{P'} - s)^2}{2\sigma^2}\right) P', \tag{2.4.1}$$

where $Z = \sum_{P'} \exp\left(-(s_{P'} - s)^2/(2\sigma^2)\right)$ is the normalizing factor and s'_P is the progress of the model P'. We choose $\sigma = 0.05$ for all the experiments and experiment with different choices of s_1 and s_2. We can now build an InPCA embedding using these 4 models, and using the procedure in Equation (2.2.9) (which is equivalent to weighted-InPCA discussed in Section 2.5.3.2) we can add our original models in Figure 2.1a into this new InPCA embedding.

Figure 2.9 shows how well these new coordinates explain pairwise Bhattacharyya distances in $D \in \mathbb{R}^{m \times m}$ for models of three configurations (AllCNN architectures trained with SGD, SGD with Nesterov's acceleration and Adam) for ten different weight initializations by calculating

$$1 - \frac{\sum_{ij} \left| D_{ij} - \|X_i - X_j\|^2 \right|}{\sum_{ij} D_{ij}} \tag{2.4.2}$$

where $X_i \in \mathbb{R}^{q, d-q}$ are the d-dimensional coordinates of the embedded points; we can calculate this



(a) InPCA using 4 points (b) Original InPCA (c) Explained Pairwise Distances

Figure 2.10: All models in Figure 2.1a with Bhattacharya distance $d_B(P, P_*) < 2$, which effectively removed the spread of points away from the train manifold (also see Figure 2.2b), were embedded using InPCA coordinates constructed using 4 points corresponding to three "bases" (straight line from the ignorance to truth, and tangents to the training trajectories at ignorance and truth) in **(a)** and using the original InPCA coordinates in Figure 2.1a computed using all models in **(b)**. The top three coordinates in both **(a)** and **(b)** are space-like. The manifold in **(a)** is structurally similar to that of **(b)**, e.g., Small and Large ResNet models are close to those of ConvMixer models, and far from fully-connected models, some ResNets and ConvMixer models are away from the main manifold at intermediate training times. **(c)** shows that the explained pairwise Bhattacharyya distances between models in the new embedding is very high, and comparable to that of the first 8 dimensions in the original InPCA. We have drawn smooth curves denoting trajectories by hand to guide the reader.

quantity that we call "explained pairwise distances" using both these new and the original InPCA coordinates. Explained pairwise distances using the original InPCA embedding (which was created using all models) and this new InPCA embedding (which was created using only the 4 points: P_0, P_* and P_{s_1}, P_{s_2} for $s_1 = 1 - s_2 = 0.3$) are both quite large—and similar to each other. The two embeddings are also consistent as to which coordinates are time-like (dimensions in Figure 2.9 are ordered by the magnitude of eigenvalues).

We next performed the same analysis but with all models in Figure 2.1a with $d_B(P, P_*) < 2$, which effectively removes models that lie away from the manifold. In Figure 2.10a, we created an InPCA embedding using 4 points: ignorance P_0, truth P_* and P_{s_1}, P_{s_2} for $s_1 = 1 - s_2 = 0.2$ by computing the average over all models P' in Equation (2.4.1), and projected the original probabilistic models into these new coordinates using the procedure in Equation (2.2.9) to visualize them. We rotated the top 3 non-trivial dimensions of this embedding to best align the embedding created using the original InPCA procedure that uses all models to compute the embedding. This alignment was done using the Kabsh-Umeyama algorithm (Lawrence et al.,

2019) which finds the optimal translation, rotation and sign-flips of the coordinates to align two sets of points; the root mean square deviation (RMSD) is 0.06. As Figure 2.10b shows, there are structural similarities in the embedding computed using only the 4 points and the one computed using all models, e.g., Small and Large ResNet models are close to those of ConvMixer models, and far from fully-connected models, some ResNets and ConvMixer models are away from the main manifold at intermediate training times. Figure 2.10c shows that the new embedding also preserves pairwise Bhattacharyya distances between the models to a similar degree.

This exercise gives us an interpretation for the low-dimensional embedding discovered by InPCA. It may point to a mechanistic explanation for our findings: the train and test manifolds are effectively low-dimensional because networks with different architectures, optimization algorithms, hyper-parameter settings and regularization mechanisms fit the same easy images in the dataset first and the same challenging images towards the end of training; this phenomenon has also been studied in Hacohen et al. (2020).

Why are the train and test manifolds effectively low-dimensional? It is remarkable that trajectories of networks with such different configurations lie on a manifold whose dimensionality is much smaller than the embedding dimension. To explore this further, we analyzed trajectories of networks trained on synthetic data: (a) sampled from a "sloppy" Gaussian, i.e., with eigenvalues of the covariance that are distributed uniformly on a logarithmic scale (this structure has been noticed in many typical problems (Yang et al., 2022; Quinn et al., 2021)), and (b) sampled from an isotropic Gaussian (non-sloppy data). We labeled these samples using a random two-layer fully-connected teacher network and trained student networks with different configurations to fit these labels. When students are initialized near ignorance P_0, train and test manifolds are effectively low-dimensional for both kinds of data (87% explained stress in top ten dimensions). When students are initialized at different initial points $\{P_0^{(k)}\}_{k=1,\ldots,10}$ similar to those in Figure 2.20, train and test manifolds are still effectively low-dimensional for both kinds of data; top ten dimensions have 85% explained stress. But the explained stress is higher in the top few dimensions if trajectories begin from near each other, e.g., from fewer initial points, or from ignorance. For sloppy input data, trajectories converge to the same manifold quickly even if they begin from very different initial points. Section 2.5.5 discusses this experiment further.

We therefore believe that the low-dimensionality of the manifold arises from (a) the structure of typical datasets (Goldt et al., 2020; d'Ascoli et al., 2021; Refinetti et al., 2021), e.g., spectral properties, and (b) the fact that typical training procedures initialize models near one specific point in the prediction space, the ignorance P_0. Along the first direction, recent work on understanding generalization (Bartlett et al., 2020; Yang et al., 2022) has argued that deep networks, as also linear/kernel models, can interpolate without overfitting if input data have a sloppy spectrum. Work in neuroscience (Simoncelli and Olshausen, 2001; Field, 1994) has also argued for visual data being effectively low-dimensional. Theories in machine learning (Vapnik, 1998; Schölkopf and Smola, 2002) and information-theory (Rissanen, 1978; Balasubramanian, 1997) for model selection are based on estimates of the number of models in a hypothesis class that are consistent with the data. In this context, our second suspect, namely initialization, suggests that even if the size of the hypothesis space might be very large for deep networks (Dziugaite and Roy, 2017; Bartlett et al., 2017), the subset of the hypothesis space explored by typical training algorithms might be much smaller.

2.5. Appendix

2.5.1. Notation

Symbol	Description
N	Number of samples
C	Number of classes
x_n	Input sample with index $n \in \{1, \dots, N\}$
y_n	Label assignment of sample with index $n \in \{1, \dots, N\}$
y_n^*	Ground-truth label of sample with index $n \in \{1, \dots, N\}$
w	Weights of the deep network
$\boldsymbol{y}^*$	Ground-truth labels for each of the N samples, $\boldsymbol{y}^* = (y_1^*, \dots, y_N^*)$
$\boldsymbol{y}$	Label assignment for each of the N samples, $\boldsymbol{y} \in \{1, \dots, C\}^N$
$p_w^n(y_n)$	Probability that sample x_n belongs to class $y_n \in \{1, \dots, C\}$, $p_w^n(y_n) \equiv p_w(y_n \mid x_n)$

$P_w(.)$	Probabilistic model with weight w; assigns a probability to every sequence $\boldsymbol{y}$
P_*	Truth ($P_* = \delta_{\boldsymbol{y}^*}(\boldsymbol{y})$)
P_0	Ignorance, has $p_0^n(c) = 1/C$ for all classes c and samples n
$\mathrm{d_B}$	Bhattacharyya distance between two probability distributions
$\mathrm{d_G}$	Geodesic distance (great circle distance) between two probability distributions
$g(w)$	Fisher Information Metric (FIM) at weight configuration w
$(\sqrt{p_u^n(c)})_{c=1,\ldots,C}$	Point on a $(C-1)$-dimensional sphere
$P_{u,v}^{\alpha}$	Geodesic between probability distributions P_u and P_v parameterized by $\alpha \in [0,1]$
T	Number of recorded checkpoints
$(w(k))_{k=0,\cdots T}$	A sequence of recorded checkpoints in the weight space
s_w	Progress of a probabilistic model P_w with weights w
α	Interpolating parameter along a geodesic, $\alpha \in [0,1]$
$\tilde{\tau}_w$	A sequence of probabilistic models recorded during training, also denoted by $(P_{w(k)})_{k=0,\cdots T}$
τ_w	A continuous curve in the space of probabilistic models, also denoted by $(P_{w(s)})_{s\in[0,1]}$
$\mathrm{d_{traj}}(\tau_u, \tau_v)$	Distance between trajectories τ_u and τ_v
D	Matrix ($\in \mathbb{R}^{m\times m}$) of pairwise Bhattacharyya distances between m probabilistic models, entries of this matrix are denoted by D_{ij}, D_{uv} etc. depending upon the context
W	Matrix ($\in \mathbb{R}^{m\times m}$) of centered pairwise Bhattacharyya distances, $W = -LDL/2$ where $L_{uv} = \delta_{uv} - 1/m$ performs the centering

X_w	Coordinates ($\in \mathbb{R}^{p,m-p}$) of the InPCA embedding of a model with weights w
$1 - \sqrt{\frac{\sum_{ij}\left(W_{ij} - \sum_{k=1}^{d} \Lambda_{kk} U_{ik} U_{kj}\right)^2}{\sum_{ij} W_{ij}^2}}$	Explained stress, used to estimate the fraction of the entries of the centered pairwise distance matrix W that are preserved by an embedding; equivalent to explained variances in standard PCA (up to the square root)
$1 - \frac{\sum_{ij}\left\lvert D_{ij} - \lVert X_i - X_j \rVert^2 \right\rvert}{\sum_{ij} D_{ij}}$	Explained pairwise distances, used to estimate the fraction of the entries of the pairwise Bhattacharyya distance matrix D that are preserved by an embedding

2.5.2. Details of the experimental setup

Datasets The experimental data in this paper was obtained by training deep networks on two datasets.

- The CIFAR-10 dataset (Krizhevsky et al., 2012) has $N = 50,000$ RGB images in the training set of size 32$\times$ 32 from $C = 10$ different categories (airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck). The test set has $N = 10,000$ images. Both train and test sets have an equal number of images in each category.

- The ImageNet dataset (Deng et al., 2009) has $C = 1000$ categories and a total of $N = 1.28 \times 10^6$ RGB images of size 224 $\times$ 224 in the training dataset. Different categories have slightly different numbers of images in the train set, but all categories have at least 1000 images. The test set consists of $N = 50,000$ images, with 50 images from each category.

Neural architectures For CIFAR-10, we used six neural architectures. These architectures were chosen and and configurations were chosen to ensure that these networks could fit all the images in the training dataset, i.e., achieve zero training error, for most training methods.

(i) A multi-layer perceptron with rectified linear unit (ReLU) nonlinearities (fully-connected network) with 4 hidden layers, of size [1024, 512, 256, 128] respectively.

(ii) An "all convolutional network" (AllCNN (Springenberg et al., 2015)) with 5 convolutional layers followed by an average pooling layer; first three layers have 96 channels and the latter two have 144 channels.

(iii) Two different wide residual networks (Zagoruyko and Komodakis, 2016). The larger one has 16 layers and [64, 256, 1024, 4096] channels for the convolutional layers in the four blocks, and the smaller network has 10 layers with [8, 32, 128, 512] channels for the four blocks. Both networks have a "widening factor" of 4. We modified the implementation at https://github.com/meliketoy/wide-resnet.pytorch.

(iv) The ConvMixer architecture (Trockman and Kolter, 2022) is a convolutional network but it uses very large receptive fields and maintains the same size for the activations across successive layers. We did not make any changes to the architecture from the original paper.

(v) The ViT architecture (Dosovitskiy et al., 2021) is a self-attention based network that uses a set of disjoint patches of size 4×4 from the input images. This network does not use convolutional operations and instead uses the so-called self-attention layer that is popularly in natural language processing. We use a linear layer size of 512, 8 self-attention heads and 6 transformer blocks (layers). We used the implementation from https://github.com/lucidrains/vit-pytorch.

We do not use Dropout (Srivastava et al., 2014) in any of the networks. All networks except ViT have a batch-normalization (Ioffe and Szegedy, 2015) layer after each convolutional or fully-connected layer, except ViT which uses layer normalization (Ba et al., 2016).

For ImageNet, we used three architectures.

(i) A smaller residual network (He et al., 2016) with 18 layers (ResNet-18). This residual network is different from the wide residual network used for CIFAR-10, primarily in that there are fewer channels in each block. A ResNet is architecturally similar to a wide residual network with a widening factor of 1. We replaced each strided convolution with a convolution followed by a BlurPool layer (Zhang, 2019).

(ii) A larger residual network with 50 layers (ResNet-50). This is one of the most popular networks for training on ImageNet and widely used as a benchmark architecture in the field.

(iii) The ViT architecture which is similar to the one used for CIFAR-10 above except that the receptive field of the first layer is larger due to the larger images in ImageNet. We trained a smaller variant of ViT called ViT-S (with 22 million weights) which was introduced in (Touvron et al., 2021). It operates on patches of size 16×16 and has 6 self-attention heads and 12 transformer blocks.

Training multiple models on ImageNet is computationally expensive. To mitigate this, we used efficient data loaders, computed gradients in half-precision, and chose effective training hyper-parameters (FFCV (Leclerc et al., 2022) for training ResNets and timm (Wightman, 2019) for training ViTs).

Training procedure For both datasets, we normalize images in the train and test sets by the channel-wise mean and variance of the images in the training dataset. For CIFAR-10, we also augmented training images by randomly cropping a region of size 32×32 after padding the original image by 4 pixels on each side, and performing horizontal flips with a probability of 0.5; our data contains models trained with and without such data augmentation.

All the networks are initialized using the default PyTorch weight initialization as follows. For fully-connected layers with an input dimension d, all weights and biases are sampled independently from a uniform distribution on $[-d^{-1/2}, d^{-1/2}]$. For convolutional layers with c channels and a $k \times k$ convolutional kernel, all weights and biases are sampled independently from a uniform distribution on $[-(ck)^{-1/2}, (ck)^{-1/2}]$.

We started with 3120 different configurations, 520 for each network architecture. Some networks did not finish training due to numerical errors during gradient updates, and we excluded them from our analysis. Figure 2.11 shows how many of the configurations did not finish training for each network architecture. Our data, with 2,296 different configurations, therefore contains fewer ViTs and Large ResNets than other architectures.

For CIFAR-10, we used three different optimization methods, stochastic gradient descent (SGD), SGD with Nesterov's momentum (with a coefficient of 0.9) and Adam (Kingma and Ba, 2015), three different

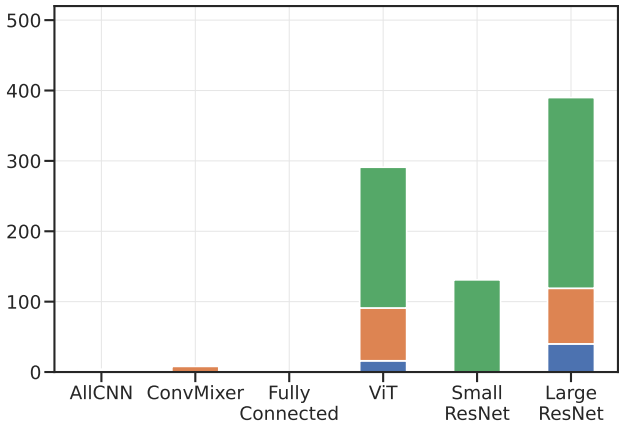


Figure 2.11: Number of networks that did not train beyond 90% error for Adam (green), SGD (blue) and SGD with Nesterov's acceleration (orange). These models are not included in our analysis.

batch sizes (200, 500 and 1000) and three different values of the weight decay coefficient (ℓ_2 regularization) ($\{0, 10^{-3}\}$ when training with SGD and SGD with Nesterov's momentum, and $\{0, 10^{-5}\}$ when training with Adam). Fully-connected networks trained on augmented data are trained for 300 epochs to achieve zero training error, all other networks are trained for 200 epochs. One epoch corresponds to using each sample in the training dataset exactly once to compute a gradient update (i.e., mini-batches are sampled without replacement). As the batch-size in SGD is increased, the stochasticity of the weight updates decreases and this makes the iterations more susceptible to converging near local minima of the loss function, and thereby obtain poor test error. It has been noticed empirically that keeping the ratio of the learning rate to batch-size invariant helps mitigate this deterioration of test error for large batch sizes (Goyal et al., 2017). This has also been argued theoretically via an analysis of the equilibrium distribution of SGD (Chaudhari and Soatto, 2018). Therefore, for SGD and SGD with Nesterov's acceleration, we fixed this ratio to 5×10^{-4}, i.e., for a batch size 200, we use a learning rate of 0.1, and increase the learning rate proportionally for larger batch sizes. For Adam, this ratio was 5×10^{-6}, i.e., we used a learning rate of 0.001 for a batch-size of 200. For all experiments, we decreased the learning rate using a cosine annealing schedule over the course of training, i.e., for all networks the learning rate decays to zero at the end of training.

Residual networks on ImageNet were trained using SGD with Nesterov's acceleration for 40 epochs with a batch-size of 1024. The learning rate was decreased linearly from 0.5. We used a weight decay coefficient of 5×10^{-5}; no weight decay was applied to parameters associated with batch-normalization. To reduce the

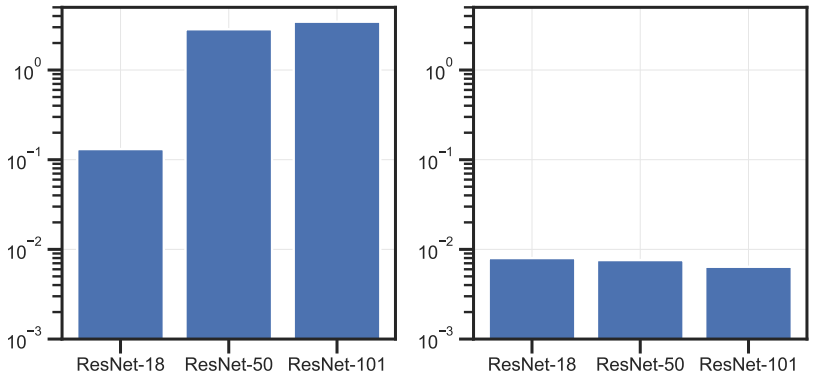


Figure 2.12: Bhattacharyya distance from ignorance P_0 for networks at the beginning of training for standard off-the-shelf implementations of ResNet (left). If we initialize the estimates of the mean and standard deviation of the batch-normalization layers by doing a forward pass on a few mini-batches, then networks are close to ignorance at the beginning of training (right).

training time, we used mixed-precision training. We also used progressive resizing, i.e., we trained on images of size 196×196 for the first 34 epochs before using the full-sized images (224×224) for the remaining 6 epochs. We use random horizontal flips and random-resize-crops for data augmentations. For datasets with a large number of classes such as ImageNet, it helps to use label smoothing (He et al., 2019), we used this with the smoothing parameter set of 0.9. This amounts to training towards a slightly different truth P_* where the correct category has a probability of 0.9 and the remainder 0.1 is distributed uniformly across the other 999 categories (instead of them being zero).

ViT architectures are difficult to train well with SGD, especially on large datasets such as ImageNet. We therefore trained ViTs on ImageNet using AdamP (Heo et al., 2021) with a cosine-annealed learning rate schedule and an initial learning rate of 0.001. We trained for 200 epochs using a batch-size of 1024 and weight decay of 0.01 without any dropout. These networks also require a more extensive set of data augmentations, we used horizontal flips with probability 0.5, cropping the image to get a patch of the desired size at a random location (images in ImageNet are not of the same size), and mixup regularization (Zhang et al., 2017b) which uses mini-batches that consist of convex combinations (with a random parameter) of images and ground-truth probability distributions.

Some ImageNet models are not initialized near ignorance P_0 We noticed that some randomly initialized models have a large Bhattacharyya distance from ignorance P_0. For example, the distance between a randomly

initialized ResNet-50 model and ignorance is as much as 0.91 times the Bhattacharyya distance between ignorance and truth $\mathrm{d_B}(P_0, P_*)$. We found that this is due to the batch-normalization layer (Ioffe and Szegedy, 2015) being incorrectly initialized at the beginning of training. Batch-normalization subtracts the channel-wise mean of the activations (computed from samples in a mini-batch) and divides the result by an estimate of the channel-wise standard deviation of the activations (computed using the samples in the mini-batch). During training, typical deep learning libraries such as PyTorch maintain an exponentially moving average of the mean and standard deviation of activations of mini-batches. And it is these averaged estimates that are used to compute the output probabilities for test data. In PyTorch, the mean is initialized to zero and the standard deviation is initialized to 1. This causes the magnitude of the activations to be very large in the final few layers at initialization and that is why the probabilistic model is very far from ignorance at initialization, as shown in Figure 2.12.

This phenomenon is seen in most popular off-the-shelf implementations of a ResNet, and could also be present in other architectures. When training in a supervised learning setting, this finding of ours is only marginally relevant because the estimates of the mean and standard deviation stabilize to reasonable values within 5–10 mini-batch updates. But there are many sub-fields of machine learning, few-shot learning (Dhillon et al., 2020), meta-learning (Thrun and Pratt, 2012) to name some, where the number of mini-batch updates of a trained model is a key parameter and where our finding has practical relevance. To fix this issue, we can initialize the batch-normalization mean and variance estimates—easily—by doing a forward pass on a few mini-batches from the training data before beginning the training. This ensures that the model starts training from near ignorance. When we collected data from our training trajectories on ImageNet, we did not have this fix. We therefore did not plot the first checkpoint for the ImageNet experiments in Figures 2.1d and 2.4d.

2.5.3. Addendum to Methods

2.5.3.1 InPCA creates an isometric embedding

InPCA, like standard PCA, relies on an embedding directed by the centered pairwise distances Equation (2.2.6). Observe that the centering in Equation (2.2.6) is the same as the centering performed in standard PCA, indeed it ensures that rows and columns of the pairwise distance matrix W sum to zero. Since InPCA involves pairwise Bhattacharyya distances, not pairwise Euclidean distances, such a centering is not trivially

equivalent to a translation of points in a vector space. We show next that the embedding created using InPCA is isometric, i.e., it satisfies Equation (2.2.7). The argument developed below also holds for other embedding techniques, e.g., the IsKL method discussed in Equation (2.5.3) that uses the symmetrized Kullback-Leibler divergence as the distance between probability distributions.

Given a real symmetric matrix $D \in \mathbb{R}^{m\times m}$, we can write $D_{ij} = \sum_k U_{ik}\Lambda_{kk}U_{jk}$ where the eigenvalues $\Lambda_{kk} \in \mathbb{R}$ and columns of U are the eigenvectors. We can define an "eigen-embedding" of such a matrix:

$$\mathbb{R} \ni X_{ik} \equiv \sqrt{|\Lambda_{kk}|}U_{ik}; \quad i,k \leq m$$

and a quasi inner-product $\langle a,b\rangle_D \doteq \sum_k \text{sign}(\Lambda_{kk})a_k b_k$ for $a,b \in \mathbb{R}^{p,m-p}$, with metric signature $(p, m-p)$ derived from the p positive eigenvalues of D. The quasi inner-product of the points in an eigen-embedding of a real symmetric matrix D allows us to reconstruct the entries of D:

$$D_{ij} = \langle X_i, X_j\rangle_D \,. \tag{2.5.1}$$

Now consider a finite symmetric premetric space $\mathcal{M} = (M, D)$ with $|M| = m$ points[4]. If D is a matrix of pairwise distances between these points, then it has a vanishing diagonal. The embedding of $-D/2$ denoted by $\{X_i \in \mathbb{R}^{p,m-p}\}_{i=1}^m$ satisfies $\langle X_i, X_i\rangle_{-D/2} = -D_{ii}/2 = 0$ for any $i \leq m$. Now observe that the distance between any X_i and X_j is the squared Minkowski interval between them, i.e.,

$$\sum_k \|X_{ik} - X_{jk}\|^2_{-D/2} = \langle X_i - X_j, X_i - X_j\rangle_{-D/2} = -(D_{ii} + D_{jj} - 2D_{ij})/2 = D_{ij}. \tag{2.5.2}$$

In other words, the m points in $\mathcal{M}$ can be isometrically embedded in a Minkowski space as the eigen-embedding of $-D/2$. The centering operation using a matrix $L_{ij} = \delta_{ij} - 1/m$ which we use to compute $W = -LDL/2$ ensures that

$$W_{ij} = \left\langle X_i - \overline{X}, X_j - \overline{X}\right\rangle_{-D/2}$$

where $\mathbb{R}^{p,m-p} \ni \overline{X} = m^{-1}\sum_i X_i$ is the mean of the eigen-embedding of $-D/2$; in other words, the

[4] A premetric space satisfies two properties: that the distance between two points is non-negative, and the distance of a point from itself is zero.

centered pairwise distance matrix is equal to the cross-covariance matrix in a Minkowski space.

Theorem 2.5.1. *Given a finite symmetric premetric space $\mathcal{M} = (M, D)$ with $|M| = m$ points, if $D \in \mathbb{R}^{m\times m}$ is the matrix of pairwise distances between these points, then the eigen-embedding of $W = -LDL/2$ where $L_{ij} = \delta_{ij} - 1/m$ is the centering matrix, is isometric to $\mathcal{M}$.*

Proof. Let the eigen-embeddings of $-D/2$ and W be $\{X_i\}_{i=1}^m$ and $\{Y_i\}_{i=1}^m$ respectively. We know that the eigen-embedding of $-D/2$ is isometric to $\mathcal{M}$. From Equation (2.5.1), we have that $\langle Y_i, Y_j\rangle = W_{ij}$ and so $\langle Y_i - Y_j, Y_i - Y_j\rangle_W = W_{ii} + W_{jj} - 2W_{ij}$. Since the centered pairwise distance matrix is equal to the cross-covariance matrix, we also have $W_{ij} = \left\langle X_i - \overline{X}, X_j - \overline{X}\right\rangle_{-D/2}$ and therefore

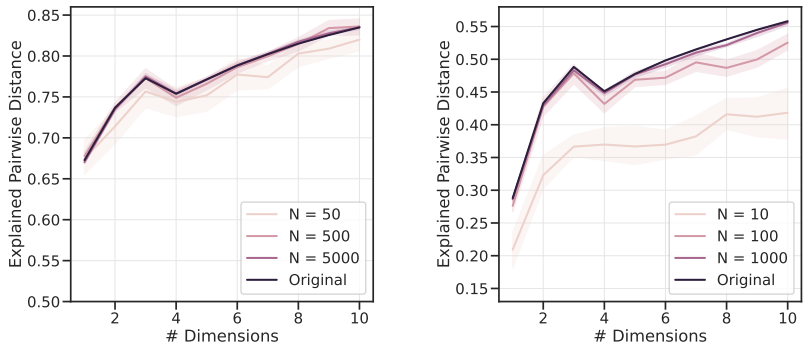
$$\begin{aligned}\langle Y_i - Y_j, Y_i - Y_j\rangle_W &= \left\langle X_i - \overline{X}, X_i - \overline{X}\right\rangle_{-D/2} + \left\langle X_i - \overline{X}, X_j - \overline{X}\right\rangle_{-D/2} - 2\left\langle X_i - \overline{X}, X_j - \overline{X}\right\rangle_{-D/2}\\ &= \langle X_i - X_j, X_i - X_j\rangle_{-D/2}\\ &= D_{ij}.\end{aligned}$$

□

2.5.3.2 Emphasizing different models using a weighted embedding

To study the details of the model manifold, we have found it useful to emphasize certain models in the visualization. There are many works (Vo et al., 2008; Gabriel and Zamir, 1979; Greenacre, 2005; Delchambre, 2015) that do similar things, e.g., those that modify the underlying objective of MDS to optimize a weighted Euclidean distance (but this does not do a good job of preserving pairwise distances between points), or those that learn a set of orthogonal transformations to highlight points of interest. We can also repeat models while computing InPCA: this shifts the center of mass and,at the same time transforms the visualization (via rotations and Lorentz boosts). It emphasizes the repeated models in the visualization. However, such a naive approach is computationally expensive because the size of the distance matrix D increases due to these repetitions.

We present a different approach called weighted-InPCA next. Let $D \in \mathbb{R}^{m\times m}$ be the matrix of pairwise Bhattacharyya distances $D_{uv} = \mathrm{d_B}(P_u, P_v)$ and let $\mu_u \in \mathbb{N}$ be multiplicity of the model with weights u,



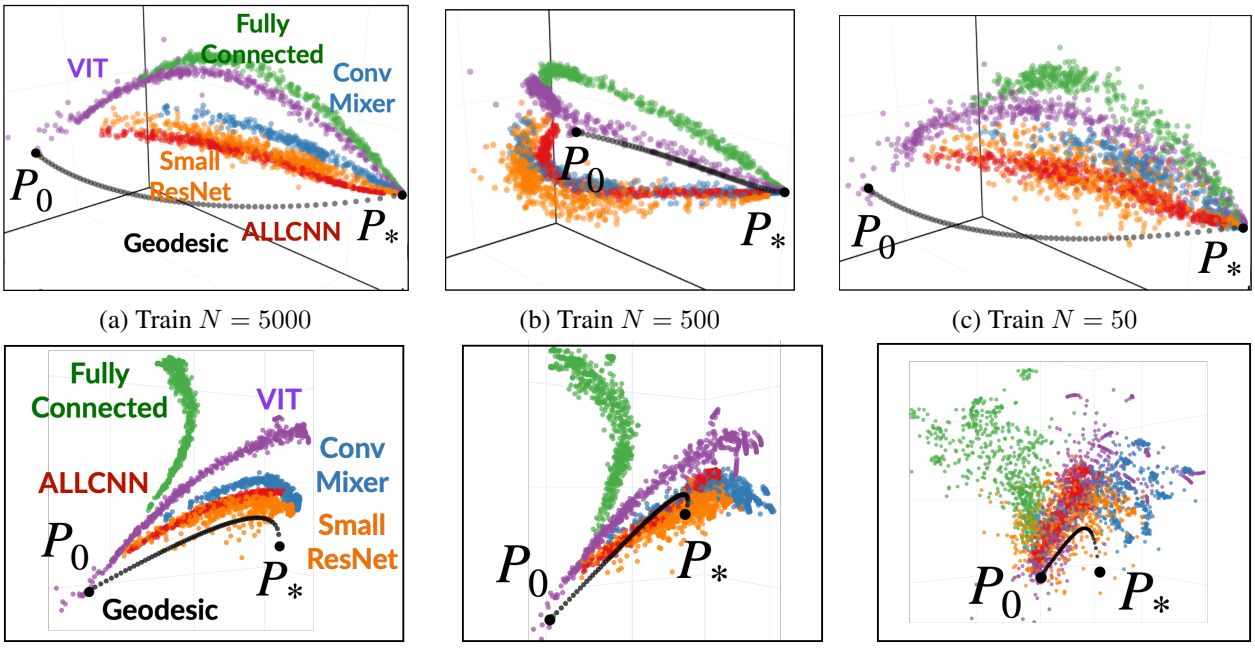
(a) Train manifold (b) Test manifold

Figure 2.13: The explained pairwise Bhattacharyya distances (computed using Equation (2.4.2)) of the embedding when projected onto the principal components computed using a subset of the samples in the train and test data. Even for very small values of N, the explained pairwise distance is close to the explained distance of the original embedding computed from all the samples.

i.e., the relative importance that we would like for it in the visualization. The normalized multiplicities are $\hat{\mu}_u = \mu_u / \sum_{v'} \mu_{v'}$. Weighted-InPCA is a modification of InPCA. It (a) uses a different centering matrix $L_{uv} = \delta_{uv} - \hat{\mu}_u$, (b) performs an eigen-decomposition of $W \text{diag}(\hat{\mu}_u)$, i.e., each column of W is multiplied by $\hat{\mu}_u$, and (c) then scales back each of the eigenvectors U_i using the expression $U_i / \sqrt{U^\top \text{diag}(\hat{\mu}) U}$. This procedure gives the same embedding as the one obtained by repeating points before calculating standard InPCA and is also equivalent to the procedure in Equation (2.2.9) when the weights μ_u of the new points are zero.

2.5.3.3 Computing pairwise distances in InPCA using only a subset of the samples gives a faithful representation of the train and test manifolds

We computed the InPCA coordinates using a subset of the samples in the train and test sets to calculate the pairwise Bhattacharyya distance matrix. Using the procedure in Equation (2.2.9), we then embedded the models in the original pairwise distance matrix computed using all samples into these InPCA coordinates. Figures 2.13 and 2.14 show that the explained pairwise distances by the top three dimensions of these new InPCA embeddings is quite high. This suggests that our visualization methods could be used effectively, even for large datasets with a large number of samples N, by sub-sampling the data before computing InPCA.



(a) Train $N = 5000$ (b) Train $N = 500$ (c) Train $N = 50$

(d) Test $N = 1000$ (e) Test $N = 100$ (f) Test $N = 10$

Figure 2.14: Projecting the original probabilistic models and pairwise Bhattacharyya distances computed on all samples into InPCA coordinates created using a distance matrix on a subset of samples (**(a-c)** for $N = 5000, 500, 50$ respectively for the train data and **(d-f)** for $N = 1000, 100, 10$ respectively for test data). On the train data, even with as few as 1% of the samples, these embeddings are qualitatively similar to the original embeddings (Figures 2.1a and 2.4a). For the test data, explained pairwise distances is low in Figure 2.13b and manifolds are more diffuse.

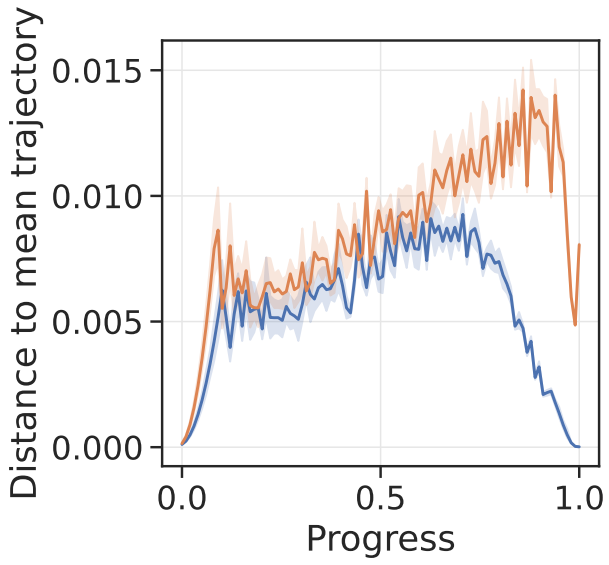
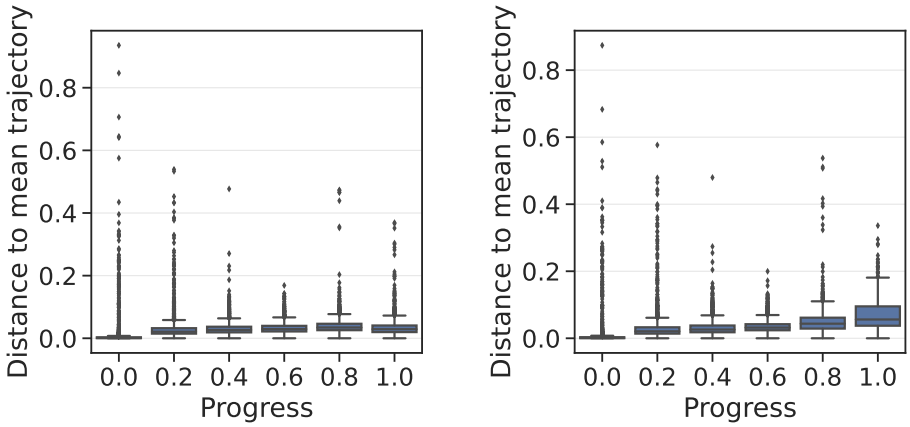


Figure 2.15: Towards the end of training at large values of progress, models trained with augmentation (orange) have larger tube widths than models trained without augmentation (blue), on the train manifold. The corresponding figure for the test manifold looks similar.

2.5.4. Addendum to Results

2.5.4.1 Further analysis of the train trajectories

Understanding the differences between the trajectories of different configurations Using the interpolated trajectories, for each configuration, we calculated the Euclidean mean of the probabilities of the models corresponding to different weight initializations at the same progress. The distance of the model to such a configuration-specific mean model gives us an understanding of the "tube width", i.e., how different in prediction space models with the same progress but corresponding to different weight initializations are. Figure 2.16a shows that—for all configurations, for all values of progress—models are very close to their respective mean model. The median tube width is about 0.05 in terms of Bhattacharyya distance throughout training; this should be compared to the abscissae of Figure 2.6a where a cut at a distance of 0.05 separates all configurations (except some AllCNNs, and very few fully-connected and ConvMixer architectures). The dendrogram in Figure 2.6a averages models for the same progress; Figures 2.16a and 2.16b indicate that such averaging is a reasonable thing to do. The test manifold in Figure 2.16b is similar, except that tube widths increase slightly with progress. This suggests that networks with different weight initializations train along very similar trajectories in prediction space.

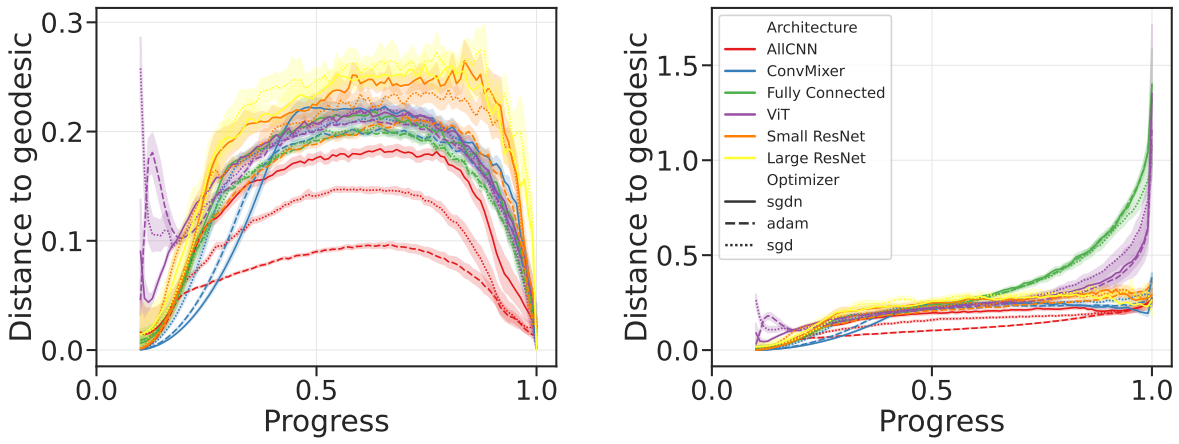


(a) (b)

Figure 2.16: A boxplot (horizontal line denotes median, boxes denote 25 percentile, whiskers denote 1.5× the inter-quantile (25–75 percentile) range) of the Bhattacharyya distance between a model and the Euclidean mean of probabilities of models with the same configuration but obtained from different weight initializations for train **(a)** and test **(b)** trajectories. There are minor differences in tube widths of different configurations and therefore we have not distinguished them here. All tube-widths are quite small, which indicates that training trajectories whose configurations only differ in weight initializations are tightly clustered together in the prediction space.

One can dig deeper into the differences in models caused by weight initialization. Tube widths of different architectures at the same progress are similar on the train manifold, but there are more pronounced differences on the test manifold. We have found that variations coming from optimization methods and regularization do not result in large tube widths. In general, towards the end of training, at large values of progress, models trained with augmentation have larger tube widths than models trained without augmentation, on both train and test manifolds (Figure 2.15). Training a deep network is a non-convex optimization problem, and as such the solution depends upon the initialization of weights in a non-trivial way. Each point in the prediction space corresponds to a large set of weight configurations that lead to this same prediction. Our results therefore suggest that, even if different weight initializations could lead to different eventual weights for these non-convex optimization problems, the probabilistic models obtained at the end of training are very similar (they are more similar on the training data than the test data).

We next study the distances of models along the interpolated trajectories to the geodesic. On the train



(a) (b)

Figure 2.17: Bhattacharyya distance of models with different configurations to the geodesic at different progress for train **(a)** and test **(b)** trajectories.

manifold (Figure 2.17a), all models are very close to the geodesic at the beginning (small progress) and at the end of training (large progress). At intermediate progress, all trajectories have large distances to the geodesic; as we discussed above this deviation away from the geodesic could be an indicator of the range of difficulties of learning different samples. Trajectories corresponding to different architectures and optimization methods are at different distances from the geodesic at intermediate progress. Train trajectories of AllCNN are closest to the geodesic; there are marked differences between the three optimization algorithms in this case. But this is not so for other architectures. For test trajectories (Figure 2.17b), the distance to the geodesic is roughly the same, and larger that that of the train manifold, for all architectures and all values of progress. At large progress, test trajectories of fully-connected and ViT networks are very far from the geodesic; this is also visible in Figure 2.4.

Models initialized at very different parts of the prediction space converge to the truth along a similar manifold

The manifold in our analysis is the set of probabilistic models explored during the training process; this is a subset of the space of all probabilistic models (which is the simplex in $[0,1]^{NC}$ and not low-dimensional). Our manifold is a subset of the manifold of all probabilistic models that can be expressed by the network $\{P_w(\boldsymbol{y}) : \forall w\}$ (which is also not expected to be low-dimensional) because the training process does not explore all parts of the weight space. To understand why our trajectories seem to lie on effectively low-

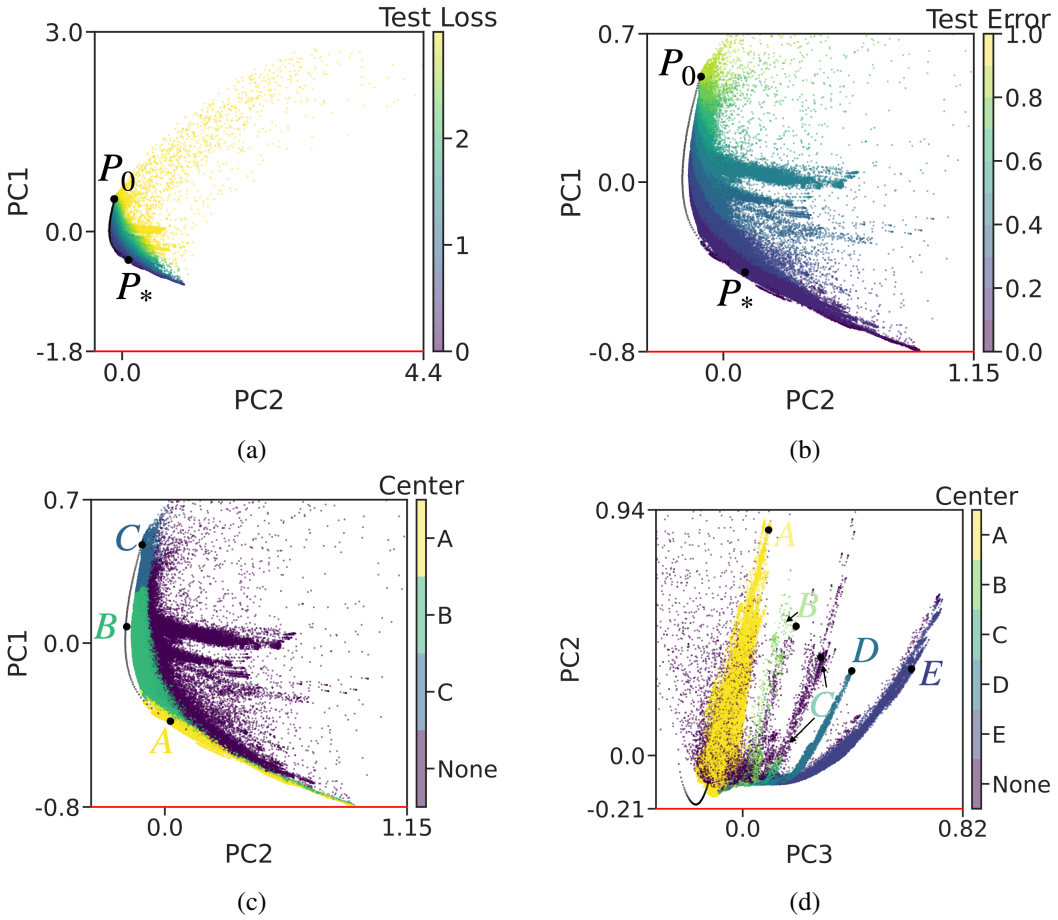


Figure 2.18: Comparison of two principal components of an InPCA embedding using test data of all models on CIFAR-10 colored by test loss **(a)**, by test error **(b)**, by whether they are within a Bhattacharyya distance < 0.3 from models marked A, B, and C on the geodesic in **(c)**, and by whether they are within a distance 0.45 from the models marked A–E in **(d)**. These figures should be studied together with Figure 2.4c.

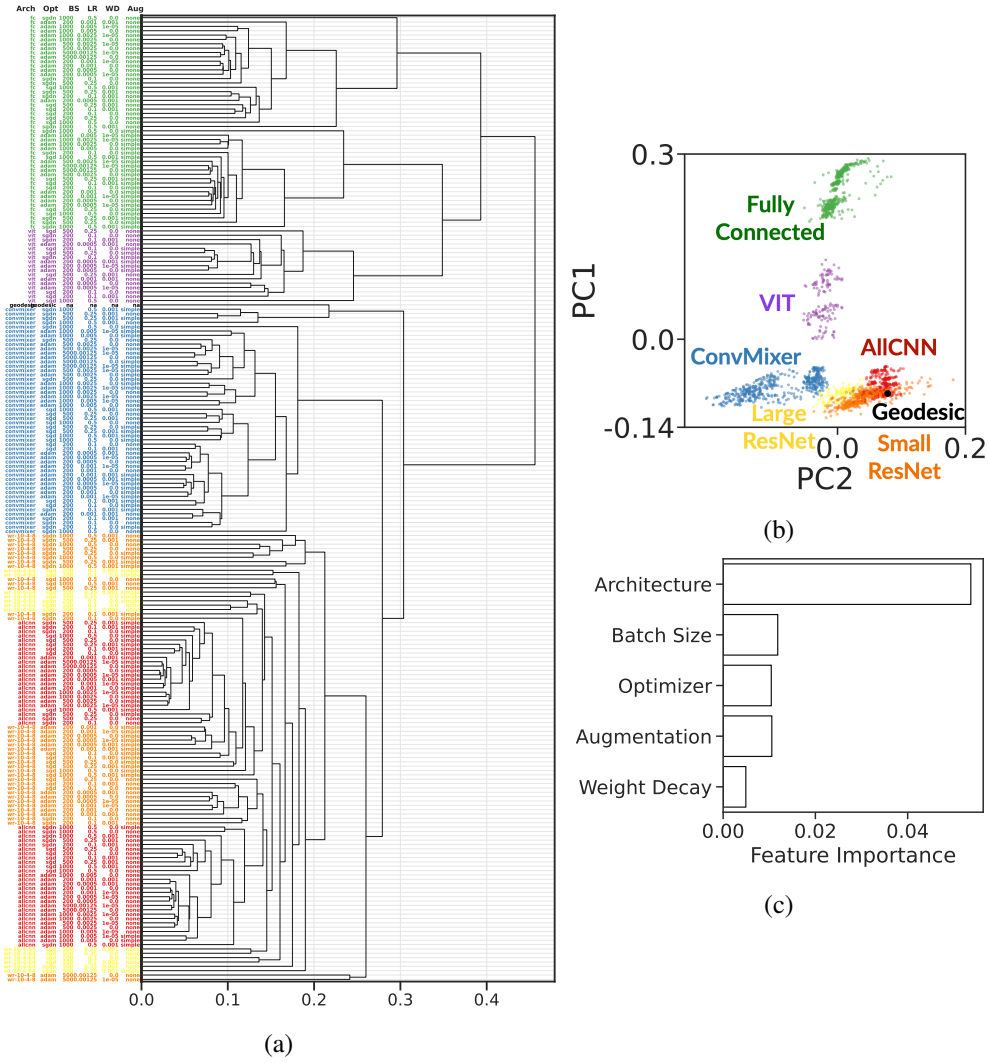
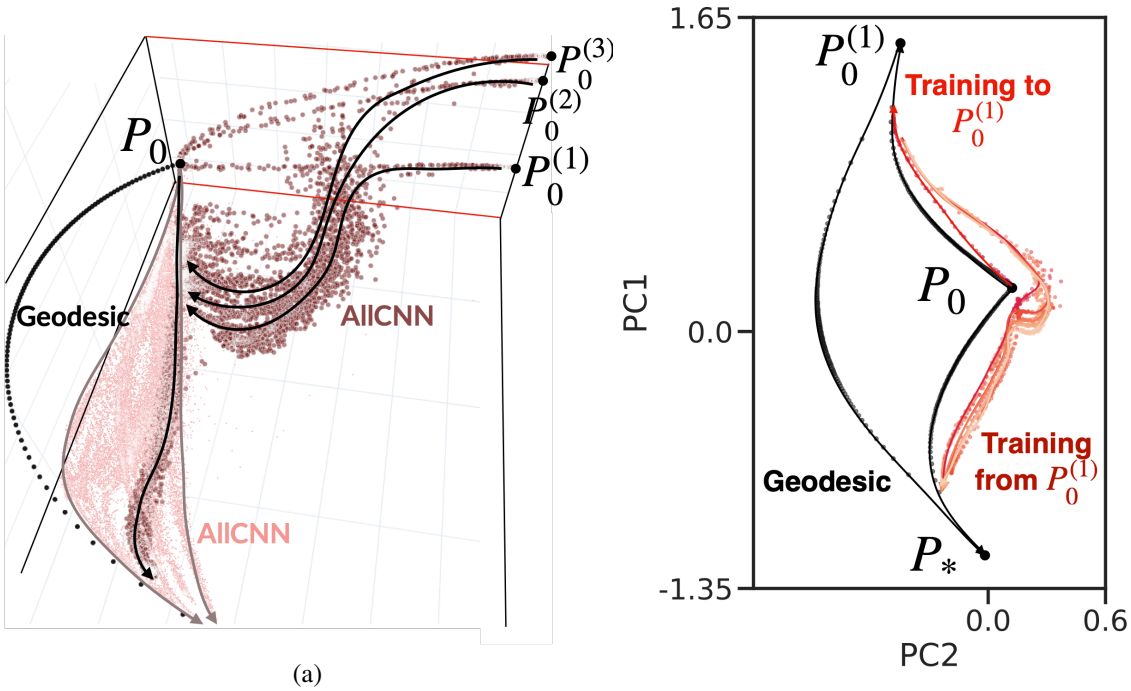


Figure 2.19: **(a)**: dendrogram obtained from hierarchical clustering of pairwise distances (averaged over weight initializations) between trajectories using distances calculated on testing samples. X-labels correspond to architecture, optimization algorithm, batch-size, learning rate, weight-decay coefficient and augmentation strategy. Compared to the equivalent figure on training data Figure 2.6a, trajectories still form clear clusters according to architecture, the distances between different trajectories are in general larger on test data, and the clusters of large and small wide ResNets are less distinguishable. **(b)** the first two components of an InPCA embedding (without averaging over weight initializations) of these trajectories, each point is one trajectory; explained stress of top two dimensions is 73.7%. **(c)** variable importance from a permutation test ($p < 10^{-6}$) using a random forest to predict pairwise distances. These three plots suggest that for test data, architecture is still the primary distinguishing factor of trajectories in the prediction space, and the picture of different trajectories is very similar to those evaluated on training data, even though they appear to have a larger difference in the InPCA embedding.



(a) (b)

Figure 2.20: **(a)** shows the top three dimensions of an InPCA embedding of some configurations with AllCNN architectures when networks are initialized near ignorance and trained to truth P_* (light brown), and when they are first trained to tasks $P_0^{(k)}$ for $k = 1, 2, 3$ with random labels (stream of brown points heading towards these corners) and then further trained to the truth P_*. Trajectories from random tasks join the original train manifold before heading to truth (black curves in **(a)** for trajectories that begin at different random tasks and red in **(b)** for trajectories corresponding to different weight initializations from the same random task). These trajectories are very different from geodesics. We have drawn smooth curves denoting trajectories by hand to guide the reader. Note that the trajectories that begin at corners with random labels rejoin the trajectories that begin from near ignorance quite close to ignorance but along paths

dimensional manifolds, using CIFAR-10, we created three different tasks by randomly assigning labels to the images, e.g., each image of a dog is labeled independently as any of the 10 possible classes. This gives us three random initial models denoted by $P_0^{(k)}$ for $k \in \{1, 2, 3\}$, and we can now train networks to fit these random labels. Both train and test manifolds of training to such random tasks are effectively low-dimensional. This suggests that the low-dimensionality is not necessarily due to there being learnable patterns in the labels.

We next performed a second stage of training where networks were initialized to the endpoints of the trajectories to $P_0^{(k)}$ for $k \in \{1, 2, 3\}$ (models do not reach these points exactly during training), and trained on

the actual CIFAR-10 task, i.e., to the actual truth P_*. In this case, we only trained one particular configuration (AllCNN architecture, SGD without Nesterov's acceleration, no augmentation or weight-decay) from 10 different weight initializations chosen to be near $P_0^{(k)}$. This two-stage training procedure also results in effectively low-dimensional train and test manifolds (Figures 2.20a and 2.21); the top three dimensions explain more than 87% of the stress. It is interesting to note that the networks don't just forget the wrong labels before learning the correct ones, trajectories rejoin the original training trajectory at a variety of points before following it to the truth.

In Figure 2.20b we show the training trajectories to (light red) and from (red) $P_0^{(1)}$, together with the geodesics connecting P_0, P_* and $P_0^{(1)}$. The geodesic from $P_0^{(1)}$ to the truth does not pass near ignorance P_0. In fact, a random task $P_0^{(k)}$ agrees with the truth on approximately $1/C$ of the samples, and the Bhattacharyya distance of the geodesic from $P_0^{(k)}$ to the truth is at least a distance $\log(C)/(2C) + ((C-1)\log(C/2))/(2C)$ (≈ 0.83 for $C = 10$) from ignorance. As a reference, the distance between training trajectories of two different configurations is about 0.15 in Figure 2.6a. Unlike the geodesic from $P_0^{(k)}$, trajectories from $P_0^{(1)}$ come much closer to ignorance; the smallest distance from P_0 ranges from 0.1–0.5 for different weight initializations. There is a large spread in the models near ignorance and trajectories with different weight initializations join along separate paths (Figure 2.20b). After progress of 0.27 ± 0.15 (which is typically achieved within 3 epochs), most models have a distance of less than 0.15 from models that began training from ignorance P_0. This suggests a remarkable picture for the train manifold: not only do trajectories that begin near ignorance P_0 lie on it, but even if trajectories begin at very different parts of the prediction space, they still join this manifold before heading to the truth. Conclusions on test data in Section 2.5.4.2 are similar.

2.5.4.2 Further analysis of trajectories on the test data

Dendrogram and InPCA embedding of test trajectories Figure 2.19 shows a dendrogram, similar to the one in Figure 2.6a, obtained from hierarchical clustering of pairwise distances (averaged over weight initializations) between trajectories using distances calculated on the test samples. Figure 2.19b shows an InPCA embedding of the test trajectories and Figure 2.19c shows a variable importance plot using a random-forest to predict the pairwise distances between test trajectories. The conclusions drawn from these plots on the test data are very similar to those on the train data in Figure 2.6 discussed in the main paper.

Characterizing the details of the test manifold We will first study the spread of points away from the test manifold. Consider Figure 2.18a, which shows points in the first two components colored by their distance to truth P_*. Points colored purple have the smallest distance and the best test loss. This is corroborated by Figure 2.18c where we took three points on the geodesic and colored models in terms of whether they are within a Bhattacharyya distance of 0.3 from these centers. Points that are away from the test manifold at early training times are colored yellow in Figure 2.18a; they consequently have high errors (90% in many cases, colored yellow in Figure 2.18b). We checked that these are the same models that are far from the train manifold near ignorance P_0 (yellow in Figure 2.2b). Some (about half) of these models did not reach zero training error, and correspondingly they also have poor test error.

In Figure 2.18a, we see that there is a large number of models that form a sliver of the manifold near truth P_*; these are primarily ConvMixer and Large ResNet architectures. Their test errors are < 10% (see Figure 2.18b), and their Bhattacharyya distance to the truth is < 1. In the train manifold, the spread in the visualization was coming due to InPCA amplifying small differences in the models, all with zero error, towards the end of training. In the test manifold, these models also have similar predictions (as seen in Figure 2.18c) but they do not have zero error. InPCA is again identifying differences in the underlying probabilistic models.

For the same error, models on the test manifold show a large spread (see Figure 2.18b) as compared to those on the train manifold in Figure 2.2c. In particular, different ConvMixer networks which eventually reach low test errors predict similarly at intermediate levels of train/test error, not only on the training data but also on the test data (blue/purple in Figure 2.18c). But fully-connected networks predict very differently from each other at intermediate errors (error of, say 0.3–0.4 in Figure 2.18b), i.e., their spread is more pronounced on the test manifold. This could indicate that architectures with strong inductive biases (e.g., convolutions) explore a smaller part of the prediction space, even on the test data. It has implications for theoretical analyses of generalization in deep learning using ideas such as algorithmic stability.

Using PC2 and PC3, in Figure 2.18d, we chose five specific endpoints, corresponding to fully-connected and ViT networks trained with and without augmentation (B–E), and for comparison, one more endpoint from the trajectory of ConvMixer trained with augmentation (A). We colored models in terms of whether

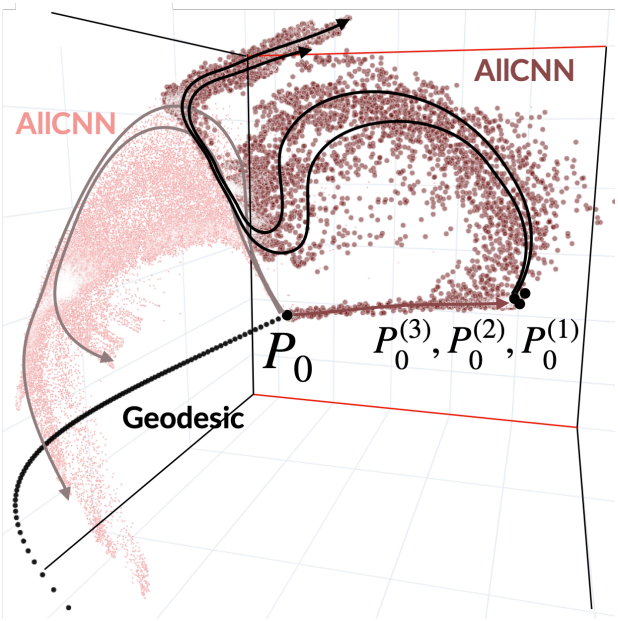


Figure 2.21: Predictions on test set of a subset of AllCNN models (the same set as in Figure 2.20) trained from ignorance (light brown) and from three different corners (dark brown). Networks trained from corners still seem to come close to the normally trained models mid-training, but they divert from the main manifold and end at a higher testing error in the later part of training.

they lie within a Bhattacharyya distance < 0.45 from their closest center. Models colored purple are far from all centers. For fully-connected and ViT networks, models having the same test error can lie in very different parts of the test manifold. For example, for test error within 0.3–0.4 (see Figure 2.18b) some models lie on the manifold (e.g., green in Figure 2.18c), some on one branch (e.g., one of the purple branches or the smaller green branch in Figure 2.4c), while some others can lie on other branches (e.g., other purple branches in Figure 2.4c).

Models initialized at very different parts of the prediction space converge to the truth along a similar manifold For the test data, there is a larger spread in how models initialized near $P_0^{(k)}$ join the main manifold, and also how their endpoints are different from endpoints of trajectories that begin near ignorance P_0 (see Figure 2.21).

2.5.4.3 Observations remain consistent with other intensive distances

We can also use other distances in place of the Bhattacharyya distance. For example, the IsKL method (Teoh et al., 2020) uses the symmetrized Kullback-Leibler (KL) divergence to compute the distances between pairs of

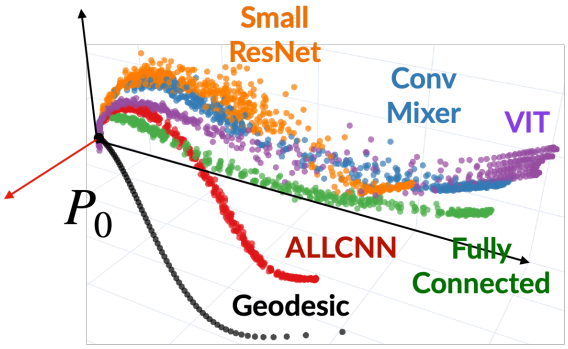


Figure 2.22: The top three dimensions of the IsKL embedding using the train data for a subset of the models trained on CIFAR-10 (this is the same subset as in Figure 2.5a). The IsKL embedding carries a different kind of information than the InPCA embedding in Figures 2.1a and 2.4a. Trajectories exhibit a larger spread towards the end of training and truth P_* (not seen here) is at infinity. The IsKL embedding emphasizes the differences among the trajectories towards the end of training.

points D in Equation (2.2.6)

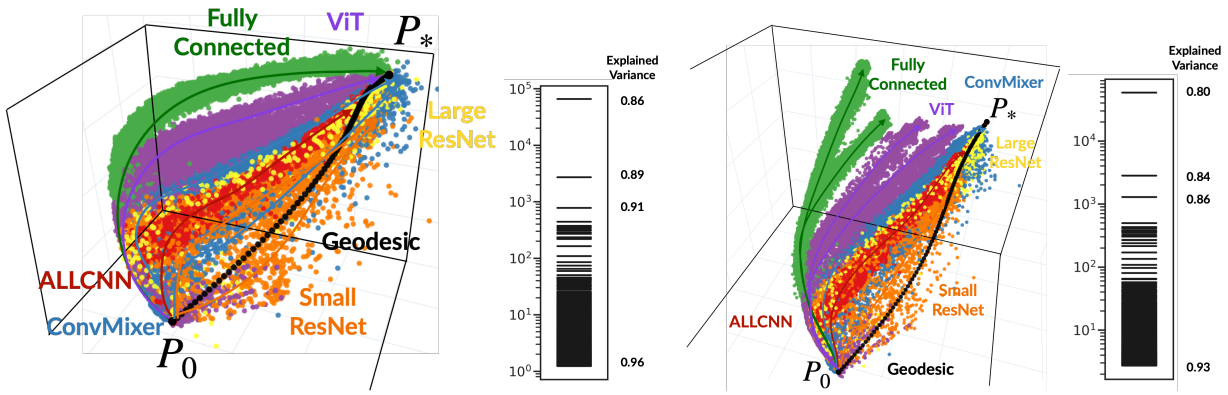
$$\mathrm{d}_{\mathrm{sKL}}(P_u, P_v) = \frac{1}{N}\sum_{n=1}^{N}\sum_{c=1}^{C}(p_u^n(c) - p_v^n(c))\log\left(\frac{p_u^n(c)}{p_v^n(c)}\right). \tag{2.5.3}$$

For exponential families, we can obtain an analytical formula for the IsKL embedding and in this case, the embedding has at most twice the number of dimensions as the dimensionality of the sufficient statistic (for CIFAR-10, this has 9×10^5 dimensions). Our models P_u and P_v are vectors that lie on a sphere of radius N (probabilities of each image sum up to 1). We could also use the geodesic distance on this sphere

$$\sqrt{N}\cos^{-1}\prod_{n=1}^{N}\sum_{c=1}^{C}\sqrt{p_u(y_n)}\sqrt{p_v(y_n)};$$

but this has poor behavior in high dimensions because points along the trajectory jump abruptly from ignorance to truth. This is similar to the saturation of the Hellinger distance in high dimensions that is discussed in the main text. Since our models live on a product space of hyper-spheres (samples in the dataset are independent of each other) we can use the geodesic distance on the product of spheres instead

$$\mathrm{d}_{\mathrm{G}}(P_u, P_v) = \frac{1}{N}\sum_{n}\cos^{-1}\sum_{c}\sqrt{p_u^n(c)}\sqrt{p_v^n(c)}. \tag{2.5.4}$$



(a) (b) (c) (d)

Figure 2.23: The top three dimensions of an embedding obtained using standard PCA for all the networks on CIFAR-10 using train data **(a)** and the test data **(c)**. The explained variance in **(b,d)** for train and test data respectively is very high but the structure of the low-dimensional manifold identified by PCA is very different from that obtained by InPCA in Figures 2.1a and 2.4a. In particular, although this embedding is low-dimensional it does not respect the natural metric in probability space because the second derivative of the divergence is not the same Fisher Information Matrix as that of, say, the Bhattacharya distance.

All the above distances respect the natural Fisher Information Metric in probability space. The IsKL, InPCA and Geodesic embeddings carry different pieces of information on the structure of the space of probability distributions. For example, IsKL places truth P_* infinitely far away, and it therefore stretches the last part of the training trajectories in our experiments. This allows us to investigate the behavior of trajectories towards the end of training in more detail (although we do not do so in this paper). We have noticed in smaller-scale experiments that IsKL captures a slightly higher explained stress in the top three dimensions that InPCA. The geodesic embedding maps geodesics to straight lines which may be useful to construct a simpler, more interpretable, set of coordinates.

Embeddings using standard principal component analysis (PCA) Our data consists of probability distributions and therefore a meaningful embedding of such data should seek to preserve distances between probability distributions. But it is reasonable to ask how well standard dimensionality reduction and embedding techniques, e.g., standard principal component analysis (PCA), can reveal the inherent low-dimensional structure in the data. For this calculation, we created a matrix of pairwise distances

$$D_{uv} = \frac{1}{N} \sum_n \sum_c (p_u^n(c) - p_v^n(c))^2$$

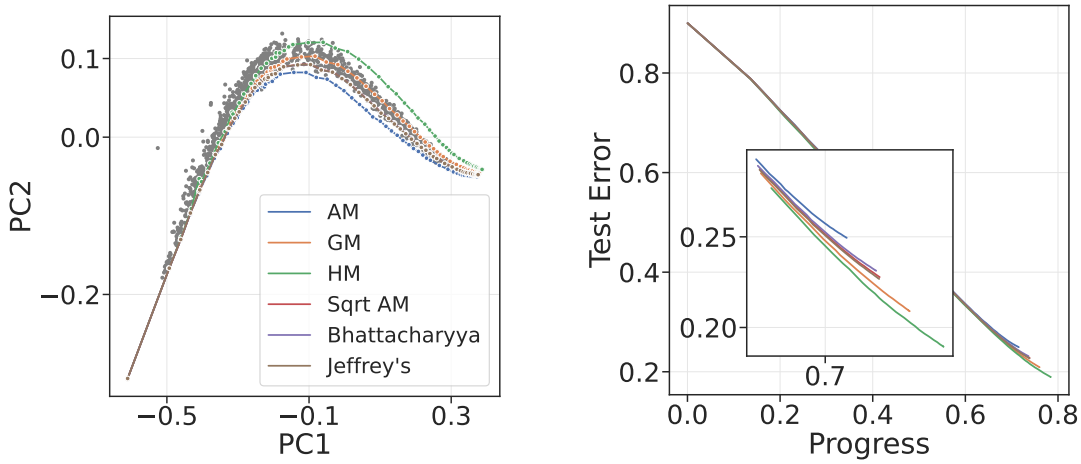
and computed the eigen-decomposition of this matrix (after centering) to get the coordinates. One should note two important choices here: (a) the Euclidean distance between the probability distributions $p_u^n(\cdot)$ and $p_v^n(\cdot)$ treats them as standard vectors in $\mathbb{R}^C$, and (b) the averaging over the samples using N^{-1} ensures that D_{uv} remains non-trivial even for a large number of samples.

We show an embedding calculated using PCA for the train and test manifolds in Figure 2.23a and Figure 2.23c respectively. In both cases, an embedding using PCA suggests that the data lies on an effectively low-dimensional manifold, the explained variance is quite large (91% and 86% in the first three dimensions for train and test manifolds respectively). This is consistent with the results we have discussed using InPCA in the main text. But because it uses an unusual distance between probability distributions, PCA distorts the structure of the manifold as compared to InPCA. The salient differences are as follows: (a) trajectories corresponding to different architectures are very close to each other in Figure 2.1a and Figure 2.6a but there are marked differences in these trajectories in Figure 2.23a; (b) the geodesic is far from all trajectories in the original data but this is not so in the PCA embedding; (c) the cloud of points that lie away from the main manifold, which we have analyzed in Figure 2.2, is not visible in the PCA embedding. For the test manifold, we see some similarities between Figures 2.4a and 2.23c: (a) there are multiple branches for fully-connected and ViT networks; and (b) networks that obtain good test error (ConvMixer and Large ResNet) are closer to the truth. There are also some differences: (a) the geodesic is far from all trajectories in the InPCA embedding while it is close to the trajectories of the Small ResNet in the PCA embedding; (b) InPCA reveals the fact that trajectories of AllCNN are closest to the geodesic in terms of the Bhattacharyya distance for both train and test manifolds (Figure 2.17) but PCA does not show this.

Altogether, while we can corroborate the claim that the trajectories explore an effectively low-dimensional manifold of predictions on both the train and test data using both methods, PCA distorts the structure of the manifold and conclusions that one may derive from the embedding are not consistent with those derived from analysis of the trajectories in the original high-dimensional space. Also, observe that InPCA distinguishes the small differences between the probability distributions towards the end of training while PCA does not.

Divergence $d(p, q)$	Centroid $(p^{(1)}, p^{(2)}, \dots)$	
$\sum_n (p_n - q_n)^2$	$\propto \sum_k p_n^{(k)}$	Arithmetic mean (AM)
$\sum_n (\sqrt{p_n} - \sqrt{q_n})^2$	$\propto \sum_k \sqrt{p_n^{(k)}}$	Sqrt. Arithmetic mean
$\sum_n (\log p_n - \log q_n)$	$\propto (\prod_k p_n^{(k)})^{1/N}$	Geometric mean (GM)
$\sum_n n(p_n^{-1} - q_n^{-1})$	$\propto (\sum_k 1/p_n^{(k)})^{-1}$	Harmonic mean (HM)
$-\log\left(\sum_n \sqrt{p_n}\sqrt{q_n}\right)$		Bhattacharyya centroid (Nielsen and Boltz, 2011)
$\sum_n (p_n - q_n)\log(p_n/q_n)$	AM/W (eAM/GM)	Jeffrey's centroid (Nielsen, 2013)

Table 2.3: **Different divergences and their corresponding centroids.** We have showed the formulae for two N-dimensional probability distributions $(p_n)_{n=1,\dots,N}$ and $(q_n)_{n=1,\dots,N}$ and the centroid of a set of distributions $\{p^{(1)}, p^{(2)}, \dots\}$. The Lambert omega function is denoted by $W(\cdot)$ and e is Euler's number.



(a) (b)

Figure 2.24: **(a)**: the top two principal components obtained from InPCA for the train data for one particular configuration on CIFAR-10 (AllCNN architecture, trained with SGD without augmentation or weight-decay). We computed the arithmetic mean (AM), geometric mean (GM), harmonic mean (HM), the arithmetic mean of the square roots of probabilities appropriately normalized (Sqrt AM), the Bhattacharya centroid and Jeffrey's centroid for models with the same progress. It is noticeable that different means do not always lie on the manifold. In particular, the arithmetic mean and the harmonic mean are the farthest away visually. **(b)**: the test error as a function of progress for the different ways of computing the mean. The test errors are AM (25.0%), GM (20.9%), HM (18.9%), Sqrt AM (22.8%), Bhattacharyya centroid (23.1%), Jeffrey's centroid (22.7%): therefore computing the harmonic mean of the probabilities of the models in the ensemble leads to a slightly better test error than computing the arithmetic mean of their probabilities which is typically done in machine learning.

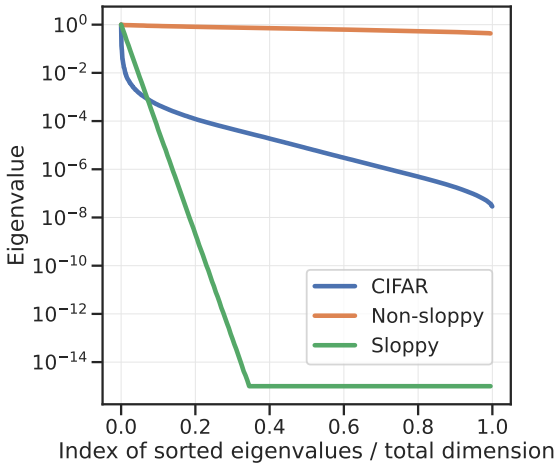


Figure 2.25: Eigenvalues of the the input correlation matrix $\mathbb{E}[xx^\top]$ for 32×32 RGB images x in CIFAR-10 (blue), i.e., $x \in \mathbb{R}^{3072}$, non-sloppy synthetic inputs ($x \in \mathbb{R}^{200}$) sampled from an isotropic zero-mean Gaussian (orange) and sloppy synthetic inputs ($x \in \mathbb{R}^{200}$) sampled from a Gaussian distribution with zero mean and covariance matrix whose eigenvalues decay as $\lambda_i = 50c\exp(-ci)$ for $c = 0.5$ (green).

2.5.4.4 Harmonic mean of an ensemble of deep networks has a better test error

We saw previously that a small network with higher eventual test error trains along the same manifold as that of a large network with lower eventual test error, more slowly. There is a classical technique that also achieves better test errors, namely ensembling. We therefore investigated whether an ensemble also exhibits higher progress towards the truth than that of the individual models that constitute the ensemble.

The standard way of building an ensemble in machine learning is to calculate the arithmetic mean of the class probabilities; this corresponds to the ℓ_2 distance in the space of probability distributions. As Table 2.3 shows, different distances correspond to different ways of computing the centroid. We choose five other candidates: (i) the arithmetic mean of the square roots of the probabilities, which corresponds to the centroid of the Hellinger distance, (ii) the geometric mean, (iii) the harmonic mean, (iv) the centroid of the Bhattacharyya distance, which can be calculated using an iterative procedure given in Nielsen and Boltz (2011), and (v) Jeffrey's centroid which corresponds to the symmetric KL-divergence which is known in closed-form (Nielsen, 2013). In Figure 2.24, for 30 different weight initializations, for both train and test trajectories pertaining to one particular configuration (AllCNN architecture, trained with SGD without augmentation or weight-decay), we show these different centroids, after the same number of mini-batch updates for each model.

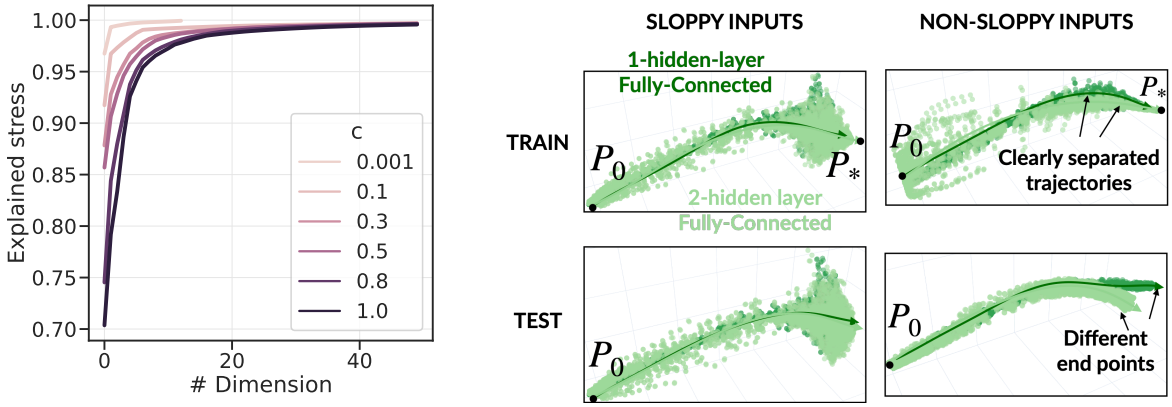
The arithmetic mean lies noticeably outside the manifold in the visualization for both train and test manifolds. Different centroids have different trajectories in the embedding. But the harmonic mean (green) makes the highest progress towards the truth on the test manifold and also has the lowest test error at the end Figure 2.24b. This suggests that ensembles that use the harmonic mean of the probabilities to compute the final model could lead to a slightly better test error.

2.5.5. Experiments using synthetic data

Datasets We sampled $N = 5000$ samples for the training set and $N = 1000$ samples for the test set from a $d = 200$ dimensional Gaussian with mean zero and a diagonal covariance $\Lambda = \text{diag}(\lambda_1, \ldots, \lambda_d)$. We experimented with two types of data: those sampled from an isotropic Gaussian ($\Lambda = I_{d\times d}$) and those sampled from a Gaussian distribution with a covariance matrix that has eigenvalues that decay linearly on a logarithmic scale, i.e., $\lambda_i = 50ce^{-ci}$. The latter setup is the so-called sloppy dataset Yang et al. (2022); Transtrum et al. (2011b). We can control the sloppiness of the dataset by choosing different values of c, i.e., larger the value of c, sharper the decay. We created a 5-class classification problem using labels from a teacher (a fully-connected network with one hidden layer of width 50). The largest logit among the 5 logits of the teacher is taken to be the ground-truth label. We train student networks of different architectures using these teacher-generated labels using the cross-entropy loss. All networks were trained with batch-normalization and without dropout.

Neural architectures and training procedure We studied the difference in training trajectories when networks are trained on data with different sloppiness. We used two values: $c = 0.001$ (which is effectively non-sloppy data) and $c = 0.5$ (which is sloppy data). We trained 160 different configurations: (1) fully connected networks of one and two hidden layers (both with a width of 512), (2) training with SGD and SGD with Nesterov's momentum of coefficient 0.9, (3) two values of batch-size 200 and 500, (4) two values of the weight decay coefficient $\{0, 10^{-5}\}$, and (5) 10 different weight initializations.

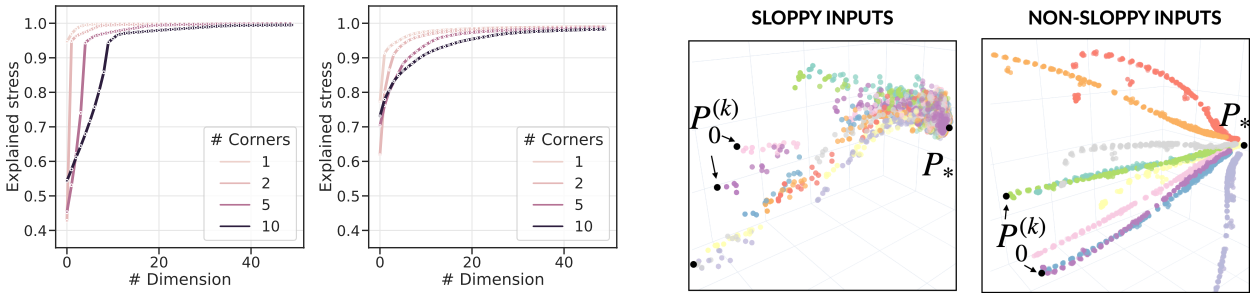
Analysis Train and test manifolds are effectively low-dimensional for both sloppy and non-sloppy data. Figure 2.26a shows how the explained stress increases in the top few dimensions of the InPCA embedding; it reaches 99% in the first 10 dimensions of an InPCA embedding. In general, when inputs are sloppy (larger value of c is more sloppy inputs), the explained stress is slightly lower. We speculate that this is due to the



(a) (b)

Figure 2.26: **(a)**: the explained stress in the top few dimensions (X-axis) of an InPCA embedding of models along training trajectories when input data are sampled from a Gaussian distributions with zero mean and covariance matrix whose eigenvalues decay as $\lambda_i = 50c\exp(-ci)$ for different values of c. For all values of c (small values indicate that inputs were sampled from a near-isotropic Gaussian and large values indicate that input data were sampled from a Gaussian with a sloppy covariance matrix), the explained stress is high. **(b)**: the top three dimensions of an InPCA embedding of models along train and test trajectories for synthetic sloppy and non-sloppy input data for two different architectures (1-hidden-layer fully-connected networks in dark green and 2-hidden-layer fully-connected networks in light green) and multiple training configurations for each architecture.

increased difficulty of the underlying optimization problem which makes the details of the optimization procedure, e.g., the learning rate, important—and thereby leads to a larger spread in the models of different configurations. The explained stress on test data is essentially the same. As the embeddings in Figure 2.26 show qualitatively, when input data is not sloppy, training trajectories show a more clear separation between different training configurations. It is therefore important to choose the architecture (in this case) when we fit models on non-sloppy data. On the other hand, if input data is sloppy, choosing the architecture or the parameters of the optimization algorithm carefully is less important. We noticed that the larger spread of the points in the InPCA embedding towards the end of training near P_* in Figure 2.26b is coming from models trained with SGD with Nesterov's acceleration. A heuristic explanation of this phenomenon, using a linear regression objective for sloppy vs. non-sloppy data, is that overshoots in the weight space caused by momentum terms in Nesterov's acceleration lead to more diverse trajectories if the underlying objective is not isotropic.



(a) Non-sloppy input data (b) Sloppy input data (c)

Figure 2.27: **(a,b)**: the explained stress of an InPCA embedding of training trajectories that are initialized at different parts ("corners") of the prediction space for non-sloppy and sloppy data respectively. For each setting we have chosen the same number of trajectories, i.e., 200 trajectories for 1 corner, 100 trajectories each from 2 corners etc. **(c)**: the top three dimensions of an InPCA embedding of models along train trajectories for sloppy and non-sloppy input data; colors indicate trajectories trained from different corners $P_0^{(k)}$. For sloppy input data, trajectories that begin at different corners quickly converge to the same manifold before heading to the truth P_*, but there is a larger spread in the points near the truth.

We next investigated the effect of initialization. We sample weights of the fully-connected layers from a standard Gaussian distribution without scaling down the variance like that in the default PyTorch initialization. Due to this, the largest output probability of the network at initialization is close to 1 (as opposed to close to 0.2 for the standard initialization when there are 5 classes). Effectively, such models are near the corners of the probability simplex. We sampled 10 such corners and 50 weight initializations using the standard initialization for each corner; this gives 50 different probabilistic models (each, for two optimization algorithm: SGD and SGD with Nesterov's acceleration, and two values of weight-decay) near each of the 10 corners to begin training from. We only used a one hidden-layer fully-connected network for training from the corners. These networks were trained towards the truth P_* with a fixed batch size (100) and two values of the weight decay coefficient (0 and 10^{-5}). For both sloppy and non-sloppy data, this gives 200 trajectories from each of the 10 corners to be used for analysis. With the number of trajectories fixed to 200, we performed an InPCA embedding of models along trajectories starting from different corners, e.g., 200 trajectories from one corner, 100 trajectories each from two corners, 40 trajectories each from 5 corners, etc.

Again, as Figures 2.27a and 2.27b show for non-sloppy and sloppy data respectively, the explained stress of an InPCA embedding of models along these trajectories in the top few dimensions is high. The explained stress captured by the top three dimensions for sloppy data is higher; this is because trajectories beginning

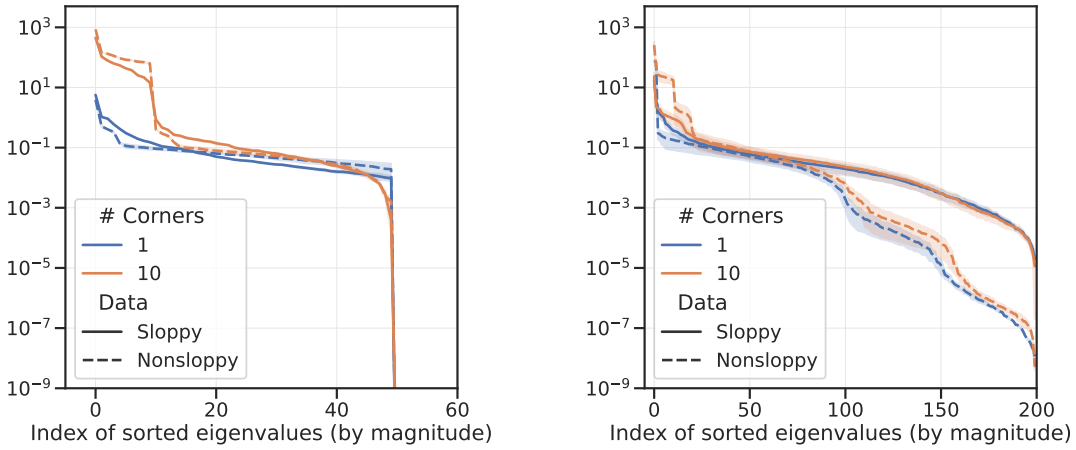


Figure 2.28: **(a,b)**: eigenvalues of the pairwise distance matrix D (see Equation (2.2.6)) of InPCA of models trained from corners at the beginning of training and after 0.5% of the training epochs, respectively. Our goal was to slice the tube of trajectories of networks with different weight initializations corresponding to the same configuration and investigate the dimensionality of the constituent models in this slice. In both cases, for both sloppy and non-sloppy input data, even if the slice is not low-dimensional the trajectories themselves in Figure 2.27 are effectively low-dimensional.

from different corners look very similar in Figure 2.27c for such data. For non-sloppy data, even if the explained stress is lower in the top three dimensions (the InPCA embedding in Figure 2.27c shows a clearer separation between trajectories), the explained stress is much higher if the embedding has more dimensions. This is indicative of the difficulty in optimization for sloppy input data (one can also see a larger spread towards the end of training in Figure 2.27c for sloppy data).

The initial 200 probability distributions (corresponding to 50 weight initializations, 1 architecture, 2 different optimization algorithms and 2 different values of weight-decay) do not lie on a low-dimensional manifold, see Figure 2.28a. In fact, the 200 probability distributions corresponding to models at an intermediate point of training (after 0.5% of the total number of epochs) also do not lie on a low-dimensional manifold, see Figure 2.28b. So it is remarkable that in Figure 2.27, the manifold formed by 200 trajectories across 4 training configurations that begin that these initializations can be embedded into a low-dimensional space faithfully (dynamics in the prediction space is clearly nonlinear). This is yet another evidence of the effectiveness of InPCA at plucking out structure in high-dimensional data.

These experiments on synthetic data suggest that, both initialization near ignorance in the prediction space and the spectral properties of the input data, could be the reason for the low-dimensionality of the train and test manifolds.

CHAPTER 3

LOW-DIMENSIONAL MANIFOLD IN THE SPACE OF TASKS

3.1. Introduction

In Chapter 2 we have been studying training towards one vertex of the model manifold, and in our current problem, the only vertex that is meaningful. Recent success in foundation models suggests models can adapt to different tasks after some finetuning. We therefore wish to understand the model manifold of the space of tasks. We adopt the techniques developed in Chapter 2 to understand the structure of the space of learnable tasks and study different representation algorithms such as supervised, transfer, meta, semi- and self-supervised learning. Unlike the previous scenario, in this section we will consider different tasks with the same input domain but possibly a different output domain. We will however assume the target domain for each task belongs to a larger unified domain.

We show in this chapter that the low-dimensionality we observed in single-task learning is also present in multi-task learning: for Imagenet, the top 3 dimensions preserve 80.02% of the pairwise distances between 2430 models trained on different sub-tasks of Imagenet with total dimension of 10^7. Similar to the separation by architecture, we also observe differences in the trajectories taken by different learning algorithms: we observe episodic meta-learning algorithms and supervised learning traverse different trajectories in the space of probabilistic models during training but learn similar models eventually; the trajectory of episodic meta-learning tends to be longer. Contrastive and semi-supervised learning methods traverse similar trajectories to that of supervised learning in the space of probabilistic models.

We also demonstrate how the techniques we developed before could help reveal interesting hidden information in the learning procedure in multiple settings. In particular, we observe that supervised learning on one task results in a surprising amount of progress on seemingly dissimilar tasks; progress on other tasks is larger if the training task has diverse classes, and structure of the space of tasks indicated by our analysis is consistent with parts of the Wordnet phylogenetic tree.

3.1.1. Contributions

We discuss theoretical and computational tools to study such probabilistic models in Section 3.2. Many of these tools are developed in more detail in Mao et al. (2024). These tools are used to visualize these very high-dimensional objects, to compute geodesics on such manifolds, to interpolate checkpoints along training trajectories into continuous curves, and to map models trained on different tasks into a unique prediction space. We point these technical tools to understanding the structure of the space of learnable tasks and study different representation algorithms such as supervised, transfer, meta, semi- and self-supervised learning. We report the following findings in Section 3.3:

(1) The manifold of probabilistic models trained on different tasks using different representation learning methods is effectively low-dimensional. This dimensionality is very small: for Imagenet where our probabilistic models are in 10^7 dimensions, the top 3 dimensions preserve 80.02% of the pairwise distances between 2430 models trained on different sub-tasks of Imagenet.

(2) Supervised learning on one task results in a surprising amount of progress on seemingly dissimilar tasks (informally, "progress" means that the representation learned on one can be used to make accurate predictions on other tasks; this is defined in Equation (2.2.4)); progress on other tasks is larger if the training task has diverse classes.

(3) Structure of the space of tasks indicated by our analysis is consistent with parts of the Wordnet phylogenetic tree.

(4) Episodic meta-learning algorithms and supervised learning traverse different trajectories in the space of probabilistic models during training but learn similar models eventually; the trajectory of episodic meta-learning for a small "way" is about $40\times$ longer in terms of its Riemann length than that of supervised learning.

(5) Contrastive and semi-supervised learning methods traverse similar trajectories to that of supervised learning in the space of probabilistic models.

(6) Fine-tuning a model upon a sub-task does not change the representation much if the model was trained

for a large number of epochs.

We present evidence and analysis of these findings using multiple neural architectures and a large number of different image-classification tasks created from the CIFAR-10 and Imagenet datasets.

3.2. Methods

To be able to compute Bhattacharyya distances between models trained on different tasks, we map the last-layer representation of the trained network to a common output space by imprinting.

Mapping a model trained on one task to another task using "imprinting" In this paper, we will consider different tasks $\{P^k\}_{k=1,\ldots,}$ with the same input domain but possibly different number of classes C^k. Given a model P^1_w parametrized by weights w for task P^1, we are interested in evaluating its learned representation on another task, say, P^2. Let $w = (w_1, w_2)$ be the weights for the backbone and the classifier respectively. The logits are $\mathbb{R}^{C^1} \ni {w_2}^\top \varphi(x; w_1)$ corresponding to an input x and features of the penultimate layer $\varphi(x; w_1)$. The network's output $p_w(c \,|\, x_n)$ for $c = 1, \ldots, C^1$ is computed using a softmax applied to the logits. If we have learned w from one task P^1, then we can re-initialize each row of the classifier weights $(w_2)'_c$ for $c = 1, \ldots, C^2$ to maximize the cosine similarity with the average feature of samples from task P^2 with ground-truth class c:

$$(w_2)'_c = h / \left\|h\right\|_2 \quad \text{where } h = \textstyle\sum_{\{x: y^*_x = c\}} \varphi(x; w_1). \tag{3.2.1}$$

The new network $w = (w_1, {w_2}')$ can be used to make predictions on P^2. Using imprinting, we can map a trajectory τ^1_w of a network being trained on P^1 to another task P^2 by mapping each point along the trajectory; let us denote this mapped trajectory by $\tau^{1\to 2}_w$.

Remark 3.2.1 (Models with different intermediate representations can have zero Bhattacharyya distance). Two models can have different internal representations and yet define identical probabilistic models. For example, a representation and a rotated version of the same representation can define identical probabilistic models if this rotation is undone before the output. The Bhattacharyya distance Equation (2.2.2) only depends on the output probabilities and would be zero if the probabilistic models are identical. Focusing the theory on the probabilistic model that makes the predictions as opposed to the feature space therefore allows us to capture many symmetries in the prediction space.

Remark 3.2.2 (Imprinting versus training the final layer or probing). There are many ways of performing such a mapping, e.g., one could fine-tune the weights using data from P^2, linear probing (Shi et al., 2016), etc. The technique described above is known as "imprinting" (Hu et al., 2015; Qi et al., 2018; Dhillon et al., 2020). In this paper, we will be mapping thousands of models across different trajectories to other tasks. Training the final layer, or a new classifier, for all these models is cumbersome and imprinting provides a simple way around this issue. Note that imprinting is not equivalent to training the classifier w_2 (with backbone w_1 fixed) using samples from the other task but we found that imprinted weights work well in practice (see Section 3.5.4).

How to choose an appropriate task to map different models to? Consider the training trajectory τ_u^1 of a model being trained on P^1 and another trajectory τ_v^2 of a model being trained on P^2. Using Equation (3.2.1), we can map these trajectories to the other task to get $\tau_u^{1\to 2}$ and $\tau_v^{2\to 1}$. This allows us to calculate, for instance, $\mathrm{d}_{\text{traj}}(\tau_u^{1\to 2}, \tau_v^2)$ using Equation (2.2.5) which is the distance of the trajectory of the model trained on P^1 and then mapped to P^2 with respect to the trajectory of a model trained on task P^2. If the two learning tasks P^1 and P^2 are very different, (e.g., Animals in CIFAR-10 and Vehicles in CIFAR-10), then this distance will be large.

Quantities like $\mathrm{d}_{\text{traj}}(\tau_u^{1\to 2}, \tau_v^2)$ or $\mathrm{d}_{\text{traj}}(\tau_v^{2\to 1}, \tau_u^1)$ are reasonable candidates to study similarities between tasks P^1 and P^2, but they are not equal to one another. We are also interested in doing such calculations with models trained on many different tasks, and mapping them to each other will lead to an explosion of quantities. To circumvent this, we map to a unique task whose output space is the union of the output spaces of the individual tasks, e.g., to study P^1 (Animals) and P^2 (Vehicles), we will map both trajectories to P^{U} which is all of CIFAR-10. We will use

$$\mathrm{d}_{\text{traj}}(\tau_u^{1\to \mathrm{U}}, \tau_v^{2\to \mathrm{U}}) \tag{3.2.2}$$

as the distance between trajectories trained on P^1 and P^2.

3.3. Results

We next describe our findings using the theoretical ideas developed in the previous section. We present a broad range of evidence and analysis using a large number of representation learning techniques, multiple neural architectures and a large number of different image-classification tasks created from the CIFAR-10

and ImageNet datasets. Experiments in this paper required about 30,000 GPU-hours. Section 3.5.1 describes the setup for these experiments in detail. One more result, Result 7: Contrastive learning methods trained on different datasets learn similar representations, is presented in Section 3.5.3.

Remark 3.3.1. All the analysis in this paper (except Figures 3.4 and 3.5) was conducted using the test data. All models were trained using the training data, but all mapped models, distances between trajectories, quantitative evaluation of progress and InPCA embeddings were computed using the test dataset. The reason for this is that we would like to study the geometry of tasks as evidenced by samples that were not a part of training. To emphasize, we do not develop any new algorithms for learning in this paper. Therefore using the test data to quantify relationships between tasks is reasonable; see similar motivations in Kaplun et al. (2022) or Ilyas et al. (2022) among others. Our findings remain valid when training data is used for analysis; this is because in most of our experiments, a representation is trained on one task but makes predictions on a completely new task after mapping.

Result 1: The manifold of models trained on different tasks, and using different representation learning methods, is effectively low-dimensional We trained multiple models on 6 different sub-tasks of ImageNet (from 5 random initializations each) to study the dimensionality of the manifold of probabilistic models along the training trajectories (100 points equidistant in progress Equation (2.2.4)) after mapping all models to all ImageNet classes ($\sim 10^8$ dimensions). We use the explained stress (defined in Equation (2.2.8)), to measure if the distances are preserved by the first k dimensions of the embedding of the models. The first 3 dimensions of InPCA (Figure 3.1a) preserve 80.02% of the explained stress (Figure 3.1b shows more dimensions). This is therefore a remarkably low-dimensional manifold. It is not exactly low-dimensional because the explained stress is not 100%, but it is an effectively low-dimensional manifold. This also indicates that the individual manifolds of models trained on one task are low-dimensional, even if they start from different random initializations in the weight space. Such low-dimensional manifolds are seen in *all* our experiments, irrespective of the specific method used for representation learning, namely, supervised, transfer (fine-tuning), meta, semi-supervised and contrastive learning.

Remark 3.3.2 (A detailed description of how we plot trajectories of representations). We provide a non-

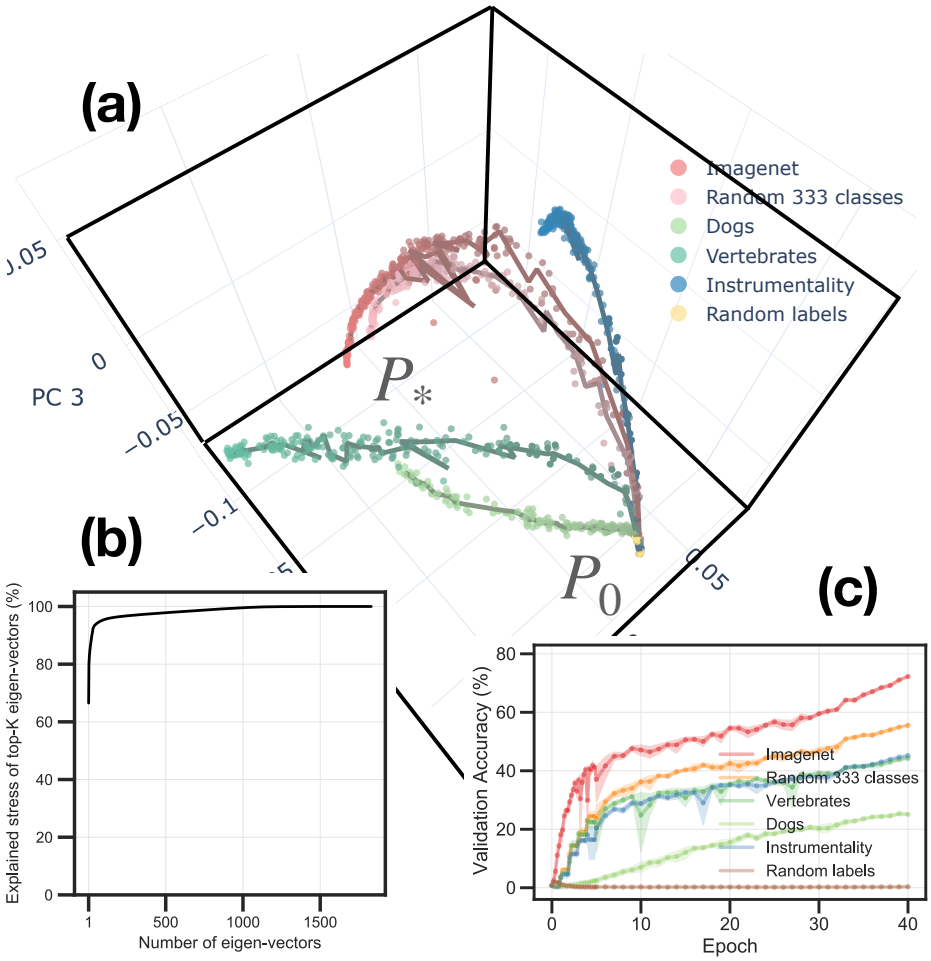


Figure 3.1: **(a)** Visualization of training trajectories of models trained on 6 tasks from ImageNet. Each point is one network, bold lines connect points along the average trajectory of each task (across 5 random weight initializations). Trajectories move towards the truth P_*, which corresponds to the ground-truth labels. Training on one task makes a remarkable amount of progress on unseen, seemingly dissimilar, classes. Trajectories of models trained on a random set of 333 classes are similar to those of the entire ImageNet. Some classes (Instrumentality) are closer to this trajectory while others such as Vertebrates and Dogs are farther away. Dogs is a semantic subset of Vertebrates; it splits at the beginning but seems to eventually reach a similar representation as one of the intermediate points of Vertebrates.

(b) Percentage explained stress Equation (2.2.8) captured by subspace spanned by the top k InPCA eigenvectors.

(c) Validation accuracy on different tasks vs. epochs.

mathematical description of how the theory in Section 3.2 was used to draw Figure 3.1a below. We train 5 different networks (random seeds for initialization) for each of the 6 tasks, and record 61 model checkpoints

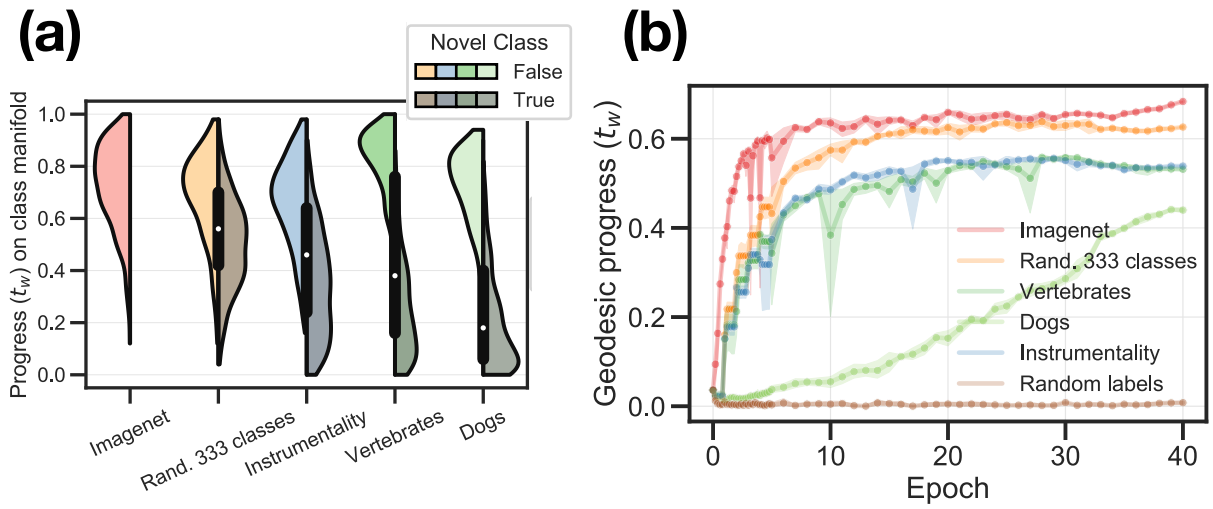


Figure 3.2: **(a)** Progress made by each model on classes seen during training (left half, lighter shade) and on novel classes (right half, darker shade). We compute t_w^c which is the progress t_w of images restricted to a single class c. This quantity t_w^c measures the quality of the representation for class c. Violin plots denote the distribution of t_w^c indicate that we make more progress on classes seen during training. If the model sees a larger diversity of classes (like with random 333 classes), more progress is made on the novel classes. Surprisingly, even if we train on just the "Dogs", we make some progress on novel classes.

(b) Progress t_w Equation (2.2.4) on the Y-axis against the number of epochs of training on the X-axis. The progress t_w increases with more epochs of training—all models make non-trivial progress towards the truth P_* ($t_w = 1$). Even if we train on only Dogs (118 classes) we make progress on the entire ImageNet.

during training; this gives 1830 checkpoints for this experiment. We re-index all checkpoints to calculate their progress using Equations (2.2.3) and (2.2.4). We then interpolate between each consecutive pair of the 61 checkpoints along each trajectory using Equation (2.2.3). The training trajectory can now be sampled at any progress $t_w \in [0, 1]$. We next calculate the "average trajectory" of the 5 networks (random seeds) of each task by averaging the output probabilities in Equation (2.2.1) at a fixed value of t_w; 100 different values of t_w spread uniformly between $[0, 1]$ are chosen. These 100 points along the average trajectory of each of the 6 tasks are also embedded together with the 1830 checkpoints (i.e., $m = 2430$ in Equation (2.2.6)). Figure 3.1a plots the top three dimensions obtained from InPCA. To clarify, the explained stress of the top 2430 dimensions would be exactly 100%.

Result 2: Supervised learning on one task results in a surprising amount of progress on seemingly dissimilar tasks. Progress on other tasks is larger if the training task has diverse classes. We studied

the progress t_w Equation (2.2.4) made by models (Figure 3.2b) trained on tasks from Result 1. Training on the task "Dogs" makes non-trivial progress on other tasks, even seemingly dissimilar ones like "Instruments" which contains vehicles, devices and clothing. In fact, it makes a remarkable amount of progress on the *entire* ImageNet, about 63.38% of the progress of a model trained directly on ImageNet. Progress is larger for larger phyla of ImageNet (Vertebrates and Instruments). But what is surprising is that if we train on a random subset of 333 classes (a third of ImageNet), then the progress on the entire ImageNet is very large (92%). This points to a strong shared structure among classes even for large datasets such as ImageNet. Note that this *does not* mean that tasks such as Vertebrates and Instruments are similar to each other. Even if training trajectories are similar for a while, they do bifurcate eventually and the final models are indeed different (see Figure 3.3b and Remark 3.3.3 on how to interpret it).

In Figure 3.2a, we studied the projections of models trained on one task onto the geodesics of unseen classes calculated using Equation (2.2.3) evaluated at the progress t_w Equation (2.2.4)). We find that a model trained on the entire ImageNet makes uneven progress on the various classes (but about 80% progress across them, progress is highly correlated with test error of different classes). Models trained on the 6 individual tasks also make progress on other unseen classes. As before, training on Instruments, Vertebrates, Dogs makes smaller progress on unseen classes compared to training on a random subset of 333 classes. This is geometric evidence that the more diverse the training dataset, the better the generalization to *unseen* classes/tasks; this phenomenon has been widely noticed and utilized to train models on multiple tasks, as we discuss further in Result 4.

Result 3: The structure of the space of tasks indicated by our visualization technique is consistent with parts of the Wordnet phylogenetic tree. To obtain a more fine-grained characterization of how the geometry in the space of learnable tasks reflects the semantics of these tasks, we selected two particular phyla of ImageNet (Animals, Artifacts) and created sub-tasks using classes that belong to these phyla (Figure 3.3a). Trajectories of models trained on Instruments and Conveyance are closer together than those of Animals. Within the Animals phylum, trajectories of Vertebrates (Dog, Reptile, Bird) are closer together than those of Invertebrates (Figure 3.3b for quantitative metrics). Effectively, we can recover a part of the phylogenetic tree of Wordnet using our training trajectories. We speculate that this may point to some shared structure between

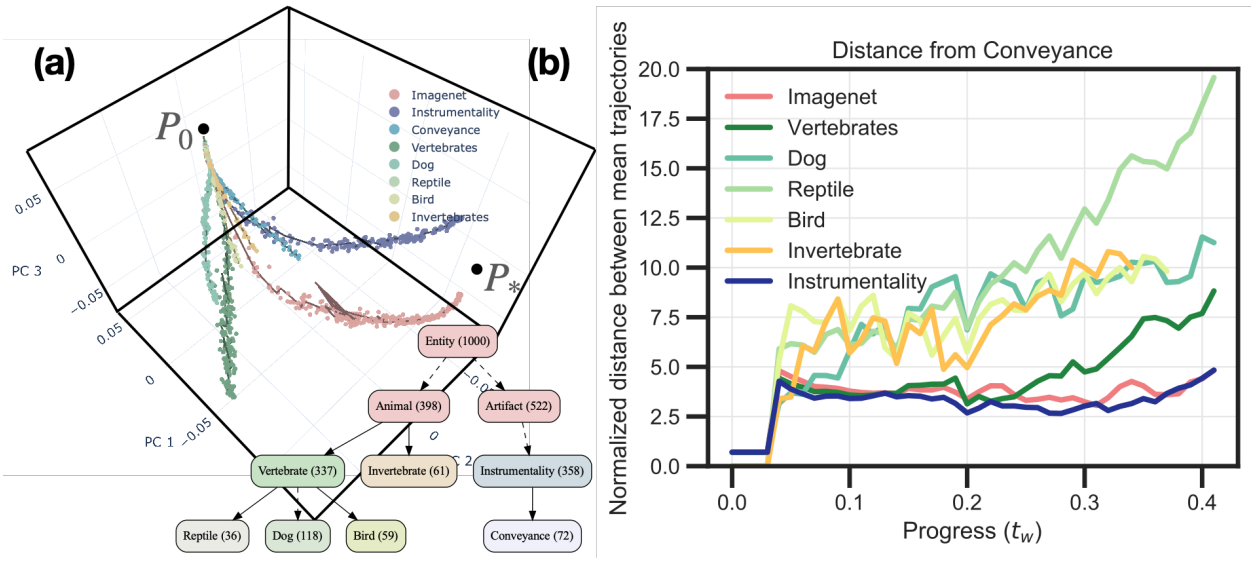


Figure 3.3: **(a)** Trajectories of models trained on different phyla of Wordnet (inset). The model manifold is again effectively low-dimensional (78.72% explained stress in 3 dimensions).
(b) We analyze the trajectories in Figure 3.3(a) and obtain a quantitative description of how trajectories of different tasks diverge from each other during training; the procedure is explained in in Remark 3.3.3. The plot depicts the Bhattacharyya distance between the mean trajectories (over random initializations) on different tasks, and the mean trajectory of Conveyance. This distance is normalized by the average of the tube radii (maximum distance of one of the 5 trajectories from the mean, computed at each progress) of the two trajectories. Such quantities allow us to make precise statements about the differences between representations and show some very surprising conclusions. Trajectories of tasks that are nearby in Wordnet are also nearby in terms of their learned representations. Further, trajectories of ImageNet (pink) are closer to Conveyance (as expected), but those of Vertebrates (red) are equally far away for more than 60% ($t_w \approx 0.25$) of the progress. In other words, training on Vertebrates (reptiles, dog, bird) makes a remarkable progress on Conveyance (cars, planes).

visual features of images and natural language-based semantics of the corresponding categories which was used to create Wordnet (Miller, 1998) of the corresponding categories. Such alignment with a natural notion of relatedness also demonstrates the soundness and effectiveness of our technical machinery.

Remark 3.3.3 (Building a precise and quantitative characterization of trajectories of representations). The precise way to understand statements like those in Result 3 is using the quantitative analysis reported in Figure 3.3b and Figure 3.10. To expand upon the caption, the X-axis of the plot is progress. For multiple models (5 random seeds) trained on two tasks (say Conveyance and Dogs), we have calculated the mean (across random seeds) of the interpolated trajectories at different progress. At each specific progress, we

have plotted the distance between the mean model trained on Conveyance (say task 1) and Dogs (say task 2) divided by the average tube radii (which is the maximum of the distance of the model corresponding to one seed from the mean):

$$2\mathrm{d_B}(\tau_{\text{mean}}^{1\to U}, \tau_{\text{mean}}^{2\to U}) / \sum_{k=1,2} \max_a [\mathrm{d_B}(\tau_a^{k\to U}, \tau_{\text{mean}}^{k\to U})].$$

The is a measure of how far away the trajectories of these two models are. If it is less than 1, then the "tubes" corresponding to models trained on tasks 1 and task 2 intersect.

Let us emphasize that we have performed such analyses for all experiments in this paper (see Figure 3.10); while the InPCA embedding gives an easy-to-understand visual description of these results for high-dimensional probabilistic models, the information geometric techniques developed in this paper enable us to make these descriptions precise and quantitative. We also include a similar step-by-step guide on how to interpret Figure 3.8b in Section 3.5.3.

Result 4: Episodic meta-learning algorithms traverse very different trajectories during training but they fit a similar model eventually. Meta-learning methods build a representation which can be adapted to a new task (Thrun and Pratt, 2012). We studied a common variant, the so-called episodic training methods (Bengio et al., 1992), in the context of few-shot learning methods (Vinyals et al., 2016). In these methods, each mini-batch consists of samples from C^w out of C classes (called "way") split into two parts: a "support set" D_s of s samples/class (called "shot"), and a "query set" D_q of q samples/class. Typical methods, say prototypical networks of Snell et al. (2017b), implement a clustering loss on features of the query samples using averaged features of the support samples $\varphi_c = s^{-1} \sum_{\{x \in D_s, y^*(x)=c\}} \varphi(x; w_1)$ for all $c = 1, \ldots, C^w$ as the cluster centroids. If features φ lie on an ℓ_2 ball of radius 1, then doing so is akin to maximizing the cosine similarity between cluster centroids and features of query samples. The same clustering loss with the learned backbone w_1 is used to predict on unseen classes (using "few" support samples to compute centroids) at test time.

To understand the representations learned by episodic meta-learning methods, we compared trajectories of episodic meta-learning to the trajectory taken by supervised learning in Figure 3.4. Supervised learning uses the cross-entropy loss over all the C classes while episodic meta-learning optimizes a loss that considers

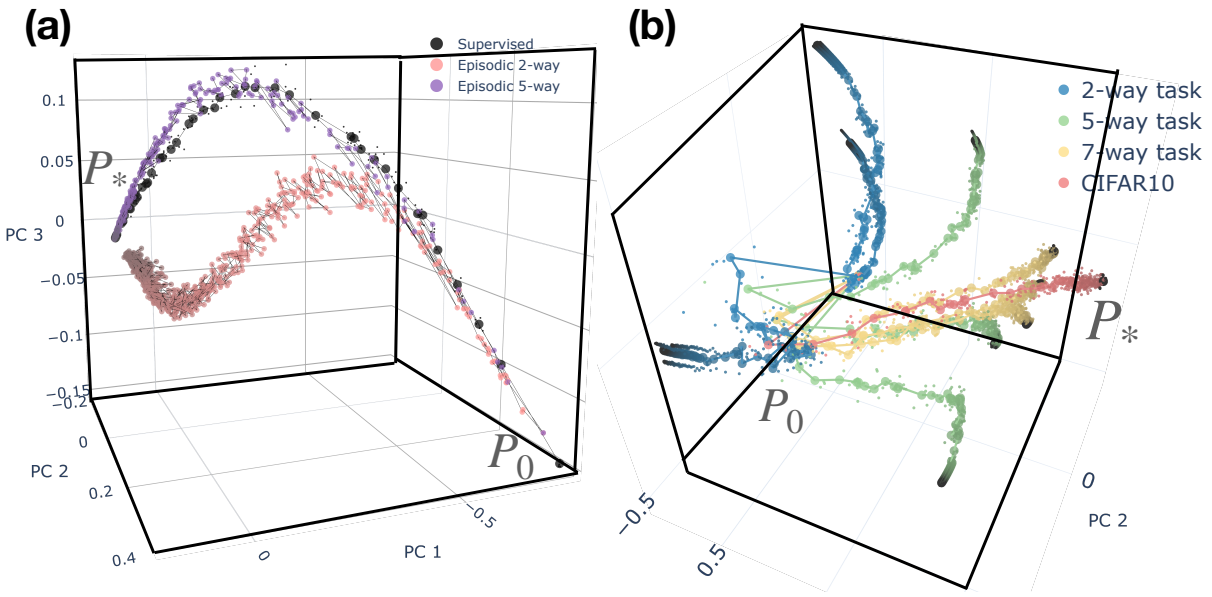


Figure 3.4: **(a)** Training trajectories for supervised learning (black), 2-way (pink) and 5-way episodic meta-learning (purple). Trajectories of 5-way meta-learning are very similar to those of supervised learning and eventually reach very similar models and high test accuracy. In contrast, 2-way meta-learning has a much longer trajectory (about 40× longer in Riemann length than black) and does not reach a good test accuracy (on all 10 CIFAR-10 classes). Representations are similar during early parts of training even if these are quite different learning mechanisms.
(b) Trajectories of 2-way (blue), 5-way (green), 7-way (yellow) tasks trained using cross-entropy loss compared to supervised learning (red). For large "way", trajectories are similar to supervised learning but they quickly deviate from the red trajectories for small ways.

all k-way classification tasks (where k is typically smaller than C), its objective differs from that used for supervised learning. Since the two objectives are different, it comes as a surprise that both arrive at the same solution; see Figure 3.4a,b and Figure 3.9 for distances between trajectories. But the Riemann trajectory length of episodic training is about 40× longer than that of supervised learning. It is worth noting that the explained stress is only 40.96% in Figure 3.4a because of larger fluctuations for episodic learning in other directions. Therefore, episodic meta-learning has a qualitatively different training trajectory in the prediction space than supervised learning. The implications of this are consistent with recent literature which has noticed that the performance of few-shot learning methods using supervised learning (followed by fine-tuning) is comparable to, or better than, episodic meta-learning (Dhillon et al., 2020; Kolesnikov et al.,

2020; Fakoor et al., 2020). Indeed, a supervised learned representation also minimizes the clustering loss.

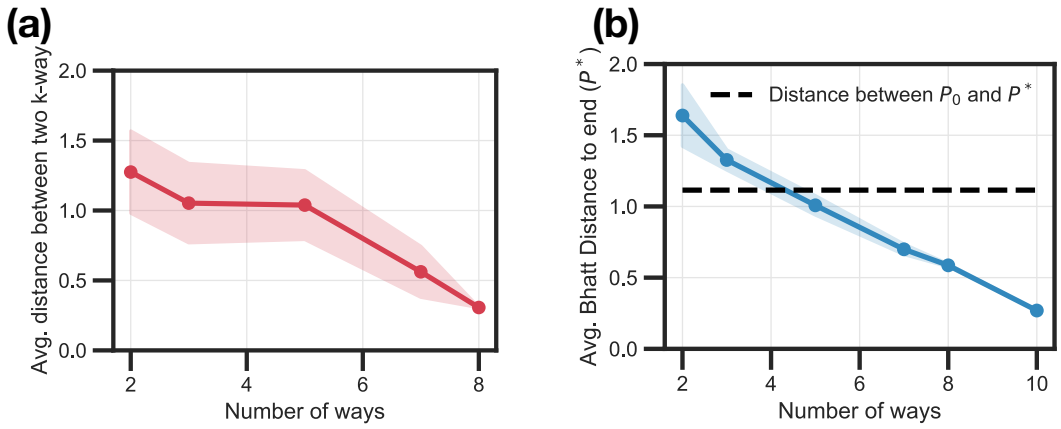


Figure 3.5: **(a)** Average distance between two k-way meta-learning trajectories decreases with k, this is a geometric evidence of the variance of predictions of learned representations.
(b) Training with a small way leads to models that predict poorly on test data (large distances from truth). These embeddings were calculated using the training dataset. The rationale being that we wanted to show how different meta-learning and supervised learning are during training.

In order to understand why few-shot accuracy of episodic training is better with a large way (Gidaris and Komodakis, 2018), we trained models on different 2-way 5-way and 7-way tasks using the cross-entropy loss (Figure 3.4b). We find that the radius of the tube that encapsulates the models of 2-way tasks around their mean trajectory is very large, almost as large as the total length of the trajectory, i.e., different models trained with a small way tasks traverse very different trajectories. Tube radius decreases as the way increases (Figure 3.5a). Further, the distance of models from the truth P_* (which is close to the end point of the supervised learning model) is higher for a small way (Figure 3.5b). This is geometric evidence of the widely used empirical practice of using a large way in episodic meta-learning. Observe in Figure 3.5b that as the way increases, the trajectory becomes more and more similar to that of supervised learning. See Figure 3.10 for a quantitative analysis of these trajectories.

Result 5: Contrastive and semi-supervised learning methods traverse trajectories similar to those of supervised learning. Contrastive learning (Becker and Hinton, 1992) learns representations without using ground-truth labels (Gutmann and Hyvärinen, 2010; Chen et al., 2020a). It has been extremely effective for

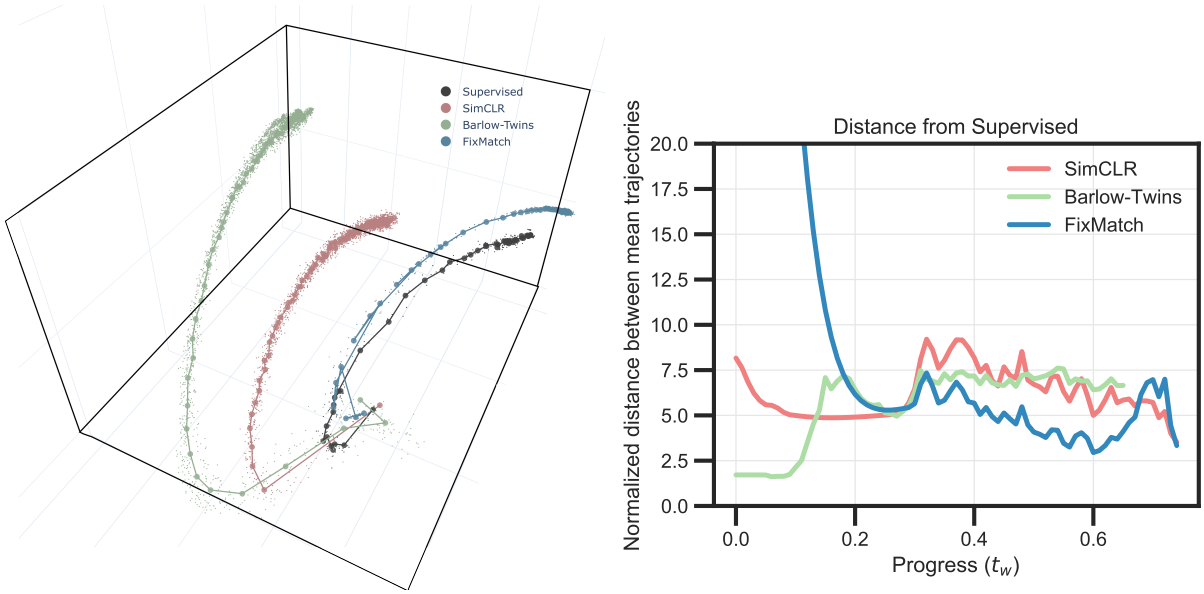


Figure 3.6: We consider 4 methods for training on CIFAR10: supervised learning, SimCLR (Chen et al., 2020a), Barlow-twins (Zbontar et al., 2021) and Fixmatch (Sohn et al., 2020). Fixmatch has access to 2500 labeled samples and 47500 unlabeled samples. SimCLR and Barlow-twins use 50,000 unlabeled samples for training.
(a) We plot the trajectories for supervised, semi-supervised and contrastive learning. The trajectory of semi-supervised learning (Fixmatch) eventually resembles supervised learning in comparison to contrastive learning methods. All methods result in remarkably similar trajectories although some of these methods are trained using only unlabeled data.
(b) Normalized distance of trajectories corresponding to contrastive and semi-supervised learning to the trajectory of supervised learning. Semi-supervised learning (Fixmatch) deviates considerably from the other methods at the beginning. We speculate that this is because the trajectory of Fixmatch is influenced by the 2500 labeled samples. As, training progresses, Fixmatch becomes increasingly similar to supervised learning as evidenced by the dip in the blue line for larger values of progress (t_w).

self-supervised learning (Doersch and Zisserman, 2017; Kolesnikov et al., 2019), e.g., prediction accuracy with 1–10% labeled data is close to that of supervised learning using all data (Chen et al., 2020b). Semi-supervised methods (Berthelot et al., 2019; Sohn et al., 2020) learn representations when ground-truth labels are available for only a small fraction of the data (0.1–1%). These methods achieve a prediction accuracy within 5% of the accuracy achieved through supervised learning. We compared representations learned using contrastive and semi-supervised learning with those from supervised learning to understand why these methods are so effective.

Consider a task P and a set of augmentations G (e.g., cropping, resizing, blurring, color/contrast/brightness

distortion etc.). Given inputs (say images) x from P, contrastive learning forces the representation $\varphi(g(x); w_1)$ and $\varphi(g'(x); w_1)$ (shortened to $\varphi(g(x))$ below) of the same input for two different augmentations g, g' to be similar. And forces it to be different from representations of other augmented inputs x' (Zbontar et al., 2021; Bachman et al., 2019; Dosovitskiy et al., 2014). Semi-supervised learning methods have access to both labeled inputs x_l and unlabeled inputs x_u. More recent methods are usually trained to fit the labeled inputs using the cross-entropy loss while enforcing consistent predictions across all augmentations (Tarvainen and Valpola, 2017; Berthelot et al., 2019) for any unlabeled input.

We compare the representations of semi-supervised (Fixmatch (Sohn et al., 2020)), contrastive (SimCLR (Chen et al., 2020a), Barlow-twins (Zbontar et al., 2021)) and supervised learning in Figure 3.6. All three trajectories are similar to the trajectory of supervised learning. We find that the trajectory of semi-supervised learning deviates from the supervised learning trajectory initially, but the two are very similar for larger values of progress (t_w). This points to a remarkable ability of semi and self-supervised learning methods to learn representations that are similar to those of supervised learning; it is not just that the accuracy of these methods is similar, they also learn similar probabilistic models.

Result 6: Fine-tuning a pre-trained model on a sub-task does not change the representation much. To understand how models train on multiple tasks, we selected two binary classification sub-tasks of CIFAR-10 (Airplane vs. Automobile, and Bird vs. Cat).

We selected models at different stages of standard supervised learning on CIFAR-10 (i.e., using 10-way output and softmax cross-entropy loss) and fine-tuned each of these models on two sub-tasks (the entire network is fine-tuned without freezing the backbone). As Figure 3.7 shows, models that were fine-tuned from earlier parts of the trajectory travel a large distance and move away from trajectories of the supervised learned CIFAR-10 models. As we fine-tune later and later models, the distance traveled away from the trajectory is smaller and smaller, i.e., changes in the representation are smaller. For a fully-trained CIFAR-10 model which interpolates the training data, the distance traveled by fine-tuning is very small (the points are almost indistinguishable in the picture); this is because both P^1 and P^2 are subsets of CIFAR-10.

Algorithms for transfer learning train on a source task before fine-tuning the model on the target task. If two tasks share a large part of their training trajectory, then we may start the fine-tuning from many shared

intermediate points—there are many such points. If the chosen point is farther along in terms of progress then the efficiency resulting from using the source task is higher because the trajectory required to fit the target task is shorter; such trajectories were used in (Gao and Chaudhari, 2021) to define a distance between tasks. As we saw in Result 2, trajectories of different tasks bifurcate after a shared part. The resultant deviation less for related tasks and more for dissimilar tasks (Figure 3.7a, Figure 3.1a,c). Therefore it is difficult to know *a priori* from which point one should start the fine-tuning from without knowing the manifold of the target task. In particular, our geometric picture indicates that fine-tuning from a fully-trained model can be detrimental to the accuracy on the target task. This has been noticed in a number of places in the transfer learning literature, e.g., Li et al. (2020), and has also been studied theoretically (Gao and Chaudhari, 2020).

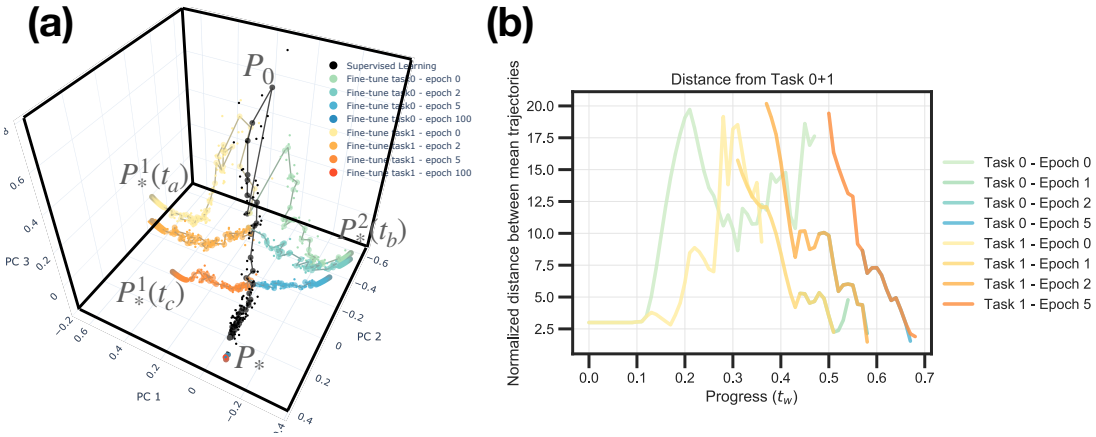


Figure 3.7: **(a)** Fine-tuning trajectories on Airplane vs. Automobile, and Bird vs. Cat sub-tasks of CIFAR-10 (warm and cold hues) pre-trained from different points along the trajectory of supervised learning. If the pretrained model has progressed further towards the truth P_*, then fine-tuning it on a sub-task does not change the representation much. The final trajectory (fine-tuning from epoch 100) is indistinguishable from P_*. **(b)** Bhattacharyya distance between the mean trajectories normalized by the average of the tube radii (like Figure 3.3b). models (say, fine-tuned after epoch 5 on task 1) go *backwards* in terms of progress, i.e., they unlearn the pre-trained representation in order to fit the new task. This occurs as early as epoch 1 here. It suggests that learning occurs extremely rapidly at the beginning and determines the efficiency of fine-tuning. Some curves here are not visible because they are overlapping heavily.

Result 7: Contrastive learning methods trained on different datasets learn similar representations

We compared representations learned using contrastive learning with those from supervised learning to understand some aspects of why the former are so effective.

We used SimCLR (Chen et al., 2020a) to perform contrastive learning on images from four sets of classes

(airplane-automobile, bird-cat, ship-truck and all of CIFAR-10). We compared the learned representation to that from supervised learning on two tasks (airplane-automobile and all of CIFAR-10) in Figure 3.8. Models trained using contrastive learning on two-class datasets learn very different representations from models trained on the same task but using supervised learning. Models trained using contrastive learning on different datasets learning similar representations (trajectories of all three two-class datasets are very close to each other). This is reasonable because contrastive learning does not use any information from the labels. It is surprising however that the trajectory of models from contrastive learning on these two-class datasets is similar to trajectories of models from contrastive learning on the entire CIFAR-10.

Let us elaborate upon this a bit more. We have color-matched the lines in Figure 3.8b with those in Figure 3.8a. The black curve is the trajectory of supervised learning on the entire CIFAR-10; red is the trajectory of SimCLR trained on the entire CIFAR-10. Figure 3.8b compares the distances of trajectories in Figure 3.8a from the red one "contrastive"; this is why there is no red trajectory in Figure 3.8b.

- The first thing to note here is that the black and red trajectories are quite close to each other; the black line in Figure 3.8b is only about 20 times far away from red as compared to their corresponding tube radii.

- Next observe that the trajectory of SimCLR on Task 1 (light blue), SimCLR on Task 2 (green) and SimCLR on Task 3 (yellow) are very similar to each other; this is seen in both Figure 3.8a and in Figure 3.8b.

- Third, they are closer to SimCLR on all of CIFAR-10 than any supervised learning trajectories (this is seen in Figure 3.8b because their curves are below everyone else). Thus, contrastive learning on datasets with different classes learns similar representations.

- The learned representation of two-class SimCLR models is similar to the one obtained using data from all classes (red) (in this experiment this occurs up to about $t_w = 0.4$ progress) but they do not go all the way to the truth (i.e., the end point of black line). This shows the benefit of having data from many classes during contrastive learning.

Also see Figure 3.10 for distances computed with respect to other trajectories which can be used to further investigate these claims.

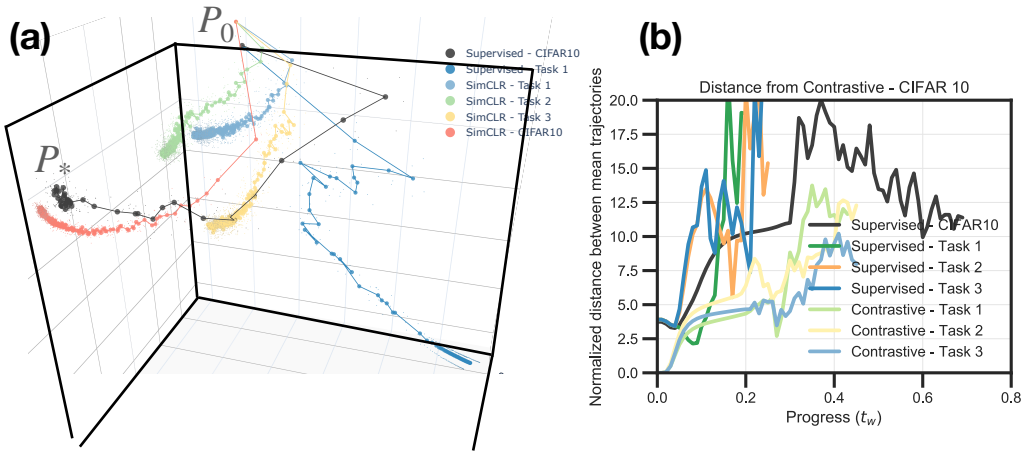


Figure 3.8: **(a)** Trajectories of contrastive learning (SimCLR) on 3 datasets (two classes each) and entire CIFAR-10 compared to those of supervised learning. SimCLR on entire CIFAR-10 learns a similar representation as that of the supervised learned model P_* (which fits the training data perfectly). SimCLR trajectories are close to each other even if different datasets were used to train them. It may seem from the embedding that SimCLR trajectories are similar to that supervised learning, which would be very surprising because the former does not use any labels, but see below.

(b) Bhattacharyya distance between the mean trajectories of all models and the mean trajectory of SimCLR on all CIFAR-10. This distance is normalized by the average of the tube radii (like Figure 3.7b). SimCLR trajectories of two-class datasets are indeed very close to each other (mean distance is $\sim 5\times$ more than their tube radii for about 45% of the way ($t_w \approx 0.2$)). This plot indicates that two-class SimCLR trajectory (light blue) is close to SimCLR on all of CIFAR-10. But two-class supervised learning trajectory (darker blue) is much farther away from SimCLR on all of CIFAR-10.

3.4. Related Work and Discussion

Understanding the space of learnable tasks A large body of work has sought to characterize relationships between tasks, e.g., domain specific methods (Zamir et al., 2018; Cui et al., 2018; Pennington et al., 2014), learning theoretic work (Baxter, 2000; Maurer, 2006; Ben-David et al., 2010; Ramesh and Chaudhari, 2022; Tripuraneni et al., 2020; Hanneke and Kpotufe, 2020; Caruana, 1997), random matrix models (Wei et al., 2022), neural tangent kernel models (Malladi et al., 2022) and information-theoretic analyses (Jaakkola and Haussler, 1999; Achille et al., 2019a,b). Broadly speaking, this work has focused on understanding the accuracy of a model on a new task when it is trained upon a related task, e.g., relationships between tasks are characterized using the excess risk of a hypothesis. Our methods also allow us to say things like "task P^1 is far from P^2 as

compared to P^3". But they can go further. We can glean a global picture of the geometric structure in the space of tasks and quantify statements such as "the divergence between P^1 and P^2 eventually is more than that of P^1 and P^3, but representations learned on these tasks are similar for 30% of the way".

There is strong structure in typical inputs, e.g., recent work on understanding generalization (Yang et al., 2022; Bartlett et al., 2020) as well as older work such as Simoncelli and Olshausen (2001); Field (1994); Marr (2010) has argued that visual data is effectively low-dimensional. Our works suggests that tasks also share a low-dimensional structure. Just like the effective low-dimensionality of inputs enables generalization on one task, effective low-dimensionality of the manifold of models trained on different tasks could perhaps explain generalization to new tasks.

Relationships between tasks in neuroscience Our results are conceptually close to those on organization and representation of semantic knowledge (Mandler and McDonough, 1993). Such work has primarily used simple theoretical models, e.g., linear dynamics of Saxe et al. (2019) (who also use MDS). Our tools are very general and paint a similar picture of ontologies of complex tasks. Concept formalization and specialization over age (Vosniadou and Brewer, 1992) also resembles our experiment in how fine-tuning models trained for longer periods changes the representation marginally. Our broad goals are similar to those of Sorscher et al. (2021) but our techniques are very different.

Information Geometry has a rich body of sophisticated ideas (Amari, 2016), but it has been difficult to wield it computationally, especially for high-dimensional models like deep networks. Our model in Equation (2.2.1) is a finite-dimensional probability distribution, in contrast to the standard object in information geometry which is an infinite-dimensional probability distribution defined over the entire domain. This enables us to compute embeddings of manifolds, geodesics, projections etc. We are not aware of similar constructions in the literature.

Visualizing training trajectories of deep networks InPCA is a variant of multi-dimensional scaling (MDS, see Cox and Cox (2008)), with the difference being that InPCA retains the negative eigenvalues which preserves pairwise distances (Quinn et al., 2019a). A large number of works have investigated trajectories of deep networks and the energy landscape during or after training using dimensionality reduction techniques (Horoi et al., 2021; Li et al., 2018; Huang et al., 2020). Gur-Ari et al. (2018);

Antognini and Sohl-Dickstein (2018) studied the dimensionality of training trajectories. The key distinction here with respect to this body of work is that we study the prediction space, not the weight space. While the weight space has symmetries (Freeman and Bruna, 2017; Garipov et al., 2018) and nontrivial dynamics (Tanaka and Kunin, 2021; Chaudhari and Soatto, 2018), the prediction space, i.e,. $[0,1]^{N\times C} \ni \{p_w(c \mid x_i)\}$, completely characterizes the output of a probabilistic model. In comparison, the loss or the error which are typically used to reason about relationships between tasks, are coarse summaries of the predictions. Any two models, irrespective of their architecture, training methodology, or even the task that they were trained on, can be studied rigorously using our techniques.

3.5. Appendix

3.5.1. Details of the experimental setup

Data

We performed experiments using two datasets.

1. CIFAR10 (Krizhevsky, 2009) has 10 classes (airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck) with RGB images of size 32×32, and

2. ImageNet (Deng et al., 2009) has 1000 classes each with about 1000 RGB images of size 224×224.

ImageNet classes are derived from the leaves of the Wordnet hierarchy (Miller, 1998) which is visualized by Bostock (2018). We use this hierarchy to create tasks using different subsets of ImageNet; We use all classes under a node to create a task. The tasks that we consider are: Dogs, Vertebrates, Invertebrates, Instrumentality, Reptile and Birds. We also consider a task with 333 randomly selected classes and unlike other tasks, it spans many different phyla of ImageNet.

Architectures We use a Wide-Resnet (Zagoruyko and Komodakis, 2016) architecture for supervised learning experiments on CIFAR-10 (WRN-16-4 with depth 16 and widening factor of 4) and a Resnet-18 (He et al., 2016) to train a model using SimCLR. All experiments on ImageNet use the Resnet-50 architecture.

All convolutional layers are initialized using the Kaiming-Normal initialization. For the Wide-Resnet, the final pooling layer is replaced with an adaptive pooling layer in order to handle input images of different

sizes.

We make three modifications to these architectures.

1. We remove the bias from the final classification layer; this helps keep the logits of the different tasks on a similar scale.

2. In the experiments for Result 3 (episodic meta-learning) and Result 6 (fine-tuning), we replace batch normalization with layer norm in the Wide-Resnet. This is because we found in preliminary experiments that batch-normalization parameters make training meta-learning models very sensitive to choices of hyper-parameters (e.g., the support or query shot), and that the learned representations of new tasks were quite different in terms of their predictions (and thereby the Bhattacharyya distance) but all the difference was coming from modifications to the BN parameters.

3. In the Resnet-50, we replace the pooling layers with BlurPool (Zhang, 2019). The bias parameter in batch normalization is set to zero with the affine scaling term set to one.

Training procedure All models are trained in mixed-precision (32-bit weights, 16-bit gradients) using stochastic gradient descent (SGD) with Nesterov's acceleration with momentum coefficient set to 0.9 and cosine annealing of the learning rate schedule. Batch-normalization parameters are excluded from weight decay.

CIFAR10 datasets use padding (4 pixels) with random cropping to an image of size 28×28 or 32×32 respectively for data augmentation. CIFAR10 images additionally have random left/right flips for data augmentation. Images are finally normalized to have mean 0.5 and standard deviation 0.25.

Supervised learning models (including fine-tuning) for CIFAR10 are trained for 100 epochs with a batch-size of 64 and weight decay of 10^{-5} using the Wide-Resnet.

Episodic meta-learners are trained using a Wide-Resnet and with the prototypical loss (Snell et al., 2017a). For the 2-way meta-learner, each episode contains 20 query samples and 10 support samples. For the 5-way meta-learner, each episode contains 50 query samples and 10 support samples. We found (Result 4) to hold across different choices of these hyper-parameters in small-scale experiments. Models are trained for around

750 epochs and the episodic learner is about 5 times slower to train with respect to wall-clock time.

We train models using SimCLR on CIFAR10 and on tasks created from CIFAR10. For the augmentations, we use random horizontal flips, random grayscale, random resized crop and color jitter. Models are trained for 200 epochs for 2-way classification problems and for 500 epochs when trained on the entirety of CIFAR10 with the Adam optimizer and an initial learning rate of 0.001.

3.5.1.1 Experiments on ImageNet

We make use of FFCV (Leclerc et al., 2022). which is a data-loading library that replaces the pytorch Dataloader. FFCV reduces the training time on ImageNet to a few hours, which allows us to train 100s of models on ImageNet, or on tasks created from it. Our implementation of ImageNet training builds on the FFCV repository [5].

ImageNet models are trained for 40 epochs with progressive resizing – the image size is increased from 160 to 224 between the epochs 29 and 34. Models are trained on 4 GPUs with a batch-size of 512. The training uses two types of augmentations – random-resized crop and random horizontal flips. Additionally, we use label smoothing with the smoothing parameter set to 0.1.

3.5.2. Implementing InPCA in very high dimensions

We calculate an InPCA embedding of models along multiple trajectories, e.g., a typical experiment has about 25 trajectories (multiple random seeds, tasks, or representation learning methods) and about 50 models (checkpoints) along each trajectory. Each model is a very high-dimensional object (with dimensionality NC where $N \sim 10^5$ and $C \sim$ 10-10^3). Even if the matrix D in Equation (2.2.6) is relatively manageable with $n \sim 1250$, each entry of D is $\mathrm{d_B}(P_u, P_v)$ and therefore requires $\sim 10^8$ operations to compute. Implementing InPCA—or even PCA—for such large matrices requires a large amount of RAM. We reduced the severity of this issue to an extent using Numpy's memmap functionality https://numpy.org/doc/stable/reference/generated/numpy.memmap.html. Also note that calculating only the top few eigenvectors of Equation (2.2.6) suffices to visualize the models, we do not need to calculate all.

The formula Equation (2.2.2) is an effective summary of the discrepancies between how the predictions

[5] https://github.com/libffcv/ffcv-imagenet/tree/main

made by two probabilistic models differ; even small differences in two models, e.g., even if both P_u and P_v make mistakes on exactly the same input samples, if $p_u^n(c)$ is slightly different than $p_v^n(c)$ for even one of n or c, the divergence is non-zero. InPCA is capable of capturing the differences between two such models Equation (2.2.6). However, when the number of classes is extremely large, the number of terms in the summation is prohibitively large and analyzing the discrepancies or calculating the embedding becomes rather difficult.

We also developed a method to work around this issue. We can use a random stochastic matrix (whose columns sum up to 1) to project the outputs for each sample $\{p_u^n(c)\}_{c=1,\ldots,C}$ into a smaller space before calculating Equation (2.2.2). This amounts to pretending as if the model predicts not the actual classes but a random linear combination of the classes (even if the model is trained on the actual classes). This is a practical trick that is necessary only when we are embedding a very large number of very high-dimensional probabilistic models. We checked in our Imagenet experiments that using this trick gives the same embeddings.

In this paper, we did not need to use this projection trick. However, we found that this tricks makes it computationally faster to compute the embeddings and we have seen it to work well in practice. We have shared the code for this procedure, since it allows other people to reproduce the results using fewer computational resources.

3.5.3. Additional Result

3.5.4. Imprinting as an alternative to training the final layer

Consider a total of C classes. We would like to find weights $\{w_c\}_{c=1}^C$ that maximize the log-probability of the samples, under the constraint that for all $c \in C$, the norm of the weights $||w_c||$ is 1. Let $\varphi(x)$ denote a internal representation of sample x. The log-probability

$$\sum_{x:y_x=c} \log p(y=c \,|\, x) = \sum_{x:y_x=c} w_c \cdot \varphi(x) - \sum_{x:y_x=c} \log \left(\sum_{j=c}^{C} \exp\left(w_c \cdot \varphi(x)\right) \right), \tag{3.5.1}$$

is proportional to the inner-product $w_c \cdot \sum_{x:y_x=c} \varphi(x)$. Maximizing just this term under the norm constraint, we get the imprinted weights $\sum_i \phi(x_i^c) / || \sum_i \phi(x_i^c) ||$ as the solution. Deriving an analytical expression for

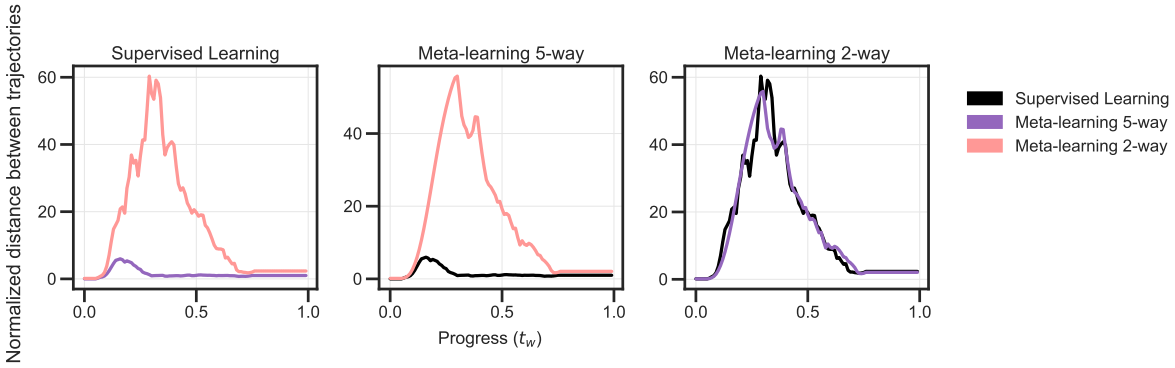


Figure 3.9: **Distance between trajectories of supervised and meta-learning at different values of progress.** Distances between the average trajectories of different algorithms (e.g., 2-way episodic learning and supervised learning, and 5-way episodic learning and supervised learning in the leftmost panel) are normalized by the average of the radii of the tubes corresponding to each trajectory. We find that trajectories of 2-way meta-learning deviate significantly from those of supervised learning for a large fraction of the trajectory. On the other hand, 5-way meta-learning is similar to the supervised learning trajectory for almost the entirety of the trajectory.

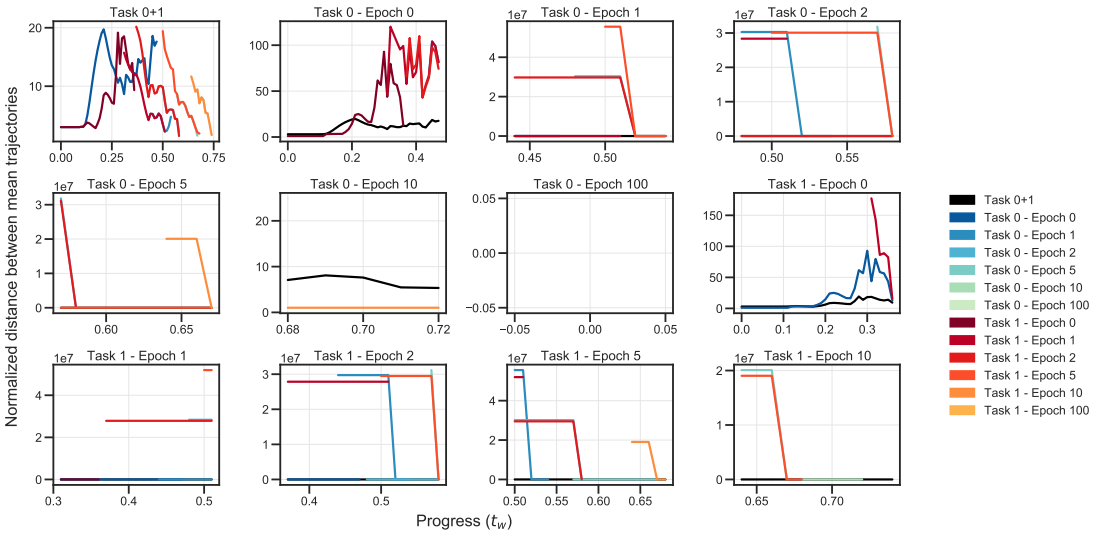
the optimal value of $\{w_c\}_{i=1}^{n_c}$ is difficult and hence we use the imprinted weights as an approximate solution. In our experiments, we found that the imprinted weights achieve an accuracy close to the optimal weights while being significantly easier to compute.

3.5.5. Invariant transformations of the internal representation

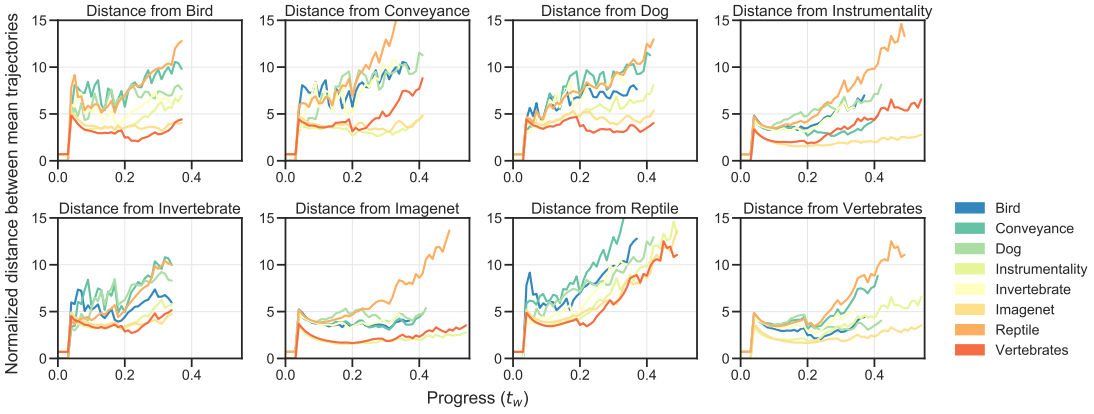
The internal representations are invariant to orthogonal transformations provided that we use imprinting to define a probabilistic model. This is because the internal representations define the same probabilistic model ever after an orthogonal transformation. Consider two internal representation ϕ and $U \cdot \phi$ where U is an orthogonal matrix. We note that the probabilistic model for $U \cdot \phi$ after imprinting is

$$\begin{aligned}\log p_2(y = c \,|\, x_i) &= \frac{U \cdot \sum_{y_x=c} \phi(x)}{||U \cdot \sum_{y_x=c} \phi(x)||} \cdot (U \cdot \phi(x_i)) - \log \left(\sum_{c=1}^{C} \exp \left(\frac{U \cdot \sum_{y_x=c} \phi(x)}{||U \cdot \sum_{y_x=c} \phi(x)||} \cdot (U \cdot \phi(x_i)) \right) \right) \\ &= \frac{\sum_{y_x=c} \phi(x)}{|| \sum_{y_x=c} \phi(x)||} \cdot \phi(x_i) - \log \left(\sum_{c=1}^{C} \exp \left(\frac{\sum_{y_x=c} \phi(x)}{|| \sum_{y_x=c} \phi(x)||} \cdot \phi(x_i) \right) \right).\end{aligned}$$

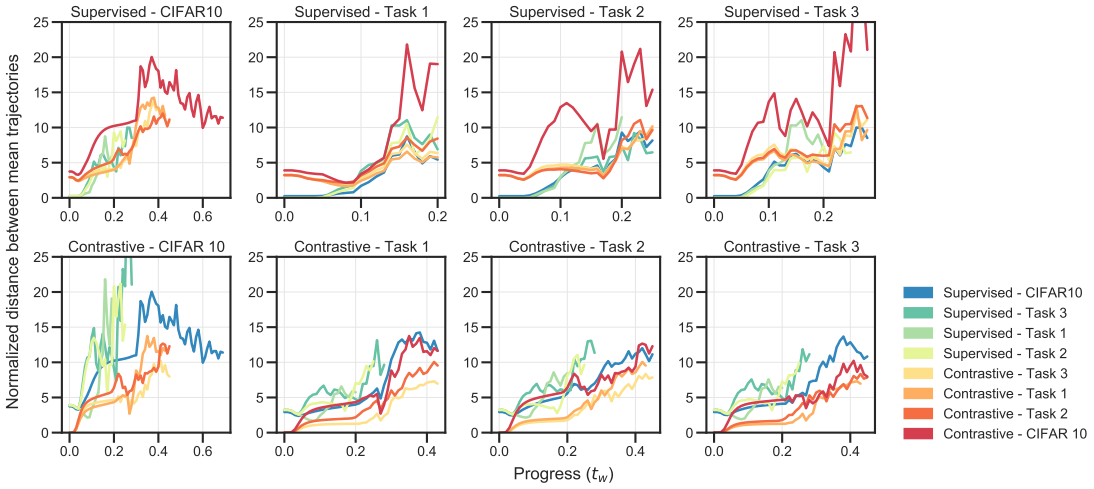
The probabilistic model for the representation $U \cdot \phi$ is identical to the probabilistic model for representation ϕ since norms and angles are preserved under orthogonal transformations. Hence the Bhattacharyya distance



(a)



(b)



(c)

Figure 3.10: This figure shows the extended version of the distances between trajectories of probabilistic models; two of them are identical to the ones in Figure 3.7b and Figure 3.8b.

between ϕ and $U \cdot \phi$ is zero.

The imprinting procedure can be thought of as removing information from the representation that is not relevant to prediction on a task. While this is true for all datasets in general, there could exist some additional structure in the data that results in more invariances (e.g., more than invariances to orthogonal transformations $O(n)$).

3.5.6. Calculating mean trajectories

We defined the distance between two trajectories to be $\mathrm{d}_{\text{traj}}(\tau_u^{1\to \mathrm{U}}, \tau_v^{2\to \mathrm{U}})$, i.e., the integral of the Bhattacharyya distance between the trajectories after mapping them to the same task and re-indexing them using the geodesic. Say we wish to compare a model trained on two tasks from CIFAR-10: Cats vs. Dogs and Airplane vs. Truck. We initialize multiple models for each of these two supervised learning problems (and we do so for every experiment in this paper) and train these 10 models. We can now calculate the mean trajectory of models on a task

$$\underset{\tau_\mu^1}{\operatorname{argmin}} \frac{1}{K} \sum_{k=1}^{K} \mathrm{d}_{\text{traj}}(\tau_{u_k}^1, \tau_\mu^1).$$

This optimization problem is very challenging because the variable is a trajectory of probabilistic models in a high-dimensional space. Even if we were to split this minimization and do it independently across time, this is still difficult because the solution is the so-called Bhattacharyya centroid on the product manifold defined in Equation (2.2.1) and cannot be computed in closed form. See (Nielsen and Boltz, 2011) for an iterative formula. We therefore simply take the arithmetic mean of the probability distributions, i.e., $P_{\mu(t)} = \frac{1}{K} \sum_{k=1}^{K} P_{w_i(t)}$. This is similar to ensembling. We use the radius of the tube around the mean trajectory, i.e.,

$$r_u = \max_k \mathrm{d}_{\text{traj}}(\tau_{u_k}^1, \tau_\mu^1)$$

to normalize distances (more precisely, we normalize using the average of the radii of the two trajectories being compared). Note that this radius depends upon time (and is computed after mapping and reindexing the trajectories). If the distance between the means of two sets of trajectories is smaller than their individual average radii, then the tubes around the means intersect each other. In such cases, one can say that the representations learned (at that time point) are not distinguishable. We next show all distances between

reindexed points along the trajectories discussed in Figures 3.1, 3.7 and 3.8. Note that each curve gives the integrands in Equation (2.2.5), not the integral.

CHAPTER 4

A POTENTIAL EXPLANATION FOR THE LOW-DIMENSIONALITY OF THE TRAINING MANIFOLD

4.1. Introduction

It is remarkable that trajectories of networks with such different configurations lie on a manifold whose dimensionality is much smaller than the embedding dimension, and the natural next question is to ask what leads to these low-dimensional dynamics. To address this question, we draw inspiration from earlier work studying the hyperribbon structure. In Transtrum et al. (2011b), the authors observed that the geodesic cross-sectional widths in the data space (corresponding to the width of a hyperribbon) is well approximated by the square root of eigenvalues of the Fisher Information Matrix (FIM), a local quantity measuring how much the prediction changes with respect to changes in parameter space. It is a nontrivial observation relating the global geometry of the network to local geometry, and it leads us to think whether such a correspondence exists in our model manifold as well.

In this chapter, we investigate this hypothesis and show the FIM of trained networks indeed have a fast-decaying eigenspectrum similar to the hyperribbon structure in the training manifold established in Chapter 2. In Figure 4.1, we show that this "sloppy" eigenspectrum, characterized by a few large eigenvalues and a large number of small eigenvalues that are distributed uniformly across an exponentially large range, is presented in the activations of different layers, Jacobian of logits with respect to the weights, gradients of the loss with respect to the activations, as also the Hessian and the FIM. In Section 4.3, we prove theoretical results relating this sloppiness to the sloppiness of data covariance structure, which is prevalent in various real-world data. We continue in Section 4.2.1 to show this sloppiness in the eigenspectrum of Hessian of trained models can be used to prove a generalization bound using PAC-Bayes theory.

In conclusion, this chapter explores the data sloppiness as a possible cause of the low-dimensional training manifold we observed, providing empirical and theiretical evidence and connect to previous works.

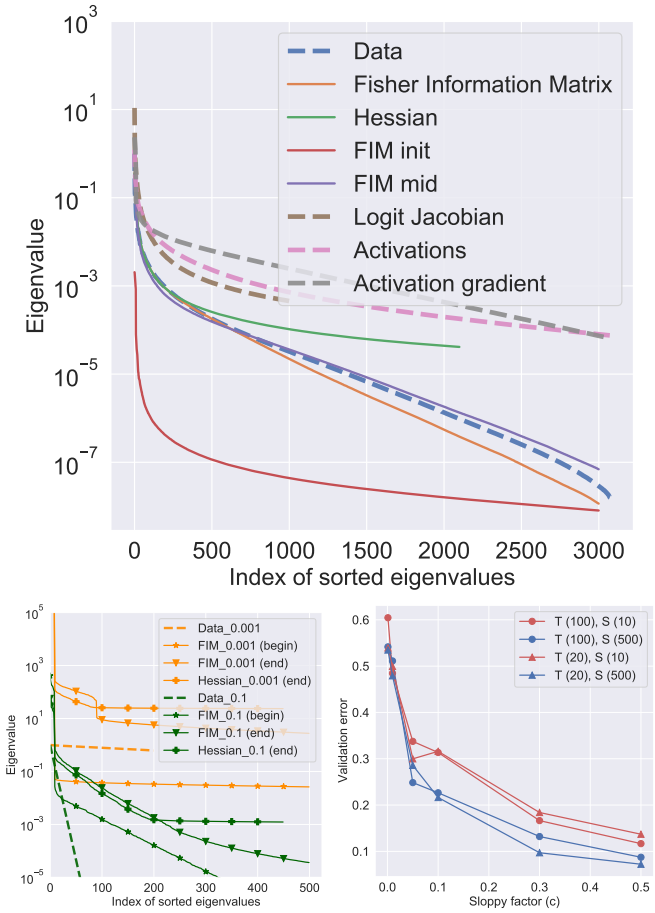


Figure 4.1: **Top:** Eigenspectra of the correlations of the inputs, activations and activation gradients, logit Jacobians and the FIM, Hessian at the end of training; for FIM we also calculate the spectra at initialization and middle of training. All eigenspectra are scaled by the largest eigenvalue of the input correlations (activation gradients are scaled up by 10^{12}). Eigenspectra corresponding to activations/activation gradients of all layers of the network, and logit Jacobians of all logits are very similar (see Section 4.7.4). Eigenspectra of these quantities are also qualitatively the same at initialization, at the middle of training (see Section 4.7.4.2 Figure 4.10). This plot is drawn for a wide residual network with 10 layers on CIFAR-10 (WRN-10-8 Zagoruyko and Komodakis (2016)), eigenspectra of other networks/datasets are qualitatively the same (see Figure 4.4 and Section 4.7.4).
Bottom Left: Eigenspectra of the input correlation matix, FIM and Hessian at begining and end of training for sloppy factor (slope of the sloppy eigenvalue decay) $c = 10^{-3}$ (orange) and $c = 10^{-1}$ (green). If inputs are not sloppy (small c) then even if there is a sharp drop after the top few eigenvalues (around 100 for orange lines), the eigenspectrum is flat. In comparison, the FIM/Hessian decay by about 3 orders of magnitude for $c = 0.1$. The details of the experiments can be found at Section 4.7.1.
Bottom Right: Validation error of a student (S) network on synthetic datasets of different sloppiness (X-axis) labeled by a teacher network (T). Numbers in brackets indicate number of hidden neurons in two-layer teachers/students. All students in this plot interpolate the training data perfectly. For non-sloppy inputs, interpolation leads to poor generalization, whereas interpolation is not detrimental to generalization for sloppy inputs. As the number of student neurons increases, fixed the teacher's size and the sloppiness factor, the validation error is better. Fixed teacher size, say 20, if inputs are sloppier (sloppy factor of 0.5 vs. 0.1) then we can generalize—roughly equally well—even if the student is smaller (10 vs. 500).

4.2. Background

Consider a dataset $D_n = \{(x_i, y_i)\}_{i=1}^n$ with n samples, $x_i \in X \subset \mathbb{R}^d$ and $y_i \in Y = \{1, \ldots, m\}$. We assume that this dataset is drawn from a joint distribution D on $X \times Y$. A classifier $h_w : X \mapsto [0,1]^m$ parameterized by weights $w \in \mathbb{R}^p$ belongs to a hypothesis space $\{h_w : w \in \mathbb{R}^p\}$; this classifier maps inputs $x \in X$ to m-dimensional categorical distributions $p_w(y \mid x) \in [0,1]^m$. Let Q be a distribution on hypotheses, which is implicitly a measure on $\mathbb{R}^p$. We define

(a) training error of a hypothesis $\hat{e}(h_w, D_n) = \frac{1}{n} \sum_{i=1}^n \mathbf{1}\{y_i \neq \text{argmax}_y(p_w(y \mid x_i))\}$;

(b) population error $e(h_w) = \mathbb{E}_{D_n \sim D^n} [\hat{e}(h_w, D_n)]$;

(c) training loss is $\check{e}(h_w, D_n) = -\frac{1}{n \log(2)} \sum_{i=1}^n \log p_w(y_i \mid x_i)$;

(d) empirical error and loss of the distribution Q of hypotheses $\hat{e}(Q, D_n) = \mathbb{E}_{w \sim Q} [\hat{e}(h_w, D_n)]$ and $\check{e}(Q, D_n) = \mathbb{E}_{w \sim Q} [\check{e}(h_w, D_n)]$, respectively;

(e) population error of distribution Q given by $e(Q) = \mathbb{E}_{D_n \sim D^n} [\hat{e}(Q, D_n)]$; and

(f) population loss is $\check{e}(Q) = \mathbb{E}_{D_n \sim D^n} [\check{e}(Q, D_n)]$.

Hessian and Fisher Information Matrix (FIM) The Hessian $H \in \mathbb{R}^{p \times p}$ is the second derivative of the empirical loss with respect to the weights w, i.e., $H_{ij} = \partial_i \partial_j \check{e}(h_w, D_n)$. The Fisher Information Matrix (FIM) $F \in \mathbb{R}^{p \times p}$ has entries

$$F_{ij} = \frac{1}{n} \sum_{k=1}^n \sum_{y=1}^m p_w(y \mid x_k) \partial_i \log p_w(y \mid x_k) \partial_j \log p_w(y \mid x_k).$$

It is important to note the expectation over the outputs y. The empirical FIM is an approximation of the FIM where one sets $y = y_k$. Both the Hessian and FIM are large matrices and it is difficult to compute them for modern deep networks. Therefore some of our experiments use a Kronecker-factor approximation (Martens and Grosse, 2016) of a block diagonal Hessian and FIM where cross-terms $\partial_i \partial_j$ across different layers of a deep network are set to zero.

4.2.1. PAC-Bayes Generalization Bounds

The PAC-Bayesian framework developed in Langford and Seeger (2001); McAllester (1999) allows us to estimate the population error of a randomized hypothesis with distribution Q using its empirical error and its Kullback-Leibler (KL) divergence with respect to some prior distribution P. For any $\delta > 0$, with probability at least $1 - \delta$ over draws of the dataset D_n, we have

$$\mathrm{kl}(\hat{e}(Q, D_n), e(Q)) \leq \frac{\mathrm{KL}(Q, P) + \log(n/\delta)}{(n-1)}, \tag{4.2.1}$$

where $\mathrm{KL}(Q, P) = \int \mathrm{d}Q(w) \log(\mathrm{d}Q/\mathrm{d}P)(w)$. We will also define a KL divergence between two Bernoulli random variables with parameters b, a as $\mathrm{kl}(b, a) = b \log(b/a) + (1-b) \log((1-b)/(1-a))$. The right hand-side of this inequality can be minimized to compute a distribution Q that has a small generalization error (Langford and Caruana, 2002; Dziugaite and Roy, 2017). Typically, we pick a simple form for distributions Q and P, say Gaussian. We can also have hyper-parameters for the prior P, say the scale ϵ of the covariance of P and search over this scale while optimizing the bound. See Section 4.7.2 for details.

4.2.2. Data-dependent PAC-Bayes Priors

The posterior Q in Equation (4.2.1) may depend upon the training samples D_n, e.g., it could be the distribution on the weight space induced by a randomized training algorithm like stochastic gradient descent (SGD). The prior P can depend upon the data distribution D, but not the samples D_n themselves. Although it is common to use priors that do not depend upon the data at all, it is has been increasingly noticed that data-distribution dependent priors may provide tighter bounds (Dziugaite and Roy, 2018). To gain intuition, recall that in the expression for the KL-divergence between two Gaussians $Q = N(w, \Sigma_q)$ and $P = N(w_0, \Sigma_p)$, we have a term of the form $(w - w_0)^\top \Sigma_p^{-1} (w - w_0)$ that depends upon the distance between trained weights w and the initialization w_0. Priors P that do not depend upon the data may therefore incur a large KL-term.

FIM and Hessian-dependent priors We can pick a prior using a subset of the training samples (Ambroladze et al., 2007), e.g., we can center the Gaussian prior on weights trained on this subset, to obtain a better PAC-Bayes bound—the theory allows this. Doing so leads to a worse denominator in Equation (4.2.1), although this may be mitigated by a smaller numerator. Parrado-Hernández et al. (2012) also define expectation-priors,

i.e., where we choose a prior that depends on the data *distribution* and, in practice, evaluate this prior using samples in the training dataset in lieu of the distribution. For example, PAC-Bayes theory allows both picking the prior covariance Σ_p to be $\Sigma_p \propto F_{w_0}$ and $\Sigma_p \propto \tilde{H}_{w_0}$ where $\tilde{H}$ is the Gauss-Newton approximation of the Hessian. But while we may use all the samples to compute the FIM, we should compute the Hessian on a separate subset of the data.

4.3. Theoretical Results

We prove how sloppiness in the Hessian and the FIM is related to sloppiness of the correlations of the activations (Section 4.3.1) and the inputs (Section 4.3.2). We then exploit sloppiness to compute PAC-Bayes generalization bounds (Section 4.3.3) and develop an expression for the effective dimensionality of a deep network (Section 4.4.1). All proofs are provided in Section 4.7.3. The theory in this section applies for general deep networks; we will remark when restrictions are in place.

4.3.1. Sloppy Input Correlation Matrix Leads to a Sloppy FIM and Hessian

Consider a deep network with L layers with weights $w = (w^0, w^1, \ldots, w^L)$. Activations of the k^{th} layer are given by $h^k = \sigma(w^{k-1}h^{k-1})$, and we set $h^0 \equiv x$. The non-linearity σ acts element-wise upon its argument and we assume that it has a bounded derivative $|\sigma'(x)| \leq a$ with $\sigma(x) = 0$ in which case $|\sigma(x)| \leq a\,|x|$; ReLU, leaky ReLUs and tanh satisfy this assumption. Preactivations (before nonlinearities) will be denoted by $u^k = w^{k-1}h^{k-1}$ for $k = 1, \cdots, L+1$, and for clarity, we use a special notation $z \equiv u^{L+1}$ to denote the logits of the network. The dimensionality of these quantities is $h^k \in \mathbb{R}^{d_k}$, $w^k \in \mathbb{R}^{d_{k+1} \times d_k}$ and $w^L \in \mathbb{R}^{m \times d_L}$. The linear map represented by w^k can model both fully-connected layers and convolutional layers. For the sake of exposition, we set all the bias terms to zero.

Theorem 4.3.1 (Trace of the FIM and Hessian are bounded by that of the input correlation matrix)**.** *For any weights, the trace of the FIM F_w and the Gauss-Newton approximation of the Hessian $\tilde{H}_w$ are both upper-bounded by*

$$2ma^{2L} tr\left(\mathbb{E}[xx^\top]\right) \prod_{j=0}^{L} \left\|w^j\right\|_2^2 \left(\sum_{j=0}^{L} \left\|w^j\right\|_2^{-2}\right). \tag{4.3.1}$$

The Gauss-Newton approximation which neglects the so-called H terms of the Hessian (Papyan, 2019) is good towards the end of training when the logits have a small entropy. For the FIM, the above bound

is remarkable however, it indicates that the trace of FIM is controlled by that the input correlations and multiplicative terms that depend upon the ℓ_2 norm of the weights.

We can also go beyond the trace and control the entire eigenspectrum. But this is difficult to do in general because both FIM and Hessian are a result of multiple nonlinear operations on the inputs. We therefore bound the eigenvalues of a block-diagonal approximation of the FIM in terms of the eigenvalues of the activations.

Lemma 4.3.2 (Block-diagonal approximation of the FIM is sloppy if the activations are sloppy)**.** *Let $spec(A)$ denote the eigenvalues $(\lambda_1(A), \ldots, \lambda_p(A))$ of a matrix A in descending order. For a constant c, let $spec(A) \preceq c\, spec(B)$ denote that $\lambda_i(A) \leq c\lambda_i(B)$ for all $i \leq p$. For any logit z_i, for all layers $k \leq L$, we have*

$$spec\left(\mathbb{E}\left[\frac{dz_i}{dw^k}\frac{dz_i}{dw^k}^\top\right]\right) \preceq a^{2(L-k)} \prod_{j=k+1}^{L} \|w^j\|_2^2 \\ spec(I_{d_{k+1}}) \otimes spec\left(\mathbb{E}\left[h^k {h^k}^\top\right]\right), \tag{4.3.2}$$

with $\prod_{j=L+1}^{L} \|w_j\|_2^2 \equiv 1$. A similar result also holds for the sum of logits $\sum_{i=1}^{m} z_i$ as in Lemma 4.3.2 (see Corollary 4.7.7). The proof of this lemma also shows that a block-diagonal approximation of the Gauss-Newton approximation of the Hessian is sloppy if the activations are sloppy.

This lemma indicates that the eigenspectrum of the block-diagonal approximation of the FIM (concatenation of the eigenspectra of different blocks) is controlled by the eigenspectrum of the activation correlations of different layers. Our experiments show that activations of all layers (except the logits) of a trained deep network are sloppy.

4.3.2. Special Cases Where Sloppy Inputs Lead to Sloppy Activations and Thereby Sloppy FIM and Hessian

Although our experiments show that activations are sloppy if the inputs are, it seems rather difficult to prove in general. We therefore discuss two special cases where this holds. The first case is for a kernel machine with an inner product kernel while the second case assumes that the width of the network goes to infinity and weights remain bounded in ℓ_2 norm.

Remark 4.3.3 (Eigenspectrum of inner product kernel is controlled by that of its inputs). Let $x_i \in \mathbb{R}^d$ for

$i \leq n$ be iid random vectors. Karoui (2010a, Theorem 2.1) shows that the Gram matrix of an inner product kernel $M_{i,j} = f\left(\frac{x_i^\top x_j}{d}\right)$ for some function f can be approximated by

$$K = \left(f(0) + f''(0)\frac{\text{tr}(\Sigma_d^2)}{2d^2}\right) 11^\top + f(0)\frac{XX^\top}{d} + v_d I_n$$
$$\text{where } v_d = f\left(\frac{\text{tr}(\Sigma_d)}{d}\right) - f(0) - f'(0)\frac{\text{tr}(\Sigma_d)}{d}.$$

More precisely $\|M - K\|_2 \to 0$ in probability when $d, n \to \infty$ for a fixed ratio d/n. Note that v_d is small when $\frac{\text{tr}(\Sigma_d)}{d}$ is small. Hence, we can see that the eigenspectrum of K, and thereby M, is controlled directly by that of $XX^\top$.

Note that this argument cannot directly be used for a deep network because correlations of activations in the network are not an inner product kernel. But this indicates that even for such a kernel machine, sloppiness of the inputs leads to sloppiness of the FIM.

Remark 4.3.4 (Infinitely wide network with bounded weight norm). If the ℓ_2 norm of the weights is bounded, we show in Lemma 4.7.2 that

$$\text{tr}\left(\mathbb{E}\left[h^k h^{k\top}\right]\right) \leq a^2 \left\|w^{k-1}\right\|_2^2 \text{tr}\left(\mathbb{E}\left[h^{k-1} h^{k-1\top}\right]\right).$$

If we iterate upon this inequality down to the last layer to $\text{tr}\mathbb{E}[xx^\top]$ on the right hand-side (which is a constant). If the width of the k^{th} layer goes to infinity, for the trace to be summable, we have that the eigenvalues of $\mathbb{E}[h^k h^{k\top}]$ decay faster than $\mathcal{O}(1/i)$.

4.3.3. Analytical Bound on Generalization

Consider a deep network trained to minimize the loss $\breve{e}(h_w, D_n)$. Assume that w is a local minimum of the objective and thus the Hessian H_w is positive semi-definite. We can write H_w as its orthonormal decomposition $H_w = U_w \Lambda_w U_w^\top$ where $\Lambda_w = \text{diag}(\lambda_1, \ldots, \lambda_p)$ with eigenvalues $\lambda_1, \geq \cdots \geq \lambda_p \geq 0$ arranged in descending order. Consider a Gaussian posterior $Q = N(\mu_q, \Sigma_q)$ with the mean $\mu_q = w$ fixed. We would like to compute the best Σ_q that gives a tight PAC-Bayes bound.

We use a loose version of the bound $e(Q) \leq L(\Sigma_q) := \breve{e}(Q, D_n) + \text{KL}(Q, P)/(2(n-1))$ to simplify the

analytical calculation and show in Section 4.7.2.2 that

$$\Sigma_q = U_w(\bar{\Lambda}_w)^{-1}U_w^\top, \tag{4.3.3}$$

$$\text{where } \bar{\lambda}_i = 2(n-1)\lambda_i + \epsilon \quad \forall i \leq p. \tag{4.3.4}$$

This posterior gives a non-vacuous bound on the generalization error (as explained in Section 4.4.2) and to our knowledge, this is the only analytical bound that is non-vacuous and does not use weight compression (e.g., Zhou et al. (2018)). For example, the bound for a fully-connected network on MNIST with one hidden layer of 600 neurons is 0.32 while the test error $e(Q)$ is ≈ 0.089. For comparison, Dziugaite and Roy (2017) *numerically optimize* Equation (4.2.1) to get a bound of 0.161.

Remark 4.3.5 (PAC-Bayes posterior is more spread out along sloppy eigenvectors). In Equation (4.3.4), we can think of the scaled prior inverse variance $\epsilon/(2(n-1))$ as a threshold beyond which the sloppy eigenvalues of the Hessian λ_i are small enough and the loss changes so little that the optimal PAC-Bayes posterior in Equation (4.2.1) focuses on accurately capturing the prior's covariance to obtain a small KL-term. For eigenvalues above this threshold, e.g., the stiff eigenvalues, the optimal posterior has to ensure that the empirical loss is not large.

4.4. Effective Dimensionality of a Deep Network

4.4.1. Definition of Effective Dimensionality

Motivated by Remark 4.3.5, we define the effective dimensionality for a deep network at weights w as the number of eigenvalues of the Hessian H_w with magnitude at least $\epsilon/(2(n-1))$, i.e.,

$$p(n,\epsilon) = \sum_{i=1}^{p} \mathbf{1}\Big\{|\lambda_i(H_w)| \geq \tfrac{\epsilon}{2(n-1)}\Big\}. \tag{4.4.1}$$

Section 4.7.2.3 gives the calculation for why this is a good definition of the dimensionality. It indicates that the threshold $\epsilon/(2(n-1))$ can be thought of as the "elbow" in the eigenspectra in Figure 4.1 (top), which separates the stiff eigenvalues which decrease quickly and the sloppy eigenvalues. This gives an easy way to compute the effective dimensionality, e.g., for the purposes of model selection.

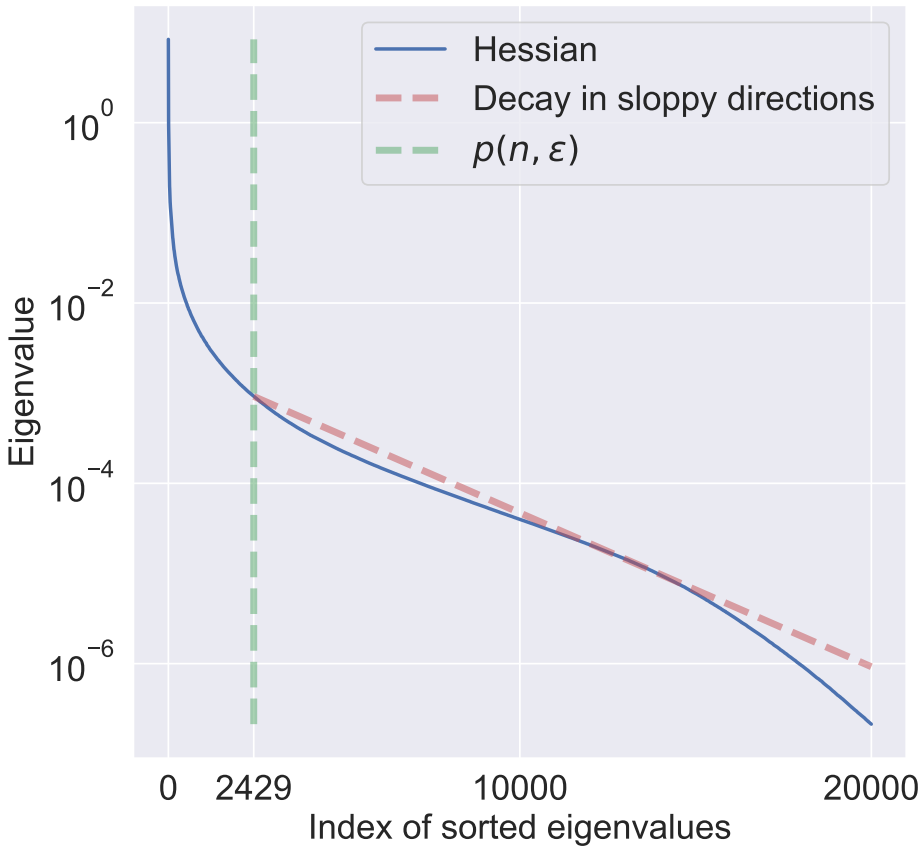


Figure 4.2: For two layer fully connected network (FC-600-2), we calculated the eigenspectrum (blue) of Kronecker-factored approximation of the Hessian at the mean of the posterior Q. The dimensionality $p(n, \epsilon)$ (green) was calculated using the ϵ obtained by the same procedure. The red line shows the linear decay of sloppy eigenvalues (slope is 0.0004). The green line is close to the elbow and effectively splits the stiff and sloppy eigenvalues.

Remark 4.4.1 (Why does the effective dimensionality depend on ϵ?). Our definition in Equation (4.4.1) may seem unusual because ϵ is a user-chosen parameter but this is only an artifact of PAC-Bayes theory. As $\epsilon \to 0$, the effective dimensionality converges to the number of weights p, but for non-zero values of ϵ, where the PAC-Bayes theory effectively restricts its predictions to a subset of the hypothesis space, this expression coupled with the analytical calculation in Equation (4.3.4) may provide a useful way to perform model selection.

Remark 4.4.2 (Why does the effective dimensionality depend on n?). The fact that $p(n, \epsilon)$ depends upon n is reminiscent of the Bayesian Information Criterion (BIC) where the the model complexity term scales with $\log n$ (Schwarz et al., 1978). The dependence on n in our cases also arises for similar reasons, from a balance between the training error $\hat{e}(Q, D_n)$ and the KL-term in Equation (4.2.1). As $n \to \infty$, we see that $p(n, \epsilon) \to p$. This is because for inputs with sloppy dimensions the model needs to capture *all* the dimensions to predict accurately.

4.4.2. Definition of Sloppiness

We next build upon Section 4.4.1 to define sloppiness.

Definition 4.4.3 (Strength factor and sloppy factor)**.** Let $\lambda_i(A)$ denote eigenvalues of a positive semi-definite matrix $A \in \mathbb{R}^{p \times p}$ in descending order $\lambda_1 \geq \cdots \geq \lambda_p$. The strength factor for a model with effective dimensionality $p(n, \epsilon)$ at a local minimum w (where H_w is positive semi-definite) is defined to be

$$s(n, \epsilon) = \sum_{i=1}^{p(n,\epsilon)} 1 + \log\left(\frac{2(n-1)\lambda_i(H_w)}{\epsilon} + 1\right). \tag{4.4.2}$$

The strength factor characterizes the stiff eigenvalues of the eigenspectrum. For a matrix A, the sloppy factor for such a model at index r is defined to be

$$c(A, r) = \sup\{c' \geq 0 : \lambda_i(A) \leq \lambda_r(A) e^{-c'(i-r)} \forall i \geq r \geq 1\} \tag{4.4.3}$$

This definition implicitly means that the small eigenvalues beyond $\lambda_r(A)$ are uniformly distributed across an exponentially large range (λ_r, λ_p) if $c(A, r) > 0$. We will be primarily interested in setting the index r to be simply $p(n, \epsilon)$. Note that sloppiness is a phenomenon pertaining to the *non-zero eigenvalues* of a matrix

and is relevant even if the matrix is singular, e.g., the FIM loses rank for non-identifiable models like deep networks (Amari et al., 2002).

How do the strength and sloppy factor affect generalization? Let us simplify notation to write the sloppy factor as $c(n,\epsilon) \equiv c(H_w, p(n,\epsilon))$. Under the assumption that the $c(n,\epsilon)$ is non-negative, when the training error $\check{e}(h_w, D_n)$ is close to zero, we show in Section 4.7.2 a loose version of PAC-Bayes bound $\check{e}(Q, D_n) + \mathrm{KL}(Q,P)/(2(n-1))$ (this was also used in Method 1 in Section 4.3.3) is

$$\frac{s(n,\epsilon) + 2/c(n,\epsilon) + \epsilon \left\|w - w_0\right\|_2^2}{4(n-1)}. \tag{4.4.4}$$

Thus, the strength and sloppy factor together determine the generalization performance. If the Hessian H_w is sloppy, then the effective dimensionality $p(n,\epsilon)$ is small. This ensures that both $s(n,\epsilon)$ and $1/c(n,\epsilon)$ are small compare to n. The third term $\epsilon \left\|w - w_0\right\|^2$ comes from the the fact that the mean of P and Q are different. It is typically not large compared to n. For example, for a two-layer fully-connected network on MINST, $p(n,\epsilon) = 2429$, $s(n,\epsilon) = 6810$, $1/c(n,\epsilon) = 2545$, and $\epsilon \left\|w - w_0\right\|^2 = 8526$, with $n = 55000, \epsilon = 101.3$). For comparison, if we have an isotropic Hessian $\lambda_i \equiv \lambda$, either $s(n,\epsilon)$ or $1/c(n,\epsilon)$ will be $\mathcal{O}(p)$ and p is about 0.8 million.

This suggests that **even if the hypothesis class of deep networks is very large, sloppiness of H_w, which is inherited from sloppiness of the input data, restricts the set of hypotheses that the trained model belongs to**, the three quantities that we have defined here $p(n,\epsilon)$, $s(n,e)$ and $c(n,\epsilon)$ together help understand this phenomenon.

4.5. Empirical Validation

We use fully-connected networks (of varying widths, and up to two hidden layers), convolutional networks (LeNet, ALL-CNN of Springenberg et al. (2015) and wide residual network of Zagoruyko and Komodakis (2016)) of varying sizes on MNIST (LeCun et al., 1990) and CIFAR-10 (Krizhevsky, 2009) for empirical validation of our theoretical results. See Section 4.7.1 for further details.

To be able to work with Hessian/FIM of large networks, in some cases, e.g., Figure 4.1 we compute fewer eigenvalues, but compute them exactly without any approximations.

Section 4.7.4.1 shows the eigenspectra of the Hessian, FIM and correlations of the activations, logit Jacobians and activation gradients for two and three-layer fully-connected networks on MNIST and All-CNN and a wide residual network on CIFAR-10. The eigenspectra are qualitatively the same as those in Figure 4.1 so we do not repeat them in the main text. Figure 4.3 studies how eigenspectra of FIM and Hessian compare to their KFAC approximations.

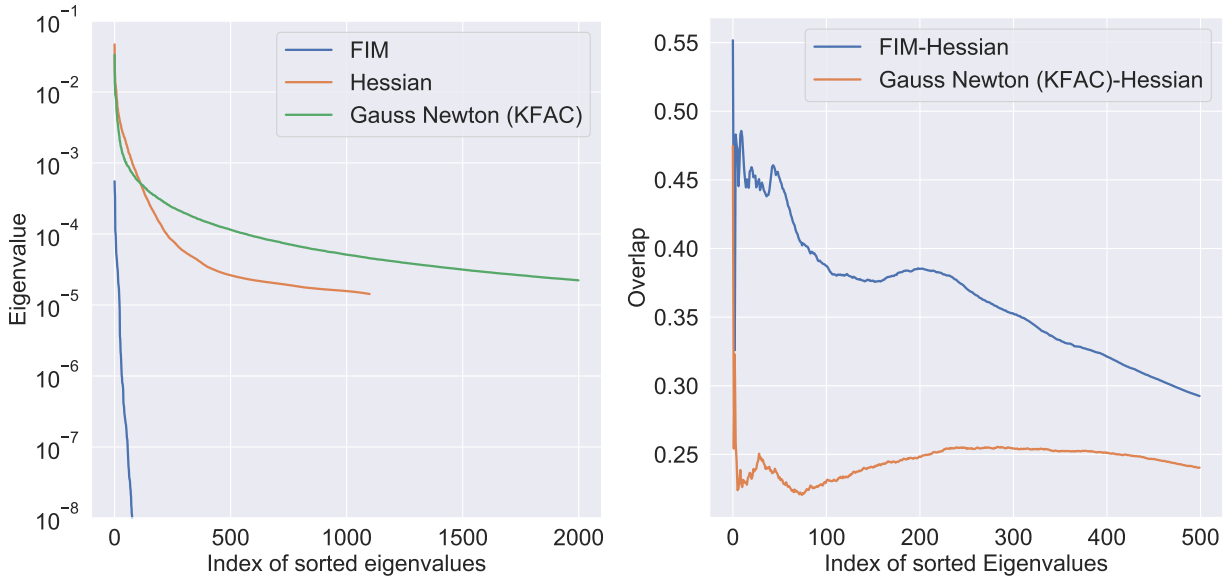


Figure 4.3: (Left) **Eigenspectra of FIM, Hessian and a KFAC approximation of the Gauss-Newton matrix for a two-layer fully-connected network on MNIST.** Even if FIM's eigenvalues are quite different, its eigenvectors have a large inner product with those of the Hessian (right), much larger than a random vector. KFAC is a good approximation for the eigenvalues of the Hessian but eigenvectors computed from KFAC are quite different from those of the Hessian. This also shows that eigenvectors of the FIM have a strong overlap with those of the Hessian.

4.6. Related Work

Sloppy models in physics and biology Our work is inspired by Brown et al. (2004); Gutenkunst et al. (2007) who noticed that regression models fitted to systems biology data have few stiff parameters that determine the outcome and a large number of sloppy parameters which only weakly determine the outcome. These authors have developed an elaborate geometric understanding of this phenomenon, see Transtrum et al. (2011b) and references therein. While sloppiness is thought to be a universal property of parametric models (Waterfall et al., 2006), the mechanism that causes models to be sloppy has not been studied yet. This work has also exclusively focused on the under-parameterized regime. We connect the sloppiness of a deep

network to the sloppiness of inputs and show that if the inputs are sloppy, then key quantities pertaining to the model, e.g., activations, FIM and Hessian etc., are also sloppy.

Hessian and the FIM of deep networks have been studied to understand the local geometry of the energy landscape and the behavior of SGD, see Hochreiter and Schmidhuber (1997); Chaudhari et al. (2017); Fort and Ganguli (2019), among others. FIM has been used to study optimization (Amari, 1998a; Martens and Grosse, 2016; Karakida et al., 2019), gradient diversity (Yin et al., 2018; Chaudhari and Soatto, 2018), and generalization (Achille et al., 2019c). A number of these works have pointed out that the Hessian and the FIM have spiky/large eigenvalues (Papyan, 2019) along with a bulk of near-zero eigenvalues (Papyan, 2018; Pennington and Bahri), and that this indicates that the energy landscape, or the prediction space, is locally flat. We focus on the decay pattern of the eigenspectra of these matrices and discover that it mirrors the decay pattern of the inputs for typical datasets. We see a strong overlap of the stiff subspace of the Hessian/FIM at initialization with that at the end of training; this is consistent with the analysis in Gur-Ari et al. (2018); Chizat et al. (2019a).

Generalization PAC-Bayes bounds for deep networks have been obtained using the methods of Langford and Caruana (2002) by Dziugaite and Roy (2017); Dziugaite (2020); Zhou et al. (2018). While analytical generalization bounds are often vacuous (Bartlett et al., 2017, 2021a; Neyshabur et al., 2017), we show that if we exploit the sloppiness of the Hessian, then we can obtain non-vacuous analytical bounds. We show that the posterior computed by the method of Dziugaite and Roy (2017) aligns well with sloppy eigenvalues of the Hessian/FIM. We build upon this work and show the benefits of sloppiness by providing data-distribution dependent PAC-Bayes bounds (also see Dziugaite and Roy (2018)).

Bartlett et al. (2020) show that a minimum norm interpolating solution of over-parameterized linear regression can predict accurately if the data matrix has a long tail of small eigenvalues. Our notion of effective dimensionality is also seen in their calculationsloppy: roughly speaking, larger our sloppiness factor c in Definition 4.4.3, better the excess risk in their linear regression, which is consistent with Figure 4.1 (bottom right). Liang and Rakhlin (2018) show similar results on the minimum-norm interpolating solution for kernel regression.

4.7. Appendix

4.7.1. Details of the experimental setup

Data We use the MNIST dataset for experiments on fully-connected networks and LeNet. We setup a binary classification problem (we map {0,1,2,3,4} to label 0 and {5,6,7,8,9} to label 1). We use 55000 samples from the training set to train the model and to optimize the PAC-Bayes bound. We set aside 5000 samples for calculating the FIM, which is used in Method 4 of PAC-Bayes bound optimization. Strictly speaking, it is not required to do so because a prior that depends upon the FIM is an expectation-prior (as discussed in Parrado-Hernández et al. (2012)) but we set aside these samples to compare in a systematic manner to existing methods in the literature that use 55,000 samples. Test error of all models is estimated using the validation set of MNIST. We use the CIFAR-10 dataset for experiments using two architectures, an All-CNN network and a wide residual network. For CIFAR-10, we use 50, 000 samples for training and 10, 000 samples for estimating the test error. No data augmentation is performed for MNIST, for CIFAR-10 we randomly flip images (left to right) with probability 0.5 and select random crops of size 32×32 after adding a padding of 4 pixels on the width and height.

Architectures For experiments on MNIST, we use LeNet-5 (this is a network with two convolutional layers of 20 and 50 channels respectively, both of 5×5 kernel size, and a fully-connected layer with 500 hidden neurons) and fully-connected net with one or two layers and 600 or 1200 neurons on each layer. The latter are denoted as FC-600-1, or FC-1200-2 in our experimental section. For CIFAR-10, we use ALL-CNN (in order to reduce the number of weights, we reduced the number of channels in the first set of blocks to 64, and in the second set of blocks to 128; this is down from 96 and 192 respectively in the original network) and wide residual net with depth 10 and a widening factor of 8. In the latter case, in order to reduce the number of weights which makes computing Hessian amenable, we reduce the number of channels in each block of the WRN to [4, 32, 64, 128], down from [16, 128, 256, 512] for a widen factor of 8.

Training procedure We train for 30 epochs on MNIST and for 100 epochs on CIFAR-10. The batch-size is fixed to 500 for both datasets. For all experiments with train with Adam and reduce the learning rate using a cosine annealing schedule starting from an initial learning rate of 10^{-3} and ending at a learning rate of 10^{-5}.

Atypical problems For atypical problems in, we constructed a training set of 50,000 samples and a validation set of 10,000 samples. Inputs $x_i \in \mathbb{R}^{200}$ were generated from distribution $N(0, \Lambda)$ where $\Lambda = (\lambda_1, ..., \lambda_{200})$. We set $\lambda_i = b \exp(-ci)$ where and $b/c = 50$; fixing the ratio b/c to be a constant keeps the trace of the data correlation matrix to be about the same for different values of c. Labels were generated by $y_i = \text{argmax}_{y \in [m]} p_w^t(y|x_i)$, where p_w^t is the teacher network randomly initialized with one hidden layer and ten output classes. We train fully-connected networks on these synthetic datasets for 50 epochs; Adam is used with a batch-size of 500 and a cosine learning rate schedule with learning rate that ranges from 10^{-3} to 10^{-5}.

We constructed datasets of Gaussian inputs of varying degrees of sloppiness by selecting decay patterns for the eigenvalues of a diagonal data correlation matrix. For $n^{-1} \text{diag}(XX^\top) = \Lambda$ where $\Lambda = (\lambda_1, \ldots, \lambda_d)$ are eigenvalues in descending order, we set $\lambda_i = b \exp(-ci)$ where b, c are constants. The trace of this correlation matrix is roughly b/c which we keep constant for different datasets. Larger the value of the "sloppy factor" c, more sharp the decay for the eigenspectrum of the data matrix. We randomly initialize a two layer fully-connected neural network with 10 output classes (called the teacher) and use it to label a dataset of such inputs. Note that since the teacher's weights in the first layer multiply the inputs, the correlation matrix of the first layer activations is non-diagonal and we are not being unduly restrictive in picking a diagonal data correlation matrix. We then fit student networks (fully-connected networks with two layers) on this data until they interpolate on the training dataset. Our goal is to study (i) how the various quantities discussed in this paper, e.g., the Hessian, FIM, activations, activation gradients, logit Jacobians , depend upon the sloppiness of the data matrix; (ii) whether the student can interpolate on sloppy datasets without over fitting. Figure 4.1 shows the results of the experiment.

4.7.2. Calculation of the effective dimensionality of a deep network

4.7.2.1 PAC-Bayes bounds

Theorem 4.7.1 (PAC-Bayes generalization bound McAllester (1999); Langford and Seeger (2001))**.** *For every $\delta > 0$, $n \in \mathbb{N}$, distribution D on $\mathbb{R}^k \times \{0, 1\}^m$, and distribution P on $\mathcal{H}$, with probability at least $1 - \delta$*

over $D_n \sim D^n$, for all distributions Q on $\mathcal{H}$,

$$kl(\hat{e}(Q, D_n), e(Q)) \leq \frac{KL(Q, P) + \log \frac{n}{\delta}}{n - 1}$$

We have the following lower-bound from Pinksker's inequality on the KL-divergence between two Bernoulli random variablesloppy:

$$2(q - p)^2 \leq \mathrm{kl}(q, p).$$

We can invert this inequality to get

$$\mathrm{kl}^{-1}(q, p) \leq q + \sqrt{p/2}.$$

When this is substituted into the above PAC-Bayes bound Equation (4.2.1), we have

$$e(Q) \leq \hat{e}(Q) + \sqrt{\frac{\mathrm{KL}(Q, P) + \log(\frac{n}{\delta})}{2(n - 1)}}.$$

Since

$$\mathbf{1}\left\{ y_i \neq \operatorname*{argmax}_y (p_w(y \mid x_i)) \right\} \leq -\frac{1}{\log 2} \log p_w(y_i | x_i)$$

we also have

$$\hat{e}(Q) \leq \breve{e}(Q).$$

Now set $\epsilon = c \exp(j/b)$, for $j \in \mathbb{N}$ and for a fixed $b, c \geq 0$, by the calculations in Section 4.7.3.1, we see that

$$e(Q) \leq \breve{e}(Q) + \sqrt{\frac{\mathrm{KL}(Q, P) + 2 \log(b \log \frac{c}{\epsilon}) + \log(\frac{\pi^2 n}{6\delta})}{2(n - 1)}},$$

holds with probability $1 - \delta$.

4.7.2.2 Calculation for the closed form expression for eigenvalues of the inverse posterior covariance in Equation (4.3.4)

The KL-divergence between two multivariate Gaussians $Q = N(\mu_q, \Sigma_q)$, $P = N(\mu_p, \Sigma_p)$ be two multivariate Gaussians is

$$\mathrm{KL}(Q, P) = \frac{1}{2}\left(\mathrm{tr}(\Sigma_p^{-1}\Sigma_q) - p + (\mu_p - \mu_q)^\top \Sigma_p^{-1}(\mu_p - \mu_q) + \log(\frac{\det \Sigma_p}{\det \Sigma_q})\right). \tag{4.7.1}$$

In order to compute the inverse posterior covariance that minimizes the right-hand side of the PAC-Bayes bound, we would like to solve the problem

$$\begin{aligned} \text{minimize} \quad & L(\Sigma_q) := \breve{e}(Q, D_n) + \frac{\mathrm{KL}(Q, P)}{2(n-1)} \\ \text{such that} \quad & Q = N(w, \Sigma_q) \\ \text{and} \quad & \Sigma_q \succeq 0. \end{aligned}$$

Observe that

$$\breve{e}(h_{w'}, D_n) = \breve{e}(h_w, D_n) + \frac{1}{2}\left\langle w' - w, H_w(w' - w)\right\rangle.$$

$$P = N(w_0, \epsilon^{-1} I).$$

For $P_w = N(w, \epsilon^{-1} I)$, we have

$$\mathrm{KL}(Q, P) = \mathrm{KL}(Q, P_w) + \frac{\epsilon}{2}\left\|w - w_0\right\|^2$$

Hence,

$$\begin{aligned} L(\Sigma_q) &= \int Q(w')\breve{e}(h_{w'}, D_n)\mathrm{d}w' + \frac{1}{2(n-1)}\int Q(w')\log\frac{Q(w')}{P_w(w')}\mathrm{d}w' + \frac{\epsilon}{4(n-1)}\left\|w - w_0\right\|^2 \\ &= \frac{1}{2(n-1)}\int\left(-\log\exp(-2(n-1)\breve{e}(w', D_n)) + \log\frac{Q(w')}{P_w(w')}\right)Q(w')\mathrm{d}w' + \frac{\epsilon}{4(n-1)}\left\|w - w_0\right\|^2 \\ &= \frac{1}{2(n-1)}\int\left(\log\frac{Q(w')}{\exp(-2(n-1)\breve{e}(w', D_n))P_w(w')/Z} - \log Z\right)Q(w')\mathrm{d}w' + \frac{\epsilon}{4(n-1)}\left\|w - w_0\right\|^2 \\ &= \frac{1}{2(n-1)}\left(\mathrm{KL}(Q, B) - \log Z\right) + \frac{\epsilon}{4(n-1)}\left\|w - w_0\right\|^2, \end{aligned}$$

where we have defined

$$B(w') = \exp(-2(n-1)\breve{e}(w', D_n))P_w(w')/Z, \quad and$$
$$Z = \int \exp(-2(n-1)\breve{e}(w', D_n))P_w(w')\mathrm{d}w'.$$

We can now see that $L(\Sigma_q)$ attains a minimum when

$$Q = B \propto \exp(-2(n-1)\breve{e}(w', D_n))P_w(w') \tag{4.7.2}$$

or $\Sigma_q^{-1} = 2(n-1)H_w + \epsilon\, I$, in other words,

$$\Sigma_q = U_w(\bar{\Lambda}_w)^{-1}U_w^\top,$$

where

$$\bar{\lambda}_i = 2(n-1)\lambda_i + \epsilon \quad \forall i \leq p.$$

4.7.2.3 Calculation for Equation (4.4.4)

Recall that the effective dimensionality of a model at a local minimum w is the number of eigenvalues of the Hessian with magnitude at least $\frac{\epsilon}{2(n-1)}$, i.e.,

$$p(n, \epsilon) = \sum_{i=1}^{p} \mathbf{1}\left\{|\lambda_i| \geq \frac{\epsilon}{2(n-1)}\right\},$$

The strength of the model at w is

$$s(n, \epsilon) = \sum_{i=1}^{p(n,\epsilon)} 1 + \log\left(\frac{2(n-1)\lambda_i}{\epsilon} + 1\right).$$

We assume that $c(H_w, p(n, \epsilon)) > 0$. i.e., denote $c(H_w, p(n, \epsilon))$ as $c(n, \epsilon)$

$$\lambda_i \leq \frac{\epsilon}{2(n-1)} \exp(-c(n, \epsilon)(i - p(n, \epsilon)))$$

We can also assume a weaker version of this decay pattern,

$$\sum_{i=p(n,\epsilon)+1}^{p} \lambda_i = \frac{\epsilon}{2(n-1)c(n,\epsilon)}.$$

We approximate the training objective in the neighborhood of w as

$$\breve{e}(h_{w'}, D_n) = \breve{e}(h_w, D_n) + \frac{1}{2}\left\langle w' - w, H_w(w' - w)\right\rangle.$$

and we assume that the model at w is a interpolation solution. In Section 4.3.3, for the posterior $Q = N(w, \Sigma_q)$ that maximizes the loose version of the PAC-Bayes bound Equation (4.2.1), where

$$\Sigma_q = U_w \bar{\Lambda}_w^{-1} {U_w}^\top,$$
$$\bar{\lambda}_i = 2(n-1)\lambda_i + \epsilon.$$

We can now calculate

$$\begin{aligned}
\breve{e}(Q, D_n) - \breve{e}(h_w, D_n) &= \frac{1}{2}\sum_{i=1}^{p} \frac{\lambda_i}{\bar{\lambda}_i} \\
&\leq \frac{p(n,\epsilon) + 1/c(n,\epsilon)}{4(n-1)}, \text{ and} \\
\frac{\mathrm{KL}(Q,P)}{2(n-1)} &= \frac{1}{4(n-1)}\left(\epsilon\left\|w - w_0\right\|^2 - p + \sum_{i=1}^{p} \log\frac{\bar{\lambda}_i}{\epsilon} + \frac{\epsilon}{\bar{\lambda}_i}\right) \\
&\leq \frac{1}{4(n-1)}\left(\epsilon\left\|w - w_0\right\|^2 + \sum_{i=1}^{p(n,\epsilon)} \log\left(\frac{2(n-1)\lambda_i}{\epsilon} + 1\right) + \sum_{i=p(n,\epsilon)+1}^{p} \frac{2(n-1)\lambda_i}{\epsilon}\right) \\
&\leq \frac{1}{4(n-1)}\left(\epsilon\left\|w - w_0\right\|^2 + \sum_{i=1}^{p(n,\epsilon)} \log\left(\frac{2(n-1)\lambda_i}{\epsilon} + 1\right) + \frac{1}{c(n,\epsilon)}\right), \text{ hence} \\
\breve{e}(Q, D_n) + \frac{\mathrm{KL}(Q,P)}{2(n-1)} &\leq \frac{s(n,\epsilon) + 2/c(n,\epsilon) + \epsilon\left\|w - w_0\right\|^2}{4(n-1)}.
\end{aligned}$$

For the KL-term, in the first inequality we have used the fact that $\log(1+x) \leq x$ to split the first summation into two parts; in the second inequality we have used the assumption that the eigenspectrum is sloppy to sum the series from $i = p(n,\epsilon) + 1$; the latter is also used in the inequality for the gap in the loss.

4.7.3. Proofs of Lemmas in Section 4.3.1

We use $\mathbb{E}$ to denote the expectation over inputs x. The following lemmas holds for all distribution of x. In particular, we can choose the distribution of x to be the point mass distribution on the dataset D_n, i.e. $x \sim \frac{1}{n}\sum_{i=1}^{n}\delta_{x_i}$, in this case, $\mathbb{E}\left[xx^\top\right] = \frac{1}{n}XX^\top \in \mathbb{R}^{d\times d}$ is the input corelation matrix.

The following lemma bounds the trace of the activation correlations and the norm of the gradient of each logit with respect to the activations.

Lemma 4.7.2 (Bounding the trace of the correlations of activations and norm of activation gradients)**.** *We have*

$$tr\left(\mathbb{E}\left[h^k h^{k\top}\right]\right) \le a^2 \left\|w^{k-1}\right\|_2^2 tr\left(\mathbb{E}\left[h^{k-1}h^{k-1\top}\right]\right), \tag{4.7.3}$$

and

$$\left\|\frac{dz_i}{dh^k}\right\|_2 \le a\left\|\frac{dz_i}{dh^{k+1}}\right\|_2 \left\|w^k\right\|_2. \tag{4.7.4}$$

Proof of Lemma 4.7.2. For the first inequality in Equation (4.7.3), observe that

$$\begin{aligned}
\text{tr}\left(\mathbb{E}\left[h^k h^{k\top}\right]\right) &\le \sum_{j=1}^{d_k}\mathbb{E}\left[\sigma(u_j^k)^2\right] \\
&\le a^2\sum_{j=1}^{d_k}\mathbb{E}\left[(u_j^k)^2\right] \\
&= a^2\text{tr}\left(\mathbb{E}\left[u^k u^{k\top}\right]\right) \\
&= a^2\text{tr}\left(\mathbb{E}\left[\left(w^{k-1}h^{k-1}\right)\left(w^{k-1}h^{k-1\top}\right)\right]\right) \\
&= a^2\text{tr}\left(w^{k-1}\mathbb{E}\left[h^{k-1}h^{k-1\top}\right]w^{k-1\top}\right) \\
&\le a^2\left\|w^{k-1}\right\|_2^2\text{tr}\left(\mathbb{E}\left[h^{k-1}h^{k-1\top}\right]\right).
\end{aligned}$$

For the second inequality in Equation (4.7.4), observe that

$$\begin{aligned}\frac{\mathrm{d}z_i}{\mathrm{d}h^k} &= \frac{\mathrm{d}z_i}{\mathrm{d}u^{k+1}} w^k \\ &= a\left(\frac{\mathrm{d}z_i}{\mathrm{d}h^{k+1}} \mathbb{1}_{u^{k+1}\geq 0}\right) w^k \\ \Rightarrow \left\|\frac{\mathrm{d}z_i}{\mathrm{d}h^k}\right\|_2 &\leq a \left\|\frac{\mathrm{d}z_i}{\mathrm{d}h^{k+1}}\right\|_2 \left\|w^k\right\|_2 .\end{aligned}$$

where $\mathbb{1}_{\text{cond}}$ is a vector of 1s at elements where the condition is true. □

The above inequalities can be used in Lemma 4.7.3 to bound the trace of the gradient correlation of any logit z_i with respect to weights of a layer w^k.

Lemma 4.7.3 (Bounding the trace of the correlation sum-of-logit Jacobian)**.** *For logit z_i, $i = 1, ..., m$*

$$tr\left(\mathbb{E}\left[\frac{dz_i}{dw^k}\frac{dz_i}{dw^k}^\top\right]\right) \leq a^{2L} tr\left(\mathbb{E}\left[xx^\top\right]\right) \prod_{j=0, j\neq k}^{L} \left\|w^j\right\|_2^2 . \tag{4.7.5}$$

for $k = 0, ..., L$. As a result,

$$tr\left(\mathbb{E}\left[\frac{dz_i}{dw}\frac{dz_i}{dw}^\top\right]\right) \leq a^{2L} tr\left(\mathbb{E}\left[xx^\top\right]\right) \prod_{j=0}^{L} \left\|w^j\right\|_2^2 \left(\sum_{j=0}^{L} \frac{1}{\left\|w^j\right\|_2^2}\right) .$$

Proof of Lemma 4.7.3. The proof follows via an application of Lemma 4.7.2. For $k = 0, 1, ..., L-1$,

$$\begin{aligned}
\operatorname{tr}\left(\mathbb{E}\left[\frac{\mathrm{d}z_i}{\mathrm{d}w^k}\frac{\mathrm{d}z_i}{\mathrm{d}w^k}^\top\right]\right) &= \operatorname{tr}\left(\mathbb{E}\left[\frac{\mathrm{d}z_i}{\mathrm{d}u^{k+1}}\frac{\mathrm{d}z_i}{\mathrm{d}u^{k+1}}^\top \otimes h^k h^{k\top}\right]\right) \\
&= \mathbb{E}\left[\operatorname{tr}\left(\frac{\mathrm{d}z_i}{\mathrm{d}u^{k+1}}\frac{\mathrm{d}z_i}{\mathrm{d}u^{k+1}}^\top\right)\operatorname{tr}\left(h^k h^{k\top}\right)\right] \\
&\leq a^2\left\|\frac{\mathrm{d}z_i}{\mathrm{d}h^{k+1}}\right\|_2^2 \operatorname{tr}\left(\mathbb{E}\left[h^k h^{k\top}\right]\right) \\
&\leq a^2\left\|\frac{\mathrm{d}z_i}{\mathrm{d}h^{L}}\right\|_2^2 \left(\prod_{j=k+1}^{L-1}\left\|w^j\right\|_2^2\right) a^{2(L-k-1)} \\
&\quad a^{2k}\prod_{j=0}^{k-1}\left\|w^j\right\|_2^2 \operatorname{tr}\left(\mathbb{E}\left[xx^\top\right]\right) \\
&\leq a^{2L}\operatorname{tr}\left(\mathbb{E}\left[xx^\top\right]\right)\prod_{j=0, j\neq k}^{L}\left\|w^j\right\|_2^2.
\end{aligned}$$

The third line comes from the fact that the matrix $\frac{\mathrm{d}z_i}{\mathrm{d}u^{k+1}}\frac{\mathrm{d}z_i}{\mathrm{d}u^{k+1}}^\top$ is rank one and its trace is the same as 2-norm. The last inequality comes from the fact that $\left\|w_i^L\right\|_2 \leq \left\|w^L\right\|_2$. For $k = L$,

$$\begin{aligned}
\operatorname{tr}\left(\mathbb{E}\left[\frac{\mathrm{d}z_i}{\mathrm{d}w^L}\frac{\mathrm{d}z_i}{\mathrm{d}w^L}^\top\right]\right) &= \operatorname{tr}\left(\mathbb{E}\left[\frac{\mathrm{d}z_i}{\mathrm{d}w_i^L}\frac{\mathrm{d}z_i}{\mathrm{d}w_i^L}^\top\right]\right) \\
&= \operatorname{tr}\left(\mathbb{E}\left[h^L h^{L\top}\right]\right) \\
&\leq a^{2L}\operatorname{tr}\left(\mathbb{E}\left[xx^\top\right]\right)\prod_{j=0}^{L-1}\left\|w^j\right\|_2^2.
\end{aligned}$$

□

Proof of Theorem 4.3.1. We first calculate an inequality for the Fisher Information Matrix (FIM)

$$\begin{aligned}
F &= \mathbb{E}\left[\sum_{y=1}^{m} p_w(y \mid x)(\partial_w \log p_w(y \mid x))(\partial_w \log p_w(y \mid x))^\top\right] \\
&= \mathbb{E}\left[\partial_w z\left[\sum_{y=1}^{m} p_w(y \mid x)\frac{\mathrm{d}\log p_w(y \mid x)}{\mathrm{d}z}\frac{\mathrm{d}\log p_w(y \mid x)}{\mathrm{d}z}^\top\right]\partial_w z^\top\right]
\end{aligned}$$

For an output distribution $p_w(y \mid x)$ obtained using the softmax operator on the logits z_y

$$p_y \equiv p_w(y \mid x) = \frac{e^{z_y}}{\sum_{y'} e^{z_{y'}}}$$

we have

$$\frac{\mathrm{d}z}{\mathrm{d}\log} p_w(y \mid x) = e_y - p$$

where e_y is the one-hot vector of the class y and $p = [p_1, ..., p_m]$.

$$\begin{aligned}
\sum_{y=1}^{m} p_w(y \mid x) \frac{\mathrm{d}\log p_w(y \mid x)}{\mathrm{d}z} \frac{\mathrm{d}\log p_w(y \mid x)}{\mathrm{d}z}^\top &\preceq \sum_{y=1}^{m} p_w(y \mid x) \left\| \frac{\mathrm{d}\log p_w(y \mid x)}{\mathrm{d}z} \right\|_2^2 I \\
&= (1 - \|p\|_2^2) I \\
&\preceq I
\end{aligned}$$

Hence we have

$$F \preceq \mathbb{E}\left[(\partial_w z) \, (\partial_w z)^\top \right].$$

In the case of the Hessian for the cross-entropy loss we make a similar calculation following the calculation of Fort and Ganguli (2019). For the calculation of Hessian, the expectation $\mathbb{E}$ denotes the expectation with respect to inputs and labels in the training set. We write

$$\begin{aligned}
(\log 2) H &\approx \mathbb{E}\left[(\partial_w z) \, \nabla_z^2 (-\log p_w(y \mid x)) \, (\partial_w z)^\top \right] \\
&= \mathbb{E}\left[(\partial_w z) \left(\mathrm{diag}(p) - p p^\top \right) (\partial_w z)^\top \right] \\
&\preceq \mathbb{E}\left[(\partial_w z) \, (\mathrm{diag}(p)) \, (\partial_w z)^\top \right] \\
&\preceq \mathbb{E}\left[(\partial_w z) \, (\partial_w z)^\top \right].
\end{aligned}$$

In the above calculation, we have kept only the so-called G-term of the Hessian and neglected an additional H-term.

$$\mathbb{E}\left[\sum_{i=1}^{m} (y_i - p_i) \frac{\partial^2 z_i}{\partial w_\alpha \partial w_\beta} \right]$$

which is typically small in practice for a well-trained network because the terms $1 - p_i$ are close to zero

for all logits (Papyan, 2019; Sagun et al., 2016)($\mathbb{E}[\sum_{i=1}^{m}|y_i - p_i|$ is 5.32×10^{-8} for FC-600-2 on MNIST). Hence, both $\text{tr}\,(F)$ and $(\log 2)\text{tr}\,(H)$ can be bounded by

$$\text{tr}\,(F)\,,(\log 2)\text{tr}\,(H) \leq \sum_{i=1}^{m} \mathbb{E}\left[\frac{\mathrm{d}z_i}{\mathrm{d}w}\frac{\mathrm{d}z_i}{\mathrm{d}w}^{\top}\right] \leq ma^{2L}\text{tr}\left(\mathbb{E}\left[xx^{\top}\right]\right)\prod_{j=0}^{L}\|w^j\|_2^2\left(\sum_{j=0}^{L}\frac{1}{\|w^j\|_2^2}\right). \tag{4.7.6}$$

Notice that the $\log 2$ factor in front of $\text{tr}(H)$ comes from the rescaling factor in the definition of $\check{e}(h_w, D_n)$.

□

Remark 4.7.4. The G-term is always positive semi-definite since the output distribution $p \in \mathbb{R}^C$ is always convex on the logits $z \in \mathbb{R}^C$, i.e., $\left(-\log\left(\frac{e^{zy}}{\sum_{y'}e^{zy'}}\right)\right)_{y=1}^{C}$ is convex in z.

Remark 4.7.5. Empirically, the trace of FIM and Hessian at the end of training (Figure 4.3) is usually much smaller than the trace of correlation matrix of logit Jacobians (Figure 4.5). In this case, the prediction of the bound in Equation (4.7.6) seems very loose. However from the above calculation, we also know that

$$\begin{aligned}
\text{tr}(F) &\leq (1 - \|p\|_2^2)\ \text{tr}\left(\sum_{i=1}^{m}\mathbb{E}\left[\frac{\mathrm{d}z_i}{\mathrm{d}w}\frac{\mathrm{d}z_i}{\mathrm{d}w}^{\top}\right]\right), \\
\text{tr}(H) &\leq \text{tr}\left(\mathbb{E}\left[(\partial_w z)\left(\text{diag}(p) - pp^{\top}\right)(\partial_w z)^{\top}\right]\right).
\end{aligned}$$

For trained network that predicts accurately, we usually get the probabilities p that are very close to one-hot vectors of the correct classes. In this case, both $1 - \|p\|_2^2$ and $\text{diag}(p) - pp^{\top}$ are close to zero. This explains why in our experiments the trace of F and H at the end of training are much smaller than that of logit Jacobians.

Proof of Lemma 4.3.2. The proof depends upon Weyl's inequality to control the eigenvalues of the sum of Hermitian matrices. It states that for Hermitian matrices $A, B, C \in \mathbb{R}^{p\times p}$, if $C = A + B$, then

$$\lambda_{i+j-1}(C) \leq \lambda_i(A) + \lambda_j(B), \quad \lambda_{p-i-j}(C) \geq \lambda_{p-i}(A) + \lambda_{p-j}(B) \tag{4.7.7}$$

for all $1 \leq i, j \leq p$. In particular if $B \succeq 0$, then $\lambda_i(C) \geq \lambda_i(A)$ for all $i \leq p$.

We can now write,

$$\begin{aligned}
\mathbb{E}\left[\frac{\mathrm{d}z_i}{\mathrm{d}w^k}\frac{\mathrm{d}z_i}{\mathrm{d}w^k}^\top\right] &= \mathbb{E}\left[\left(\frac{\mathrm{d}z_i}{\mathrm{d}h^{k+1}}\odot\frac{\mathrm{d}h^{k+1}}{\mathrm{d}u^{k+1}}\right)\left(\frac{\mathrm{d}z_i}{\mathrm{d}h^{k+1}}\odot\frac{\mathrm{d}h^{k+1}}{\mathrm{d}u^{k+1}}\right)^\top\otimes h^k h^{k\top}\right] \\
&\preceq \mathbb{E}\left[a^2\left\|\frac{\mathrm{d}z_i}{\mathrm{d}h^{k+1}}\right\|^2 I_{d_{k+1}}\otimes h^k h^{k\top}\right] \\
&= a^2\left\|\frac{\mathrm{d}z_i}{\mathrm{d}h^{k+1}}\right\|^2 I_{d_{k+1}}\otimes\mathbb{E}\left[h^k h^{k\top}\right] \\
&= a^{2(L-k)}\left(\prod_{j=k+1}^{L}||w_j||^2\right) I_{d_{k+1}}\otimes\mathbb{E}\left[h^k h^{k\top}\right]
\end{aligned}$$

Hence, by Equation (4.7.7)

$$\operatorname{spec}\left(\mathbb{E}\left[\frac{\mathrm{d}z_i}{\mathrm{d}w_k}\frac{\mathrm{d}z_i}{\mathrm{d}w_k}^\top\right]\right) \preceq \operatorname{spec}\left(a^{2(L-k)}\prod_{j=k+1}^{L}||w_j||^2 I_{d_{k+1}}\otimes\mathbb{E}\left[h^k h^{k\top}\right]\right)$$

so we have

$$\operatorname{spec}\left(\mathbb{E}\left[\frac{\mathrm{d}z_i}{\mathrm{d}w_k}\frac{\mathrm{d}z_i}{\mathrm{d}w_k}^\top\right]\right) \preceq a^{2(L-k)}\prod_{j=k+1}^{L}||w_j||^2\operatorname{spec}\left(I_{d_{k+1}}\right)\otimes\operatorname{spec}(\mathbb{E}\left[h^k h^{k\top}\right])$$

□

Remark 4.7.6 (Modification using sloppiness of activation gradients). Figure 4.1 shows that the slope of decay of FIM and the activations are essentially the same. However, in Equation (4.3.2) if spec $\left(\mathbb{E}\left[h^k h^{k\top}\right]\right)$ decays as $\mathcal{O}(\exp(-ci))$, the decay of spec $\left(\mathbb{E}\left[\frac{\mathrm{d}z_i}{\mathrm{d}w^k}\frac{\mathrm{d}z_i}{\mathrm{d}w^k}^\top\right]\right)$ is $\mathcal{O}(\exp(-ci/d_{k+1}))$. This is a loose bound, especially when d_{k+1} is large, e.g., the spectrum could decay much more faster. But note that if we can write a KFAC-approximation

$$\mathbb{E}\left[\frac{\mathrm{d}z_i}{\mathrm{d}w^k}\frac{\mathrm{d}z_i}{\mathrm{d}w^k}^\top\right] \approx \mathbb{E}\left[\frac{\mathrm{d}z_i}{\mathrm{d}u^{k+1}}\frac{\mathrm{d}z_i}{\mathrm{d}u^{k+1}}^\top\right]\otimes\mathbb{E}\left[h^k h^{k\top}\right].$$

then we obtain a stronger decay for the logit gradient when d_{k+1} is large, if we assume that the activations *gradients* are sloppy. If spec $\left(\mathbb{E}\left[\frac{\mathrm{d}z_i}{\mathrm{d}u^{k+1}}\frac{\mathrm{d}z_i}{\mathrm{d}u^{k+1}}^\top\right]\right)$ decays as $\exp -c_1 i$ and spec $\left(\mathbb{E}\left[h^k h^{k\top}\right]\right)$ decays as

$\exp -c_2 j$, then the $(i+j)^2$th largest eigenvalue of $\mathbb{E}\left[\frac{\mathrm{d}z_i}{\mathrm{d}w^k}\frac{\mathrm{d}z_i}{\mathrm{d}w^k}^\top\right]$ is smaller than $\exp(-\min\{c_1,c_2\}(i+j))$, hence the kth largest eigenvalue of $\mathbb{E}\left[\frac{\mathrm{d}z_i}{\mathrm{d}w^k}\frac{\mathrm{d}z_i}{\mathrm{d}w^k}^\top\right]$ is smaller than $\exp(-\min\{c_1,c_2\}\sqrt{k})$. Hence, the decay rate of spec $\left(\mathbb{E}\left[\frac{\mathrm{d}z_i}{\mathrm{d}w^k}\frac{\mathrm{d}z_i}{\mathrm{d}w^k}^\top\right]\right)$ is $\mathcal{O}\left(\exp(-\min\{c_1,c_2\}\sqrt{k})\right)$.

Corollary 4.7.7. *Denote the FIM and Hessian with respect to the kth layer $F(w_k), H(w_k)$ respectively, then we have,*

$$\textit{spec}\left(F(w_k)\right), \textit{spec}\left(H(w_k)\right) \preceq 2ma^{2(L-k)} \prod_{j=k+1}^{L} \left\|w^j\right\|_2^2 \textit{spec}(I_{d_{k+1}}) \otimes \textit{spec}\left(\mathbb{E}\left[h^k h^{k\top}\right]\right).$$

As in Lemma 4.3.2, $\prod_{j=L=1}^{L} \left\|w^j\right\|_2^2 = 1$.

Proof. From Lemma 4.7.3 we know that

$$F(w_k), H(w_k) \preceq 2\mathbb{E}\left[(\partial_{w_k} z)(\partial_{w_k} z)^\top\right]$$

Let $s = \sum_{i=1}^m z_i$ be the sum of logits, then we have

$$\begin{aligned} F(w_k), H(w_k) &\preceq 2\mathbb{E}\left[\left(\frac{\mathrm{d}s}{\mathrm{d}w_k}\right)\left(\frac{\mathrm{d}s}{\mathrm{d}w_k}\right)^\top\right] \\ &\preceq 2ma^{2(L-k)} \prod_{j=k+1}^{L} \left\|w^j\right\|_2^2 \mathrm{spec}(I_{d_{k+1}}) \otimes \mathrm{spec}\left(\mathbb{E}\left[h^k h^{k\top}\right]\right) \end{aligned}$$

The second inequality comes from a similar calculation as in Lemma 4.3.2 for network with one added layer where $h_{L+1} = u_{L+1} = z$, $u_{L+2} = w_{L+1}h_{L+1}$, and $w_{L+1} = [1, ..., 1]$, $\|w_{L+1}\|_2^2 = m$. □

4.7.3.1 Optimizing parameters of the prior in the PAC-Bayes bound

The prior should be fixed before looking at the training set, but for all methods above, we optimize the scale of the prior. We do this by adding an additional penalty in the KL term. Assume that a^i for $i = 1, \ldots, m'$ are the number of parameters in the prior that we can select, we choose $a^i = (1/c^i)\exp(-j^i/b^i)$ for $j^i \in \mathbb{N}$. We reindex j^i as a single index $k = (\sum_i j^i)^{m'}$, then if the PAC-Bayes bound for each index k is designed to hold with probability at least $1 - \frac{6\delta}{\pi^2 k^2}$, then by union bound, it will hold uniformly for all $k \in \mathbb{N}$ with

probability at least $1-\delta$. For a bound that holds with probability $1-\delta'$, the penalty we should add is $\log \frac{n}{\delta'}$, hence, using the relation

$$a^i = (1/c^i)\exp(j^i/b^i), \quad \delta' = \frac{6\delta}{\pi^2 k^2}, \quad k = (\sum_i j^i)^{m'}$$

we add the penalty

$$\varphi(a^1, ..., a^{m'}) = 2m' \log(\sum_i b_i \log(c^i a^i)) + \log \frac{\pi^2 n}{6\delta}$$

Similarly, for any positive or negative integer j^i, we can set $k = (\sum_i 2|j|^i)^{m'}$ to get the penalty

$$\varphi(a^1, ..., a^{m'}) = 2m' \log(2 \sum_i \left|b_i \log(c^i a^i)\right|) + \log \frac{\pi^2 n}{6\delta}$$

4.7.4. Further experimental studies

4.7.4.1 Additional results on the sloppiness of different architectures and datasets

MNIST in spite of its lower dimensionality has roughly the same range of eigenvalues but it has a very small threshold r in Definition 4.4.3 which indicates that data has a lower number of effective dimensions than CIFAR-10. The FIM (empirical FIM is essentially the same line) shows a very strong decay for MNIST; since the trace of the FIM has been used as an indicator of the information stored in the weights (Achille et al., 2018), this indicates that the weights have to store very little information to predict MNIST well. The Hessian and FIM have very different eigenvalues for MNIST but as Figure 4.3 indicates the two matrices have a larger overlap in their top eigenvectors. Eigenspectra of other networks on MNIST are similar to Figure 4.4 while those of CIFAR-10 are similar to Figure 4.1.

In Figure 4.5, we compare the correlation matrices of logit Jacobian for different logits, which shows that the eigenspectra for different logits are similar. In Figure 4.6 and Figure 4.7 we compare the correlation matrices of activations and their gradients. From the figures, we can see that the eigenspectra are similar for different layers, which shows that the sloppiness is preserved as we getting into higher layers of neural network. In 4.8 and 4.9 we ploted the eigenspectra for different networks. The similarity of eigenspectra of matrices

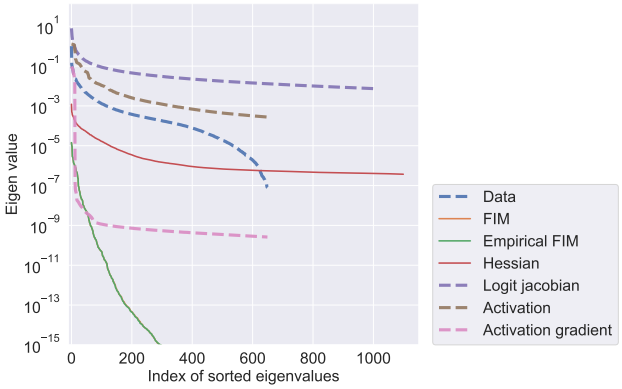


Figure 4.4: Eigenspectra for a two-layer fully-connected network on MNIST. The eigenspectra are qualitatively the same as those of Figure 4.1, e.g., there is a sharp drop at the beginning and a long, linear tail of small eigenvalues follows. Slopes of the eigenspectra of activations, activation gradients, Jacobians and Hessian mirror those of the data. In contrast to Figure 4.1, the slope of the FIM is quite different here. The Empirical FIM and FIM overlaps with each other since the model is trained to nearly perfect train and validation error.

calculated on same dataset but different architectures strongly indicates that the sloppiness of Hessian, FIM, correlations of logit Jacobians, activations and gradients of activations are all inherited from the sloppiness of the data set.

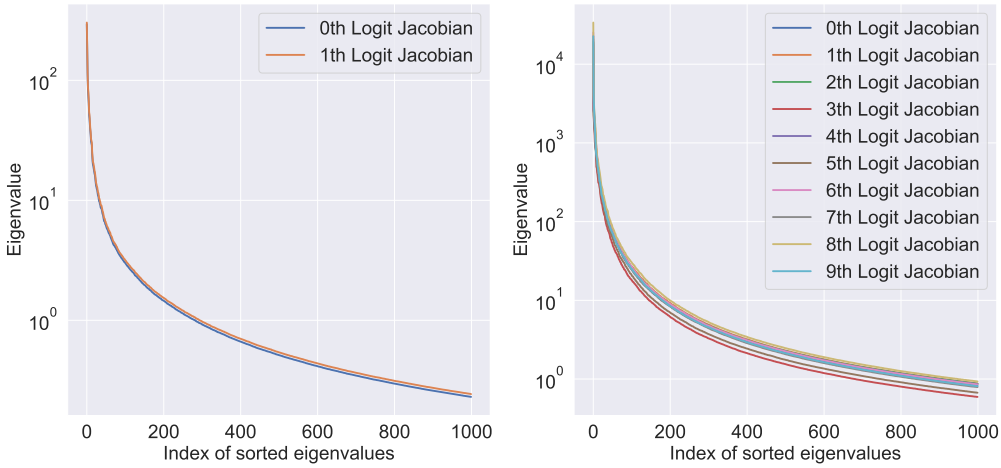


Figure 4.5: Eigenspectra of the correlation matrices of Jacobian of logits for FC-600-2 on MNIST (Left) and wide residual net on CIFAR-10 (Right). The eigenspectra are similar for different logits.

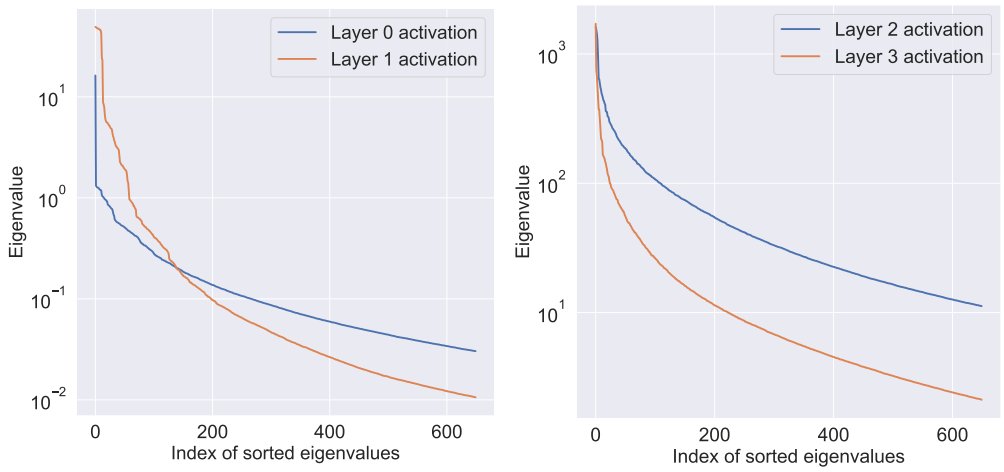


Figure 4.6: Eigenspectra of the correlation matrices of activations of different layers for FC-600-2 on MNIST (Left) and wide residual net on CIFAR-10 (Right). For different layers, the eigenspectra are similar.

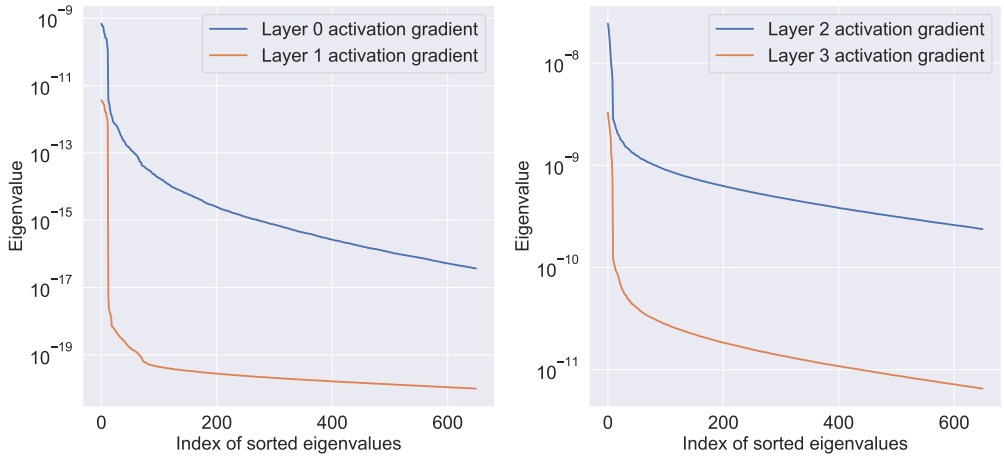


Figure 4.7: Eigenspectra of the correlation matrices of gradients with respect to the activations of different layers for FC-600-2 on MNIST (Left) and wide residual net on CIFAR-10 (Right). For different layers, the eigenspectra are qualitatively similar, and as we move into higher layers of neural networks, the eigenvalues becomes smaller for gradient of activations.

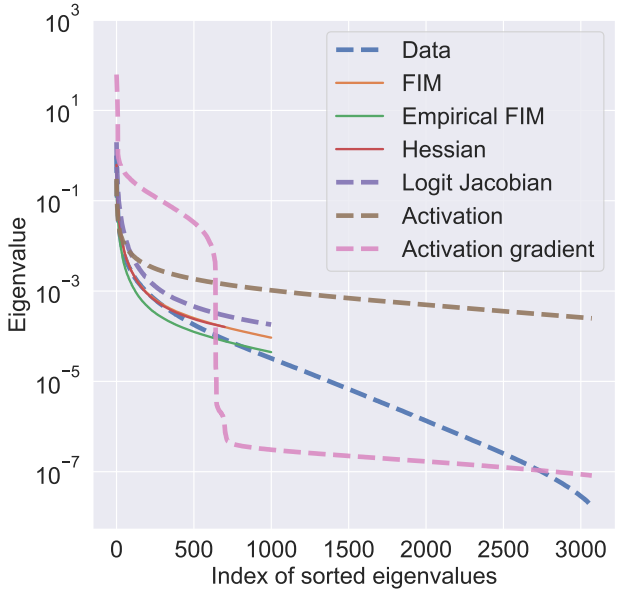


Figure 4.8: Eigenspectra for ALL-CNN on CIFAR-10. The eigenspectra are qualitatively the same as those of Figure 4.1 for a wide residual network on CIFAR-10.

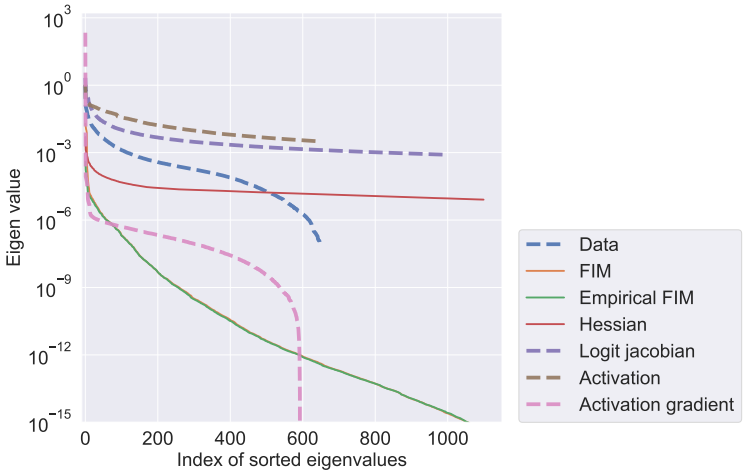


Figure 4.9: Eigenspectra for FC-1200-1 on MNIST. The eigenspectra are qualitatively the same as those of Figure 4.4 for FC-600-2 on MNIST.

4.7.4.2 Eigenspectra at the middle of training

Figure 4.10 shows the eigenspectra for FIM and Hessian of a wide residual net on CIFAR-10 during training, which shows that the eigenspectra are qualitatively the same throughout training.

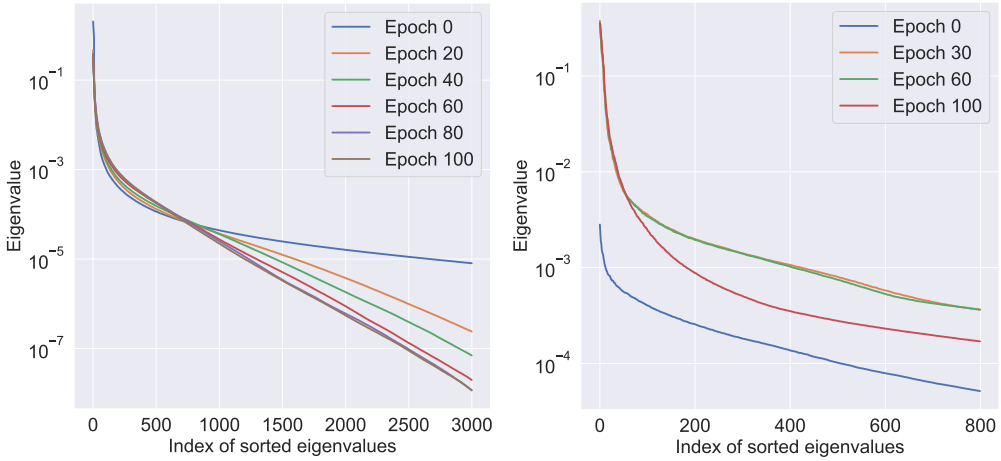


Figure 4.10: (Left) Eigenspectra for FIM of WRN on CIFAR-10 throughout training. The eigenspectrum for epoch 0 are scaled up by 10^3 to bring it to this scale. (Right) Eigenspectra for Hessian of WRN on CIFAR-10 throughout training. We take the absolute value of the eigenvalues. The eigenspectra are qualitatively the same.

4.7.4.3 Eigenspectrum at the end of training for data sets with random labels

Figure 4.11 shows the eigenspectra of empirical FIM at the end of training, in which the eigenspectra is less sloppy for data sets with random labels, and the top eigenvalues increases as we increase the fraction of random labels. This shows that even if we have the same input data set, the sloppiness and the top few eigenvectors can still be affected by the task. The eigenspectra becomes less sloppy and has larger head when the the model is used to learn more difficult tasks (more random labels).

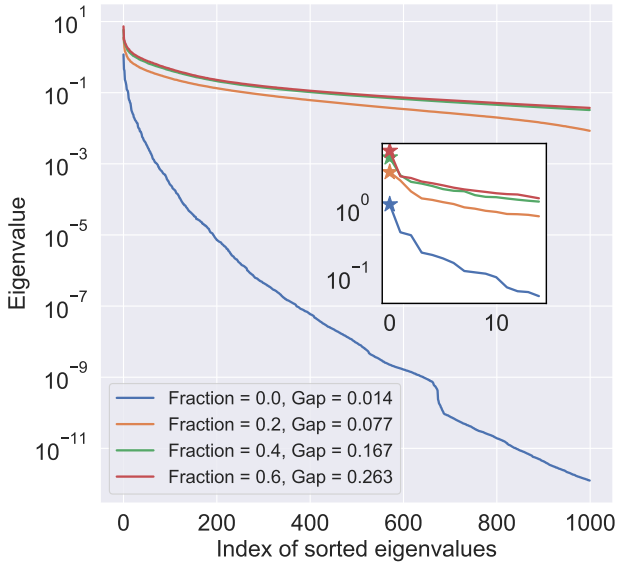


Figure 4.11: **Eigenspectra at the end of training for data sets with random labels.** The plots shows the eigenspectra of empirical FIM at the end of training. The experiment is done on MNIST using fully connected net FC-600-2. The label "Fraction=a" indicates the data set with random label of fraction a. The inset plot shows the top 15 eigenvalues. The line for Fraction=0.0 are scaled up by 10^7. The plots shows that the FIM at the end of training is less sloppy for data sets with random labels, and the top eigenvalues increases as we increase the fraction of random labels.

CHAPTER 5

AN ANALYTICAL CHARACTERIZATION OF SLOPPINESS IN NEURAL NETWORKS: INSIGHTS FROM LINEAR MODELS

5.1. Introduction

Combining the analysis in Section 2.5.5 and Chapter 4, we hypothesize that the following factors could be contributing to the low-dimensionality we observed in the training manifold (i) typical datasets have a sloppy eigenspectrum, possibly leading to a sloppy Fisher Information Matrix (FIM) (ii) the FIM eigenspace is not changing significantly during training, making the FIM eigenspectrum a good approximation of the widths of the model manifold and (iii) typical training procedures initialize models near one specific point in the prediction space, the ignorance P_0. To understand whether these conditions can result in hyperribbon-like structure, we study the training manifold of linear models, where the FIM is just the data covariance matrix and is invariant during training.

This chapter is organized as follows: In Section 5.2 we constrain the geometry of linear uni-variate regression trained with gradient descent through the lens of principal component analysis (PCA). We provide an analytical expression of the PCA matrix of the training trajectories and provides bounds on the decay of eigenvalues for each term in the expression. We then identify the three key factors determining the dimensionality of the training manifold: the training time, data sloppiness, and initialization variance relative to the truth. We study the scenarios under which the training manifold of linear networks is low-dimensional with a phase diagram in Figure 5.3. In Section 5.3 we extend our analysis to the training of kernel machines, stochastic gradient descent in linear models, and deep linear models.

5.2. Training manifold for linear regression

Consider a dataset $\{(x_i', y_i)\}_{i=1}^n$ that consists of inputs $x_i' \in \mathbb{R}^{d-1}$ and outputs $y_i \in \mathbb{R}$. We will focus on the case with a scalar output in this paper for clarity of exposition, all results hold for multi-dimensional output. Let $x_i \equiv [x_i', 1]$ denote the input with a constant appended to it and consider a linear model $y_i = w^\top x_i$ with

$w \in \mathbb{R}^d$ trained to minimize

$$C(w) = \frac{1}{2n} \sum_{i=1}^{n} r_i(w)^2. \tag{5.2.1}$$

Here the residuals $r_i(w) = \hat{y}_i - y_i$ for $i \in \{1, \ldots, n\}$ denote the difference between the predictions and targets $y_i \in \mathbb{R}$. We will assume that the targets corresponds to unknown true weights $w^* \in \mathbb{R}^d$. Discrete-time gradient descent to minimize this objective updates with a step-size (learning rate) α can be written as $w_{t+1} = w_t - \alpha\, \partial_w C(w_t)$, starting from some $w_0 \in \mathbb{R}^d$ for all $t = 1, 2, \ldots$. We denote the n-dimensional vector of residuals computed at weights w_t by $r_t \equiv [r_1(w_t), \ldots, r_n(w_t)]^\top \in \mathbb{R}^n$. This vector of residuals r_t evolves as the weights are updated by gradient descent:

$$r_{t+1} = (I - \alpha K) r_t = (I - \alpha K)^{t+1} r_0, \tag{5.2.2}$$

where $I \in \mathbb{R}^{n \times n}$ is the identity and $K \in \mathbb{R}^{n \times n}$ is a symmetric positive semi-definite matrix with entries $K_{ij} = x_i^\top x_j / n$ for $i, j \in \{1, \ldots, n\}$. The matrix $K = XX^\top/n$ where the i^{th} row of $X \in \mathbb{R}^{n \times d}$ contains the input datum x_i. It is the input-correlation matrix, or the neural tangent kernel (NTK (Jacot et al., 2018a)) for a linear model. Let the i^{th} largest eigenvalue of K be denoted by $\lambda_i^K \geq 0$. The shorthand

$$K_d \equiv I - \alpha K,$$

will be useful to simplify our expressions. It is the first-order approximation of $\exp(-\alpha K)$. We make three assumptions in our analysis.

(i) **Input data and ground-truth targets:** We will assume that the input data are sloppy, i.e.,

$$\lambda_i^K = \exp(-(i-1)c),$$

for some $c > 0$ for all $i = 1, \ldots, n$, with $c \gg 1/n$. We will also assume that the unknown true weights w^* are a random variable with zero mean and isotropic variance $(\sigma_*)^2 I \in \mathbb{R}^d$. This will indirectly be an assumption on the norm of the targets y_i.

(ii) **Model:** The model is over-parameterized, i.e., the number of samples is smaller than the dimensionality

of the input data $n < d$. Weights w are under-determined if $n < d$ and therefore there is an infinite set of solutions that achieve $C(w) = 0$. Therefore we effectively assume that $\text{rank}(K) = n$ for a positive definite K. This assumption is not unduly restrictive. Our analysis can also be conducted in the image space of K if K is rank deficient.

(iii) **Training method:** For a large part of the analysis we will be interested in training methods that resemble gradient descent, or its variants. We will assume that the step-size $\alpha < 1/\lambda_1^K$. This is a standard assumption used in the analysis of gradient descent algorithms (Bottou, 2012). It ensures that $\|I - \alpha K\|_2 < 1$ and therefore $\|r_t\|_2 \to 0$ monotonically as $t \to \infty$.

5.2.1. Principal component analysis of the training manifold

Consider N randomly initialized models with weights $\{w_0^{(i)}\}_{i=1}^N$ sampled from a probability distribution supported on $\mathbb{R}^d$ with zero mean and an isotropic variance, i.e., $\mathbb{E}[w_0] = 0$ and $\mathbb{E}[w_0 w_0^\top] = \sigma_w^2 I$. The initial residual vectors satisfy

$$\mathbb{E}\left[r_0^{(i)}\right] = -y, \quad \mathbb{E}\left[r_0^{(i)} {r_0^{(j)}}^\top\right] = \delta_{ij}\sigma_w^2 K + yy^\top,$$

for all $i, j \in \{1, \ldots, N\}$ where $y = [y_1, \ldots, y_n]^\top \in \mathbb{R}^n$ is the vector of targets and $\delta_{ij} = \mathbf{1}_{\{i=j\}}$ is the delta function. The prediction space $\mathbb{R}^n$ has Euclidean geometry. We can therefore capture the geometry of the training manifold, which is a subset of $\mathbb{R}^n$, using principal component analysis (PCA) of points on trajectories $\{r_t^{(i)}\}_{t=0,\,i=1}^{T-1,\,N}$. The covariance matrix corresponding to PCA is

$$\begin{aligned} P(N,T) &= \frac{1}{NT}\sum_{i,t}(r_t^{(i)} - \bar{r})(r_t^{(i)} - \bar{r})^\top, \\ &= \frac{1}{N}\left(\sum_{i=1}^{N}\frac{1}{T}\sum_{t=0}^{T-1} r_t^{(i)} {r_t^{(i)}}^\top\right) - \bar{r}\,\bar{r}^\top, \end{aligned} \tag{5.2.3}$$

where the mean residual is $\bar{r} \equiv \bar{r}(N,T) = \frac{1}{NT}\sum_{i=1}^N \sum_{t=0}^{T-1} r_t^{(i)}$. The mean residual evolves according to the equation $\bar{r}(N,T) = K_T \bar{r}(N,0)$ with

$$K_T \equiv \frac{1}{T}\sum_{t=0}^{T-1} K_d^t = \frac{1}{\alpha T} K^{-1}(I - K_d^T). \tag{5.2.4}$$

As the number of random initializations N goes to infinity, we can separate the PCA matrix P into two components,

$$\begin{aligned} P(T) &= \lim_{N\to\infty} P(N,T) \\ &= \underbrace{\frac{1}{T}\sum_{t=0}^{T-1} K_d^t(\sigma_w^2 K + yy^\top)K_d^t}_{P_1(T)\equiv P_1^{\sigma_w}(T)+P_1^y(T)} - \underbrace{K_T yy^\top K_T}_{P_2(T)}. \end{aligned} \tag{5.2.5}$$

For any two positive semi-definite matrices $A, B \in \mathbb{R}^{n\times n}$, Weyl's inequality says that

$$\lambda_i^A + \lambda_n^B \le \lambda_i^{A+B} \le \lambda_i^A + \lambda_1^B, \tag{5.2.6}$$

for eigenvalues λ_i ordered in decreasing order of their magnitude. The second term $P_2(T)$ in Equation (5.2.5) is an outer product of y with itself and has a single non-vanishing eigenvalue $\lambda^{P_2} = \|K_T y\|^2$. Therefore, eigenvalues of $P(T)$ are sandwiched by the eigenvalues of $P_1(T)$:

$$\max\left(\lambda_{i+1}^{P_1}, \lambda_i^{P_1} - \lambda^{P_2}\right) \le \lambda_i^P \le \lambda_i^{P_1}. \tag{5.2.7}$$

From Equation (5.2.4),

$$\|K_T\|_2 \le \frac{1-(1-\alpha\lambda_1^K)^T}{\alpha T \lambda_n^K} \le \frac{1}{\alpha T \lambda_n^K}. \tag{5.2.8}$$

This suggests that $\lambda^{P_2} = \mathcal{O}(1/T)$, and the approximation becomes tighter for long training times. To characterize the geometry of the training manifold, it suffices to focus on the eigenspectrum of $P_1(T)$.

5.2.2. The geometry of the training manifold

The goal of this section will be to characterize the geometry of the training manifold, i.e., eigenvalues of the PCA matrix $P(T)$ in Equation (5.2.5). We will analyze its three components:

$$P(T) = -P_2(T) + \underbrace{P_1^{\sigma_w}(T) + P_1^y(T)}_{=P_1(T)}.$$

The first term $P_2(T)$ has unit rank, the second term $P_1^{\sigma_w}(T)$ depends on the variance of initial weights σ_w^2, and the third term $P_1^y(T)$ again depends on the targets y, but it is not unit rank. To simplify the notation, we

will denote their eigenvalues by λ^{P_2}, $\lambda_i^{\sigma_w}$ and λ_i^y, respectively, for all $i = 1, \ldots, n$. We can write $P_1(T)$ as follows:

$$TP_1(T) = \underbrace{\sigma_w^2 \sum_{t=0}^{T-1} K_d^t K K_d^t}_{TP_1^{\sigma_w}(T)} + \underbrace{\sum_{t=0}^{T-1} K_d^t y y^\top K_d^t}_{TP_1^y(T)}, \tag{5.2.9}$$

Contributions to the eigenspectrum coming from weight initialization. Note that $K_d = I - \alpha K$ commutes with K. We can therefore write $P_1^{\sigma_w}(T)$ to be the geometric sum

$$P_1^{\sigma_w}(T) = \sigma_w^2 \sum_{t=0}^{T-1} K_d^{2t} K = \frac{\sigma_w^2}{\alpha}(I + K_d)^{-1}(I - K_d^{2T}),$$

and calculate its eigenvalues explicitly as

$$\lambda_i^{\sigma_w} = \frac{\sigma_w^2}{\alpha}\left(\frac{1 - (1 - \alpha\lambda_i^K)^{2T}}{2 - \alpha\lambda_i^K}\right).$$

Due to the inequality $1 - (1 - x)^a \leq \min\{1, ax\}$, which holds for $|x| < 1$ and $a \geq 1$, under assumption (iii), the eigenvalue $\lambda_i^{\sigma_w}$ is bounded above by

$$\lambda_i^{\sigma_w} \leq \frac{\sigma_w^2}{\alpha}\left(\frac{\min\{1, 2T\alpha\lambda_i^K\}}{2 - \alpha\lambda_i^K}\right). \tag{5.2.10}$$

Figure 5.1 uses Equation (5.2.10) to explain how eigenvalues $\lambda_i^{\sigma_w}$ of different indices i depend upon the training duration T and the sloppy eigenspectrum of the input correlation matrix K. From Equation (5.2.9), it is immediate that if the initialization variance σ_w^2 is small (relative to σ_*^2 which controls $\|y\|$), the contribution of $P_1^{\sigma_w}(T)$ to the dimensionality of the hyper-ribbon is small for all times T. For all times T, the head of the eigenspectrum $P_1^{\sigma_w}$ decays rather quickly. For small times T, eigenvalues in the tail of $P_1^{\sigma_w}$ are quite small. The implication of this is that, everything else (i.e., P_1^y) being the same, models trained for long times have a higher-dimensional hyper-ribbon due to the variations caused by the initialization of weights. For short times, the hyper-ribbon has a smaller dimensionality.

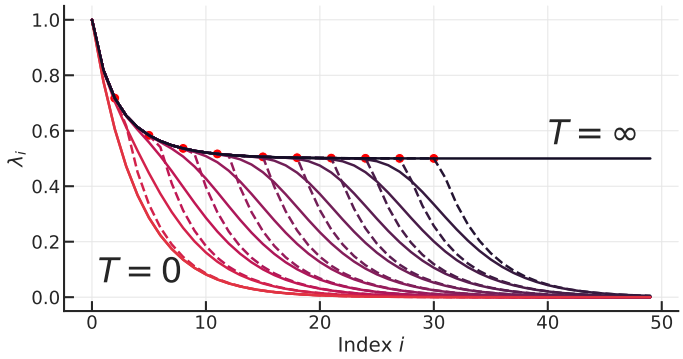


Figure 5.1: **Contributions to the eigenspectrum of the PCA matrix coming from weight initialization computed using the bound in Equation (5.2.10) (dotted) and numerical computation of the corresponding term in Equation (5.2.9) (bold) with $\sigma_w^2 = 1$, $\alpha = 1$ and $c = 0.5$.** For very large training times $T \gg \lambda_1^K/\lambda_n^K$, the eigenvalue $\lambda_i^{\sigma_w}$ corresponds to the minimum in the numerator being 1 for any i. This means that $\lambda_i^{\sigma_w}$ decays to $\sim \sigma_w^2/2\alpha$ as the index i increases, at a rate determined by the decay of λ_i^K. From Equation (5.2.10), for $T \ll \lambda_1^K/\lambda_n^K$ and $i < \ln(2T\alpha)/c$, the minimum in the numerator is 1. This gives $\lambda_i^{\sigma_w} \leq \sigma_w^2/(2\alpha - \alpha^2\lambda_i^K)$. This decays at the same rate as the previous case. For larger values of i, the minimum comes from the other term in the numerator, and therefore $\lambda_i^{\sigma_w}$ decays to much smaller values $\sim \frac{1}{(2/\lambda_n^K - \alpha)}$. This is the lower envelope of the curves above. The limit $T \to 0$ corresponds to the eigenvalues of $\sigma_w^2 K$ and therefore reflects the sloppy decay in the input correlation matrix.

Contributions to the eigenspectrum coming from the targets. The second term corresponding to $P_1^y(T)$ in Equation (5.2.9) resembles the so-called reachability Gramian in systems theory. It is well-known that $P = \lim_{T\to\infty} P_1^y(T)$ is the unique solution to the discrete algebraic Lyapunov equation (DATTA, 2004)

$$K_d P K_d^\top - P + yy^\top = 0.$$

In systems theory, this concept is used for model reduction, i.e., to identify a low-dimensional dynamical system that captures time-varying data from a larger system. The rate of decay of the singular values of the reachability Gramian characterizes the quality of this approximation. Singular values of the Gramian decay quickly (Penzl, 2000; Antoulas, 2005; Townsend and Wilber, 2018; Beckermann and Townsend, 2016) when K_d has some nice properties (e.g., normal, well-conditioned), and $yy^\top$ is approximately low rank. This is precisely the setting of our paper. To study $P_1^y(T)$, which is a finite sum, we write it as the difference between

two Gramians:

$$TP_1^y(T) = \sum_{t=0}^{\infty} K_d^t yy^\top K_d^t - \sum_{t=0}^{\infty} K_d^t \left(K_d^T yy^\top K_d^T\right) K_d^t$$
$$= \sum_{t=0}^{\infty} K_d^t \left(yy^\top - K_d^T yy^\top K_d^T\right) K_d^t.$$

The following lemma shows that the eigenspectrum of $P_1^y(T)$ decays quickly.

Lemma 5.2.1. *We have*

$$\frac{\lambda_{1+2i}^y}{\lambda_1^y} \leq \frac{4\rho^{-2i}}{(1+\rho^{-4i})^2} < 4\rho^{-2i},$$

where

$$\rho = \exp\left(\frac{\pi^2}{2\ln\left(8\lambda_1^K/\lambda_n^K - 4\right)}\right).$$

For the input correlation matrix $K = US^2U^\top$ with $\lambda_i^K = e^{-c(i-1)}$ like we have assumed, Lemma 5.2.1 is non-vacuous when $cn > \ln 4$ which ensures that $\rho > 1$. We should also note that the largest eigenvalue of $\tilde{K}_d$ is bounded away from zero, i.e., $a > 0$. The following lemma now gives a lower bound on the eigenvalue λ_1^y.

Lemma 5.2.2. *If we denote* $\tilde{\lambda}_i = \sum_{t=0}^{T-1}(1-\alpha\lambda_i^K)^{2t}$*, then*

$$\|y\|^2 \leq \lambda_1^y \leq \tilde{\lambda}_n \|y\|^2,$$

where the norm of targets concentrates around the value

$$\|y\|^2 \approx \sigma_*^2 \sum_{i=1}^{n} \lambda_i^K. \tag{5.2.11}$$

Notice that $\tilde{\lambda}_i = \lambda_i^{\sigma_w}/(\sigma_w^2\lambda_i^K)$.

Lemma 5.2.1 suggests that the decay of the eigenspectrum of P_1^y is $\exp(-i\pi^2/nc)$. In contrast to the decay of $P_1^{\sigma_w}$, this rate is independent of the training time T. The rate of decay of P_1^y is comparable to that of $P_1^{\sigma_w}$ for small indices i. Both are proportional to $\exp(-ia)$ for a constant a that depends on c and n. Suppose now that σ_w^2 is small relative to σ_*^2. Since $\lambda_1^y \geq \|y\|^2 = \sigma_*^2 \text{tr}(K)$, the head of the eigenspectrum of P_1^y can be much taller than that of $P_1^{\sigma_w}$. Even if the decay of the two is similar. In other words, the head of

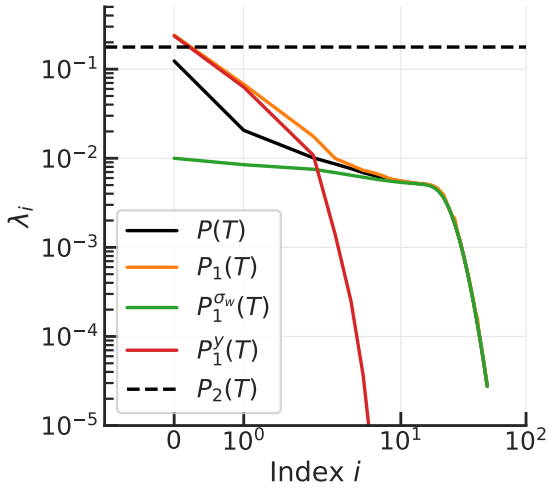


Figure 5.2: **The tail of the eigenspectrum of $P_1(T)$ is well-approximated by the contribution coming from the weight initializations $P_1^{\sigma_w}(T)$, while the head is well approximated by the contribution coming from the targets $P_1^y(T)$.** These eigenvalues were computed for $d = 100$ dimensional data with slope $c = 0.2$ for the eigenvalues of the input correlation matrix, after fitting $n = 50$ samples for $T = 50$ iterations with initialization variance $\sigma_w^2 = 0.1$ and variance of the ground-truth weights being $\sigma_*^2 = 2$.

eigenspectrum of $P_1(T) = P_1^{\sigma_w}(T) + P_1^y(T)$ is determined by P_1^y and the tail is determined by $P_1^{\sigma_w}$.

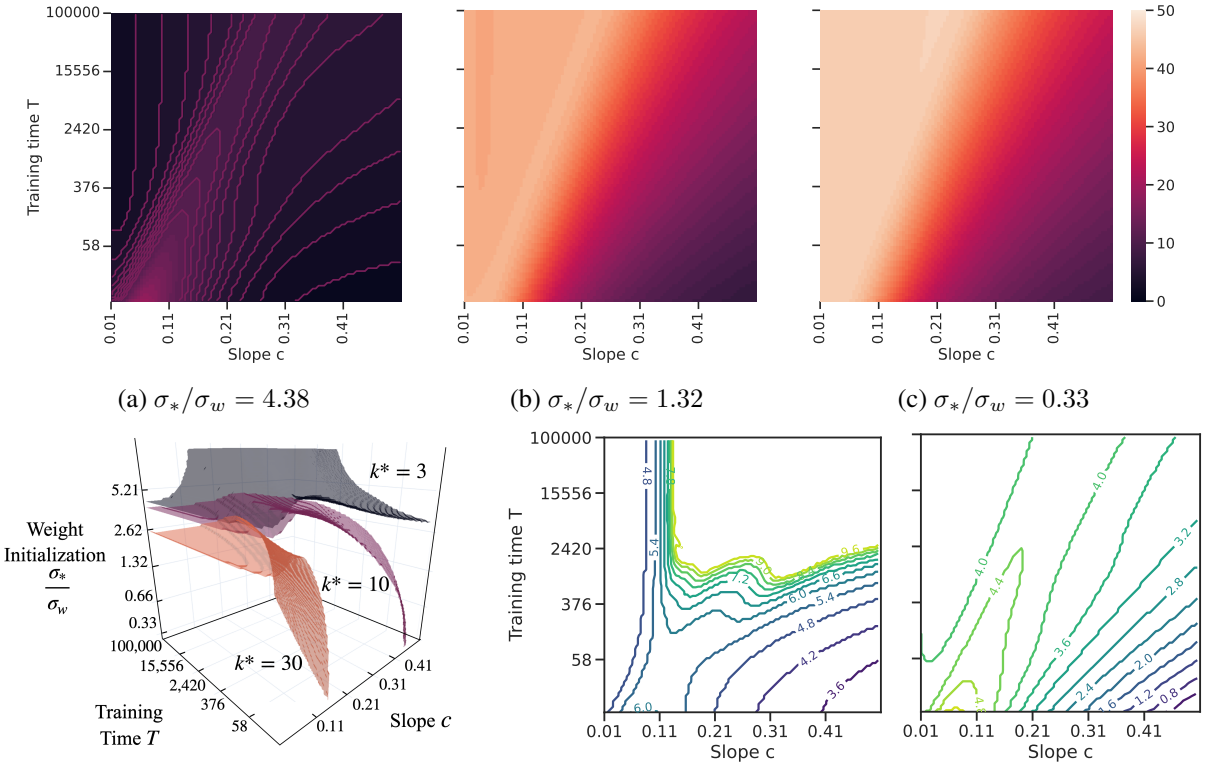
It might seem counter-intuitive that sloppier data, i.e., large c, leads to a slower decay in the eigenspectrum of P_1^y. But notice in Figure 5.1 that the threshold upon the index i after which the eigenspectrum of $P_1^{\sigma_w}(T)$ decreases quickly is $i^* < \ln(2T\alpha)/c$. This threshold scales as $1/c$. Therefore, if one trains for small times T, the eigenspectrum of the sum $P_1(T) = P_1^{\sigma_w}(T) + P_1^y(T)$ still decays after i^*, essentially dominated by $P_1^{\sigma_w}$. In other words, for the same T, sloppier the data, smaller the threshold i^* after which the eigenspectrum of $P_1(T)$ decays. We should note that although Lemma 5.2.1 does show that the eigenspectrum of P_1^y decays, it is a somewhat loose upper bound. In our experiments, the decay of the eigenspectrum is typically about twice as fast.

Combining the two parts to obtain the eigenspectrum of $\mathbf{P_1(T)}$. The following lemma combines the technical development in the previous two subsections.

Lemma 5.2.3. *The eigenvalues of $TP_1(T)$ in Equation (5.2.9) satisfy*

$$\frac{\lambda_i^{P_1}}{\lambda_1^{P_1}} \leq \min\left\{1, 4\rho^{-(i-1)} + \frac{\sigma_w^2}{\alpha \|y\|^2}\right\} \qquad \text{if } i \leq 2k^*,$$
$$\leq 4\rho^{-(k^*-1)} + \frac{\sigma_w^2}{\alpha \|y\|^2} \min\{1, 2\alpha T \lambda_{i-k^*+1}^K\} \qquad \text{else.}$$

for all $i = 1, \ldots, n$, where $k^* = \min\left\{\ln\left(\frac{2T\alpha}{2c}\right), \frac{n}{2}\right\}$.



(a) $\sigma_*/\sigma_w = 4.38$ (b) $\sigma_*/\sigma_w = 1.32$ (c) $\sigma_*/\sigma_w = 0.33$

(d) Transitions between phases of the hyper-ribbon (e) $k^* = 3$ dimensions (f) $k^* = 10$ dimensions

Figure 5.3: **A phase diagram for linear regression that describes the number of dimensions in the hyper-ribbon, i.e., the number of dimensions required to capture 95% of the variance of the points on the training manifold.** This is studied with respect to three parameters: (i) the training time T, (ii) slope c, and (iii) the relative magnitude of weight initialization σ_*/σ_w. **(a-c)** show a heat-map of the dimensionality of the hyper-ribbon for different training times T and slopes c for three different regimes of weight initialization. **(d)** is a three dimensional plot that depicts the boundaries of the different phases, defined by the dimensionality of the hyper-ribbon (3 dimensions in black, 10 in pink and 30 in orange). **(e-f)** show contours for different values of σ_*/σ_w for three and ten- dimensional hyper-ribbons, respectively. See the narrative for an elaborate discussion.

Figure 5.3 summarizes the development of this section using a phase diagram that describes the geometry of the hyper-ribbon in terms of the relevant parameters, the training time T, the slope c and the relative magnitude of the weight initialization σ_*/σ_w. Consider Figure 5.3 (a). For small initialization variance $\sigma_*/\sigma_w \gg 1$, the hyper-ribbon is very low-dimensional for most training times T and slope c. The eigenspectrum is dominated by P_1^y in this case and its fast decay allows for lower-dimensional hyper-ribbons. There appears to be a straight line ($\log T \propto c$) along which the dimensionality is larger, due to relative magnitudes of $P_1^{\sigma_w}$ and P_1^y in Equation (5.2.9). The latter results in a higher-dimensionality for small c and small T, while the former is the cause of higher-dimensionality at larger c and large T. If the initialization variance is small, short training times do not fit the data well. For large c, this causes the hyper-ribbon to be low-dimensional (roughly, because the condition number of optimization is large and different models end up being rather similar). For small c, this is evident as a higher-dimensional hyper-ribbon (roughly, because models are initialized in different subspaces of the data). For longer training times T, different models fit the data very well when c is small (again, because of the condition number). This is evident as a low-dimensional hyper-ribbon above the straight line. The majority of experiments in Mao et al. (2024) lie in this regime.

Next consider Figure 5.3 (b-c). As the initialization variance increases, the apparent straight line $\log T \propto c$ that distinguishes low-dimensional hyper-ribbons from higher-dimensional ones, is still present. The upper-left region is increasingly higher-dimensional. For small slope c the hyper-ribbon is high-dimensional for all training times T. Because, models are initialized in very different subspaces of the data, and this is true for all three plots, except that it becomes more apparent as σ_*/σ_w decreases. For large c, for small times, the hyper-ribbon may be low-dimensional but we need much longer times to fit this data well. (Mao et al., 2024, Fig. 10, S.10, S.16) showed that when neural networks are initialized very far away from the true data distribution, the hyper-ribbon is not low-dimensional. The dimensionality increases when data is not sloppy. These experiments of theirs lie in regimes Figure 5.3 (b-c).

Next consider Figure 5.3 (d) A three dimensional plot that depicts the boundaries of the different phases, defined by the dimensionality of the hyper-ribbon (3 dimensions in black, 10 in pink and 30 in orange). Some broad trends are apparent in the 3D plot, e.g., (i) large σ_*/σ_w leads to a low-dimensional hyper-ribbon, (ii) the geometry of the hyper-ribbon is very sensitive to other parameters when the slope c is small. As one goes

from small T, large slope c and large σ_*/σ_w, to large times, small slope and small σ_*/σ_w, the dimensionality of the hyper-ribbon increases. The other panels in this figure are obtained by projecting this phase diagram upon different axes.

Figure 5.3 (e-f) show contours for different values of σ_*/σ_w for three and ten- dimensional hyper-ribbons, respectively. Focus on the contour marked 4.8, the two left and right wings of this contour together lead to a slice of Figure 5.3 (a) at a fixed dimension of three. Figure 5.3 (e) indicates that there is a contiguous region in $(T, c, \sigma_*/\sigma_w)$-space with $\sigma_*/\sigma_w \gtrapprox 9$ where the hyper-ribbon has fewer than three dimensions. In Figure 5.3 (f) such contiguous regions occur at small values of σ_*/σ_w.

Altogether, Figure 5.3 (d-f) shed light on thumb-rules for identifying the complexity of models that would be required to fit data in these different regimes. Given a dataset (a proxy for its complexity would be c), a training recipe (a proxy of which would be σ_*/σ_w) and training budget (a proxy of which would be T), the boundary of the phase diagram indicates the smallest model that one needs to achieve a good fit. For example, if our regime lies below the orange surface, we need to fit a model with a larger number of parameters.

5.3. Variants of the linear model

We next extend our analysis to a few more general settings.

Stochastic Gradient Descent (SGD). Let us now consider stochastic gradient descent for the linear predictive model. In this case, the weights $w \in \mathbb{R}^d$ are updated, not using the gradient on the entire objective as before $w_{t+1} = w_t - \alpha\partial_w C(w_t)$, but instead as

$$w_{t+1} = w_t - \frac{\alpha}{2b}\partial_w \left\{ \sum_{i=1}^{b} r^2_{\omega_i}(w_t) \right\}.$$

where the random variable ω_i is uniformly distributed on $\{1, \ldots, n\}$ and the batch-size is b. We can model SGD as gradient descent perturbed by state-dependent Langevin noise

$$w_{t+1} = w_t - \alpha\partial_w C(w_t) + (\alpha/\sqrt{b})\xi_t, \tag{5.3.1}$$

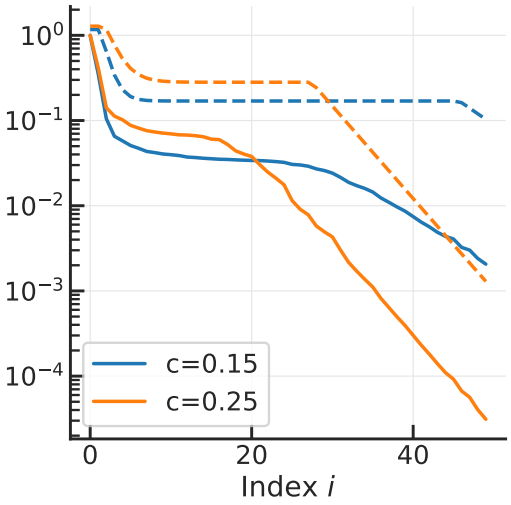


Figure 5.4: **Comparison of the bound in Lemma 5.2.3 with eigenvalues of $P_1(T)$ computed directly from Equation (5.2.9) for different values of sloppy decay rate c.** The proof of Lemma 5.2.3 works by computing the ideal point to apply Weyl's inequality. This enables us to separately calculate the decay in the head of the eigenspectrum and the tail, for both small and large training times T in spite of the fact that different parts of $P_1(T)$ in Equation (5.2.9) dominate in different regimes.

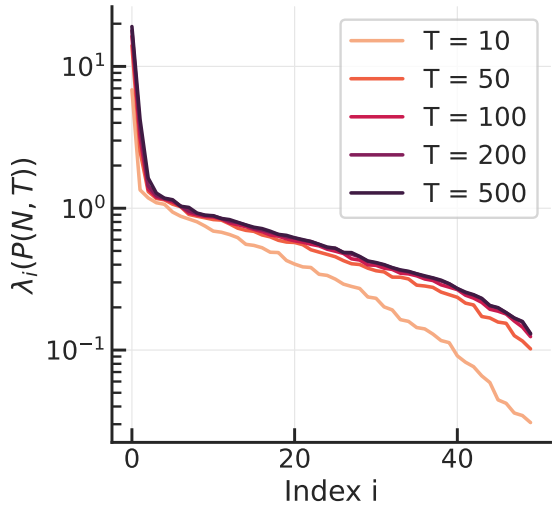


Figure 5.5: **The eigenspectrum of $P(N,T)$ for different training times T from numerical experiments on linear models** with $d = 100$ dimensional data with $n = 50$ training samples, slope $c = 0.1$, initialization variance $\sigma_w^2 = 1$ and learning rate $\alpha = 1/\lambda_1^K$. As training time T increases, the eigenvalues in the tail increase in magnitude, this is because $P_1^{\sigma_w}(T)$. See Figure 5.1. This also causes an increase in the largest eigenvalue in the head, due to the diminishing magnitude of $P_2(T)$ in Equation (5.2.8).

where $\xi_t \sim N(0, D)$ where $D = X^\top X/n - \bar{x}^\top \bar{x}$ with $\bar{x} = n^{-1}\sum_{i=1}^n x_i$ is the covariance matrix of the inputs (Chaudhari and Soatto, 2018). Under this dynamics of the weights, the residuals evolve as

$$r_{t+1} = (I - \alpha K) r_t + (\alpha/\sqrt{b}) X \xi_t.$$

The PCA matrix $P(N, T)$ as the number of random initializations N goes to infinity becomes

$$P(T) = \frac{1}{T}\sum_{t=0}^{T-1} \mathbb{E}\left[r_t r_t^\top\right] - K_T y y^\top K_T,$$

where K_T is the same matrix as in Equation (5.2.5). Notice here the randomness of r_t comes from both random initialization and noise from Langevin dynamics, and we are taking expectation with both sources of randomness, assuming they are independent. We now have

$$\begin{aligned}
&\mathbb{E}\left[r_{t+1} r_{t+1}^\top\right] \\
&= K_d \mathbb{E}\left[r_t r_t^\top\right] K_d^\top + 2(\alpha/\sqrt{b}) K_d \mathbb{E}\left[r_t\right] \mathbb{E}\left[\xi_t\right] X^\top \\
&\quad + (\alpha^2/b) X \mathbb{E}\left[\xi_t \xi_t^\top\right] X^\top \\
&= K_d \mathbb{E}\left[r_t r_t^\top\right] K_d^\top + (\alpha^2/b) X X^\top X X^\top \\
&= K_d \mathbb{E}\left[r_t r_t^\top\right] K_d + (\alpha^2/b) K^2,
\end{aligned}$$

where we recall that $K_d = I - \alpha K$.

If we define $P_\xi = (\alpha^2/b)\sum_{t=0}^\infty K_d^t K^2 K_d^t$, then P_ξ solves the Lyapunov equation $K_d P_\xi K_d^\top - P_\xi + (\alpha^2/b) K^2 = 0$, and it can be checked by induction that

$$\begin{aligned}
&\sum_{t=0}^{T-1} \mathbb{E}\left[r_t r_t^\top\right] \\
&= T P_\xi + \sum_{t=0}^{T-1} K_d^t \left(\mathbb{E}\left[r_0 r_0^\top\right] - P_\xi\right) K_d^t \\
&= \sum_{t=0}^{T-1} K_d^t \left(\sigma_w^2 K + y y^\top\right) K_d^t + T P_\xi - \sum_{t=0}^{T-1} K_d^t P_\xi K_d^t,
\end{aligned}$$

which has two additional terms compared to Equation (5.2.5). Notice that K_d commutes with K^2 so P_ξ has the explicit expression

$$P_\xi = (\alpha^2/b)(I - K_d^2)^{-1}K^2 = (\alpha/b)(2I - \alpha K)^{-1}K.$$

Since P_ξ commutes with K_d, we can simplify

$$\begin{aligned}\sum_{t=0}^{T-1} K_d^t P_\xi K_d^t &= (\alpha/b)\sum_{t=0}^{T-1} K_d^{2t}(2I - \alpha K)^{-1}K \\ &= (1/b)(2I - \alpha K)^{-2}(I - K_d^{2T})\end{aligned}$$

For $\alpha\lambda_i^K < 1$ the i^{th} eigenvalue of the above matrix is $\simeq T\lambda_i^K/(2b)$. In Lemma 5.2.3, the upper bound on the eigenvalues of $P_1(T)$ for $i \leq 2k^*$ is perturbed by largest eigenvalue of the two additional terms. This is at most the largest eigenvalue of P_ξ, which is simply $\lambda_1(P_\xi) = (\alpha/b)(\lambda_1^K/(2 - \alpha\lambda_1^K)) \leq \alpha/b$ for our setting where $\lambda_1^K = 1$ and $\alpha < 1$ (which ensures the convergence of the infinite sum in P_ξ). In other words in Lemma 5.2.3, we will have,

$$\frac{\lambda_i^{P_1}}{\lambda_1^{P_1}} \leq \min\left\{1, 4\rho^{-(i-1)} + \frac{1}{\|y\|^2}\left(\frac{\sigma_w^2}{\alpha} + \frac{\alpha}{b}\right)\right\}$$

for $i \leq 2k^*$. This indicates an interesting relationship between the variance of weight initialization σ_w^2, the learning rate α and the batch-size b. Suppose we wish to keep the volume of the ensemble of trajectories, as measured by the volume of the hyper-ribbon in residual space, the same. This is a reasonable because it indicates the propensity to identify good fits within the ensemble. Suppose we are in the regime where $\alpha \propto b$, which is very common while training neural networks. If we pick a batch-size that is twice as large, the first term σ_w^2/α shrinks by a factor of two. To compensate—to keep the decay rate and effectively the dimensionality of the hyper-ribbon the same—we must pick a weight initialization variance that is twice as large.

Kernel machines. Consider predictions given by $\hat{y}_i = f(x_i)$ where the predictor f is not linear, but it belongs to a class of functions $\mathcal{F}$ with some regularity properties. A classical example of such a class of

functions is called reproducing kernel Hilbert space (RKHS) which is a Hilbert space with the "reproducing kernel property". This property states that for any input datum x, there exists a function $\varphi_x \in \mathcal{F}$ such that the evaluation of $f \in \mathcal{F}$ at the input x can be written as an inner product $f(x) = \langle f, \varphi_x \rangle_{\mathcal{F}} = \int f(x')\varphi_x(x')\mathrm{d}x'$. The function

$$k(x, x') = \langle \varphi_x, \varphi_{x'} \rangle_{\mathcal{F}} \geq 0$$

is called the reproducing kernel. Gradient descent in RKHS (Yao et al., 2007) to minimize the objective in Equation (5.2.1) corresponds to updates of the form

$$\forall x: \; f_{t+1}(x) = f_t(x) - \frac{\alpha}{n} \sum_{i=1}^{n} \left(f_t(x_i) - y_i\right) k(x_i, x).$$

Notice that the residuals $r_i = f_t(x_i) - y_i$ for all $i = 1, \ldots, n$ follow linear dynamics, same as those in Equation (5.2.2), namely, $r_{t+1} = (I - \alpha K) r_t$ except that we now have $K_{ij} = k(x_i, x_j)/n$ for any i, j. This matrix is called the Gram matrix and it is positive semi-definite by Mercer's theorem (Schölkopf and Smola, 2002). In other words, all our development in the previous section holds for trajectories of kernel machines initialized from different initial conditions.

If the input correlation matrix is sloppy, then the Gram matrix is sloppy. This is easiest to see if x comes from a high dimensional distribution, i.e., $d \to \infty$ as $n \to \infty$ with a fixed d/n. Results in random matrix theory (Karoui, 2010b) state that the Gram matrix K can be well-approximated by the sample covariance matrix in such cases. And therefore, kernel machines are rather similar to linear models. If inputs x are drawn from a distribution with density $p(x)$ supported on $\mathbb{R}^d$, as $n \to \infty$, the k-th eigenvalue of the Gram matrix K converges to the k-the eigenvalue of the integral operator $T[\varphi](x) = \int k(x, x')\varphi(x')p(x')\mathrm{d}x'$ (Koltchinskii and Giné, 2000). The eigenspectrum of such integral operators has been studied, e.g., if $p(x)$ is Gaussian and $k(x, x') \propto \exp(-\|x - x'\|^2 / d)$ is highly local, then eigenvalues of the Gram matrix K decay exponentially (Zhu et al., 1998). For translation invariant kernels, the decay is related to how quickly $p(x)$ goes to zero with increasing $\|x\|$ and to the Fourier transform of the kernel k (Widom, 1963). In other words, the Gram matrix K can also be sloppy even if the input correlation matrix is not.

Let us now consider the space of training trajectories corresponding to different kernel models. Let $\{K^{(m)}\}_{m=1}^{M}$ be M different kernel machines with trajectories in the residual space defined by $r_{t+1}^{(i,m)} = (I - \alpha K^{(m)})^{t+1} r_0^{(i,m)}$ for the i^{th} initialization. This suggests that we should study the covariance matrix

$$P(N, M, T) = \frac{1}{NMT} \sum_{i,m,t} \left(r_t^{(i,m)} - \bar{r} \right) \left(r_t^{(i,m)} - \bar{r} \right)^{\top},$$

where the mean $\bar{r} = \frac{1}{MNT} \sum_{i,m,t} r_t^{(i,m)}$. Taking $N \to \infty$ as before, we get

$$P(M, T) \equiv \frac{1}{M} \sum_{m=1}^{M} P_1^{(m)}(T) - P_2(M, T),$$

where $P_2(M, T) = K_{T,M} y y^{\top} K_{T,M}$ with

$$K_{T,M} = \frac{1}{MT} \sum_{m=1}^{M} \sum_{t=0}^{T-1} {K_d^{(m)}}^t.$$

The above equation is the analog of Equation (5.2.5). Similar to our previous analysis, $P_2(M, T)$ has a single eigenvalue that vanishes for large T. We can thus obtain a result that is rather similar to the one in Lemma 5.2.3 with appropriate substitutions $\rho \to \rho^{(m)}$ and $\lambda_i^K \to \lambda_i^{K^{(m)}}$, assuming different kernels have different slopes $c^{(m)}$ but the same largest eigenvalue, i.e., the same learning rate α. Now by Weyl's inequality,

$$\lambda_{M(i-1)+1} \left(\sum_{m=1}^{M} P_1^{(m)}(T) \right) \leq \sum_{m=1}^{M} \lambda_i \left(P_1^{(m)}(T) \right)$$

and the fact that $\lambda_1(\sum_m P_1^m) \geq M \left\| y \right\|^2$, the right-hand side of the per-kernel version of Lemma 5.2.3 can be summed up over m. Altogether, the eigenvalues of $TP_1(M, T) \equiv TP_1$ satisfy

$$\frac{\lambda_i^{P_1}}{\lambda_1^{P_1}} \leq \min \left\{ 1, \frac{4}{M} \sum_m \rho^{(m)^{-\lfloor \frac{i-1}{M} \rfloor}} + \frac{\sigma_w^2}{\alpha \left\| y \right\|^2} \right\} \tag{5.3.2}$$

if $i \leq 2k^*$ and otherwise we have

$$\frac{\lambda_i^{P_1}}{\lambda_1^{P_1}} \leq \frac{1}{M} \left\{ \sum_m 4\rho^{(m)^{-(k^*-1)}} + \frac{\sigma_w^2 \min\{1, 2\alpha T \lambda^{K^{(m)}}_{\lfloor \frac{i-1}{M} \rfloor - k^* + 2}\}}{\alpha \left\| y \right\|^2} \right\}$$

for all $i = 1, \ldots, n$, where $k^* = \min_m \ln(2T\alpha)/(2c^{(m)})$. This is a loose upper bound, because it uses the floor $\lfloor \frac{i-1}{M} \rfloor$ in the exponent of $\rho^{(m)}$. But due to the averaging over m, if different kernels have similar condition numbers, i.e., similar $\rho^{(m)}$, then the decay rate of eigenvalues $\lambda_i^{P_1}$ is shallower by a factor of M. But they do decay, and we should expect the hyper-ribbon to be low-dimensional. For the second expression when $i > 2k^*$, the second term (coming from $P_1^{\sigma_w}$) dominates the eigenspectrum. It has become worse due to the presence of $\lambda^{K^{(m)}}_{\lfloor \frac{i-1}{M} \rfloor - k^* + 2}$. But we can see that it still indicates a decay in the eigenspectrum at large i.

This narrative gives intuition into the experiments of Mao et al. (2024) which are discussed in Section 5.1, where the training manifold for different neural architectures was computed to find that the explained variance of the top few dimensions was quite high. Our discussion suggests that this can arise only if the "effective kernels" of all those networks have Gram matrices that decay quickly. If some of them do not decay quickly, then the explained variance would be rather low.

Weight Decay The least squares objective Equation (5.2.1) is often "regularized" to be $C(w) + \frac{\lambda}{2} \|w\|_2^2$ to obtain a fit where the weights have a small ℓ_2-norm. This is important in situations when the model is over-parameterized, i.e., $n < d$, where there may be multiple solutions to the non-regularized problem. The residual dynamics can be seen to be $r_{t+1} = (I - \alpha K_\lambda) r_t$ where $K_\lambda = XX^\top + \lambda I_{n \times n}$. All our calculations in Section 5.2 hold with K replaced by K_λ, i.e., with each $\lambda_i^{K_\lambda} = \lambda_i^K + \lambda$.

Deep linear networks One can also study a two-layer linear model $\hat{y}_n = u^\top v x_i$ where $v \in \mathbb{R}^{p \times d}$ and $u \in \mathbb{R}^p$ and minimize the squared residuals $C(u, v) = 1/2 \sum_i r_i(u, v)^2$ where $r_i = u^\top v x_i - y_i$. This is no longer a convex optimization, but it is, of course, simply a reparameterization of the original problem with $uv = w^\top$. Gradient descent updates for both variables can be written as

$$u_{t+1} = u_t - \alpha \sum_i r_i v_t x_i,$$
$$v_{t+1} = v_t - \alpha \sum_i r_i u x_i^\top.$$

Up to first order in the step-size α, the residuals evolve as

$$r_{t+1} = r_t - \alpha K(u, v)\, r_t. \tag{5.3.3}$$

where $K(u,v)_{ij} = x_i^\top(v^\top v + u^\top u)x_j$. This dynamics is no longer linear because the kernel K depends upon the parameters u, v. Unlike linear models or kernel machines, the dynamics of deep linear networks follows a nonlinear trajectory in the prediction space, even if the deep linear network can be re-parameterized using $w^\top \equiv u^\top v$ and seen to be just linear (Tarmoun et al., 2021). If we use the so-called balanced initialization (Fukumizu, 1998; Saad and Solla, 1995; Saxe et al., 2014) $v_0 v_0^\top = u_0 u_0^\top$ then

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(v_t v_t^\top\right) = \frac{\mathrm{d}}{\mathrm{d}t}\left(u_t u_t^\top\right),$$

for all times t. In other words, the kernel $K(u,v) \equiv K$ is invariant for balanced initializations. This holds approximately even for discrete-time updates to the weights. Similar considerations apply for multi-layer linear networks where successive layers of the network are initialized to be balanced (Arora et al., 2018).

In some learning scenarios, gradient descent on a nonlinear objective function gives an almost linear trajectory in the function space (Chizat et al., 2019b). Some works describe a learning scenarios that can be decomposed into two parts, a first part that learns features and a second part that is equivalent to regression using the learnt features (Atanasov et al., 2022; Jacot et al., 2022).

5.4. Proofs

Proof of Lemma 5.2.1. We will denote $TP_1^y(T)$ as P in this proof for clearer exposition. We know that P solves the discrete algebraic Lyapunov equation

$$K_d P K_d^\top - P + \left(yy^\top - K_d^T yy^\top K_d^T\right) = 0.$$

See (Antoulas, 2005, Chapter 4.3.3), such a P also solves the continuous Lyapunov equation $\tilde{K_d}P + P\tilde{K_d}^\top + BB^\top = 0$ with

$$\tilde{K_d} = (K_d + I)^{-1}(K_d - I),$$
$$BB^\top = 2(K_d + I)^{-1}(yy^\top - K_d^T yy^\top K_d^T)(K_d + I)^{-1}.$$

Notice that $\tilde{K_d}$ is normal and has a bounded spectrum:

$$\sigma(K_d) \subseteq [-b, -a],$$

with $0 < a < b < \infty$. Notice that $BB^\top$ has a rank of at most 2. From (Beckermann and Townsend, 2016, Theorem 2.1 and Corollary 3.2) we can conclude that

$$\lambda^P_{1+2i} \leq 4\rho^{-2i}\lambda^P_1, \text{ with } \rho = \exp\left(\frac{\pi^2}{2\log(4b/a)}\right).$$

Notice that $\lambda_i^{\tilde{K}_d} = -\frac{\alpha\lambda_i^K}{2-\alpha\lambda_i^K}$, so with our assumption (iii) which states that $\alpha < 1/\lambda_1^K$, we have

$$\frac{b}{a} = \frac{\lambda_1^K\,(2-\alpha\lambda_n^K)}{\lambda_n^K\,(2-\alpha\lambda_1^K)} \leq \frac{2\lambda_1^K}{\lambda_n^K} - 1.$$

Proof of Lemma 5.2.2. The lower bound is obtained by seeing that

$$\lambda_1^y \geq \max_t \lambda_1(K_d^t y y^\top K_d^t) \geq \|y\|^2,$$

and the upper bound is given by

$$\lambda_1^y \leq \sum_t \lambda_1(K_d^t y y^\top K_d^t) = \tilde{\lambda}_n \|y\|^2.$$

Since the true weights w^* are drawn from an isotropic normal distribution with covariance $\sigma_*^2 I$, the norm of the targets concentrates around the trace of the data correlation matrix $\sigma_*^2 \sum_{i=1}^n \lambda_i^K$. Indeed, $\|y\|^2 = (V^\top w^*)^\top S^2 (V^\top w^*)$, where V and S, are given by the singular value decomposition (SVD) of the data matrix $X = USV^\top$. Since entries of $V^\top w^*$ are independent random variables, $\|y\|^2$ is concentrated around the above value by the Hanson-Wright inequality

$$\mathbb{P}\left(\left|\|y\|^2 - \sigma_*^2 \text{tr}(K)\right| > t\right) \leq 2\exp\left[-a\min\left(\frac{t^2}{\sigma_*^4 \text{tr}(K)}, \frac{t}{\sigma_*^2}\right)\right]$$

where $a > 0$ is a constant independent of t, K and σ_*.

Proof of Lemma 5.2.3. To keep the notation clear, in the proof we will refer to $P_1^{\sigma_w}(T)$ and $P_1^y(T)$ as P^{σ_w} and P^y respectively.

Small times. Notice that if $T < \frac{\lambda_1^K}{2\lambda_n^K}$, then $2T\alpha < \alpha\frac{\lambda_1^K}{\lambda_n^K} < e^{c(n-1)} < e^{cn}$. In this case we have $k^* = \frac{\ln(2T\alpha)}{2c} < \frac{n}{2}$, so we can obtain the following upper bound on $\lambda_i^{P_1}$ by Weyl's inequality:

$$\lambda_i^{P_1} \leq \begin{cases} \lambda_i^y + \lambda_1^{\sigma_w} & \text{for } i \leq 2k^* \\ \lambda_{k^*}^y + \lambda_{i-k^*+1}^{\sigma_w} & \text{for } i > 2k^*. \end{cases}$$

For $i \leq 2k^*$, we have

$$\frac{\lambda_i^{P_1}}{\lambda_1^{P_1}} \leq \frac{\lambda_i^{P_1}}{\lambda_1^y} \leq 4\rho^{-(i-1)} + \frac{\lambda_1^{\sigma_w}}{\lambda_1^y} \leq 4\rho^{-(i-1)} + \frac{\sigma_w^2 \min\{1, 2T\alpha\lambda_1^K\}}{\alpha \|y\|^2}$$

where we have used $\lambda_1^{P_1} \geq \lambda_1^y$ since P^{σ_w} is positive definite. Note that under assumptions (i-ii), $\lambda_1^K = 1$ and $\alpha < 1$, therefore we do not need the minimum in the second term. For $i > 2k^*$, we have

$$\begin{aligned} \frac{\lambda_i^{P_1}}{\lambda_1^{P_1}} &\leq \frac{\lambda_{i-k^*+1}^{\sigma_w}}{\lambda_1^y} + 4\rho^{-(k^*-1)} \\ &\leq \frac{\sigma_w^2 \min\{1, 2\alpha T\lambda_{i-k^*+1}^K\}}{\alpha \|y\|^2} + 4\rho^{-(k^*-1)} \end{aligned}$$

where we have used Equation (5.2.10) and assumption (ii).

Large times. For $T \geq \frac{\lambda_1^K}{2\lambda_n^K}$, we can choose the splitting point for Weyl's inequality to be $n/2$. We now have

$$\frac{\lambda_i^{P_1}}{\lambda_1^{P_1}} \leq \frac{\lambda_{i-n/2+1}^{\sigma_w}}{\lambda_1^y} + \frac{\lambda_{n/2}^y}{\lambda_1^y} \leq \frac{\lambda_{i-n/2+1}^{\sigma_w}}{\lambda_1^y} + 4\rho^{-(n/2-1)}.$$

If $T \geq \frac{\lambda_1^K}{2\alpha\lambda_n^K}$, then $2T\alpha\lambda_i^K > 1$ for all $i \geq 1$, so by Equation (5.2.10)

$$\lambda_{i-n/2+1}^{\sigma_w} \leq \frac{\sigma_w^2}{2\alpha - \alpha^2\lambda_{i-n/2+1}^K}$$

and

$$\frac{\lambda_{i-n/2+1}^{\sigma_w}}{\lambda_1^y} \leq \frac{\sigma_w^2}{\alpha \|y\|^2} \frac{1}{(2 - \alpha\lambda_{i-n/2+1}^K)} \leq \frac{\sigma_w^2}{\alpha \|y\|^2}.$$

For $\frac{\lambda_1(K)}{2\lambda_n(K)} \leq T < \frac{\lambda_1(K)}{2\alpha\lambda_n(K)}$, we may still use the split at $k^* = n/4$ in the above calculation, and take the minimum of the above two bounds for $i > n/2$.

CHAPTER 6

CONCLUSION AND FUTURE DIRECTIONS

In conclusion, this thesis investigates the model manifold explored by standard training of neural network training and observes that it lives in a surprisingly low-dimensional subspace, resembling the hyper-ribbon structure observed in many physics and biology models Transtrum (2011). In contrast to classical approaches in Information Geometry Amari (2016), which focus on infinite-dimensional statistical manifolds, our approach adopts a finite-dimensional view of models, allowing a more detailed understanding of the global geometry of the model manifold.

While we provide some preliminary hypotheses and analysis to understand this low-dimensionality, many open questions remain. For instance, does the low dimensionality provide practical implications that can be used to improve actual learning processes?

One possible direction is to understand whether the low dimensionality can serve as a compressed representation of a network. To facilitate that, we would need to define a notion of "basis" in the space of model prediction. We propose here two possible approaches to look for such a basis, in the function space and the data space respectively.

First, we explore the idea of using trained models as candidates for the bases. Concretely, suppose we have a given set of data points $\{x_i\}_{i=1}^N$, and $\{f_j\}_{j=1}^M$ a set of functions we are interested in (f_j is a probability distribution supported on $\{x_i\}$, for classification problem it can be thought of as a vector in a probability simplex). We would like to find a set of functions $\{g_i\}_{i=1}^K$ such that there exists $\{w_j\}_{j=1}^M$, $w_j \in \Delta^K$ (the K-dimensional simplex) with $d_B(f_{j_1}, f_{j_2}) \approx d_B(\sum_i w_{j_1 i} g_i, \sum_j w_{j_2 i} g_i)$. In other words, we want to find a set of model bases such that existing models fall into the convex hull of existing models. Notice this problem always has the trivial solution that takes $\{g_i\} \equiv \{f_j\}$, and the hope is that there exists a solution set with cardinality $K \ll M$ due to the low-dimensionality observed in our experiments.

In Mao et al. (2024), we attempted to understand the InPCA coordinates by constructing a 3-dimensional subspace using the following directions: the straight line that joins ignorance and truth, tangents to a training

trajectory at ignorance and at truth. This new embedding preserves pairwise Bhattacharyya distances between the original models to a similar degree as that of the original InPCA embedding. We speculated that these vectors represent learning easy samples, hard samples, and the general direction from ignorance to truth. It would be interesting to know what the basis models we found using this method represent. The initial tangents are related to the NTK. Finding this basis might provide insights into what is missing from the NTK characterization of the dynamics.

Another possible approach follows the observation that the native coordinates of our manifold correspond to model outputs on a fixed set of data points on which we evaluate our models. The fact that the model manifold has a hyper-ribbon structure suggests that the coordinates are entangled, i.e., model outputs on different samples are highly correlated. We would like to construct a basis for the model manifold by finding another set of "datapoints" so that the model outputs on those samples have a more direct correspondence to the coordinates of the model manifold. Notice that for a set of K samples, the model manifold, according to our definition, is naturally K-dimensional (for simplicity, consider scalar output functions for now). Intuitively, we want to find $\{x_i\}_{i=1}^K$ such that the model predictions on those samples given by $\{f_j(x_i)\}$ best preserves the pairwise distances of the original models. We hope this set of samples provides insight into understanding the structure of data space and could possibly speed up training on new samples.

BIBLIOGRAPHY

Emmanuel Abbe, Enric Boix Adsera, and Theodor Misiakiewicz. The merged-staircase property: A necessary and nearly sufficient condition for SGD learning of sparse functions on two-layer neural networks. In *Proceedings of Thirty Fifth Conference on Learning Theory*, pages 4782–4887. PMLR, June 2022.

Emmanuel Abbe, Enric Boix-Adsera, and Theodor Misiakiewicz. Sgd learning on neural networks: leap complexity and saddle-to-saddle dynamics. (arXiv:2302.11055), August 2023. URL http://arxiv.org/abs/2302.11055. arXiv:2302.11055 [cs, stat].

Alessandro Achille and Stefano Soatto. Emergence of invariance and disentanglement in deep representations. *The Journal of Machine Learning Research*, 19(1):1947–1980, 2018.

Alessandro Achille, Matteo Rovere, and Stefano Soatto. Critical learning periods in deep networks. In *International Conference on Learning Representations*, 2018.

Alessandro Achille, Michael Lam, Rahul Tewari, Avinash Ravichandran, Subhransu Maji, Charless C Fowlkes, Stefano Soatto, and Pietro Perona. Task2vec: Task embedding for meta-learning. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 6430–6439, 2019a.

Alessandro Achille, Glen Mbeng, and Stefano Soatto. Dynamics and Reachability of Learning Tasks. *arXiv:1810.02440 [cs, stat]*, May 2019b.

Alessandro Achille, Giovanni Paolini, and Stefano Soatto. Where is the information in a deep neural network? *arXiv preprint arXiv:1905.12213*, 2019c.

Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. A Convergence Theory for Deep Learning via Over-Parameterization. *arXiv:1811.03962 [cs, math, stat]*, June 2019.

Shun-ichi Amari. Natural gradient works efficiently in learning. *Neural Computation*, 10(2):251–276, 1998a.

Shun-ichi Amari. Natural Gradient Works Efficiently in Learning. *Neural Computation*, 10(2):251–276, February 1998b. ISSN 0899-7667. doi: 10.1162/089976698300017746.

Shun-ichi Amari. *Information Geometry and Its Applications*, volume 194 of *Applied Mathematical Sciences*. Tokyo, 2016.

Shun-ichi Amari, Hyeyoung Park, and Tomoko Ozeki. Geometrical singularities in the neuromanifold of multilayer perceptrons. *Advances in neural information processing systems*, 1:343–350, 2002.

Amiran Ambroladze, Emilio Parrado-Hernández, and John Shawe-Taylor. Tighter pac-bayes bounds. *Advances in neural information processing systems*, 19:9, 2007.

Joseph Antognini and Jascha Sohl-Dickstein. PCA of high dimensional random walks with comparison to neural network training. *Advances in Neural Information Processing Systems*, 31, 2018.

Athanasios C. Antoulas. *Approximation of Large-Scale Dynamical Systems*. Society for Industrial and Applied Mathematics, January 2005. ISBN 978-0-89871-529-3. doi: 10.1137/1.9780898718713. URL http://epubs.siam.org/doi/book/10.1137/1.9780898718713.

Sanjeev Arora, Simon S Du, Wei Hu, and Zhiyuan Li. On Exact Computation with an Infinitely Wide Neural Net.

Sanjeev Arora, Nadav Cohen, Noah Golowich, and Wei Hu. A convergence analysis of gradient descent for deep linear neural networks. *arXiv preprint arXiv:1810.02281*, 2018.

Sanjeev Arora, Nadav Cohen, Noah Golowich, and Wei Hu. A convergence analysis of gradient descent for deep linear neural networks. (arXiv:1810.02281), October 2019a. doi: 10.48550/arXiv.1810.02281. URL http://arxiv.org/abs/1810.02281. arXiv:1810.02281 [cs, stat].

Sanjeev Arora, Nadav Cohen, Wei Hu, and Yuping Luo. Implicit regularization in deep matrix factorization. (arXiv:1905.13655), October 2019b. doi: 10.48550/arXiv.1905.13655. URL http://arxiv.org/abs/1905.13655. arXiv:1905.13655 [cs, stat].

Sanjeev Arora, Simon Du, Wei Hu, Zhiyuan Li, and Ruosong Wang. Fine-Grained Analysis of Optimization and Generalization for Overparameterized Two-Layer Neural Networks. In *International Conference on Machine Learning*, pages 322–332, May 2019c.

Alexander Atanasov, Blake Bordelon, and Cengiz Pehlevan. Neural Networks as Kernel Learners: The Silent Alignment Effect. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=uz5uw6gM0m.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer Normalization. *arXiv:1607.06450 [cs, stat]*, July 2016.

Philip Bachman, R Devon Hjelm, and William Buchwalter. Learning representations by maximizing mutual information across views. *Advances in neural information processing systems*, 32, 2019.

Vijay Balasubramanian. Statistical Inference, Occam's Razor, and Statistical Mechanics on the Space of Probability Distributions. *Neural Computation*, 9(2):349–368, February 1997. ISSN 0899-7667, 1530-888X.

P. Baldi and K. Hornik. Neural networks and principal component analysis: Learning from examples without local minima. *Neural Networks*, 2:53–58, 1989.

Peter L Bartlett, Dylan J Foster, and Matus J Telgarsky. Spectrally-normalized margin bounds for neural networks. In *Advances in Neural Information Processing Systems*, pages 6240–6249, 2017.

Peter L Bartlett, Philip M Long, Gábor Lugosi, and Alexander Tsigler. Benign overfitting in linear regression. *Proceedings of the National Academy of Sciences*, 117(48):30063–30070, 2020.

Peter L Bartlett, Andrea Montanari, and Alexander Rakhlin. Deep learning: A statistical viewpoint. *arXiv preprint arXiv:2103.09177*, 2021a.

Peter L. Bartlett, Andrea Montanari, and Alexander Rakhlin. Deep learning: A statistical viewpoint. *Acta Numerica*, 30:87–201, 2021b.

Jonathan Baxter. A model of inductive bias learning. *Journal of artificial intelligence research*, 12:149–198, 2000.

Suzanna Becker and Geoffrey E Hinton. Self-organizing neural network that discovers surfaces in random-dot stereograms. *Nature*, 355(6356):161–163, 1992.

Bernhard Beckermann and Alex Townsend. On the singular values of matrices with displacement structure. (arXiv:1609.09494), September 2016. URL http://arxiv.org/abs/1609.09494. arXiv:1609.09494 [math].

Mikhail Belkin. Fit without fear: Remarkable mathematical phenomena of deep learning through the prism of interpolation. *arXiv preprint arXiv:2105.14368*, 2021.

Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32): 15849–15854, 2019.

Shai Ben-David, John Blitzer, Koby Crammer, Alex Kulesza, Fernando Pereira, and Jennifer Wortman Vaughan. A theory of learning from different domains. *Machine learning*, 79(1):151–175, 2010.

Samy Bengio, Yoshua Bengio, Jocelyn Cloutier, and Jan Gecsei. On the optimization of a synaptic learning rule. In *Preprints Conf. Optimality in Artificial and Biological Neural Networks*, pages 6–8, 1992.

David Berthelot, Nicholas Carlini, Ian Goodfellow, Nicolas Papernot, Avital Oliver, and Colin A Raffel. MixMatch: A holistic approach to semi-supervised learning. *Advances in Neural Information Processing Systems*, 32, 2019.

Christopher M Bishop et al. *Neural Networks for Pattern Recognition*. 1995.

Mike Bostock. Imagenet hierarchy. https://observablehq.com/@mbostock/imagenet-hierarchy, 2018.

L. Bottou. Stochastic gradient descent tricks. In *Neural Networks: Tricks of the Trade*, pages 421–436. 2012.

Richard P. Brent. An algorithm with guaranteed convergence for finding a zero of a function. *The computer journal*, 14(4):422–425, 1971.

Kevin S Brown, Colin C Hill, Guillermo A Calero, Christopher R Myers, Kelvin H Lee, James P Sethna, and Richard A Cerione. The statistical mechanics of complex signaling networks: Nerve growth factor signaling. *Physical biology*, 1(3):184, 2004.

Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

Rich Caruana. Multitask learning. *Machine learning*, 28(1):41–75, 1997.

Pratik Chaudhari and Stefano Soatto. Stochastic gradient descent performs variational inference, converges to limit cycles for deep networks. In *Proc. of International Conference of Learning and Representations (ICLR)*, 2018.

Pratik Chaudhari, Anna Choromanska, Stefano Soatto, Yann LeCun, Carlo Baldassi, Christian Borgs, Jennifer Chayes, Levent Sagun, and Riccardo Zecchina. Entropy-SGD: Biasing gradient descent into wide valleys. In *Proc. of International Conference of Learning and Representations (ICLR)*, 2017.

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International Conference on Machine Learning*, pages 1597–1607, 2020a.

Ting Chen, Simon Kornblith, Kevin Swersky, Mohammad Norouzi, and Geoffrey E Hinton. Big self-supervised models are strong semi-supervised learners. *Advances in Neural Information Processing Systems*, 33:22243–22255, 2020b.

Lenaic Chizat and Francis Bach. Implicit bias of gradient descent for wide two-layer neural networks trained with the logistic loss. *arXiv preprint arXiv:2002.04486*, 2020.

Lénaïc Chizat, Edouard Oyallon, and Francis Bach. On lazy training in differentiable programming. *Advances in Neural Information Processing Systems*, 32:2937–2947, 2019a.

Lénaïc Chizat, Edouard Oyallon, and Francis Bach. On Lazy Training in Differentiable Programming. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d\textquotesingle Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 2937–2947. Curran Associates, Inc., 2019b.

Michael AA Cox and Trevor F Cox. Multidimensional scaling. In *Handbook of Data Visualization*, pages 315–347. 2008.

Yin Cui, Yang Song, Chen Sun, Andrew Howard, and Serge Belongie. Large scale fine-grained categorization and domain-specific transfer learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4109–4118, 2018.

Alex Damian, Eshaan Nichani, and Jason D. Lee. Self-stabilization: The implicit bias of gradient descent at the edge of stability. (arXiv:2209.15594), April 2023. doi: 10.48550/arXiv.2209.15594. URL http://arxiv.org/abs/2209.15594. arXiv:2209.15594 [cs, math, stat].

Alexandru Damian, Jason Lee, and Mahdi Soltanolkotabi. Neural Networks can Learn Representations with Gradient Descent. In *Proceedings of Thirty Fifth Conference on Learning Theory*, pages 5413–5452.

PMLR, June 2022.

Stéphane d'Ascoli, Marylou Gabrié, Levent Sagun, and Giulio Biroli. On the interplay between data structure and loss function in classification problems. *Advances in Neural Information Processing Systems*, 34: 8506–8517, 2021.

BISWA NATH DATTA. Chapter 8 - numerical solutions and conditioning of lyapunov and sylvester equations. In BISWA NATH DATTA, editor, *Numerical Methods for Linear Control Systems*, pages 245–303. Academic Press, San Diego, 2004.

Vin De Silva and Joshua B Tenenbaum. Sparse multidimensional scaling using landmark points. Technical report, Technical Report, Stanford University, 2004.

Ludovic Delchambre. Weighted principal component analysis: a weighted covariance eigendecomposition approach. *Monthly Notices of the Royal Astronomical Society*, 446(4):3545–3555, 2015.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

Benoit Dherin, Michael Munn, and David G. T. Barrett. The Geometric Occam's Razor Implicit in Deep Learning. *arXiv:2111.15090 [cs, stat]*, November 2021.

Guneet S Dhillon, Pratik Chaudhari, Avinash Ravichandran, and Stefano Soatto. A baseline for few-shot image classification. In *Proc. of International Conference of Learning and Representations (ICLR)*, 2020.

Carl Doersch and Andrew Zisserman. Multi-task self-supervised visual learning. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2051–2060, 2017.

Alexey Dosovitskiy, Jost Tobias Springenberg, Martin Riedmiller, and Thomas Brox. Discriminative unsupervised feature learning with convolutional neural networks. *Advances in neural information processing systems*, 27, 2014.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.

Simon S Du and Xiyu Zhai. Gradient Descent Provably Optimizes Over-Parameterized Neural Networks. page 19, 2019.

Simon S Du, Chi Jin, Jason D Lee, Michael I Jordan, Aarti Singh, and Barnabas Poczos. Gradient Descent Can Take Exponential Time to Escape Saddle Points. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

Gintare Karolina Dziugaite. *Revisiting Generalization for Deep Learning: PAC-Bayes, Flat Minima, and Generative Models*. PhD thesis, University of Cambridge, 2020.

Gintare Karolina Dziugaite and Daniel M. Roy. Computing Nonvacuous Generalization Bounds for Deep (Stochastic) Neural Networks with Many More Parameters than Training Data. In *Proc. of the Conference on Uncertainty in Artificial Intelligence (UAI)*, 2017.

Gintare Karolina Dziugaite and Daniel M Roy. Data-dependent PAC-Bayes priors via differential privacy. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 8440–8450, 2018.

Rasool Fakoor, Pratik Chaudhari, Stefano Soatto, and Alexander J Smola. Meta-Q-Learning. In *Proc. of International Conference of Learning and Representations (ICLR)*, 2020.

Yu Feng and Yuhai Tu. The inverse variance–flatness relation in stochastic gradient descent is critical for finding flat minima. *Proceedings of the National Academy of Sciences*, 118(9):e2015617118, Mar 2021. doi: 10.1073/pnas.2015617118.

David J Field. What is the goal of sensory coding? *Neural computation*, 6(4):559–601, 1994.

Stanislav Fort and Surya Ganguli. Emergent properties of the local geometry of neural loss landscapes. *arXiv:1910.05929 [cs, stat]*, October 2019.

Jonathan Frankle and Michael Carbin. The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks. *arXiv:1803.03635 [cs]*, March 2019.

C Daniel Freeman and Joan Bruna. Topology and geometry of half-rectified network optimization. In *ICLR*, 2017.

Kenji Fukumizu. Effect of batch learning in multilayer neural networks. *Gen*, 1(04):1E–03, 1998.

K Ruben Gabriel and Shmuel Zamir. Lower rank approximation of matrices by least squares with any choice of weights. *Technometrics*, 21(4):489–498, 1979.

Yansong Gao and Pratik Chaudhari. A free-energy principle for representation learning. In *Proc. of International Conference of Machine Learning (ICML)*, 2020.

Yansong Gao and Pratik Chaudhari. An information-geometric distance on the space of tasks. In *Proc. of International Conference of Machine Learning (ICML)*, 2021.

Timur Garipov, Pavel Izmailov, Dmitrii Podoprikhin, Dmitry Vetrov, and Andrew Gordon Wilson. Loss

surfaces, mode connectivity, and fast ensembling of dnns. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 8803–8812, 2018.

Rong Ge, Furong Huang, Chi Jin, and Yang Yuan. Escaping from saddle points: Online stochastic gradient for tensor decomposition. In *Journal of Machine Learning Research*, volume 40. Microtome Publishing, 2015.

Rong Ge, Jason D. Lee, and Tengyu Ma. Learning One-hidden-layer Neural Networks with Landscape Design. In *International Conference on Learning Representations*, February 2018.

Behrooz Ghorbani, Shankar Krishnan, and Ying Xiao. An investigation into neural net optimization via hessian eigenvalue density. In *Proceedings of the 36th International Conference on Machine Learning*, 2019a.

Behrooz Ghorbani, Song Mei, Theodor Misiakiewicz, and Andrea Montanari. Limitations of Lazy Training of Two-layers Neural Network. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019b.

Behrooz Ghorbani, Song Mei, Theodor Misiakiewicz, and Andrea Montanari. When Do Neural Networks Outperform Kernel Methods? *arXiv:2006.13409 [cs, math, stat]*, June 2020.

Spyros Gidaris and Nikos Komodakis. Dynamic few-shot visual learning without forgetting. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4367–4375, 2018.

Sebastian Goldt, Marc Mézard, Florent Krzakala, and Lenka Zdeborová. Modeling the influence of data structure on learning in neural networks: The hidden manifold model. *Physical Review X*, 10(4):041044, 2020.

Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch SGD: Training ImageNet in 1 hour. *arXiv:1706.02677*, 2017.

Michael Greenacre. Weighted metric multidimensional scaling. In *New Developments in Classification and Data Analysis: Proceedings of the Meeting of the Classification and Data Analysis Group (CLADAG) of the Italian Statistical Society, University of Bologna, September 22–24, 2003*, pages 141–149. Springer, 2005.

Suriya Gunasekar, Jason Lee, Daniel Soudry, and Nathan Srebro. Characterizing implicit bias in terms of optimization geometry. In *International Conference on Machine Learning*, page 1832–1841, July 2018. URL http://proceedings.mlr.press/v80/gunasekar18a.html.

Guy Gur-Ari, Daniel A. Roberts, and Ethan Dyer. Gradient descent happens in a tiny subspace. (arXiv:1812.04754), December 2018. URL http://arxiv.org/abs/1812.04754. arXiv:1812.04754 [cs, stat].

Guy Gur-Ari, Daniel A. Roberts, and Ethan Dyer. Gradient Descent Happens in a Tiny Subspace. *arXiv:1812.04754 [cs, stat]*, December 2018.

Guy Gur-Ari, Daniel A Roberts, and Ethan Dyer. Gradient descent happens in a tiny subspace. *arXiv preprint arXiv:1812.04754*, 2018.

Ryan N Gutenkunst, Joshua J Waterfall, Fergal P Casey, Kevin S Brown, Christopher R Myers, and James P Sethna. Universally sloppy parameter sensitivities in systems biology models. *PLoS Computational Biology*, 3(10):e189, 2007.

Michael Gutmann and Aapo Hyvärinen. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 297–304. JMLR Workshop and Conference Proceedings, 2010.

Guy Hacohen, Leshem Choshen, and Daphna Weinshall. Let's agree to agree: Neural networks share classification order on real datasets. In *International Conference on Machine Learning*, pages 3950–3960, 2020.

Benjamin D. Haeffele and Rene Vidal. Global Optimality in Tensor Factorization, Deep Learning, and Beyond. *arXiv:1506.07540 [cs, stat]*, June 2015.

Steve Hanneke and Samory Kpotufe. A no-free-lunch theorem for multitask learning. *arXiv preprint arXiv:2006.15785*, 2020.

Moritz Hardt and Tengyu Ma. Identity matters in deep learning. (arXiv:1611.04231), July 2018. doi: 10.48550/arXiv.1611.04231. URL http://arxiv.org/abs/1611.04231. arXiv:1611.04231 [cs, stat].

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

Tong He, Zhi Zhang, Hang Zhang, Zhongyue Zhang, Junyuan Xie, and Mu Li. Bag of tricks for image classification with convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 558–567, 2019.

Byeongho Heo, Sanghyuk Chun, Seong Joon Oh, Dongyoon Han, Sangdoo Yun, Gyuwan Kim, Youngjung Uh, and Jung-Woo Ha. Adamp: Slowing down the slowdown for momentum optimizers on scale-invariant weights. In *International Conference on Learning Representations (ICLR)*, 2021.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *NIPS Deep Learning and Representation Learning Workshop*, 2015.

Sepp Hochreiter and Jürgen Schmidhuber. Flat minima. *Neural Computation*, 9(1):1–42, 1997.

Stefan Horoi, Jessie Huang, Guy Wolf, and Smita Krishnaswamy. Visualizing high-dimensional trajectories on the loss-landscape of ANNs. *arXiv preprint arXiv:2102.00485*, 2021.

Junlin Hu, Jiwen Lu, and Yap-Peng Tan. Deep transfer metric learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 325–333, 2015.

W Ronny Huang, Zeyad Emam, Micah Goldblum, Liam Fowl, Justin K Terry, Furong Huang, and Tom Goldstein. Understanding generalization through visualizations. 2020.

Andrew Ilyas, Sung Min Park, Logan Engstrom, Guillaume Leclerc, and Aleksander Madry. Datamodels: Predicting predictions from training data. *arXiv preprint arXiv:2202.00622*, 2022.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

Sosuke Ito and Andreas Dechant. Stochastic time evolution, information geometry, and the Cramér-Rao bound. *Physical Review X*, 10(2):021056, 2020.

Tommi Jaakkola and David Haussler. Exploiting generative models in discriminative classifiers. In *Advances in Neural Information Processing Systems*, pages 487–493, 1999.

Arthur Jacot, Franck Gabriel, and Clement Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in Neural Information Processing Systems 31*, pages 8571–8580. 2018a.

Arthur Jacot, Franck Gabriel, and Clement Hongler. Neural Tangent Kernel: Convergence and Generalization in Neural Networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 8571–8580. Curran Associates, Inc., 2018b.

Arthur Jacot, François Ged, Berfin Şimşek, Clément Hongler, and Franck Gabriel. Saddle-to-Saddle Dynamics in Deep Linear Networks: Small Initialization Training, Symmetry, and Sparsity, January 2022.

Ziwei Ji and Matus Telgarsky. Directional convergence and alignment in deep learning. *arXiv:2006.06657 [cs, math, stat]*, June 2020.

Chi Jin, Rong Ge, Praneeth Netrapalli, Sham M. Kakade, and Michael I. Jordan. How to Escape Saddle Points Efficiently. *arXiv:1703.00887 [cs, math, stat]*, March 2017.

Dimitris Kalimeris, Gal Kaplun, Preetum Nakkiran, Benjamin Edelman, Tristan Yang, Boaz Barak, and Haofeng Zhang. Sgd on neural networks learns functions of increasing complexity. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, page 3496–3506. Curran Associates, Inc., 2019. URL http://papers.nips.cc/paper/8609-sgd-on-neural-networks-learns-functions-of-increasing-complexity.pdf. tex.ids: nakkiranSGDNeuralNetworks2019 arXiv: 1905.11604.

Gal Kaplun, Nikhil Ghosh, Saurabh Garg, Boaz Barak, and Preetum Nakkiran. Deconstructing distributions: A pointwise framework of learning. *arXiv preprint arXiv:2202.09931*, 2022.

Ryo Karakida, Shotaro Akaho, and Shun-ichi Amari. Universal Statistics of Fisher Information in Deep Neural Networks: Mean Field Approach. *arXiv:1806.01316 [cond-mat, stat]*, October 2019.

Noureddine El Karoui. The spectrum of kernel random matrices. *Annals of Statistics*, 38:1–50, 2010a.

Noureddine El Karoui. The spectrum of kernel random matrices. *The Annals of Statistics*, 38(1), Feb 2010b. ISSN 0090-5364. doi: 10.1214/08-AOS648. URL http://arxiv.org/abs/1001.0492. arXiv: 1001.0492.

K. Kawaguchi. Deep learning without poor local minima. In *NIPS*, 2016.

Kenji Kawaguchi and Leslie Pack Kaelbling. Elimination of all bad local minima in deep learning. *arXiv:1901.00279 [cs, math, stat]*, January 2020. URL http://arxiv.org/abs/1901.00279. arXiv: 1901.00279.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference for Learning Representations*, 2015.

Alexander Kolesnikov, Xiaohua Zhai, and Lucas Beyer. Revisiting self-supervised visual representation learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1920–1929, 2019.

Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Joan Puigcerver, Jessica Yung, Sylvain Gelly, and Neil Houlsby. Big Transfer (BiT): General Visual Representation Learning. *arXiv:1912.11370 [cs]*, May 2020.

Vladimir Koltchinskii and Evarist Giné. Random matrix approximation of spectra of integral operators. *Bernoulli*, 6(1):113–167, 2000.

A. Krizhevsky. *Learning Multiple Layers of Features from Tiny Images*. PhD thesis, Computer Science, University of Toronto, 2009.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.

John Langford and Rich Caruana. (Not) bounding the true error. *Advances in Neural Information Processing Systems*, 2:809–816, 2002.

John Langford and Matthias Seeger. Bounds for averaging classifiers. 2001.

Julian Laub and Klaus-Robert Müller. Feature discovery in non-metric pairwise data. *The Journal of Machine Learning Research*, 5:801–818, 2004.

Jim Lawrence, Javier Bernal, and Christoph Witzgall. A purely algebraic justification of the Kabsch-Umeyama algorithm. *Journal of research of the National Institute of Standards and Technology*, 124:1, 2019.

Guillaume Leclerc, Andrew Ilyas, Logan Engstrom, Sung Min Park, Hadi Salman, and Aleksander Madry.

ffcv. https://github.com/libffcv/ffcv/, 2022. commit xxxxxxx.

Yann LeCun, Bernhard E Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne E Hubbard, and Lawrence D Jackel. Handwritten digit recognition with a back-propagation network. In *Advances in Neural Information Processing Systems*, pages 396–404, 1990.

Hao Li, Zheng Xu, Gavin Taylor, and Tom Goldstein. Visualizing the loss landscape of neural nets. In *ICLR*, 2018.

Hao Li, Pratik Chaudhari, Hao Yang, Michael Lam, Avinash Ravichandran, Rahul Bhotika, and Stefano Soatto. Rethinking the hyper-parameters for fine-tuning. In *Proc. of International Conference of Learning and Representations (ICLR)*, 2020.

Yuanzhi Li and Yingyu Liang. Learning Overparameterized Neural Networks via Stochastic Gradient Descent on Structured Data. *arXiv:1808.01204 [cs, stat]*, August 2019.

Tengyuan Liang and Alexander Rakhlin. Just interpolate: Kernel "ridgeless" regression can generalize. *CoRR*, abs/1808.00387, 2018. URL http://arxiv.org/abs/1808.00387.

Tengyuan Liang and Alexander Rakhlin. Just interpolate: Kernel “ridgeless” regression can generalize. *The Annals of Statistics*, 48(3):1329–1347, 2020.

Hong Liu, Sang Michael Xie, Zhiyuan Li, and Tengyu Ma. Same pre-training loss, better downstream: Implicit bias matters for language models. In *Proceedings of the 40th International Conference on Machine Learning*, page 22188–22214. PMLR, July 2023. URL https://proceedings.mlr.press/v202/liu23ao.html.

Shengchao Liu, Dimitris Papailiopoulos, and Dimitris Achlioptas. Bad global minima exist and sgd can reach them. *arXiv:1906.02613 [cs, stat]*, June 2019. URL http://arxiv.org/abs/1906.02613. arXiv: 1906.02613.

Kaifeng Lyu and Jian Li. Gradient descent maximizes the margin of homogeneous neural networks. September 2019. URL https://openreview.net/forum?id=SJeLIgBKPS.

Sadhika Malladi, Alexander Wettig, Dingli Yu, Danqi Chen, and Sanjeev Arora. A kernel-based view of language model fine-tuning. *arXiv preprint arXiv:2210.05643*, 2022.

Jean M Mandler and Laraine McDonough. Concept formation in infancy. *Cognitive development*, 8(3): 291–318, 1993.

Jialin Mao, Itay Griniasty, Han Kheng Teoh, Rahul Ramesh, Rubing Yang, Mark K. Transtrum, James P. Sethna, and Pratik Chaudhari. The training process of many deep networks explores the same low-dimensional manifold. *Proceedings of the National Academy of Sciences*, 121(12):e2310002121, March 2024. doi: 10.1073/pnas.2310002121.

David Marr. *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*. 2010.

James Martens and Roger Grosse. Optimizing Neural Networks with Kronecker-factored Approximate Curvature. *arXiv:1503.05671 [cs, stat]*, May 2016.

Andreas Maurer. Bounds for linear multi-task learning. *The Journal of Machine Learning Research*, 7: 117–139, 2006.

David A McAllester. PAC-Bayesian model averaging. In *Proceedings of the Twelfth Annual Conference on Computational Learning Theory*, pages 164–170, 1999.

Leland McInnes, John Healy, Nathaniel Saul, and Lukas Großberger. UMAP: Uniform manifold approximation and projection. *Journal of Open Source Software*, 3(29):861, 2018.

Song Mei, Theodor Misiakiewicz, and Andrea Montanari. Mean-field theory of two-layers neural networks: Dimension-free bounds and kernel limit. In *Conference on Learning Theory*, pages 2388–2464, 2019.

Song Mei, Theodor Misiakiewicz, and Andrea Montanari. Generalization error of random feature and kernel methods: Hypercontractivity and kernel matrix concentration. *Applied and Computational Harmonic Analysis*, 59:3–84, July 2022. ISSN 1063-5203. doi: 10.1016/j.acha.2021.12.003.

George A Miller. *WordNet: An Electronic Lexical Database*. 1998.

Behnam Neyshabur, Srinadh Bhojanapalli, David McAllester, and Nathan Srebro. Exploring Generalization in Deep Learning. *arXiv:1706.08947 [cs]*, July 2017.

Eshaan Nichani, Yu Bai, and Jason D. Lee. Identifying good directions to escape the ntk regime and efficiently learn low-degree plus sparse polynomials. October 2022. URL https://openreview.net/forum?id=052QkenIdSI.

Frank Nielsen. Jeffreys centroids: A closed-form expression for positive histograms and a guaranteed tight approximation for frequency histograms. *IEEE Signal Processing Letters*, 20(7):657–660, 2013.

Frank Nielsen and Sylvain Boltz. The burbea-rao and bhattacharyya centroids. *IEEE Transactions on Information Theory*, 57(8):5455–5466, 2011.

Vardan Papyan. The full spectrum of deepnet hessians at scale: Dynamics with sgd training and sample size. *arXiv preprint arXiv:1811.07062*, 2018.

Vardan Papyan. Measurements of three-level hierarchical structure in the outliers in the spectrum of deepnet hessians. *arXiv preprint arXiv:1901.08244*, 2019.

Emilio Parrado-Hernández, Amiran Ambroladze, John Shawe-Taylor, and Shiliang Sun. PAC-Bayes bounds with data dependent priors. *The Journal of Machine Learning Research*, 13(1):3507–3531, 2012.

Jeffrey Pennington and Yasaman Bahri. Geometry of Neural Network Loss Surfaces via Random Matrix Theory. page 9.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

Thilo Penzl. Eigenvalue decay bounds for solutions of Lyapunov equations: The symmetric case. *Systems & Control Letters*, 40(2):139–144, June 2000. ISSN 0167-6911. doi: 10.1016/S0167-6911(00)00010-4.

Hang Qi, Matthew Brown, and David G Lowe. Low-shot learning with imprinted weights. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5822–5830, 2018.

Katherine N Quinn, Colin B Clement, Francesco De Bernardis, Michael D Niemack, and James P Sethna. Visualizing probabilistic models and data with intensive principal component analysis. *Proceedings of the National Academy of Sciences*, 116(28):13762–13767, 2019a.

Katherine N. Quinn, Heather Wilber, Alex Townsend, and James P. Sethna. Chebyshev approximation and the global geometry of model predictions. *Physical Review Letters*, 122(15):158302, April 2019b. doi: 10.1103/PhysRevLett.122.158302.

Katherine N. Quinn, Michael C. Abbott, Mark K. Transtrum, Benjamin B. Machta, and James P. Sethna. Information geometry for multiparameter models: New perspectives on the origin of simplicity. page arXiv:2111.07176, 2021.

Rahul Ramesh and Pratik Chaudhari. Model Zoo: A Growing "Brain" That Learns Continually. In *Proc. of International Conference of Learning and Representations (ICLR)*, 2022.

Maria Refinetti, Sebastian Goldt, Florent Krzakala, and Lenka Zdeborová. Classifying high-dimensional gaussian mixtures: Where kernel methods fail and neural networks succeed. In *International Conference on Machine Learning*, pages 8936–8947, 2021.

Jorma Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978.

D. Saad and S. A. Solla. Exact solution for on-line learning in multilayer neural networks. *Physical Review Letters*, 74(21):4337, 1995.

Itay Safran and Ohad Shamir. Spurious Local Minima are Common in Two-Layer ReLU Neural Networks. In *Proceedings of the 35th International Conference on Machine Learning*, pages 4433–4441. PMLR, July 2018.

Levent Sagun, Leon Bottou, and Yann LeCun. Singularity of the hessian in deep learning. *arXiv:1611:07476*, 2016.

Levent Sagun, Leon Bottou, and Yann LeCun. Eigenvalues of the hessian in deep learning: Singularity and beyond. (arXiv:1611.07476), October 2017. doi: 10.48550/arXiv.1611.07476. URL http://arxiv.org/abs/1611.07476. arXiv:1611.07476 [cs].

Andrew M Saxe, James L McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. page 22.

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.

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, 2019.

Bernhard Schölkopf and Alexander J Smola. *Learning with Kernels*. 2002.

Gideon Schwarz et al. Estimating the dimension of a model. *Annals of statistics*, 6(2):461–464, 1978.

Xing Shi, Inkit Padhi, and Kevin Knight. Does string-based neural MT learn source syntax? In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1526–1534, 2016.

Ravid Shwartz-Ziv and Naftali Tishby. Opening the Black Box of Deep Neural Networks via Information. *arXiv:1703.00810 [cs]*, April 2017.

Eero P Simoncelli and Bruno A Olshausen. Natural image statistics and neural representation. *Annual review of neuroscience*, 24(1):1193–1216, 2001.

Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 4080–4090, 2017a.

Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. In *Advances in Neural Information Processing Systems*, pages 4077–4087, 2017b.

Kihyuk Sohn, David Berthelot, Nicholas Carlini, Zizhao Zhang, Han Zhang, Colin A Raffel, Ekin Dogus Cubuk, Alexey Kurakin, and Chun-Liang Li. FixMatch: Simplifying semi-supervised learning with consistency and confidence. *Advances in Neural Information Processing Systems*, 33, 2020.

Mahdi Soltanolkotabi, Adel Javanmard, and Jason D. Lee. Theoretical Insights Into the Optimization Landscape of Over-Parameterized Shallow Neural Networks. *IEEE Transactions on Information Theory*, 65(2):742–769, February 2019. ISSN 0018-9448, 1557-9654. doi: 10.1109/TIT.2018.2854560.

Ben Sorscher, Surya Ganguli, and Haim Sompolinsky. The geometry of concept learning. *bioRxiv : the preprint server for biology*, 2021.

Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for Simplicity: The All Convolutional Net. *arXiv:1412.6806 [cs]*, April 2015.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *JMLR*, 15(1):1929–1958, 2014.

Ke Sun and Frank Nielsen. A Geometric Modeling of Occam's Razor in Deep Learning, December 2023.

Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on Machine Learning*, page 1139–1147. PMLR, May 2013. URL https://proceedings.mlr.press/v28/sutskever13.html.

Hidenori Tanaka and Daniel Kunin. Noether's learning dynamics: The role of kinetic symmetry breaking in deep learning. *arXiv preprint arXiv:2105.02716*, 2021.

Salma Tarmoun, Guilherme Franca, Benjamin D. Haeffele, and Rene Vidal. Understanding the dynamics of gradient flow in overparameterized linear models. In *Proceedings of the 38th International Conference on Machine Learning*, page 10153–10161. PMLR, July 2021.

Antti Tarvainen and Harri Valpola. Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

Han Kheng Teoh, Katherine N. Quinn, Jaron Kent-Dobias, Colin B. Clement, Qingyang Xu, and James P. Sethna. Visualizing probabilistic models in Minkowski space with intensive symmetrized Kullback-Leibler embedding. *Physical Review Research*, 2(3):033221, August 2020. ISSN 2643-1564.

Sebastian Thrun and Lorien Pratt. *Learning to Learn*. 2012.

Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *International conference on machine learning*, pages 10347–10357. PMLR, 2021.

Alex Townsend and Heather Wilber. On the singular values of matrices with high displacement rank. *Linear Algebra and its Applications*, 548:19–41, July 2018.

M. K. Transtrum, B. B. Machta, and J. P. Sethna. Why are nonlinear fits so challenging? September 2009.

Mark Transtrum. *Information Geometry For Nonlinear Least-Squares Data Fitting And Calculation Of The Superconducting Superheating Field*. PhD thesis, Cornell University, August 2011.

Mark K. Transtrum, Benjamin B. Machta, and James P. Sethna. Geometry of nonlinear least squares with applications to sloppy models and optimization. *Physical Review E*, 83(3):036701, Mar 2011a. doi: 10.1103/PhysRevE.83.036701.

Mark K. Transtrum, Benjamin B. Machta, and James P. Sethna. The geometry of nonlinear least squares with applications to sloppy models and optimization. *Physical Review E*, 83(3):036701, March 2011b. ISSN 1539-3755, 1550-2376.

Nilesh Tripuraneni, Chi Jin, and Michael I Jordan. Provable meta-learning of linear representations. *arXiv preprint arXiv:2002.11684*, 2020.

Asher Trockman and J Zico Kolter. Patches are all you need? *arXiv preprint arXiv:2201.09792*, 2022.

Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of machine learning research*, 9(11), 2008.

Vladimir Vapnik. *Statistical Learning Theory*. 1998.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.

Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. In *Advances in Neural Information Processing Systems*, pages 3630–3638, 2016.

Nhat Vo, Duc Vo, SungYoung Lee, and Subhash Challa. Weighted nonmetric MDS for sensor localization. In *2008 International Conference on Advanced Technologies for Communications*, pages 391–394. IEEE, 2008.

Stella Vosniadou and William F Brewer. Mental models of the earth: A study of conceptual change in childhood. *Cognitive psychology*, 24(4):535–585, 1992.

Joshua J. Waterfall, Fergal P. Casey, Ryan N. Gutenkunst, Kevin S. Brown, Christopher R. Myers, Piet W. Brouwer, Veit Elser, and James P. Sethna. Sloppy-Model Universality Class and the Vandermonde Matrix. *Physical Review Letters*, 97(15):150601, October 2006. doi: 10.1103/PhysRevLett.97.150601.

Alexander Wei, Wei Hu, and Jacob Steinhardt. More than a toy: Random matrix models predict how real-world neural representations generalize. *arXiv preprint arXiv:2203.06176*, 2022.

Colin Wei, Jason D Lee, Qiang Liu, and Tengyu Ma. Regularization Matters: Generalization and Optimization of Neural Nets v.s. their Induced Kernel. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d\textquotesingle Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 9712–9724. Curran Associates, Inc., 2019.

Harold Widom. Asymptotic behavior of the eigenvalues of certain integral equations. *Transactions of the American Mathematical Society*, 109(2):278–295, 1963.

Ross Wightman. Pytorch image models. https://github.com/rwightman/pytorch-image-models, 2019.

Blake Woodworth, Suriya Gunasekar, Jason D. Lee, Edward Moroshko, Pedro Savarese, Itay Golan, Daniel Soudry, and Nathan Srebro. Kernel and rich regimes in overparametrized models. In *Proceedings of Thirty Third Conference on Learning Theory*, page 3635–3673. PMLR, July 2020. URL https://proceedings.mlr.press/v125/woodworth20a.html.

Rubing Yang, Jialin Mao, and Pratik Chaudhari. Does the data induce capacity control in deep learning? In *Proc. of International Conference of Machine Learning (ICML)*, 2022.

Yuan Yao, Lorenzo Rosasco, and Andrea Caponnetto. On early stopping in gradient descent learning. *Constructive Approximation*, 26(2):289–315, 2007.

Dong Yin, Ashwin Pananjady, Max Lam, Dimitris Papailiopoulos, Kannan Ramchandran, and Peter Bartlett. Gradient diversity: A key ingredient for scalable distributed learning. In *International Conference on Artificial Intelligence and Statistics*, pages 1998–2007, 2018.

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *British Machine Vision Conference 2016*, 2016.

Amir R Zamir, Alexander Sax, William Shen, Leonidas J Guibas, Jitendra Malik, and Silvio Savarese. Taskonomy: Disentangling task transfer learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3712–3722, 2018.

Jure Zbontar, Li Jing, Ishan Misra, Yann LeCun, and Stéphane Deny. Barlow twins: Self-supervised learning via redundancy reduction. In *International Conference on Machine Learning*, pages 12310–12320, 2021.

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. In *International Conference on Learning Representations (ICLR)*, 2017a.

Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017b.

Richard Zhang. Making convolutional networks shift-invariant again. In *International conference on machine learning*, pages 7324–7334. PMLR, 2019.

Wenda Zhou, Victor Veitch, Morgane Austern, Ryan P Adams, and Peter Orbanz. Non-vacuous generalization bounds at the ImageNet scale: A PAC-Bayesian compression approach. In *International Conference on Learning Representations*, 2018.

Huaiyu Zhu, Christopher K I Williams, Richard Rohwer, and Michal Morciniec. Gaussian regression and optimal finite dimensional linear models. 1998.