Supporting Information for

Your main manuscript title

Jialin Mao, Itay Griniasty, Han Kheng Teoh, Rahul Ramesh, Rubing Yang, Mark K. Transtrum, James P. Sethna, Pratik Chaudhari

Corresponding Author: Pratik Chaudhari (pratikac@seas.upenn.edu)

This PDF file includes:

Figs. S1 to S18
Table S1
SI References

S.1. Notation

Symbol	Description
N	Number of samples
C	Number of classes
x_n	Input sample with index $n \in \{1, \ldots, N\}$
y_n	Label assignment of sample with index $n \in \{1, \ldots, N\}$
y_n^*	Ground-truth label of sample with index $n \in \{1, \ldots, N\}$
w	Weights of the deep network
$\vec{y}^*$	Ground-truth labels for each of the N samples, $\vec{y}^* = (y_1^*, \ldots, y_N^*)$
$\vec{y}$	Label assignment for each of the N samples, $\vec{y} \in \{1, \ldots, C\}^N$
$p_w^n(y_n)$	Probability that sample x_n belongs to class $y_n \in \{1, \ldots, C\}$, $p_w^n(y_n) \equiv p_w(y_n \mid x_n)$
$P_w(.)$	Probabilistic model with weight w; assigns a probability to every sequence $\vec{y}$
P_*	Truth ($P_* = \delta_{\vec{y}^*}(\vec{y})$)
P_0	Ignorance, has $p_0^n(c) = 1/C$ for all classes c and samples n
$\mathsf{d_B}$	Bhattacharyya distance between two probability distributions
$\mathsf{d_G}$	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 $(P_{w(s)})_{s\in[0,1]}$
$\mathsf{d_{traj}}(\tau_u, \tau_v)$	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

S.2. Derivation of the joint probability of predictions and the Bhattacharyya distance

The quantity in [1] is the joint likelihood of all the labels given the weights. Observe that

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

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., $p(y_n \mid x_1, \ldots, x_n) = p(y_n \mid x_n)$, and (c) the samples are frozen to the ones in the training set for the analysis, i.e., the distribution $p(x_1, \ldots, x_n) \equiv \left(\frac{1}{N} \sum_{n=1}^N \delta_{x_n}(x_n)\right) = 1$. So we actually do not need to use the assumption that the training samples $x_1, \ldots, x_N$ 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 (1, Equation 1.61).

The expression for the Bhattacharyya distance between two probability distributions P_u and P_v in [2] can be derived as follows. Note that $\vec{y}$ can take a total of C^N distinct values, and each $y_n \in \{1, \ldots, C\}$.

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

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.

S.3. 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 (2) 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 (3) 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 (4)) 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 (5). 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 (6) 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 (7) 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 (8) in any of the networks. All networks except ViT have a batch-normalization (9) layer after each convolutional or fully-connected layer, except ViT which uses layer normalization (10).

For ImageNet, we used three architectures.

(i) A smaller residual network (11) 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 (12).

(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 (13). 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 (14) for training ResNets and timm (15) 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. Fig. S.1 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.

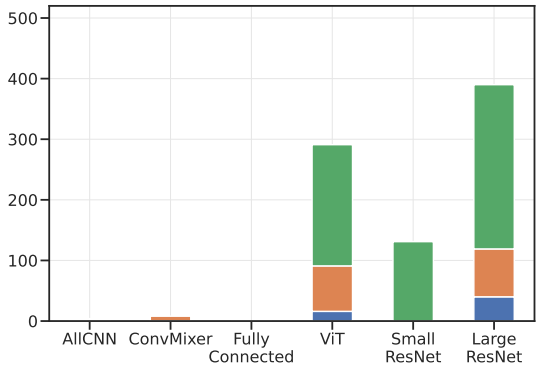


Fig. S.1. Number of networks that did not train beyond 90% error for Adam (green), SGD (blue) and SGD with Nesterov's acceleration (orange). These models are not included in our analysis.

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 (16), three different 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 (17). This has also been argued theoretically via an analysis of the equilibrium distribution of SGD (18). 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 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 (19), 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 (20) 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 (21) 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 (9) 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 Fig. S.2.

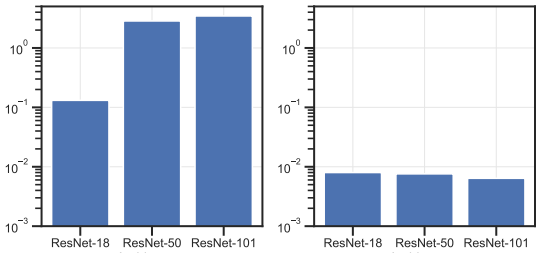


Fig. S.2. Bhattacharyya distance from ignorance P_0 for networks at the beginning of training for standard off-the-shelf implementations of ResNet (left). If we initialize the estimates of the mean and standard deviation of the batch-normalization layers by doing a forward pass on a few mini-batches, then networks are close to ignorance at the beginning of training (right).

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 (22), meta-learning (23) 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 Figs. 2d and 5d.

S.4. Addendum to Methods

S.4.1. InPCA creates an isometric embedding. InPCA, like standard PCA, relies on an embedding directed by the centered pairwise distances [6]. Observe that the centering in [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 [7]. The argument developed below also holds for other embedding techniques, e.g., the IsKL method discussed in [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 $\langle a, b \rangle_D \doteq \sum_k \mathrm{sign}(\Lambda_{kk}) a_k b_k$ 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:

$$D_{ij} = \langle X_i, X_j \rangle_D \,. \quad [1]$$

Now consider a finite symmetric premetric space $\mathcal{M} = (M, D)$ with $|M| = m$ points*. If D is a matrix of pairwise distances between these points, then it has a vanishing diagonal. The embedding of $-D/2$ denoted by $\left\{X_i \in \mathbb{R}^{p,m-p}\right\}_{i=1}^m$ 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.,

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

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

Theorem 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 [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

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

□

S.4.2. 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

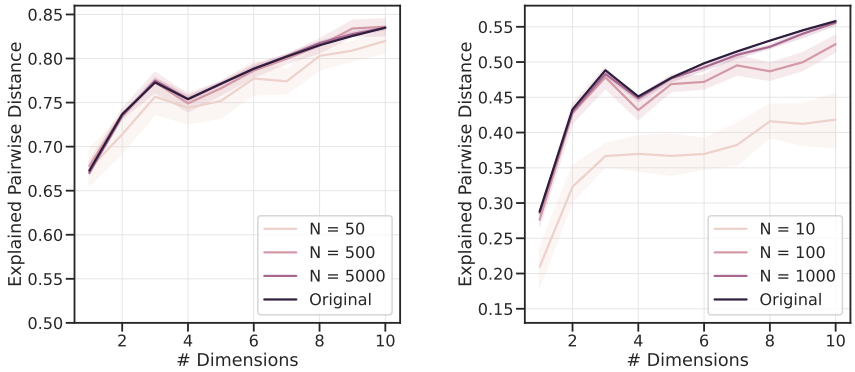
$$\begin{aligned}
\alpha^* &= \operatorname*{argmin}_{\alpha \in [0,1]} \mathrm{d_G}(P, P^\alpha_{0,*}) \\
&= (1 - \epsilon) \cos^{-1}\left(\frac{\sin((1-\alpha)\mathrm{d_G^n})}{\sin(\mathrm{d_G^n})} \cos(\mathrm{d_G^n}) + \frac{\sin(\alpha \mathrm{d_G^n})}{\sin(\mathrm{d_G^n})} \right) + \epsilon \cos^{-1}\left(\frac{\sin((1-\alpha)\mathrm{d_G^n})}{\sin(\mathrm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

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 $\mathrm{d_G^n} = \cos^{-1}(1/\sqrt{C})$ 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 Fig. 8).

S.4.3. 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 (24–27) 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.

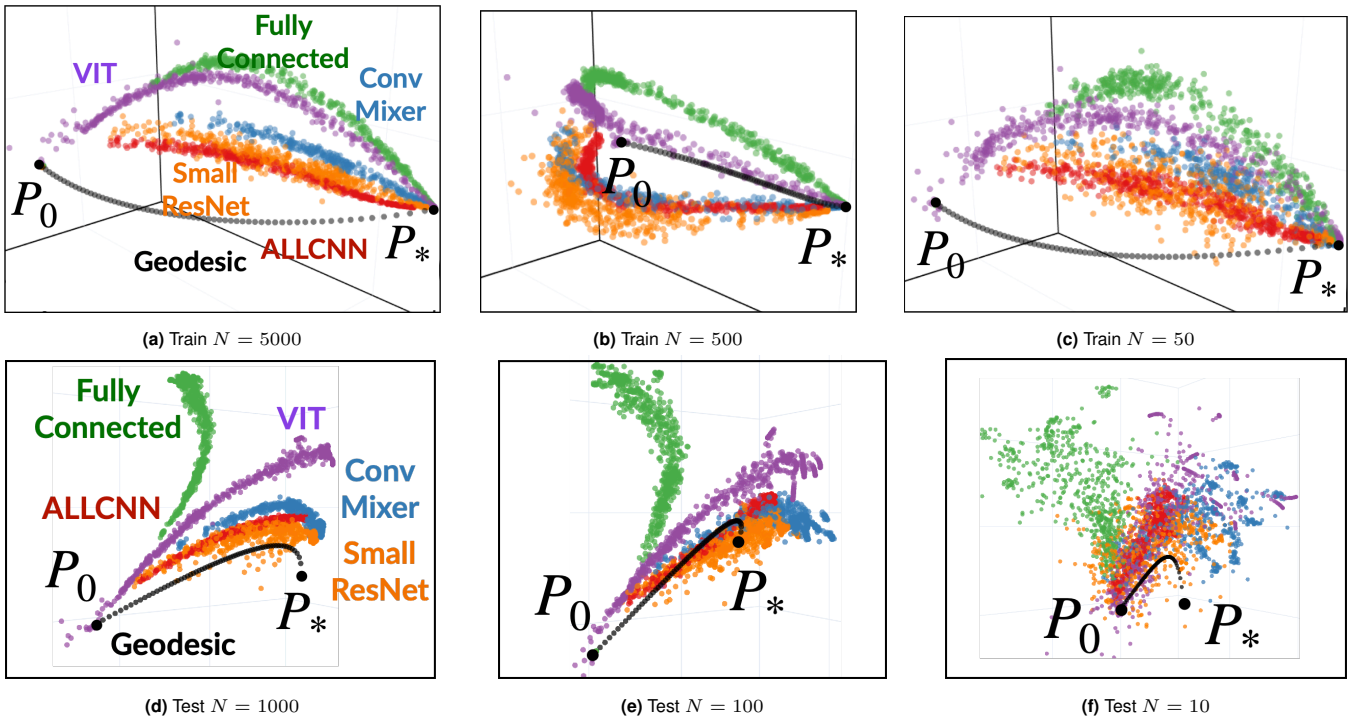
We present a different approach called weighted-InPCA next. Let $D \in \mathbb{R}^{m \times m}$ be the matrix of pairwise Bhattacharyya distances $D_{uv} = \mathrm{d_B}(P_u, P_v)$ and let $\mu_u \in \mathbb{N}$ be multiplicity of the model with weights u, i.e., the relative importance that we would like 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 $W\mathrm{diag}(\hat{\mu}_u)$, 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 $U_i / \sqrt{U^\top \mathrm{diag}(\hat{\mu}) U}$. 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 [9] when the weights μ_u of the new points are zero.

*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.



(a) Train manifold **(b)** Test manifold

Fig. S.3. The explained pairwise Bhattacharyya distances (computed using [12]) 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.



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

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

Fig. S.4. 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 (Figs. 2a and 5a). For the test data, explained pairwise distances is low in Fig. S.3b and manifolds are more diffuse.

S.4.4. 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 [9], we then embedded the models in the original pairwise distance matrix computed using all samples into these InPCA coordinates. Figs. S.3 and S.4 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.

S.5. Addendum to Results

S.5.1. 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. Fig. S.6a 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 Fig. 7a 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 Fig. 7a averages models for the same progress; Figs. S.6a and S.6b indicate that such averaging is a reasonable thing to do. The test manifold in Fig. S.6b 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.

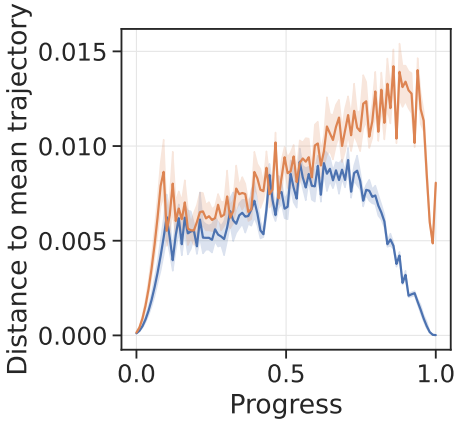


Fig. S.5. Towards the end of training at large values of progress, models trained with augmentation (orange) have larger tube widths than models trained without augmentation (blue), on the train manifold. The corresponding figure for the test manifold looks similar.

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 (Fig. S.5). 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 manifold (Fig. S.7a), 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 (Fig. S.7b), 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 Fig. 5.

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 $\{P_w(\vec{y}) : \forall w\}$ (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-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 (Figs. S.10a and S.11); the top three

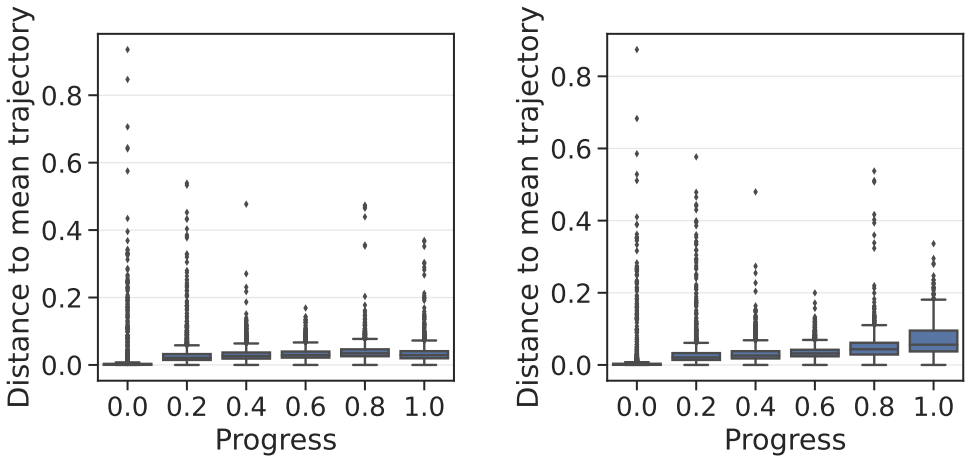


Fig. S.6. A boxplot (horizontal line denotes median, boxes denote 25 percentile, whiskers denote 1.5$\times$ the inter-quantile (25–75 percentile) range) of the Bhattacharyya distance between a model and the Euclidean mean of probabilities of models with the same configuration but obtained from different 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.

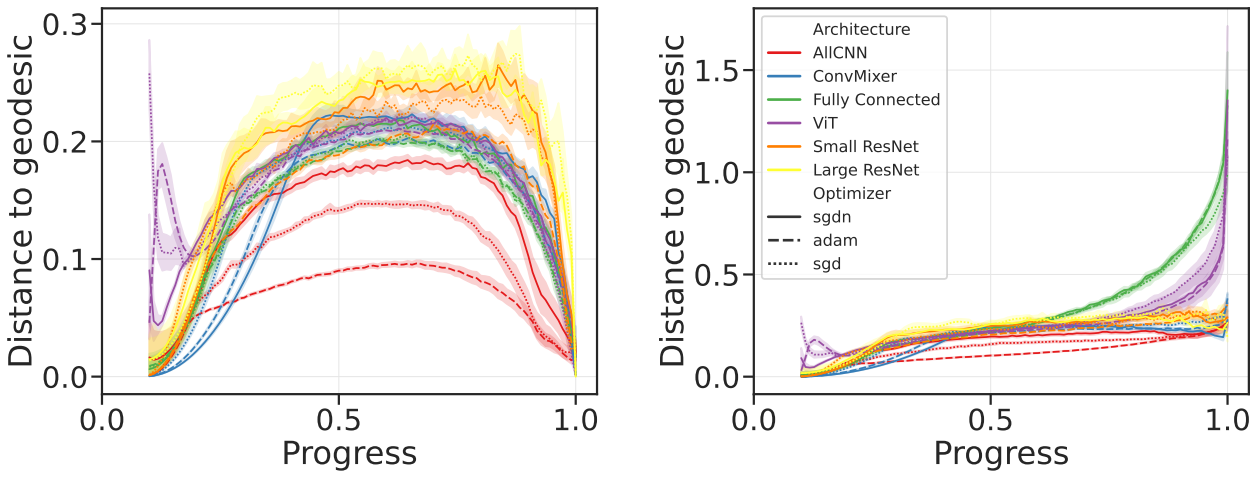


Fig. S.7. Bhattacharyya distance of models with different configurations to the geodesic at different progress for train **(a)** and test **(b)** trajectories.

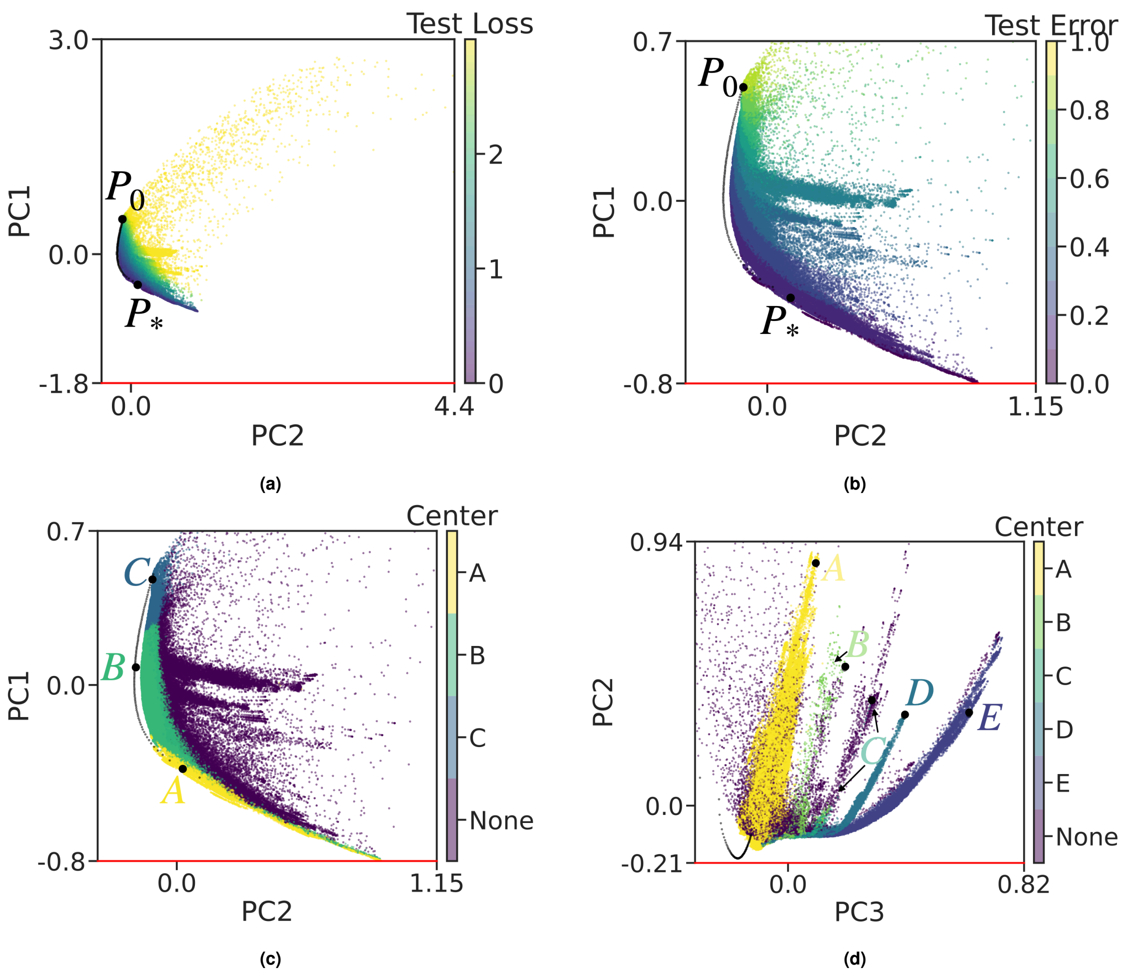


Fig. S.8. Comparison of two principal components of an InPCA embedding using test data of all models on CIFAR-10 colored by test loss **(a)**, by test error **(b)**, by whether they are within a Bhattacharyya distance < 0.3 from models marked A, B, and C on the geodesic in **(c)**, and by whether they are within a distance 0.45 from the models marked A–E in **(d)**. These figures should be studied together with Fig. 5c.

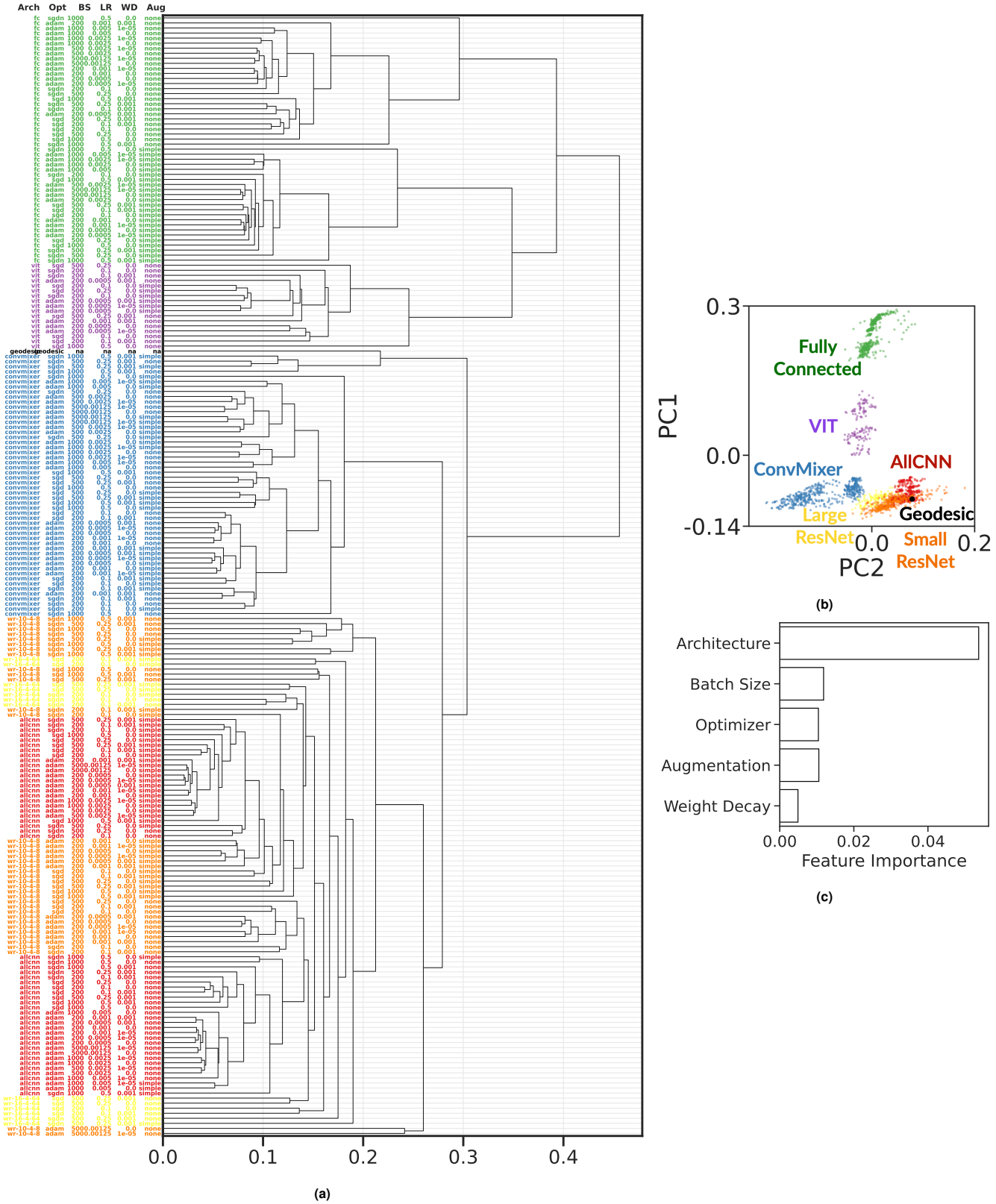


Fig. S.9. (a): dendrogram obtained from hierarchical clustering of pairwise distances (averaged over weight initializations) between trajectories using distances calculated on testing samples. X-labels correspond to architecture, optimization algorithm, batch-size, learning rate, weight-decay coefficient and augmentation strategy. Compared to the equivalent figure on training data Fig. 7a, 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.

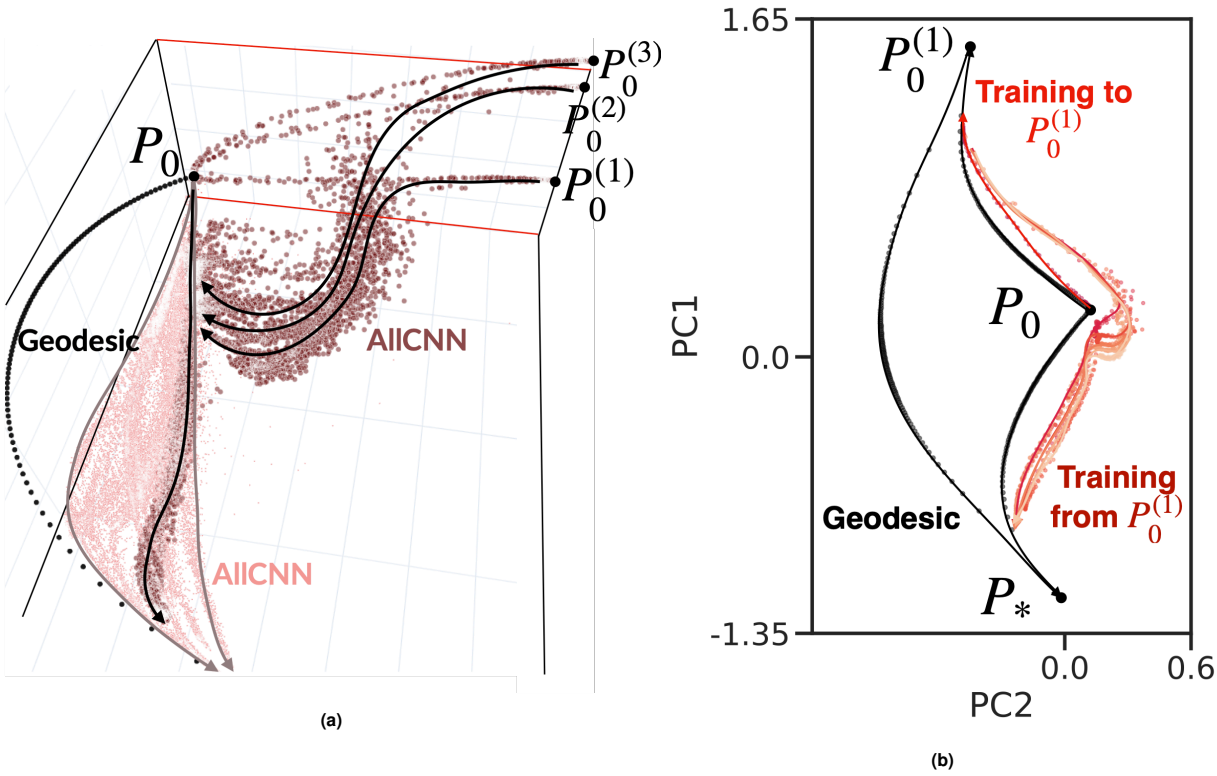


Fig. S.10. (a) shows the top three dimensions of an InPCA embedding of some configurations with AllCNN architectures when networks are initialized near ignorance and trained to truth P_* (light brown), and when they are first trained to tasks $P_0^{(k)}$ for $k = 1, 2, 3$ with random labels (stream of 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

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 Fig. S.10b 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 Fig. 7a. 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 (Fig. S.10b). 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 §S.5.2 are similar.

S.5.2. Further analysis of trajectories on the test data.

Dendrogram and InPCA embedding of test trajectories Fig. S.9 shows a dendrogram, similar to the one in Fig. 7a, obtained from hierarchical clustering of pairwise distances (averaged over weight initializations) between trajectories using distances calculated on the test samples. Fig. S.9b shows an InPCA embedding of the test trajectories and Fig. S.9c 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 Fig. 7 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 Fig. S.8a, 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 Fig. S.8c 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 Fig. S.8a; they consequently have high errors (90% in many cases, colored yellow in Fig. S.8b). We checked that these are the same models that are far from the train manifold near ignorance P_0 (yellow in Fig. 3b). Some (about half) of these models did not reach zero training error, and correspondingly they also have poor test error.

In Fig. S.8a, 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 Fig. S.8b), 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 Fig. S.8c) 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 Fig. S.8b) as compared to those on the train manifold in Fig. 3c. 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 Fig. S.8c). But fully-connected networks predict very differently from each other at intermediate errors (error of, say 0.3–0.4 in Fig. S.8b), 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 Fig. S.8d, 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 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 Fig. S.8b) some models lie on the manifold (e.g., green in Fig. S.8c), some on one branch (e.g., one of the purple branches or the smaller green branch in Fig. 5c), while some others can lie on other branches (e.g., other purple branches in Fig. 5c).

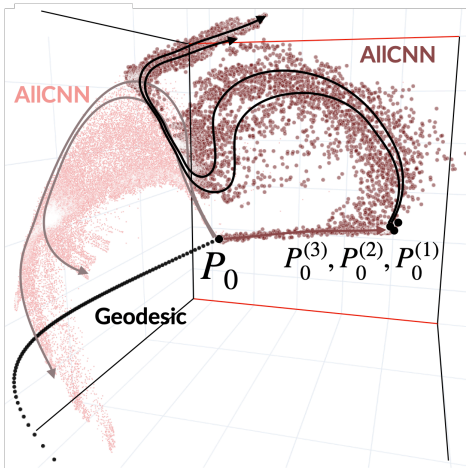


Fig. S.11. Predictions on test set of a subset of AllCNN models (the same set as in Fig. S.10) trained from ignorance (light brown) and from three different corners (dark brown). Networks trained from corners still seem to come close to the normally trained models mid-training, but they divert from the main manifold and end at a higher testing error in the later part of training.

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 end-

points are different from endpoints of trajectories that begin near ignorance P_0 (see Fig. S.11).

S.5.3. Observations remain consistent with other intensive distances. We can also use other distances in place of the Bhattacharyya distance. For example, the IsKL method (28) uses the symmetrized Kullback-Leibler (KL) divergence to compute the distances between pairs of points D in [6]

$$\mathrm{d_{sKL}}(P_u, P_v) = \frac{1}{N}\sum_{n=1}^{N}\sum_{c=1}^{C}(p_u^n(c) - p_v^n(c))\log\left(\frac{p_u^n(c)}{p_v^n(c)}\right). \quad [3]$$

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

$$\mathrm{d_G}(P_u, P_v) = \frac{1}{N}\sum_{n}\cos^{-1}\sum_{c}\sqrt{p_u^n(c)}\sqrt{p_v^n(c)}. \quad [4]$$

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.

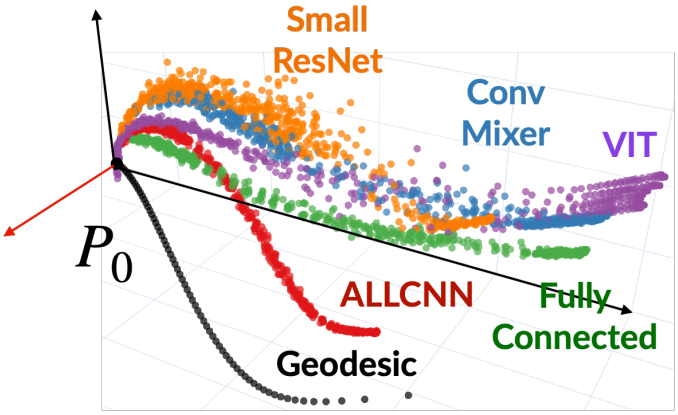


Fig. S.12. The top three dimensions of the IsKL embedding using the train data for a subset of the models trained on CIFAR-10 (this is the same subset as in Fig. 6a). The IsKL embedding carries a different kind of information than the InPCA embedding in Figs. 2a and 5a. Trajectories exhibit a larger 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.

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 Fig. S.13a and Fig. S.13c 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 Fig. 2a and Fig. 7a but there are marked differences in these trajectories in Fig. S.13a; (b) the geodesic is far from all trajectories in the original data but this is not so

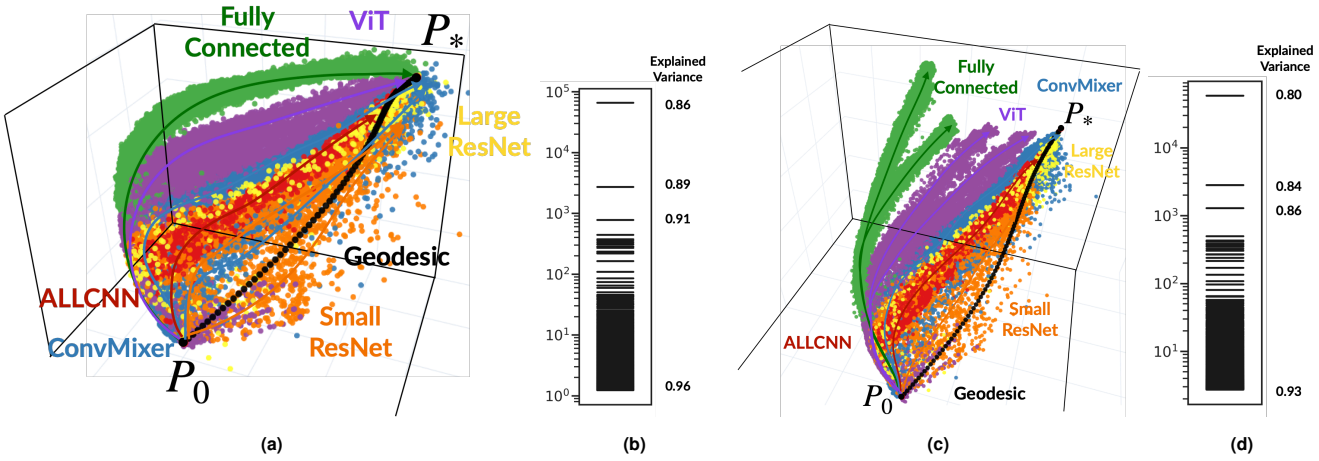


Fig. S.13. The top three dimensions of an embedding obtained using standard PCA for all the networks on CIFAR-10 using train data **(a)** and the test data **(c)**. The explained variance in **(b,d)** for train and test data respectively is very high but the structure of the low-dimensional manifold identified by PCA is very different from that obtained by InPCA in Figs. 2a and 5a. 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.

in the PCA embedding; (c) the cloud of points that lie away from the main manifold, which we have analyzed in Fig. 3, is not visible in the PCA embedding. For the test manifold, we see some similarities between Figs. 5a and S.13c: (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 (Fig. S.7) 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.

S.5.4. 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 S.1 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 (29), and (v) Jeffrey's centroid which corresponds to the symmetric KL-divergence which is known in closed-form (30). In Fig. S.14, 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 Fig. S.14b. 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.

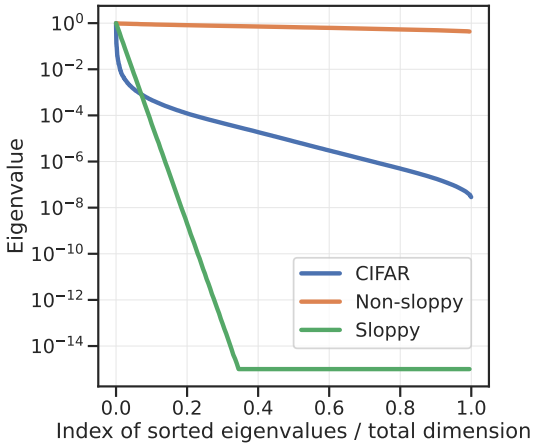


Fig. S.15. Eigenvalues of the the input correlation matrix $\mathbb{E}[xx^\top]$ for 32×32 RGB images x in CIFAR-10 (blue), i.e., $x \in \mathbb{R}^{3072}$, non-sloppy synthetic inputs ($x \in \mathbb{R}^{200}$) sampled from an isotropic zero-mean Gaussian (orange) and sloppy synthetic inputs ($x \in \mathbb{R}^{200}$) 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).

S.6. Experiments using synthetic data

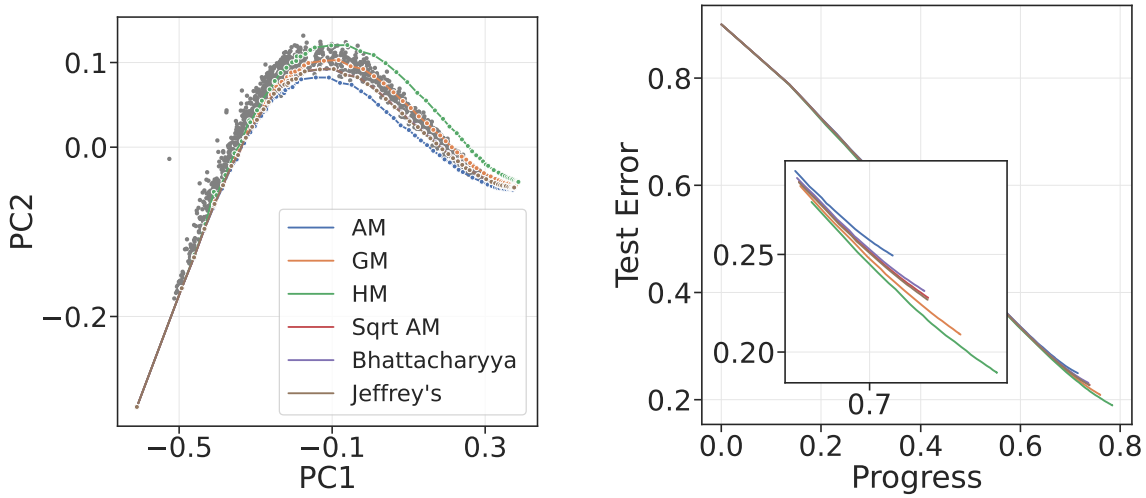
Datasets We sampled $N = 5000$ samples for the training set and $N = 1000$ samples for the test set from a $d = 200$ dimensional Gaussian with mean zero and a diagonal covariance $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_d)$. We experimented with two types of data: those sampled from an isotropic 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 (31, 32). 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. Fig. S.16a 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 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

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 (29)
$\sum_n (p_n - q_n)\log(p_n/q_n)$	AM/W (eAM/GM)	Jeffrey's centroid (30)

Table S.1. 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)

Fig. S.14. (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.

configurations. The explained stress on test data is essentially the same. As the embeddings in Fig. S.16 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 Fig. S.16b 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.

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 Figs. S.17a and S.17b 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 from different corners look very similar in Fig. S.17c for such data. For non-sloppy data, even if the explained stress is lower in the top three dimensions (the InPCA embedding in Fig. S.17c 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 Fig. S.17c 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 Fig. S.18a. 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 Fig. S.18b. So it is remarkable that in Fig. S.17, 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.

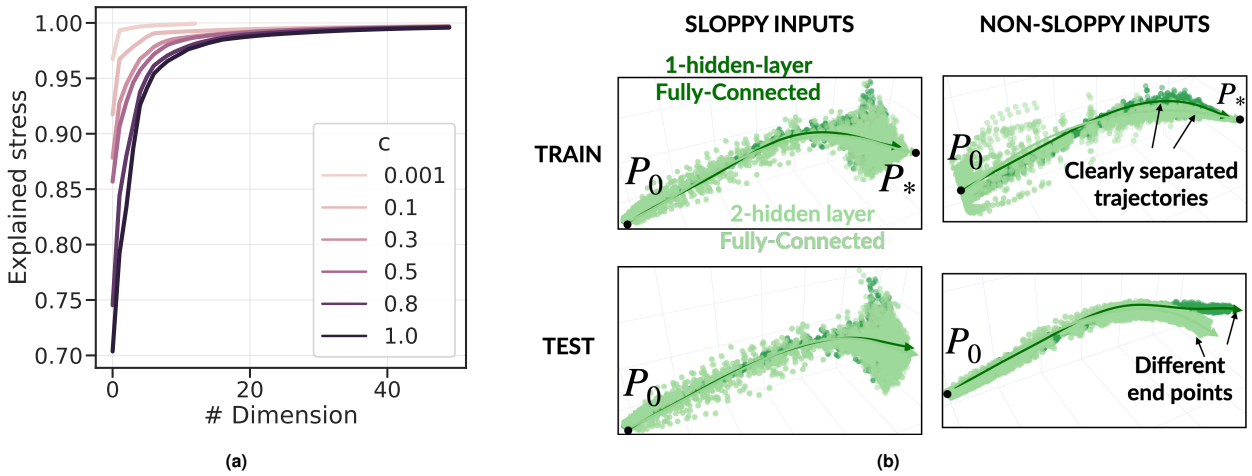
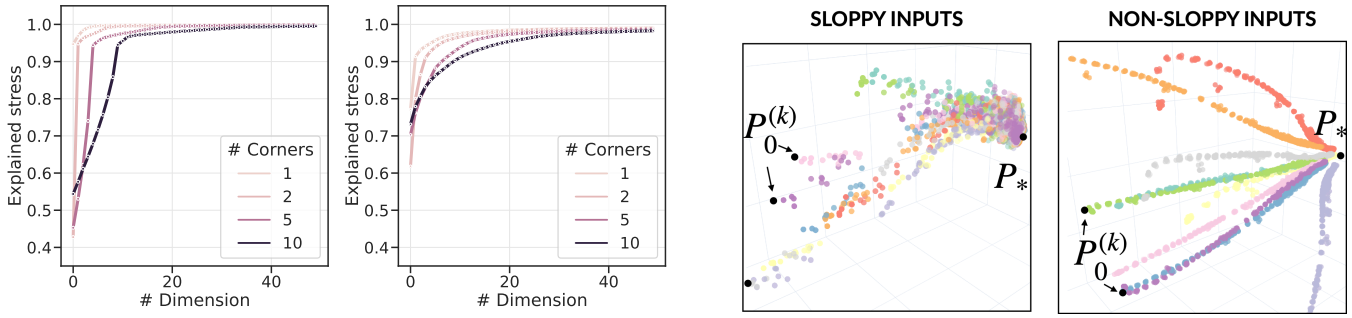


Fig. S.16. (a): the explained stress in the top few dimensions (X-axis) of an InPCA embedding of models along training trajectories when input data are sampled from a Gaussian distributions with zero mean and covariance matrix whose eigenvalues decay as $\lambda_i = 50c\exp(-ci)$ for different values 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.



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

Fig. S.17. (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.

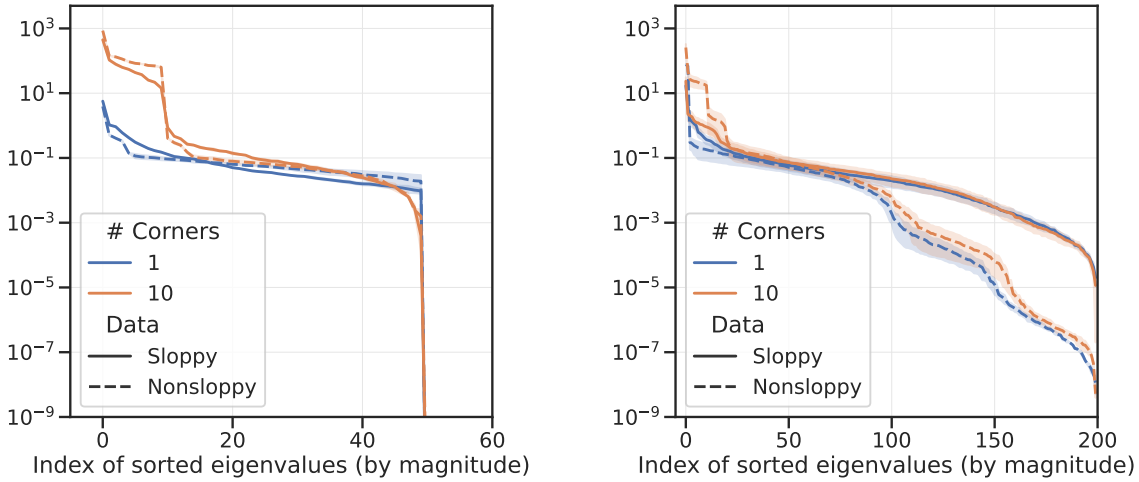


Fig. S.18. (a,b): eigenvalues of the pairwise distance matrix D (see [6]) of InPCA of models trained from corners at the beginning of training and after 0.5% of the training epochs, respectively. Our goal was to slice the tube of trajectories of networks with different weight initializations corresponding 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 Fig. S.17 are effectively low-dimensional.

References

1. Christopher M Bishop et al. *Neural Networks for Pattern Recognition*. 1995.
2. 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.
3. 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.
4. Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for Simplicity: The All Convolutional Net. *arXiv:1412.6806 [cs]*, April 2015.
5. Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *British Machine Vision Conference 2016*, 2016.
6. Asher Trockman and J Zico Kolter. Patches are all you need? *arXiv preprint arXiv:2201.09792*, 2022.
7. 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.
8. 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.
9. 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.
10. Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer Normalization. *arXiv:1607.06450 [cs, stat]*, July 2016.
11. 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.
12. Richard Zhang. Making convolutional networks shift-invariant again. In *International conference on machine learning*, pages 7324–7334. PMLR, 2019.
13. 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.
14. Guillaume Leclerc, Andrew Ilyas, Logan Engstrom, Sung Min Park, Hadi Salman, and Aleksander Madry. ffcv. https://github.com/libffcv/ffcv/, 2022.
15. Ross Wightman. Pytorch image models. https://github.com/rwightman/pytorch-image-models, 2019.
16. Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference for Learning Representations*, 2015.
17. 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.
18. 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.
19. 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.
20. 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.
21. Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017.
22. 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.
23. Sebastian Thrun and Lorien Pratt. *Learning to Learn*. 2012.
24. 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.
25. 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.
26. 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.
27. Ludovic Delchambre. Weighted principal component analysis: a weighted covariance eigendecomposition approach. *Monthly Notices of the Royal Astronomical Society*, 446(4):3545–3555, 2015.
28. 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.
29. Frank Nielsen and Sylvain Boltz. The burbea-rao and bhattacharyya centroids. *IEEE Transactions on Information Theory*, 57(8):5455–5466, 2011.
30. 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.
31. 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.
32. 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 2011. ISSN 1539-3755, 1550-2376.