PNAS RESEARCH ARTICLE | PHYSICS

The training process of many deep networks explores the same low-dimensional manifold

Jialin Mao[a], Itay Griniasty[b], Han Kheng Teoh[b], Rahul Ramesh[c], Rubing Yang[a], Mark K. Transtrum[d], James P. Sethna[b], and Pratik Chaudhari[e,1]

Edited by Herbert Levine, Northeastern University, Boston, MA; received June 15, 2023; accepted February 12, 2024

We develop information-geometric techniques to analyze the trajectories of the predictions of deep networks during training. 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, 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.

deep learning | information geometry | optimization | principal component analysis | visualization

Significance

Training a deep neural network involves solving a high-dimensional, large-scale, and nonconvex optimization problem and should be prohibitively hard—but it is quite tractable in practice. To shed light upon this paradox, we develop tools for the analysis and visualization of the prediction space of high-dimensional probabilistic models. Our experimental data show that the training process explores a low-dimensional manifold in the prediction space. Networks with many different architectures, trained with different optimization procedures, and regularization techniques traverse the same manifold. This suggests that the optimization problem in deep learning is inherently low dimensional.

We show that training trajectories of multiple deep neural networks with different architectures, optimization algorithms, hyperparameter settings, and regularization methods evolve on a remarkably low-dimensional manifold in the space of probability distributions. 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. 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 $\vec{y} = (y_1, \dots, y_N) \in \{1, \dots, C\}^N$ 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

$$P_w(\vec{y}) = \prod_{n=1}^{N} p_w^n(y_n), \qquad [1]$$

where w are the parameters of the network and we have used the shorthand $p_w^n(y_n) \equiv p_w(y_n \mid x_n)$. This is the joint likelihood of all the N labels given the inputs and the parameters w; see *SI Appendix*, S.2 for details. The probability distribution in Eq. **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." We develop techniques to analyze such high-dimensional probabilistic models and embed these models into lower-dimensional spaces for visualization.

We first show, using experimental data (with $NC \sim 10^6$ to 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 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 nonconvex objective, be feasible.

We next 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

Author contributions: J.M., I.G., H.K.T., R.R., R.Y., M.K.T., J.P.S., and P.C. designed research; J.M., I.G., H.K.T., R.R., and R.Y. performed research; J.M., I.G., H.K.T., R.R., R.Y., M.K.T., J.P.S., and P.C. contributed new reagents/analytic tools; J.M., I.G., H.K.T., R.R., R.Y., M.K.T., J.P.S., and P.C. analyzed data; and J.M., I.G., H.K.T., R.R., R.Y., M.K.T., J.P.S., and P.C. wrote the paper.

The authors declare no competing interest.

This article is a PNAS Direct Submission.

[1]To whom correspondence may be addressed. Email: pratikac@seas.upenn.edu.

This article contains supporting information online at https://www.pnas.org/lookup/suppl/doi:10.1073/pnas.2310002121/-/DCSupplemental.

Published March 12, 2024.

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. We find that models initialized at very different parts of the prediction space, e.g., by first fitting them to random labels, train along trajectories that merge quickly, approaching the true labels along the same manifold.

Methods

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 $P_* = \delta_{\vec{y}^*}(\vec{y})$ where $\vec{y}^*$ 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 is

$$\begin{aligned} \mathrm{d_B}(P_u, P_v) &= -N^{-1}\log\sum_{\vec{y}}\prod_{n=1}^{N}\sqrt{p_u^n(y_n)}\sqrt{p_v^n(y_n)} \\ &\overset{(*)}{=} -N^{-1}\log\prod_{n=1}^{N}\sum_{c=1}^{C}\sqrt{p_u^n(c)}\sqrt{p_v^n(c)}; \\ &= -N^{-1}\sum_n \log\sum_c \sqrt{p_u^n(c)}\sqrt{p_v^n(c)}. \end{aligned} \qquad [2]$$

Here, $(*)$ follows because samples are independent; see *SI Appendix*, S.2. for more details. 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 (1), or the geodesic distance on the product space (*SI Appendix*, S.5.3). 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 toward 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 nonzero 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 Eq. **1**. Geodesics in this space using the Fisher Information Metric (FIM) (2) 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 $P_{u,v}^{\alpha}(\vec{y}) = \prod_n p_{u,v}^{n,\alpha}(y_n)$ satisfies (3, Eq. **47**)

$$\sqrt{p_{u,v}^{n,\alpha}} = \frac{\sin((1-\alpha)\mathrm{d}_G^n)}{\sin(\mathrm{d}_G^n)}\sqrt{p_u^n} + \frac{\sin(\alpha \mathrm{d}_G^n)}{\sin(\mathrm{d}_G^n)}\sqrt{p_v^n}, \qquad [3]$$

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

$$s_w = \underset{\alpha\in[0,1]}{\operatorname{argmin}}\ \mathrm{d_G}(P_w, P_{0,*}^{\alpha}), \qquad [4]$$

where

$$\mathrm{d_G}(P_u, P_v) = N^{-1}\sum_n \cos^{-1}\sum_c \sqrt{p_u^n(c)}\sqrt{p_v^n(c)}$$

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 as in Fig. 1. We discuss the relationship between progress and error in *SI Appendix*, S.4.2. To find a point's progress, we solve Eq. **4** using a bisection search (4).

We would now like to convert each trajectory $\tilde{\tau}_u = (P_{u(k)})_{k=0,\dots,T}$ 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 Eq. **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 Eq. **3** the geodesically interpolated probability distribution $P_{u(k),u(k+1)}^{\alpha}$ 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

$$\mathrm{d_{traj}}(\tau_u, \tau_v) = \int_0^1 \mathrm{d_B}(P_{u(s)}, P_{v(s)})\,\mathrm{d}s, \qquad [5]$$

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) (1, 5) which is closely related to multidimensional scaling MDS (6) 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 $D_{uv} = \mathrm{d_B}(P_u, P_v)$ and

$$W = -LDL/2, \qquad [6]$$

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 $|\Lambda_{00}| \geq |\Lambda_{11}| \geq \dots$ allows us to compute the

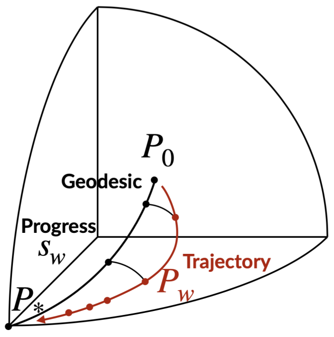


Fig. 1. A schematic of the procedure in Eq. **4** used to compute progress s_w by projecting a model P_w along a training trajectory onto the geodesic between ignorance P_0 and truth P_*.

*To aid the reader, *SI Appendix*, S.1. collects all the notation in one place.

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 $\mathbb{R}^{p,m-p} \ni X = U\sqrt{|\Lambda|}$[†]. In standard PCA, the embedding is always Euclidean since the eigenvalues of W are guaranteed to be nonnegative. 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 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.,

$$\|X_u - X_v\|^2 = \mathrm{d_B}(P_u, P_v) \geq 0 \qquad \textbf{[7]}$$

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 *SI Appendix*, S.4.1 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 (7) or UMAP (8) which approximately preserve local pairwise distances but distort 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

$$1 - \sqrt{\frac{\sum_{ij}(W_{ij} - \sum_{k=1}^{d} \sqrt{\Lambda_{kk}} U_{ik} \sqrt{\Lambda_{kk}} U_{kj})^2}{\sum_{ij} W_{ij}^2}} = 1 - \sqrt{\frac{\sum_{k=d+1}^{m} \Lambda_{kk}^2}{\sum_i \Lambda_{ii}^2}}; \qquad \textbf{[8]}$$

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 $m \sim 10^3$ to 10^5 models with $NC \sim 10^6$ to 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 Eq. **6** by magnitude suffices for the purpose of visualization. *SI Appendix*, S.5.3 discusses embeddings using other methods.

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 Eq. **6** to be

$$W_{uv} = -\frac{\mathrm{d_B}(P_u, P_v)}{2} + \frac{1}{2m} \sum_{u'} (\mathrm{d_B}(P_u, P_{u'}) + \mathrm{d_B}(P_v, P_{u'}) - \frac{1}{m} \sum_{v'} \mathrm{d_B}(P_{u'}, P_{v'})), \qquad \textbf{[9]}$$

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 uth 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. We discuss a generalization of this approach in *SI Appendix*, S.4.3. 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 ref. 9.

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 $\mathrm{argmin}_{P_w} m^{-1} \sum_i \mathrm{d_B}(P_i, P_w)$ (10). 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 (see Fig. S.14. from *SI Appendix*, 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.

Results

Experimental Data. We[‡] trained 2,296 different configurations on the CIFAR-10 dataset (11) corresponding to networks [§] with different a) network architectures fully connected, convolutional: AllCNN (12), residual: Wide ResNet (13), and ConvMixer (14), self-attention-based: ViT (15), b) network sizes (a small residual network and a large residual network), c) optimization methods SGD, SGD with Nesterov's acceleration and Adam (16), d) hyperparameters (learning rate and batch-size), e) regularization mechanisms with and without weight-decay (17), f) data augmentation (mean-SD-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.

We also performed a smaller scale experiment on ImageNet using a) three different architectures a small residual network: ResNet-18 (18), 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 (19), 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 1 summarizes the train and test errors of models used in our analysis. *SI Appendix*, S.3 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. Fig. 2*A* shows the first three dimensions of the InPCA embedding of the probabilistic model in Eq. **1** computed over samples in the training set. Each point corresponds to one model (i.e., one architecture, optimization algorithm, hyperparameters, regularization, weight initialization and a particular checkpoint along the training trajectory) and is colored by the architecture. The explained stress Eq. **8** of the first three dimensions is 76% as shown in Fig. 2*B*; it increases to 98%

[†]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 $\mathbb{C}^m$.

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

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

Table 1. Median (and 25 to 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*

	Fully connected (3.8M)	AllCNN (0.4M)	Small ResNet (0.3M)	Large ResNet (43.9M)	ConvMixer (0.6M)	ViT (9.5M)
CIFAR-10						
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)

	ResNet-18 (11.6M)	ResNet-50 (25.6M)	ViT-S (22M)
ImageNet			
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)

*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.

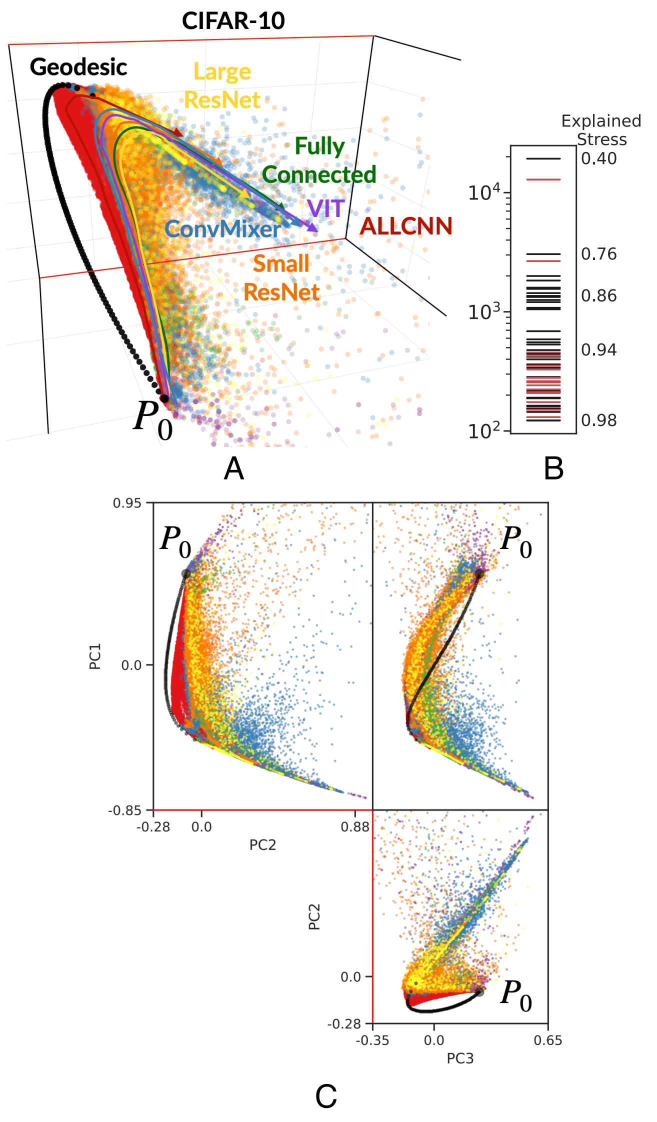
within the first 50 dimensions. The prediction space for CIFAR-10 has 4.5×10^5 dimensions ($N = 5 \times 10^4$ and $C = 10$); the rank of the distance matrix in InPCA is at most 151,407. For ImageNet, all networks are trained on the entire training set ($N = 1.28 \times 10^6$), but we use a subset of 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 Fig. 2*D*; 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 (Fig. 3*B*). The black curve denotes the embedding of the geodesic between P_0 and P_* calculated using Eq. **3**. Typical weight initialization schemes initialize models near P_0 irrespective of the configuration. Toward the end of training, 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. Submanifolds corresponding to each configuration seem to be rather similar; we will analyze this quantitatively in Fig. 7*A*. 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 toward 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 Fig. S.7 *A* and *B* in *SI Appendix*. The geodesic corresponds to the trajectory of natural gradient descent (21), 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 Fig. 2*A* 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. Submanifolds spanned by ViTs and ResNets appear different from each other while submanifolds of the smaller and larger ResNet are quite similar; we will see in Fig. 7*A* 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 Eq. **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).

Characterizing the details of the train manifold. Fig. 3*A* shows a pairwise comparison for the first three principal components of InPCA (same data as that of Fig. 2*A*). 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 Fig. 2. The third principal component is again space-like. All models that train well have small Bhattacharyya distances to the truth P_* toward 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 emphasizes these subtle differences using all coordinates, including the imaginary coordinate corresponding to the negative eigenvalue. Fig. 3*B* shows the train loss of all models (colored by purple for small,



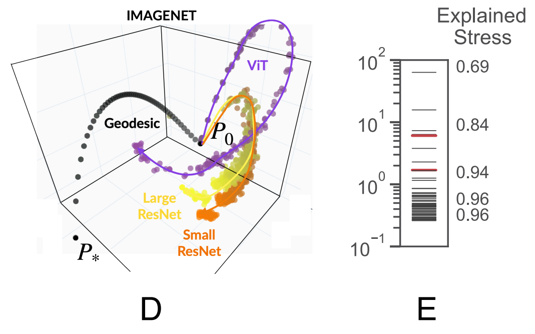


Fig. 2. The manifold of models along training trajectories of networks with different configurations (architectures denoted by different colors, optimization algorithms, hyperparameters, and regularization mechanisms) is effectively low dimensional for networks trained on CIFAR-10 (*A*). Different configurations train along similar trajectories but are quite different from the geodesic between ignorance P_0 and truth P_* (not seen here). (*B*) The manifold is hyperribbon-like (20): Eigenvalues of the InPCA distance matrix Eq. **6** for CIFAR-10 are spread over a large range with the top few dimensions capturing a large fraction of the stress Eq. **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*)). Results for models trained on ImageNet are similar (*D* and *E*). In (*A* and *D*), we have drawn smooth curves denoting trajectories by hand to guide the reader.

yellow for large). Even if the truth looks far away from them visually (>4 in a Euclidean sense), models colored purple in Fig. 3*B* have small distances from the truth $d_B(P_w, P_*) < 0.2$; incidentally their Minkowski distance to the truth in the top three coordinates is negative.

In Fig. 3*B*, the spread of points (yellow) near P_0 consists of some models that have 90% error (same as that of ignorance). There are 1,500 such points, coming from 370 different trajectories (over 85% of points are from 145 trajectories). Over half of these high error deviating networks (Fig. 4*B*) 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 Fig. 3*D*. 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). Fig. 3*C* 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 Fig. 3*A*; as Fig. 3*D* shows, the distance of a bulk of these ConvMixer models to point A is small (< 0.1). And Fig. 3*C* suggests that these models have error <10% (some also have larger errors). In this region, the spread of the points in the visualization is coming predominantly from the small differences in the probabilities.

Fig. 4*A* studies models that are away from the manifold, with $d_B(P, P_*) > 2$ (yellow in Fig. 3*B*). For ConvMixer and the two residual networks, a majority of these models were trained by Adam. No AllCNN models were away from the manifold.

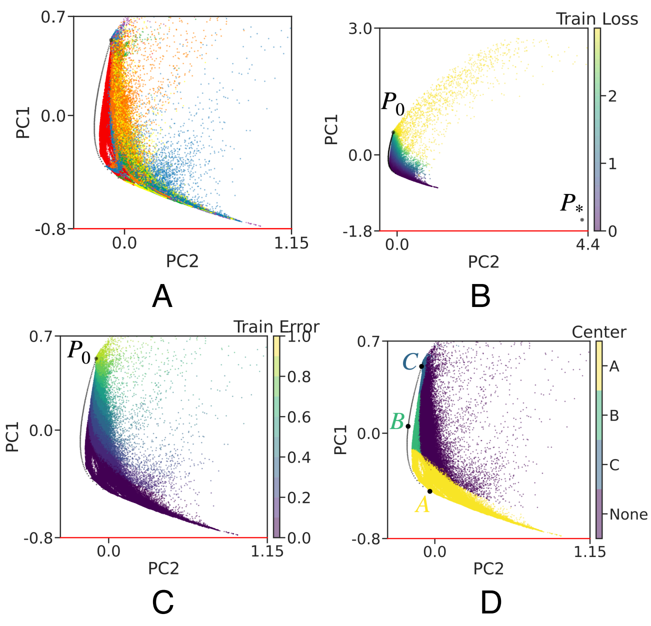


Fig. 3. Comparison of the top two principal components of an InPCA embedding of all models on CIFAR-10 colored by the architectures (*A*) (same as Fig. 2*C*), train loss (*B*), which is two times the Bhattacharyya distance $d_B(P, P_*)$ for classification tasks like ours, train error in (*C*), and by 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 Fig. 2*C*.

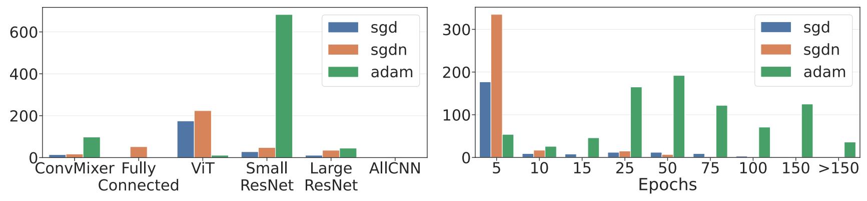


Fig. 4. Number of models P with $d_B(P, P_*) > 2$ (that are away from the main manifold) stratified by (*A*) architectures and (*B*) the number of epochs.

Fig. 4*B* 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. Fig. 5*A* shows the first three dimensions of the InPCA embedding of predictions on the test data using the same networks as that of Fig. 2*A*. 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 $C = 1{,}000$). 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 Fig. 2*A*. Trajectories begin near ignorance ($d_B(P, P_0) < 0.6$ 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 Fig. 5*A* than those in Fig. 2*A*; networks of different architectures have more distinctive test trajectories. We have analyzed these differences quantitatively in Fig. S.9*A* in *SI Appendix*. 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.

SI Appendix S.5.2 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 Eq. **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 toward the truth in the train space results in progress toward 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 Eq. **9** to embed test models into these coordinates as follows. Let us denote by P'_u the model on the test data for the same weights u. Calculate

$$W'_{uv} = -\frac{d_B(P'_u, P'_v)}{2} + \frac{1}{2m}\left(\sum_{u'} d_B(P_u, P_{u'}) + d_B(P_v, P_{u'}) - \frac{1}{m}\sum_{v'} d_B(P_{u'}, P_{v'})\right), \qquad [10]$$

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 Eq. **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 Eq. **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 Eq. **10** is close to its counterpart in Eq. **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 Eq. **10** to calculate the coordinates of the test models and obtained Fig. 6*A*. Observations drawn from this procedure are qualitatively the same as those from Figs. 2 and 5, 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 Fig. 5*A*; 0.52 vs. 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 Eq. **2** calculated using only a subset

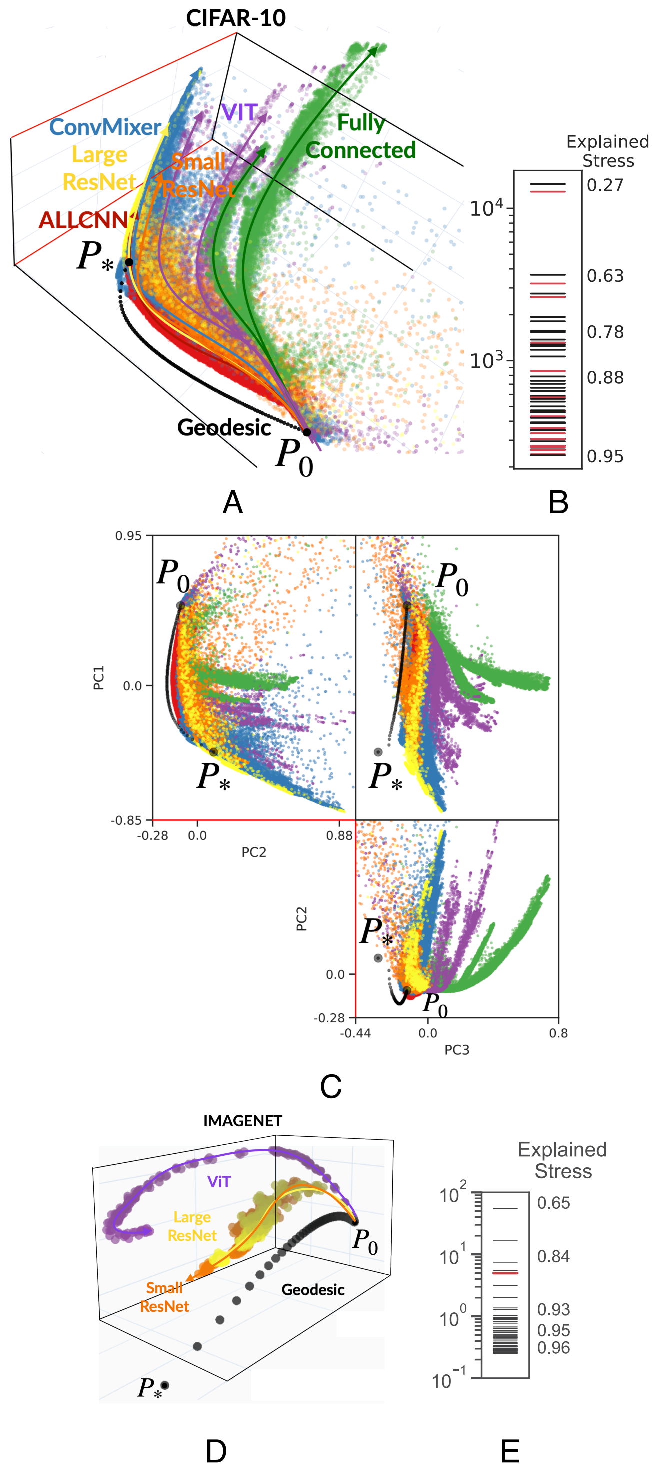


Fig. 5. Predictions on the test data of networks with different configurations (architectures denoted by different colors, different optimization algorithms, and regularization mechanisms) trained on CIFAR-10 (*A*) is also effectively low-dimensional. Trajectories of different architectures are distinctive on the test data. (*B*) Test manifold is also hyperribbon-like: eigenvalues of the InPCA distance matrix Eq. **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 Eq. **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 Fig. 2*C* look quite similar to those of (*C*). Results for models trained on ImageNet are similar (*D* and *E*). In (*A* and *D*), we have drawn smooth curves denoting trajectories by hand to guide the reader.

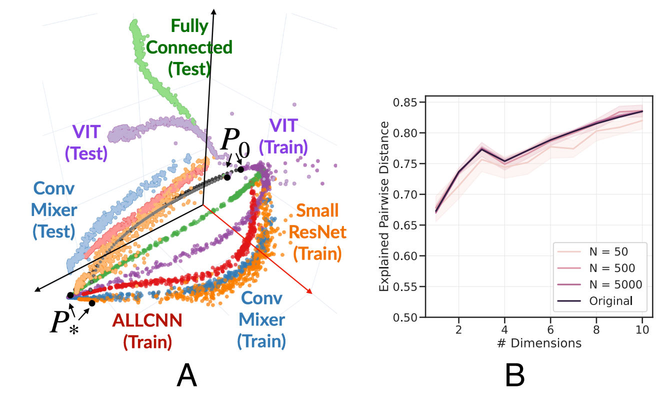


Fig. 6. (*A*) A joint embedding of a subset of networks on CIFAR-10 using their predictions on samples from both the train (bold) and test (translucent) sets. (*B*) The explained pairwise Bhattacharyya distances computed using Eq. **12** is quite high for models on the train data after embedding them into an InPCA embedding computed using a small number of samples ($N = 5{,}000$, $N = 500$, and $N = 50$) in the train data. *SI Appendix*, S.4.4 discusses this further.

of the samples. Figs. S.3 and S.4 in *SI Appendix* show the result of using the procedure in Eq. **10** 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 $C = 1{,}000$, 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 Eq. **4** toward the truth P_*. Using these interpolations, we calculated the distance between trajectories corresponding to different configurations using Eq. **5**, averaged over the weight initializations. Fig. 7*A* 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 Figs. 2*A* and 5*A*). 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 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, hyperparameters, 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.

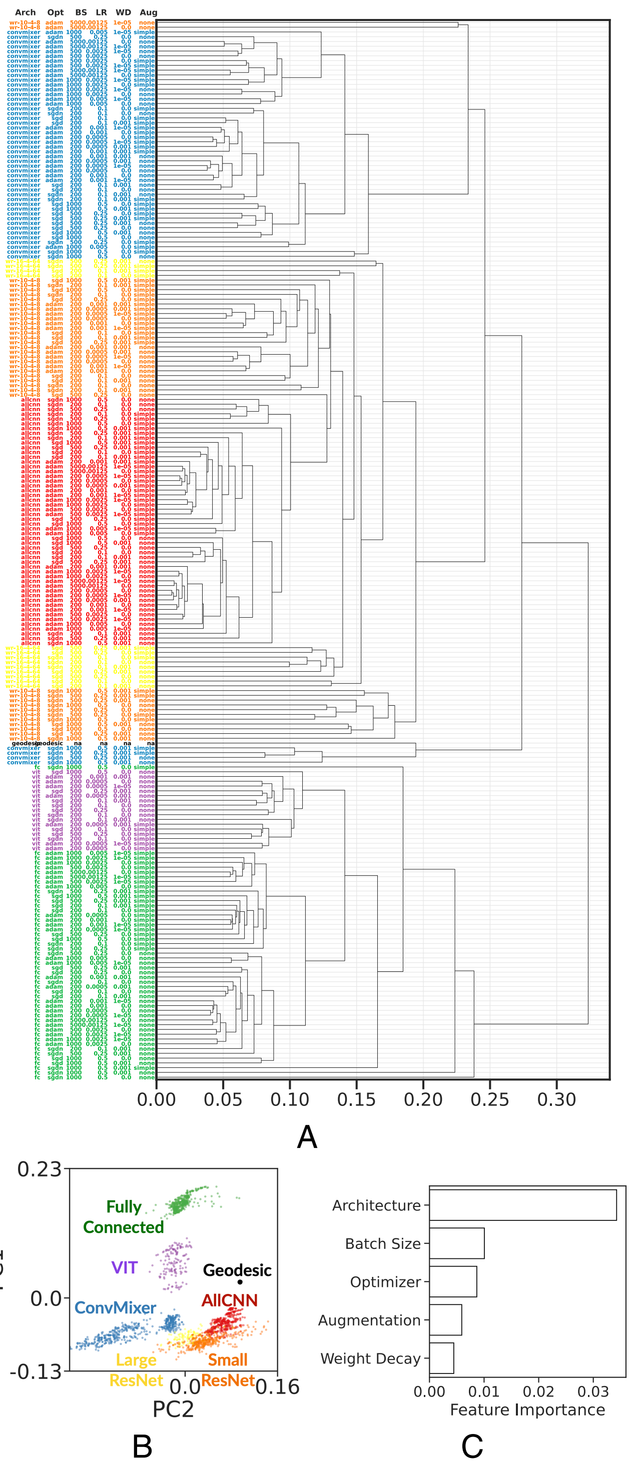


Fig. 7. (*A*) dendrogram obtained from hierarchical clustering of pairwise distances (averaged over weight initializations) between training trajectories (computed using Eq. **5**) of networks with different configurations (X-labels correspond to architecture, optimization algorithm, batch-size, learning 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 (Fig. S.9 in *SI Appendix*).

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; Fig. 9 shows that the larger network progresses faster toward 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 nonconvolutional 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 rates 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 Fig. 7*B*, 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 Fig. 2*C* depicts a low-dimensional manifold, Fig. 7*B* 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 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 Fig. 7*C* 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.

SI Appendix, S.5.1 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 toward the end of training all test models are quite far from truth. As S.5.2 and Fig. S.9*A* in *SI Appendix* 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 toward the Truth for the Same Number of Gradient Updates. Networks with different configurations make progress toward the truth P_* at different rates. As Fig. 8 shows, progress

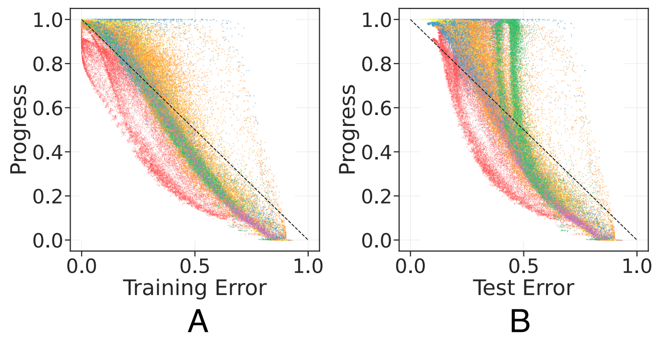


Fig. 8. Progress of models with different configurations (color scheme is same as that of Fig. 2*A*) is strongly correlated with (*A*) train error ($R^2 = 0.95$), and (*B*) test error ($R^2 = 0.88$).

is strongly correlated with both train error ($R^2 = 0.95$) and test error ($R^2 = 0.88$). Progress toward the train truth and toward the test truth is also highly correlated with each other ($R^2 = 0.99$). This suggests that progress, which can be calculated easily using Eq. **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. Fig. 8 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 Fig. 2*A* and Fig. S.7*A* in *SI Appendix* 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.

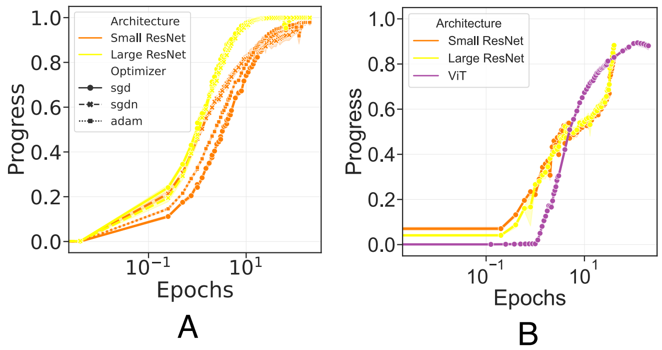


Fig. 9. A Large ResNet makes more progress toward the truth than a Small ResNet for the same number of gradient updates on CIFAR-10 (*A*) and ImageNet (*B*), irrespective of the optimization algorithms. Since the manifold of train and test trajectories for the two architectures are very similar (Fig. 7*A*), this suggests that larger networks and smaller networks make the same kind of predictions but the larger ones simply learn faster.

We saw in Fig. 7*A* that trajectories of the Large ResNet lie on the same submanifold as that of the Small ResNet; see Fig. S.6 in *SI Appendix* for the tube widths. The trend for the test manifold in Fig. S.9*A* in *SI Appendix* is similar. After the same number of gradient updates, the Large ResNet makes more progress toward the truth than the Small ResNet on CIFAR-10 (Fig. 9*A*). Fig. 9*B* 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.

Discussion

An 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 (22, 23) or analytical models (24–26), analyzing representations of different layers (27, 28), properties of stochastic optimization methods (29) 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 find 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 (30–32). A large body of work has sought to analyze this phenomenon (33–35), and it has also been argued that we need to rethink our understanding of generalization in machine learning (36). We have found that a Large ResNet trains along the same manifold as that of a Small ResNet. It proceeds further toward the truth in the later parts of the trajectory. In view of the effectiveness of pruning and knowledge distillation (37, 38), 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 (39), or by arguing that the minibatch gradient has a large overlap with the subspace spanned by the top few eigenvectors of the Hessian during training for networks without batch-normalization (40–42). 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 parameterization of deep networks.

Computational Information Geometry. Information Geometry (2) 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 Eq. **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 (43). Our visualization techniques, that build upon InPCA and IsKL (1, 5), work around this issue using multidimensional scaling (6, 44) 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 (45). Together with these visualization techniques, the theory developed in this paper gives 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 Fig. 2*A*. We next offer an interpretation of this phenomenon. Our probabilistic models are an N-product of probability distributions corresponding to points $(\sqrt{p_u^n(1)}, \dots, \sqrt{p_u^n(C)})$ 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. The latter two are stand-ins for the tangents to the trajectories at P_0 and P_* and they are calculated using

$$P_s = \frac{1}{Z}\sum_{P'} \exp\left(\frac{-(s_{P'}-s)^2}{2\sigma^2}\right) P', \qquad [11]$$

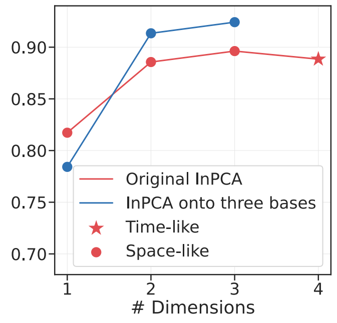


Fig. 10. The procedure in Eq. **9** was used to add original models used for Fig. 2*A* into an InPCA embedding created using four points corresponding to three "bases" (straight line from ignorance to truth, and tangents to the training trajectories at ignorance and truth) for three configurations, all 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 four points, only the first three dimensions are nontrivial.

where $Z = \sum_{P'} \exp(-(s_{P'}-s)^2/(2\sigma^2))$ 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 four models, and using the procedure in Eq. **9** (which is equivalent to weighted-InPCA discussed in *SI Appendix*, S.4.3), we can add our original models in Fig. 2*A* into this new InPCA embedding.

Fig. 10 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

$$1 - \frac{\sum_{ij} |D_{ij} - \|X_i - X_j\|^2|}{\sum_{ij} D_{ij}}, \qquad [12]$$

where $X_i \in \mathbb{R}^{q,d-q}$ are the d-dimensional coordinates of the embedded points; we can calculate this 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 four 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 Fig. 10 are ordered by the magnitude of eigenvalues).

We next performed the same analysis but with all models in Fig. 2*A* with $d_B(P, P_*) < 2$, which effectively removes models that lie away from the manifold. In Fig. 11a, we created an InPCA embedding using four 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 Eq. **11**, and projected the original probabilistic models into these new coordinates using the procedure in Eq. **9** to visualize them. We rotated the top 3 nontrivial 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 Kabsch–Umeyama algorithm (46) 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 Fig. 11*B* shows, there are structural similarities in the embedding computed using only the four points and the

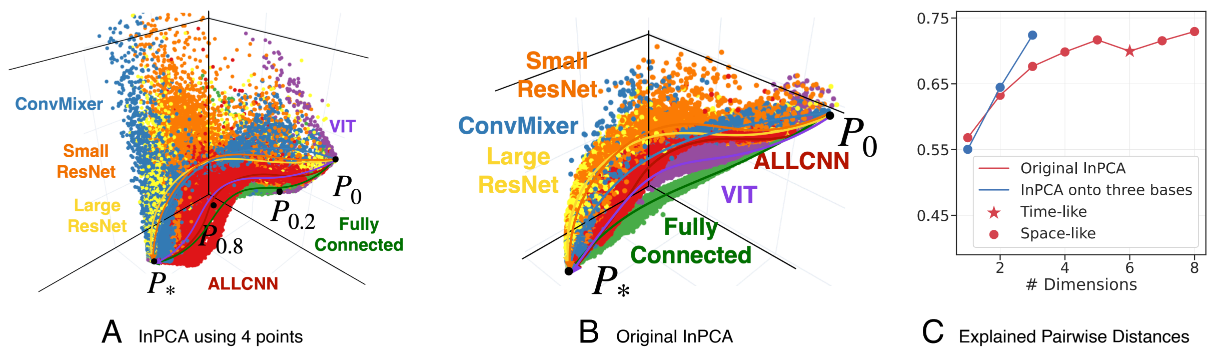


Fig. 11. All models in Fig. 2*A* with Bhattacharya distance $d_B(P, P_*) < 2$, which effectively removed the spread of points away from the train manifold (Fig. 3*B*), were embedded using InPCA coordinates constructed using four points corresponding to three "bases" (straight line from the ignorance to truth, and tangents to the training trajectories at ignorance and truth) in (*A*) and using the original InPCA coordinates in Fig. 2*A* 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 eight dimensions in the original InPCA. We have drawn smooth curves denoting trajectories by hand to guide the reader.

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. Fig. 11*C* 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 identified 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, hyperparameter settings, and regularization mechanisms fit the same easy images in the dataset first and the same challenging images toward the end of training; this phenomenon has also been studied in ref. 47.

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 (48, 49), and b) sampled from an isotropic Gaussian (nonsloppy 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 $\{P_0^{(k)}\}_{k=1,\dots,10}$ similar to those in Fig. S.10 in *SI Appendix*, 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. *SI Appendix*, S.6 discusses this experiment further.

We therefore believe that the low-dimensionality of the manifold arises from a) the structure of typical datasets (50–52), 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 (48, 53) has argued that deep networks, as also linear/kernel models, can interpolate without overfitting if input data have a sloppy spectrum. Work in neuroscience (54, 55) has also argued for visual data being effectively low-dimensional. Theories in machine learning (56, 57) and information-theory (58, 59) 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 (60, 61), the subset of the hypothesis space explored by typical training algorithms might be much smaller.

Data, Materials, and Software Availability. Code data have been deposited in Github (https://github.com/grasp-lyrl/low-dimensional-deepnets) (62).

ACKNOWLEDGMENTS. J.M., R.R., R.Y., and P.C. were supported by grants from the NSF (IIS-2145164, CCF-2212519), the Office of Naval Research (N00014-22-1-2255), and cloud computing credits from Amazon Web Services. I.G. was supported by the NSF (DMREF-89228, EFRI-1935252) and Eric and Wendy Schmidt AI in Science Postdoctoral Fellowship. H.K.T. was supported by the NIH (1R01NS116595-01). J.P.S. was supported by the NSF (DMR-1719490), M.K.T. was supported by the NSF (DMR-1753357). We would like to acknowledge Itai Cohen and Jay Spendlove for helpful comments on this material and manuscript.

Author affiliations: [a]Applied Mathematics and Computational Sciences, University of Pennsylvania, Philadelphia, PA 19104; [b]Laboratory of Atomic and Solid State Physics, Cornell University, Ithaca, NY 14853; [c]Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA 19104; and [d]Department of Physics and Astronomy, Brigham Young University, Provo, UT 84604; and [e]Department of Electrical and Systems Engineering, University of Pennsylvania, Philadelphia, PA 19104

1. H. K. Teoh *et al.*, Visualizing probabilistic models in Minkowski space with intensive symmetrized Kullback-Leibler embedding. *Phys. Rev. Res.* **2**, 033221 (2020).
2. S. Amari, *Information Geometry and Its Applications. Applied Mathematical Sciences, Tokyo* (2016), vol. 194.
3. S. Ito, A. Dechant, Stochastic time evolution, information geometry, and the Cramér-Rao bound. *Phys. Rev. X* **10**, 021056 (2020).
4. R. P. Brent, An algorithm with guaranteed convergence for finding a zero of a function. *Comput. J.* **14**, 422–425 (1971).

5. K. N. Quinn, C. B. Clement, F. De Bernardis, M. D. Niemack, J. P. Sethna, Visualizing probabilistic models and data with intensive principal component analysis. *Proc. Natl. Acad. Sci. U.S.A.* **116**, 13762–13767 (2019).
6. M. A. A. Cox, T. F. Cox, "Multidimensional scaling" in *Handbook of Data Visualization* (2008), pp. 315–347.
7. L. Van der Maaten, G. Hinton, Visualizing data using t-SNE. *J. Mach. Learn. Res.* **9** (2008).
8. L. McInnes, J. Healy, N. Saul, L. Großberger, UMAP: Uniform manifold approximation and projection. *J. Open Source Softw.* **3**, 861 (2018).
9. V. De Silva, J. B. Tenenbaum, "Sparse multidimensional scaling using landmark points" (Tech. Rep., Stanford University, 2004).
10. F. Nielsen, S. Boltz, The Burbea-Rao and Bhattacharyya centroids. *IEEE Trans. Inf. Theory* **57**, 5455–5466 (2011).
11. A. Krizhevsky, Learning multiple layers of features from tiny images. PhD thesis, Computer Science, University of Toronto (2009).
12. J. T. Springenberg, A. Dosovitskiy, T. Brox, M. Riedmiller, Striving for simplicity: The all convolutional net. arXiv [Preprint] (2015). http://arxiv.org/abs/1412.6806 (Accessed 23 October 2023).
13. S. Zagoruyko, N. Komodakis, "Wide residual networks" in *British Machine Vision Conference 2016* (2016).
14. A. Trockman, J. Z. Kolter, Patches are all you need? *Trans. Mach. Learn. Res.* (2023).
15. A. Dosovitskiy *et al*., An image is worth 16x16 words: Transformers for image recognition at scale. arXiv [Preprint] (2020). http://arxiv.org/abs/2010.11929.
16. D. Kingma, J. Ba, "Adam: A method for stochastic optimization" in *International Conference for Learning Representations* (2015).
17. S. Ioffe, C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift" in *International Conference on Machine Learning*, F. Bach, D. Blei, Eds. (Proceedings of Machine Learning Research (PMLR), 2015), pp. 448–456.
18. K. He, X. Zhang, S. Ren, J. Sun, "Deep residual learning for image recognition" in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2016), pp. 770–778.
19. B. Heo *et al*., "Adamp: Slowing down the slowdown for momentum optimizers on scale-invariant weights" in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2021).
20. M. K. Transtrum, B. B. Machta, J. P. Sethna, The geometry of nonlinear least squares with applications to sloppy models and optimization. *Phys. Rev. E* **83**, 036701 (2011).
21. S. Amari, Natural gradient works efficiently in learning. *Neural Comput.* **10**, 251–276 (1998).
22. P. Baldi, K. Hornik, Neural networks and principal component analysis: Learning from examples without local minima. *Neural Netw.* **2**, 53–58 (1989).
23. T. Liang, A. Rakhlin, Just interpolate: Kernel "ridgeless" regression can generalize. *Ann. Stat.* **48**, 1329–1347 (2020).
24. S. Mei, T. Misiakiewicz, A. Montanari, "Mean-field theory of two-layers neural networks: Dimension-free bounds and kernel limit" in *Conference on Learning Theory* (2019), pp. 2388–2464.
25. L. Chizat, F. Bach, "Implicit bias of gradient descent for wide two-layer neural networks trained with the logistic loss" in *Conference on Learning Theory* (2020), pp. 1305–1338.
26. A. Jacot, F. Gabriel, C. Hongler, Neural tangent kernel: Convergence and generalization in neural networks. *Adv. Neural Inf. Process. Syst.* **31**, 8571–8580 (2018).
27. R. Shwartz-Ziv, N. Tishby, Opening the black box of deep neural networks via information. arXiv [Preprint] (2017). http://arxiv.org/abs/1703.00810 (Accessed 23 October 2023).
28. A. Achille, S. Soatto, Emergence of invariance and disentanglement in deep representations. *J. Mach. Learn. Res.* **19**, 1947–1980 (2018).
29. P. Chaudhari, S. Soatto, "Stochastic gradient descent performs variational inference, converges to limit cycles for deep networks" in *Proceedings of International Conference of Learning and Representations (ICLR)* (2018).
30. T. Brown *et al*., Language models are few-shot learners. *Adv. Neural. Inf. Process. Syst.* **33**, 1877–1901 (2020).
31. A. Vaswani *et al*., "Attention is all you need" in *Advances in Neural Information Processing Systems*, I. Guyon *et al*., Eds. (Curran Associates, Inc., 2017).
32. A. Dosovitskiy *et al*., "An image is worth 16x16 words: Transformers for image recognition at scale" in *International Conference on Learning Representations* (2021).
33. M. Belkin, Fit without fear: Remarkable mathematical phenomena of deep learning through the prism of interpolation. *Acta Numer.* **30**, 203–248 (2021).
34. M. Belkin, D. Hsu, S. Ma, S. Mandal, Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proc. Natl. Acad. Sci. U.S.A.* **116**, 15849–15854 (2019).
35. P. L. Bartlett, A. Montanari, A. Rakhlin, Deep learning: A statistical viewpoint. *Acta Numer.* **30**, 87–201 (2021).
36. C. Zhang, S. Bengio, M. Hardt, B. Recht, O. Vinyals, "Understanding deep learning requires rethinking generalization" in *International Conference on Learning Representations (ICLR) 2017* (2017).
37. J. Frankle, M. Carbin, The lottery ticket hypothesis: Finding sparse, trainable neural networks. arXiv [Preprint] (2019). http://arxiv.org/abs/1803.03635.
38. G. Hinton, O. Vinyals, J. Dean, "Distilling the knowledge in a neural network" in *NIPS Deep Learning and Representation Learning Workshop* (2015).
39. Y. Feng, Y. Tu, The inverse variance-flatness relation in stochastic gradient descent is critical for finding flat minima. *Proc. Natl. Acad. Sci. U.S.A.* **118**, e2015617118 (2021).
40. G. Gur-Ari, D. A. Roberts, E. Dyer, Gradient descent happens in a tiny subspace. arXiv [Preprint] (2018). http://arxiv.org/abs/1812.04754 (Accessed 23 October 2023).
41. B. Ghorbani, S. Krishnan, Y. Xiao, "An investigation into neural net optimization via Hessian eigenvalue density" in *Proceedings of the 36th International Conference on Machine Learning*, K. Chaudhuri, R. Salakhutdinov, Eds. (Proceedings of Machine Learning Research (PMLR), 2019).
42. L. Sagun, L. Bottou, Y. LeCun, Eigenvalues of the Hessian in deep learning: Singularity and beyond. arXiv [Preprint] (2017). http://arxiv.org/abs/1611.07476 (Accessed 23 October 2023).
43. J. Antognini, J. Sohl-Dickstein, PCA of high dimensional random walks with comparison to neural network training. *Adv. Neural. Inf. Process. Syst.* **31** (2018).
44. A. M. Saxe, J. L. McClelland, S. Ganguli, A mathematical theory of semantic development in deep neural networks. *Proc. Natl. Acad. Sci. U.S.A.* **116**, 11537–11546 (2019).
45. J. Laub, K.-R. Müller, Feature discovery in non-metric pairwise data. *J. Mach. Learn. Res.* **5**, 801–818 (2004).
46. J. Lawrence, J. Bernal, C. Witzgall, A purely algebraic justification of the Kabsch-Umeyama algorithm. *J. Res. Nat. Inst. Stand. Technol.* **124**, 1 (2019).
47. G. Hacohen, L. Choshen, D. Weinshall, "Let's agree to agree: Neural networks share classification order on real datasets" in *International Conference on Machine Learning* (2020), pp. 3950–3960.
48. R. Yang, J. Mao, P. Chaudhari, "Does the data induce capacity control in deep learning?" in *Proceedings of International Conference of Machine Learning (ICML)* (2022).
49. K. N. Quinn, M. C. Abbott, M. K. Transtrum, B. B. Machta, J. P. Sethna, Information geometry for multiparameter models: New perspectives on the origin of simplicity. *Rep. Prog. Phys.* **86**, 035901 (2022).
50. S. Goldt, M. Mézard, F. Krzakala, L. Zdeborová, Modeling the influence of data structure on learning in neural networks: The hidden manifold model. *Phys. Rev. X* **10**, 041044 (2020).
51. S. d'Ascoli, M. Gabrié, L. Sagun, G. Biroli, On the interplay between data structure and loss function in classification problems. *Adv. Neural. Inf. Process. Syst.* **34**, 8506–8517 (2021).
52. M. Refinetti, S. Goldt, F. Krzakala, L. Zdeborová, "Classifying high-dimensional Gaussian mixtures: Where kernel methods fail and neural networks succeed" in *International Conference on Machine Learning*, M. Meila, T. Zhang, Eds. (Proceedings of Machine Learning Research (PMLR), 2021), pp. 8936–8947.
53. P. L. Bartlett, P. M. Long, G. Lugosi, A. Tsigler, Benign overfitting in linear regression. *Proc. Natl. Acad. Sci. U.S.A.* **117**, 30063–30070 (2020).
54. E. P. Simoncelli, B. A. Olshausen, Natural image statistics and neural representation. *Annu. Rev. Neurosci.* **24**, 1193–1216 (2001).
55. D. J. Field, What is the goal of sensory coding? *Neural Comput.* **6**, 559–601 (1994).
56. V. Vapnik, *Statistical Learning Theory* (1998).
57. B. Schölkopf, A. J. Smola, *Learning with Kernels* (2002).
58. J. Rissanen, Modeling by shortest data description. *Automatica* **14**, 465–471 (1978).
59. V. Balasubramanian, Statistical inference, occam's razor, and statistical mechanics on the space of probability distributions. *Neural Comput.* **9**, 349–368 (1997).
60. G. K. Dziugaite, D. M. Roy, "Computing nonvacuous generalization bounds for deep (stochastic) neural networks with many more parameters than training data" in *Proceedings of Conference of Uncertainity in Artificial Intelligence* (2017).
61. P. L. Bartlett, D. J. Foster, M. J. Telgarsky, "Spectrally-normalized margin bounds for neural networks" in *Advances in Neural Information Processing Systems*, I. Guyon *et al*., Eds. (Curran Associates, Inc., 2017), pp. 6240–6249.
62. J. Mao *et al*., Data from "low-dimensional-deepnets". Code. Github. https://github.com/grasp-lyrl/low-dimensional-deepnets. Deposited 17 June 2023.