Walking the Weight Manifold: a Topological Approach to Conditioning Inspired by Neuromodulation

Ari S. Benjamin* **Kyle Daruwalla*** **Christian Pehle*** **Anthony Zador**
Cold Spring Harbor Laboratory
Cold Spring Harbor, NY 11724
{benjami,daruwal,pehle,zador}@cshl.edu

Abstract

One frequently wishes to learn a range of similar tasks as efficiently as possible, re-using knowledge across tasks. In artificial neural networks, this is typically accomplished by *conditioning* a network upon task context by injecting context as input. Brains have a different strategy: the parameters themselves are *modulated* as a function of various neuromodulators such as serotonin. Here, we take inspiration from neuromodulation and propose to learn weights which are smoothly parameterized functions of task context variables. Rather than optimize a weight vector, i.e. a single point in weight space, we optimize a smooth manifold in weight space with a predefined topology. To accomplish this, we derive a formal treatment of optimization of manifolds as the minimization of a loss functional subject to a constraint on volumetric movement, analogous to gradient descent. During inference, conditioning selects a single point on this manifold which serves as the effective weight matrix for a particular sub-task. This strategy for conditioning has two main advantages. First, the topology of the manifold (whether a line, circle, or torus) is a convenient lever for inductive biases about the relationship between tasks. Second, learning in one state smoothly affects the entire manifold, encouraging generalization across states. To verify this, we train manifolds with several topologies, including straight lines in weight space (for conditioning on e.g. noise level in input data) and ellipses (for rotated images). Despite their simplicity, these parameterizations outperform conditioning identical networks by input concatenation and better generalize to out-of-distribution samples. These results suggest that modulating weights over low-dimensional manifolds offers a principled and effective alternative to traditional conditioning.

1 Introduction

Conditioning is essential in the neural network toolbox. For example, an image generator might be conditioned on which category of image to generate or with which style. A running robot might be conditioned on a desired running speed or its goal location. In each of these examples, the sub-tasks are so closely related—relative to all possible functions—that it is more efficient to use a single neural network, provided that one can ensure that learning in one context appropriately generalizes to learning in other contexts.

One way to encourage cross-task generalization would be to take advantage of the clear relationships between these tasks. Here, we pay particular attention to the *topology* of sub-tasks. Running speed is a 1D line through the task space of ambulating. Navigation on a map, conditioned on location, is a 2D operation over a sheet. Classifying images of 3D objects taken from arbitrary orientations lies on the surface of a sphere in task space. An inductive bias which captures these topologies would

*Denotes equal contribution, alphabetical order

Preprint.

ensure that knowledge transfers between tasks while enforcing that the true task topology constrains the possible learned input/output mappings.

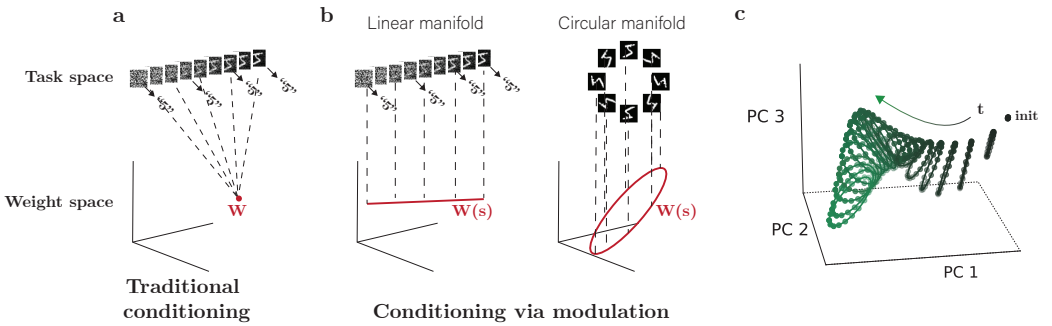

Figure 1: **a.** Traditional approaches map all conditions in task space to a single network in weight space. Here, conditioning corresponds to the noise added to an image (i.e. input uncertainty) in a classification task. **b.** Our approach maps various conditions to a parameterized manifold in weight space with known topology chosen to match the task topology. For the task in **a**, this corresponds to a line, but for alternative tasks like rotations of an input image, an ellipse is more appropriate. **c.** Manifolds are optimized via our proposed steepest descent rule such that they minimize the volumetric distance between steps. Here, we illustrate this by projecting an ellipse manifold of weights into principal component space during learning. The weights correspond to the first convolutional layer of a network trained to classify CIFAR-10 images.

Here, we realize this strategy by hypothesizing that for each manifold of tasks there exists a manifold of corresponding topology in weight space which implements that range of tasks. For example, a topological sphere in weight space likely implements the set of functions for classifying objects rotated in 3D space. More formally, assuming smoothness over the weight and task manifolds, and a topology over the task manifold, our "manifold hypothesis" postulates a homeomorphism between both manifolds. Fig. 1a and b illustrate this hypothesis and how it differs from a traditional view on neural network weights and conditioning.

To empirically evaluate this hypothesis, we formalize this notion and derive a learning rule for optimizing manifolds. This rule is analogous to gradient descent, whose implicit learning biases are arguably the key to success in high-dimensional optimization. Just as gradient descent effectively minimizes the path length through weight space, our rule pushes a manifold in the direction of steepest descent while minimizing its total volumetric movement through weight space. This ensures that learning pressure at one location of the manifold does not push other locations on the manifold in very far directions. This learning rule is designed to respect Occam's razor, minimally changing the function across the manifold in response to new data anywhere.

While it might not be obvious that lines, circles, or other simple shapes exist in weight space that implement realistic functions, we find empirically that such solutions do indeed exist. Fig. 1c demonstrates how the rule manipulates the structure of the manifold while taking minimal volumetric steps through weight space (shown here for an image classification network's first layer weights when trained on CIFAR-10 [16]). Furthermore, we demonstrate a potential benefit of conditioning via modulation: generalization to contexts not seen during training. Just as humans must learn while calm but perform while nervous or angry, the modulation of artificial networks allows them to generalize performance to areas in which the involved task slightly differs.

In brief, our major contributions include:

- a neuro-inspired formalism for topologically constrained weight manifolds
- a corresponding steepest descent rule for updating manifolds in principled manner
- computationally tractable instances of our rule for simple manifolds (e.g. a line, ellipse, etc.)
- practical implementations of our rule that leverage existing automatic differentiation libraries
- experimental evaluations demonstrating when the proposed rule is effective and ineffective; notably, that it generalizes to novel conditions unseen in the training data relative to vanilla gradient descent and traditional conditioning

1.1 Inspiration from neuroscience

The proposed abstraction of weight manifolds reflects a wide set of studies on the physiological properties of neurons and small circuits. After it was observed in the early 20th century that neurons integrate information from their synaptic inputs—directly leading to the earliest generations of neural network models [20]—it later become clear that nearly all key parameters of neural systems are, in fact, functions of various neuromodulators [12]. While this work vastly complicated the modeling of small biological circuits, it revealed a general biological capability for reusing neural circuits for different purposes in different behavioral states [19].

The mechanistic effects of neuromodulation are profound and diverse. The excitability of neurons are affected, as well as specific synapses and specific ion channels [15, 17]. Effects vary depending upon the cell type in question, sometimes in opposite directions for the same neuromodulator [10]. Furthermore, the delivery of neuromodulation can be extremely targeted to single neurons (i.e. cotransmission) [22]. Together these endow neuromodulation with the potential for strongly affecting the behavior of a single neural network at all functional scales.

In light of this astounding diversity, we aim to abstract only a very general principal of operation. Notably, we are not attempting to model particular functions of various neuromodulators in different circuits. Instead, we ask what advantage this general capability provides to artificial systems.

1.2 Modulating a network $\subset$ conditioning a network

As a note of clarity, conditioning information may be either **high-dimensional** (for example, CLIP embeddings to generate images from natural language descriptions) or **low-dimensional** (for example, desired running speed of a robot). Here, we are solely interested in the low-dimensional case, which we believe admits special treatment. To distinguish low-dimensional conditioning cases, we will refer to this as *modulating* a neural network.

Low-dimensional modulations of a network are quite common. A helpful example can be found in generative modeling via diffusion (DDPMs) where the denoiser is conditioned on the magnitude of injected noise, or more specifically, the timestep of the diffusion process [13]. This allows a single denoising neural network to denoise both low-noise and high-noise examples, but transfer knowledge between noise regimes.

2 Optimizing a weight manifold: a formal treatment

Neural networks are typically optimized as a single point in weight space; however, our approach requires optimizing an entire manifold of weights simultaneously. This section formalizes this approach while providing intuition for the underlying concepts.

2.1 Weights as parameterized functions

In traditional neural networks, each weight (synapse) is described by a single value. Here, we make each weight a function of the conditioning variable. Formally, consider a smooth manifold $\mathcal{M}(s, \mathbf{P})$ parametrized by $s \in [0, 1]$ and depending on parameters $\mathbf{P} \in \mathbb{R}^p$. The parameter s represents the **modulator** or conditioning value, and the parameters $\mathbf{P}$ are learnable. Then, $\mathcal{M} : \mathbb{R} \times \mathbb{R}^p \to \mathbb{R}^d$ is map from points on the manifold, selected by s, to the weights of the network.

Example: A simple manifold is a straight line segment through weight space. Just as a line can be uniquely defined by its two endpoints, we can parametrize this manifold as a linear interpolation:

$$\mathcal{M}(s, \mathbf{P}) = (1 - s)\mathbf{P}_1 + s\mathbf{P}_2 \tag{1}$$

where $\mathbf{P} = (\mathbf{P}_1, \mathbf{P}_2)$ contains the endpoints. When $s = 0$, we get the weights $\mathbf{P}_1$; when $s = 1$, we get $\mathbf{P}_2$; and for values in between, we get a smooth interpolation along the line.

2.2 Optimization

Unlike standard neural network training that minimizes a loss at a single weight configuration, we need to minimize a loss **functional** $L[\mathcal{M}]$ that evaluates the entire manifold:

$$L[\mathcal{M}] = \int_0^1 \ell(\mathcal{M}(s, \mathbf{P}))\, \mathrm{d}s \tag{2}$$

where $\ell : \mathbb{R}^d \to \mathbb{R}$ is a function measuring the loss at each point along the manifold.

Intuition: We're essentially averaging the performance across all possible conditioning values. This ensures that our network performs well across the entire conditioning spectrum, not just at isolated points.

2.2.1 The variational problem

At each optimization step, we need to find the best way to update our parameters $\mathbf{P}$. This means finding the optimal perturbation, $\Delta\mathbf{P}$, that moves the entire manifold in a beneficial direction.

$$\begin{aligned} \Delta\mathbf{P} = \arg\min_{\Delta\mathbf{P}} \quad & L[\mathcal{M}(s, \mathbf{P} + \Delta\mathbf{P})] \\ \text{such that} \quad & d^2(\mathcal{M}(s, \mathbf{P} + \Delta\mathbf{P}), \mathcal{M}(s, \mathbf{P})) = C_0 \end{aligned} \tag{3}$$

where d^2 denotes the total squared distance over the manifold (i.e. the Euclidean distance at every point integrated over s). The distance constraint ensures that the manifold does not move too far in a single step.

Key insight: Just as gradient descent minimizes distance traveled through weight space, our approach minimizes the **volumetric movement** of the entire manifold. This ensures that learning at one point on the manifold does not cause excessive changes elsewhere.

2.2.2 Solving for $\Delta\mathbf{P}$

First, we note that small perturbations affect our manifold approximately linearly:

$$\mathcal{M}(s, \mathbf{P} + \Delta\mathbf{P}) \approx \mathcal{M}(s, \mathbf{P}) + \frac{\partial \mathcal{M}}{\partial \mathbf{P}} \Delta\mathbf{P} \tag{4}$$

Thus, the perturbation $\Delta\mathbf{P}$ affects the manifold dependent upon the Jacobian of the parameterization, $\frac{\partial \mathcal{M}}{\partial \mathbf{P}}$. To express the optimal update, it is helpful to introduce the following notation:

$$\mathbf{M}(s) = \left(\frac{\partial \mathcal{M}}{\partial \mathbf{P}}\right)^T \frac{\partial \mathcal{M}}{\partial \mathbf{P}} \qquad \text{(local metric tensor)} \tag{5}$$

$$\mathbf{g}(s) = \left(\frac{\partial \mathcal{M}}{\partial \mathbf{P}}\right)^T \nabla \ell(\mathcal{M}(s)) \qquad \text{(local gradient w.r.t. } \mathbf{P}\text{)} \tag{6}$$

The metric tensor $\mathbf{M}(s)$ captures how parameter changes affect the manifold at each point s, while $\mathbf{g}(s)$ represents the direction of steepest descent of the loss at each point.

Through Lagrangian optimization on Eq. 4 (detailed derivation in the supplemental information), we arrive at the optimal parameter update:

$$\Delta\mathbf{P} = -\frac{1}{2\lambda} \left[\int_0^1 \mathbf{M}(s)\, \mathrm{d}s\right]^{-1} \int_0^1 \mathbf{g}(s)\, \mathrm{d}s \tag{7}$$

This update has an intuitive interpretation: we're averaging gradients across the entire manifold and then applying a correction based on the manifold's geometry.

In practice, the integral over the gradients $\int_0^1 \mathbf{g}(s)\ \mathrm{d}s$ can be computed via sampling without modifications to stochastic gradient descent frameworks. The term in brackets, $\left[\int_0^1 \mathbf{M}(s)\ \mathrm{d}s\right]^{-1}$, is the inverse of the integrated metric tensor, which we will denote this as $\overline{\mathbf{M}}^{-\mathbf{1}}$. For general manifolds, $\overline{\mathbf{M}}^{-\mathbf{1}}$ may be challenging to calculate as it represents the inverse of a very large matrix (with as many rows as network weights). Luckily, this term is analytically computable in several special cases that are relevant for practice, such as lines, ellipses, or any parameterized manifold expressible as a weighted sum of certain basis points.

2.2.3 Manifolds with analytic $\overline{\mathbf{M}}^{-\mathbf{1}}$

For many manifold types, we can analytically determine the inverse integrated metric tensor $\overline{\mathbf{M}}^{-\mathbf{1}} = \left[\int_0^1 \mathbf{M}(s)\ \mathrm{d}s\right]^{-1}$ and its matrix-vector product. This allows for efficient optimization without needing to compute and invert large matrices.

Example: Straight Line Manifold To illustrate this, let's consider again the straight line manifold parametrized as a linear interpolation between two points $\mathcal{M}(s, \mathbf{P}) = (1-s)\mathbf{P}_1 + s\mathbf{P}_2$.

To compute the metric tensor at any point s, we need the Jacobian:

$$\frac{\partial \mathcal{M}}{\partial \mathbf{P}} = [(1-s)\mathbf{I} \quad s\mathbf{I}] \tag{8}$$

where I is the identity matrix with the same dimension as all network parameters, flattened. The metric tensor is then:

$$\mathbf{M}(s) = \left(\frac{\partial \mathcal{M}}{\partial \mathbf{P}}\right)^T \frac{\partial \mathcal{M}}{\partial \mathbf{P}} = \begin{bmatrix} (1-s)^2\mathbf{I} & s(1-s)\mathbf{I} \\ s(1-s)\mathbf{I} & s^2\mathbf{I} \end{bmatrix} \tag{9}$$

The inverse of this matrix after integrating over s from 0 to 1 is our desired $\overline{\mathbf{M}}^{-\mathbf{1}}$:

$$\overline{\mathbf{M}}^{-\mathbf{1}} = \left[\int_0^1 \mathbf{M}(s)\ \mathrm{d}s\right]^{-1} = \begin{bmatrix} \frac{1}{3}\mathbf{I} & \frac{1}{6}\mathbf{I} \\ \frac{1}{6}\mathbf{I} & \frac{1}{3}\mathbf{I} \end{bmatrix}^{-1} = \begin{bmatrix} 4\mathbf{I} & -2\mathbf{I} \\ -2\mathbf{I} & 4\mathbf{I} \end{bmatrix} \tag{10}$$

This may seem like too a large matrix to hold in memory; however, it never needs to be instantiated—all that is required to calculate ΔP is its matrix-vector product with the integrated local gradient. This is simple to evaluate. Plugging into Equation 7, we obtain:

$$\Delta \mathbf{P} = -\frac{1}{2\lambda} \begin{bmatrix} 4\nabla_{\mathbf{P}_1}\ell(\mathcal{M}(s)) - 2\nabla_{\mathbf{P}_2}\ell(\mathcal{M}(s)) \\ -2\nabla_{\mathbf{P}_1}\ell(\mathcal{M}(s)) + 4\nabla_{\mathbf{P}_2}\ell(\mathcal{M}(s)) \end{bmatrix} \tag{11}$$

Thus, this method is easy to implement and consists only of fast linear combinations of gradients computed through automatic differentiation. By comparison, if we had not considered the metric penalty, the update rule would simply be the gradient $[\nabla_{\mathbf{P}_1} l(\mathcal{M}(s)) \quad \nabla_{\mathbf{P}_2} l(\mathcal{M}(s))]^T$, without any mixing of the gradients with respect to each endpoint.

Summary of Analytical Cases: Many useful manifold parameterizations have closed-form expressions for $\overline{\mathbf{M}}^{-\mathbf{1}}$, making them computationally efficient. Table 1 summarizes key examples.

2.3 Practical Implementation: Efficient Optimization of Weight Manifolds

During learning and inference, one sees a batch of examples with varying levels of conditioning, $\{(\mathbf{x}_i, s_i)\}_i^B$. The challenge is to obtain the output for each example, noting that each different values of s correspond to the outputs of effectively different neural network with weights $\mathcal{M}(s, \mathbf{P})$. It would be inefficient to instantiate each neural network separately to process each example, as the memory requirements would scale linearly with the batch size, B.

Table 1: Analytical forms of $\overline{\mathbf{M}}^{-1}$ for common manifold types

Manifold Type	**Parameterization**	$\overline{\mathbf{M}}^{-1}$
Straight Line	$(1-s)\mathbf{P}_1 + s\mathbf{P}_2$	$\begin{bmatrix} 4\mathbf{I} & -2\mathbf{I} \\ -2\mathbf{I} & 4\mathbf{I} \end{bmatrix}$
Ellipse	$\mathbf{P}_1 + \mathbf{P}_2\cos(2\pi s) + \mathbf{P}_3\sin(2\pi s)$	$\begin{bmatrix} \mathbf{I} & 0 & 0 \\ 0 & 2\mathbf{I} & 0 \\ 0 & 0 & 2\mathbf{I} \end{bmatrix}$
Tethered Rod	$(1-s)\mathbf{P}_1 + s\mathbf{P}_2$	$\begin{bmatrix} 0 & 0 \\ 0 & 3\mathbf{I} \end{bmatrix}$
Cubic B-Splines	$\sum_{i=0}^{n} N_{i,k}(s)\mathbf{P}_i$	$\mathbf{D}^{-1}$ where $\mathbf{D}$ is banded diagonal; $\frac{1}{60n}[120, 78, 24, 3]$ along diagonals

The key insight of our implementation is that for all of the manifolds shown in Table 1, the weight vector at a specific point on the manifold, $\mathcal{M}(s, \mathbf{P})$, is a **linear combination of n "basis points" $\mathbf{P}_i$**. Each basis point defines a particular instance of the network.

$$\mathcal{M}(s, \mathbf{P}) = \sum_{i=0}^{n} a_i(s)\mathbf{P}_i \tag{12}$$

Conveniently, most learnable operations in neural networks (e.g. fully-connected layers, convolutions, embeddings, sub-operations of self-attention, etc.) are also linear in that they are additive and homogenous—$f(\alpha a + \beta b) = \alpha f(a) + \beta f(b)$. Nonlinear operations such as ReLU generally do not contain any learnable components. This property along with Eq. 12 allows us to process all B examples in a batch while only ever holding the n basis points in memory.

The reason for this is that within each layer, the effective s-dependent weight matrix $\mathbf{W}(s)$ never needs to be instantiated to obtain the matrix-vector product $\mathbf{W}(s)\mathbf{x}$. Instead, one can apply each basis point separately, then linearly combine them:

$$\mathbf{W}(s)\mathbf{x} = a_1(s)\mathbf{W}_1\mathbf{x} + a_2(s)\mathbf{W}_2\mathbf{x} + \ldots \tag{13}$$

Often, n is small enough that each term in the linear combination can be computed in parallel. This procedure can then be extended layer-by-layer to cover an entire neural network. Thus, with minimal memory overhead and no runtime overheads, we can compute a full batch of examples over the manifold despite $B \gg n$.

Algorithm 1 (in appendix) provides a high-level overview of our approach.

3 Empirical results

In evaluating our proposed approach, we must consider when our neuromodulation-inspired abstraction is useful for training artificial networks. To this end, we study two settings where topological constraints on network weights are effective but also where they are ineffective. First, Sec. 3.1 demonstrates the generalization ability of our approach to unseen data augmentation conditions when the augmentation topology is known. Surprisingly, this occurs even for simple, rigid manifolds, experimentally proving our "manifold hypothesis." Second, Sec. 3.2 attempts to leverage manifolds to regularize networks in the face of uncertainty where the mapping from conditioning value to task manifold is more complex. *Our aim in these experiments goes beyond validating the correctness of our approach; we hope to highlight both the use cases and pitfalls of using modulation as a mechanism to address a learning challenge.* Please refer to the appendix for complete details on reproducing these results.

3.1 Generalization to unseen conditions

Here, we design a setting where data augmentation is applied the inputs to the network, and the conditioning describes the augmentation that was applied. Specifically, we rotate CIFAR-10 images

passed to a moderately sized convolutional network (CNN) and the conditioning, s, encodes the angle of the rotation from $[0, 2\pi]$. The network must classify these images despite the rotation. To test generalization, we only sample a fixed subset of the possible angles during training (e.g. 10% of $[0, 2\pi]$). During evaluation, we sample the full range and report the test accuracy as a function of the sparsity of the training conditions. Fig. 2a illustrates the overall setting.

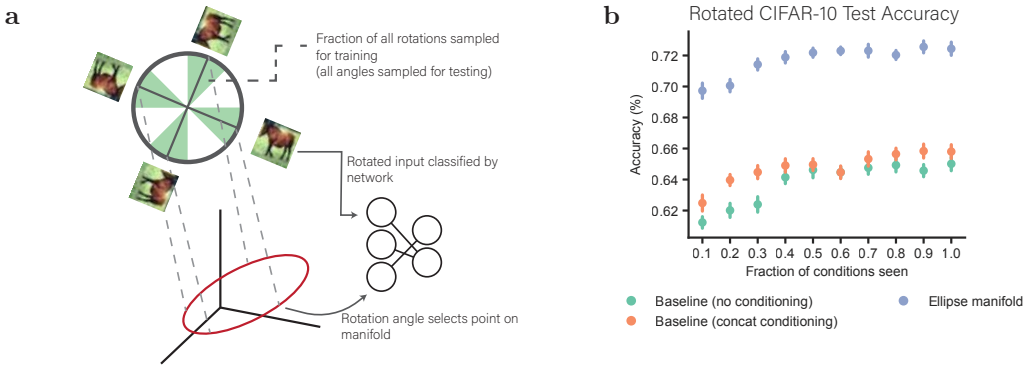

Figure 2: **a.** An illustration of the training paradigm used to test the generalization abilities of our manifold approach. We sample a sparse subset of the possible conditions (rotation angles of the input) during training and test on the full set of conditions. **b.** Performance of the ellipse manifold network on the test set vs. a baseline network with and without conditioning.

As shown in Fig. 2b, training an ellipse manifold generalizes to unseen angles even at extremely sparse sampling of conditions during training. Furthermore, the performance of the manifold network surpasses conventional networks with and without conditioning input. We also note that the baselines lose performance as the training conditions become sparser, while the manifold networks maintains its peak performance for a wider range of sampling sparsities.

We emphasize that these results are non-trivial. They demonstrate, empirically, that the "manifold hypothesis" is true, and that we can match the appropriate homeomorphism between the task and weight manifolds with even a simple parametrized ellipse. The manifold used here is simple in that it is rigid—it can rotate and stretch in high dimensional weight space, but it cannot bend and must lie in a plane. This is an extremely strong constraint on space the weights can occupy while still performing the task correctly and generalizing. Yet, when the topological structure in the data is clear (such as rotation invariance of images), modulating weights with even a simple mechanism is sufficient to exploit this structure.

3.2 Controllable regularization of networks

While known data transformations such as augmentation have clearly definable topologies that can be exploited, this is not the only form of conditioning that our approach could target. An unknown data transformation that must frequently be dealt with is input noise, and models must robustly perform in the face of this uncertainty. Regularization of a networks weights is a common attempt for dealing with noisy data. Traditionally, a single network is regularized by a fixed amount which may be overly conservative when the noise varies from sample to sample.

Here, we study a setting where a network classifies images from CIFAR-10 with additive Gaussian noise. Specifically, for each example, we sample a noise level $s \sim \text{Unif}(0, S)$ where S denotes the maximum noise level. Then, we augment the input image as $\hat{\mathbf{x}} = (1 - s)\mathbf{x} + s\eta$ where $\eta \sim \mathcal{N}(0, \mathbf{I})$. Thus, s reflects the uncertainty of the example. We train a CNN whose weights are constrained to a line manifold with L_2-regularization. For a given sample, $(\mathbf{x}_i, \mathbf{y}_i, s_i)$, the loss of that sample is

$$\ell_{\mathcal{M}}(\mathbf{x}_i, \mathbf{y}_i, s_i; \mathcal{M}(s_i, \mathbf{P})) = \ell(\mathbf{x}_i, \mathbf{y}_i; \mathcal{M}(s_i, \mathbf{P})) + s_i \lambda \|\mathcal{M}(s_i, \mathbf{P})\|_2^2 \tag{14}$$

where ℓ is the original loss (e.g. cross entropy), $\mathcal{M}(s_i, \mathbf{P})$ are the network weights used for the i-th sample, and λ is a regularization coefficient. Thus, our uncertainty conditioning controls the level of regularization applied to each example.

Fig. 3 shows the test performance on CIFAR-10 (which is also noised in the same manner). While the manifold network does slightly outperform the baseline cases, its advantage is minimal. We

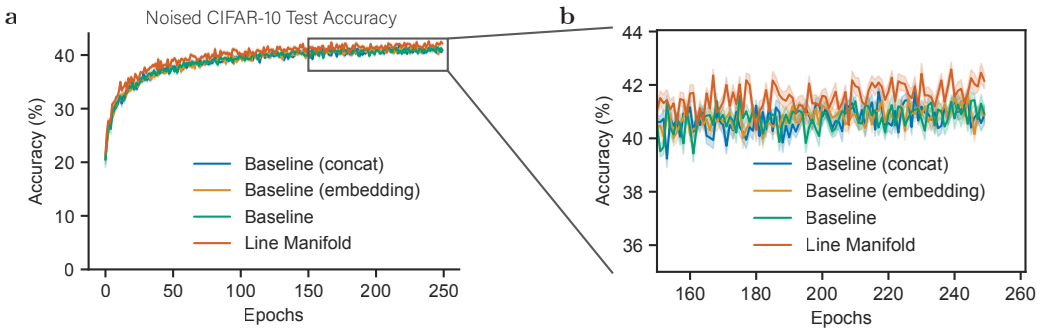

Figure 3: **a.** Noised CIFAR-10 test accuracies for baseline networks and the line manifold. Noise procedure is described in Sec. 3.2. **b.** A zoomed-in view of **a**.

hypothesize that this is because of the mismatch between our specification of uncertainty and the true uncertainty manifold. Namely, only the few examples closed to the learned decision boundary are relevant to the model's uncertainty, and thus, our chosen manifold spans a much wider range of the task space than the true manifold. These results illustrate cases where modulation is not useful—when the topology is difficult to define or properly infer and relate to the conditioning value.

4 Related work

Dynamic weights: Our work and several other works share the general idea that weight matrices might be adaptive, rather than fixed. For example, fast weight programmers, dynamic filter networks, and linear transformers can all be seen as having weights which are themselves a function of the input [2, 14, 25]. These methods differ from our approach in that the contexts are inferred rather than supplied as a method for external conditioning. Furthermore, these methods do not constrain the movement of the implied manifold over weights, nor ensure that its topology matches the task at hand.

Linear modes and distributions over networks: Weight manifolds are one way to establish a set of networks that work well. In this way, it is closely related to hypernetworks [11] and Bayesian networks [18, 3], which both establish distributions over weight space. More close in spirit is work which documents the existence of "linear mode connectivity," i.e. that paths of everywhere-low loss exist in weight space which connect separate learning trajectories [8]. Such paths are also empirically found connecting minima from related tasks, i.e. connecting the pretrained multitask solution with a fine-tuned solution [21]. Here, rather than find such solutions empirically, we provide a framework for directly training lines in weight space, and in general, any low-dimensional manifold. Interestingly, our results provide proof that there exist straight lines that connecting modes, not only the smoothly curved lines found *post hoc* after training as in prior works.

Conditioning methods: Although conditioning via input concatenation and embedding are standard, several other methods for conditioning exist. One closely related work is FiLM, which learns to apply an affine transformation to the network's intermediate activations which is a function of the conditioning variable [24]. Other methods with a similar philosophy include Conditional Instance Norm [9] and Conditional Batch Norm [7], which adapt standard layer normalization layers to be functions of the conditioning information. These methods effectively establish a manifold over the biases of each layer, which is either an affine manifold (in the case of FiLM) or curved according to the divisive normalization scheme. Our approach can be seen as a generalization of these methods.

Models of neuromodulation: Weight manifolds can be seen as an abstraction and generalization of many computational models of neuromodulation. While no single paragraph can summarize all such models, here we highlight those models in which neuromodulation acts as a functional knob upon circuit behavior [1, 5, 26, 28, 23, 6, 27]. For example, one classic model describes the effect of acetylcholine in the hippocampus as a knob upon top-down/bottom-up gain in a generative model, effectively correlating acetylcholine to perceptual uncertainty [29, 30]. While details differ, these

papers generally supply specific mechanisms in which neuromodulation affects circuit behavior. We argue that each of these models are equivalent to a manifold in weight space of a single network, and furthermore that it is productive to consider the abstract properties of such manifolds for learning and computation.

5 Broader impact

This work is focused on foundational theoretical research for optimizing manifolds of model weights. As such, it does not have direct deployment considerations or immediate negative harms. Still, a plausible positive impact of this work is better control, interpretation, and constraint of learned neural network functions. This should not be misconstrued for safety guarantees—networks learned with our approach are only constrained along the conditioning axes specified by the researcher. Misalignment between the intended axes and the specification can result in unexpected behavior, and more importantly, the model is not constrained on unspecified axes. Finally, an common application of conditioning is in large language models and generative models (e.g. image diffusion models), so the improvements in this work endow these models with additional capabilities, potentially exacerbating existing harms and misuse.

6 Discussion

Here, we demonstrated how the general principle behind neuromodulation can be ported to artificial neural networks by analogy to low-dimensional manifolds in weight space. We showed experimentally that simple manifolds (such as straight lines and ellipses) that solve tasks indeed exist in weight space, and furthermore, can be trained using a novel steepest rule for manifolds. Our approach provides a robust and principled procedure to create collectives of networks that learn together yet differ from one another in meaningful ways.

We conjectured that the advantage of conditioning by modulating weights instead of injecting input would be the ability to exploit the topology of the data. Thus, the choice of manifold provides a programmable inductive bias for conditioning. In support of this claim, we demonstrated that this allows for out-of-distribution generalization to unseen conditioning values better than conditioning via injecting input. On the other hand, when the chosen manifold and task topology are misspecified, modulating weights provides limited advantages. Thus, our experimental results demonstrate that modulating weights is useful computational primitive when the topology of the task is clear, and it is easily exploited by simple modulation schemes (i.e simple manifolds).

Our theoretical formulation provides a foundation for many potential applications beyond those that we studied in this paper. In particular, we focused on a few simple manifold types, but future extensions could include more complex manifolds that can be bent and distorted in weight space to permit more flexible topology embeddings. Alternatively, two, three, or higher dimensional topologies would permit the exploration of how different conditions across tasks interact in weight space. Furthermore, we study settings where the conditioning value is explicit and known, but this is the rarely the case for biological networks. Instead, a more realistic case should explore inferring or controlling the conditioning value through learned experience. Finally, an important use case for conditioning is encouraging a model to be invariant or equivariant to data transformations. Unlike many existing methods, our framework allows a network to target either case and potentially explore the trade-off between the two. Ultimately, the success of deep learning models has been their ability to decipher and exploit structure in the world. While this is typically statistically gleaned from data, inductive biases that are strong yet flexible allow networks to learn more efficiently and generalize. Our work, through a formal treatment of the connection between functions embedded in weight space and topologies in task space, enables a new generation of programmable, flexible, and controllable inductive biases for neural networks.

Code availability

The code for all figures in this paper were written in Jax [4] and will be made available shortly.

References

[1] L F Abbott. Modulation of function and gated learning in a network memory. *Proceedings of the National Academy of Sciences*, 87(23):9241–9245, December 1990.

[2] Jimmy Ba, Geoffrey E Hinton, Volodymyr Mnih, Joel Z Leibo, and Catalin Ionescu. Using Fast Weights to Attend to the Recent Past. In *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

[3] David Barber and Christopher M Bishop. Ensemble learning in Bayesian neural networks. *Nato ASI Series F Computer and Systems Sciences*, 168:215–238, 1998. Publisher: Springer Verlag.

[4] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018.

[5] V. Brezina and K. R. Weiss. Analyzing the functional consequences of transmitter complexity. *Trends in Neurosciences*, 20(11):538–543, November 1997.

[6] Julia C. Costacurta, Shaunak Bhandarkar, David M. Zoltowski, and Scott W. Linderman. Structured flexibility in recurrent neural networks via neuromodulation. *bioRxiv*, page 2024.07.26.605315, July 2024.

[7] Harm de Vries, Florian Strub, Jeremie Mary, Hugo Larochelle, Olivier Pietquin, and Aaron C Courville. Modulating early visual processing by language. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[8] Felix Draxler, Kambis Veschgini, Manfred Salmhofer, and Fred Hamprecht. Essentially No Barriers in Neural Network Energy Landscape. In *Proceedings of the 35th International Conference on Machine Learning*, pages 1309–1318. PMLR, July 2018. ISSN: 2640-3498.

[9] Vincent Dumoulin, Jonathon Shlens, and Manjunath Kudlur. A Learned Representation For Artistic Style, February 2017. arXiv:1610.07629 [cs].

[10] Charles R. Gerfen and D. James Surmeier. Modulation of Striatal Projection Systems by Dopamine. *Annual Review of Neuroscience*, 34(Volume 34, 2011):441–466, July 2011. Publisher: Annual Reviews.

[11] David Ha, Andrew Dai, and Quoc V. Le. HyperNetworks, December 2016. arXiv:1609.09106 [cs].

[12] Ronald M. Harris-Warrick and Eve Marder. Modulation of neural networks for behavior. *Annual Review of Neuroscience*, 14:39–57, 1991. Place: US Publisher: Annual Reviews.

[13] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising Diffusion Probabilistic Models. In *Advances in Neural Information Processing Systems*, volume 33, pages 6840–6851. Curran Associates, Inc., 2020.

[14] Xu Jia, Bert De Brabandere, Tinne Tuytelaars, and Luc V Gool. Dynamic Filter Networks. In *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

[15] M Klein, J Camardo, and E R Kandel. Serotonin modulates a specific potassium current in the sensory neurons that show presynaptic facilitation in Aplysia. *Proceedings of the National Academy of Sciences*, 79(18):5713–5717, September 1982. Publisher: Proceedings of the National Academy of Sciences.

[16] Alex Krizhevsky. Learning Multiple Layers of Features from Tiny Images. page 60, 2009.

[17] I B Levitan. Modulation of ion channels in neurons and other cells. *Annual review of neuroscience*, 11:119–136, January 1988.

[18] David JC MacKay. A practical Bayesian framework for backpropagation networks. *Neural computation*, 4(3):448–472, 1992. Publisher: MIT Press.

[19] Eve Marder. Neuromodulation of Neuronal Circuits: Back to the Future. *Neuron*, 76(1):1–11, October 2012.

[20] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5:115–133, 1943. Publisher: Springer.

[21] Seyed Iman Mirzadeh, Mehrdad Farajtabar, Dilan Gorur, Razvan Pascanu, and Hassan Ghasemzadeh. Linear Mode Connectivity in Multitask and Continual Learning, October 2020. arXiv:2010.04495 [cs].

[22] Michael P. Nusbaum and Mark P. Beenhakker. A small-systems approach to motor pattern generation. *Nature*, 417(6886):343–350, May 2002. Publisher: Nature Publishing Group.

[23] Mohammed Abdal Monium Osman, Kai Fox, and Joshua Isaac Stern. A Hopfield network model of neuromodulatory arousal state, September 2024. Pages: 2024.09.15.613134 Section: New Results.

[24] Ethan Perez, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron Courville. FiLM: Visual Reasoning with a General Conditioning Layer, December 2017. arXiv:1709.07871 [cs].

[25] Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. Linear Transformers Are Secretly Fast Weight Programmers, June 2021. arXiv:2102.11174 [cs].

[26] Jake P. Stroud, Mason A. Porter, Guillaume Hennequin, and Tim P. Vogels. Motor primitives in space and time via targeted gain modulation in cortical networks. *Nature Neuroscience*, 21(12):1774–1783, December 2018. Publisher: Nature Publishing Group.

[27] Ben Tsuda, Stefan C. Pate, Kay M. Tye, Hava T. Siegelmann, and Terrence J. Sejnowski. Neuromodulators generate multiple context-relevant behaviors in a recurrent neural network by shifting activity flows in hyperchannels. *bioRxiv*, 2024. Publisher: Cold Spring Harbor Laboratory _eprint: https://www.biorxiv.org/content/early/2024/09/12/2021.05.31.446462.full.pdf.

[28] Nicolas Vecoven, Damien Ernst, Antoine Wehenkel, and Guillaume Drion. Introducing neuromodulation in deep neural networks to learn adaptive behaviours. *PLOS ONE*, 15(1):e0227922, January 2020. Publisher: Public Library of Science.

[29] Angela Yu and Peter Dayan. Acetylcholine in cortical inference. *Neural Networks*, 15(4-6):719–730, 2002. Publisher: Elsevier.

[30] Angela J. Yu and Peter Dayan. Uncertainty, neuromodulation, and attention. *Neuron*, 2005.

A Supplement: Mathematical details of manifold optimization

A.1 Theorem 1: Optimal Manifold Perturbation

Theorem 1. *Given a parametrized manifold $\mathcal{M}(s, \mathbf{P})$ with loss functional $L[\mathcal{M}] = \int_0^1 \ell(\mathcal{M}(s, \mathbf{P}))ds$, the optimal perturbation $\Delta\mathbf{P}$ that minimizes the first-order approximation of $L[\mathcal{M}(s, \mathbf{P} + \Delta\mathbf{P})]$ subject to a constraint on the total squared distance $\int_0^1 |\mathcal{M}(s, \mathbf{P} + \Delta\mathbf{P}) - \mathcal{M}(s, \mathbf{P})|^2 \, \mathrm{d}s = C_0$ is given by:*

$$\Delta\mathbf{P} = -\frac{1}{2\lambda} \left[\int_0^1 \mathbf{M}(s) \, \mathrm{d}s \right]^{-1} \int_0^1 \mathbf{g}(s) \, \mathrm{d}s \tag{15}$$

where $\mathbf{M}(s) = \left(\frac{\partial \mathcal{M}}{\partial \mathbf{P}}\right)^T \frac{\partial \mathcal{M}}{\partial \mathbf{P}}$ is the local metric tensor, $\mathbf{g}(s) = \left(\frac{\partial \mathcal{M}}{\partial \mathbf{P}}\right)^T \nabla \ell(\mathcal{M}(s))$ is the local gradient, and λ is a Lagrange multiplier that controls the step size.

A.2 Proof of Theorem 1

We begin by establishing the linearization of the loss functional around the current manifold $\mathcal{M}(s, \mathbf{P})$. For a small perturbation $\Delta\mathbf{P}$, the manifold changes approximately as:

$$\mathcal{M}(s, \mathbf{P} + \Delta\mathbf{P}) \approx \mathcal{M}(s, \mathbf{P}) + \frac{\partial \mathcal{M}}{\partial \mathbf{P}} \Delta\mathbf{P} \tag{16}$$

Using this linearization, we can approximate the loss functional:

$$\begin{aligned} L[\mathcal{M}(s, \mathbf{P} + \Delta\mathbf{P})] &\approx \int_0^1 \ell \left(\mathcal{M}(s, \mathbf{P}) + \frac{\partial \mathcal{M}}{\partial \mathbf{P}} \Delta\mathbf{P} \right) \, \mathrm{d}s && (17) \\ &\approx \int_0^1 \left[\ell(\mathcal{M}(s, \mathbf{P})) + \nabla \ell(\mathcal{M}(s, \mathbf{P})) \cdot \frac{\partial \mathcal{M}}{\partial \mathbf{P}} \Delta\mathbf{P} \right] \, \mathrm{d}s + \mathcal{O}(\|\Delta\mathbf{P}\|^2) && (18) \\ &= L[\mathcal{M}] + \int_0^1 \nabla \ell(\mathcal{M}(s, \mathbf{P})) \cdot \frac{\partial \mathcal{M}}{\partial \mathbf{P}} \Delta\mathbf{P} \, \mathrm{d}s + \mathcal{O}(\|\Delta\mathbf{P}\|^2) && (19) \end{aligned}$$

Here, we've used a first-order Taylor expansion of ℓ around $\mathcal{M}(s, \mathbf{P})$.

Now, let's consider the distance constraint. The squared distance between the original and perturbed manifolds is:

$$\begin{aligned} d^2(\mathcal{M}(s, \mathbf{P} + \Delta\mathbf{P}), \mathcal{M}(s, \mathbf{P})) &= \int_0^1 |\mathcal{M}(s, \mathbf{P} + \Delta\mathbf{P}) - \mathcal{M}(s, \mathbf{P})|^2 \, \mathrm{d}s && (20) \\ &\approx \int_0^1 \left| \frac{\partial \mathcal{M}}{\partial \mathbf{P}} \Delta\mathbf{P} \right|^2 \, \mathrm{d}s + \mathcal{O}(\|\Delta\mathbf{P}\|^3) && (21) \\ &= \int_0^1 \Delta\mathbf{P}^T \left(\frac{\partial \mathcal{M}}{\partial \mathbf{P}} \right)^T \left(\frac{\partial \mathcal{M}}{\partial \mathbf{P}} \right) \Delta\mathbf{P} \, \mathrm{d}s + \mathcal{O}(\|\Delta\mathbf{P}\|^3) && (22) \end{aligned}$$

For notational convenience, let's define:

$$\mathbf{M}(s) = \left(\frac{\partial \mathcal{M}}{\partial \mathbf{P}} \right)^T \frac{\partial \mathcal{M}}{\partial \mathbf{P}} \qquad \text{(local metric tensor)} \tag{23}$$

$$\mathbf{g}(s) = \left(\frac{\partial \mathcal{M}}{\partial \mathbf{P}} \right)^T \nabla \ell(\mathcal{M}(s, \mathbf{P})) \qquad \text{(local gradient)} \tag{24}$$

Our optimization problem now becomes:

$$\begin{aligned} &\text{minimize} \quad \int_0^1 \mathbf{g}(s)^T \Delta\mathbf{P} \, \mathrm{d}s \\ &\text{subject to} \quad \int_0^1 \Delta\mathbf{P}^T \mathbf{M}(s) \Delta\mathbf{P} \, \mathrm{d}s = C_0 \end{aligned} \tag{25}$$

To solve this constrained optimization problem, we form the Lagrangian:

$$\mathcal{L}(\Delta\mathbf{P}, \lambda) = \int_0^1 \mathbf{g}(s)^T \Delta\mathbf{P} \, \mathrm{d}s + \lambda \left(\int_0^1 \Delta\mathbf{P}^T \mathbf{M}(s) \Delta\mathbf{P} \, \mathrm{d}s - C_0 \right) \tag{26}$$

Taking the functional derivative with respect to $\Delta\mathbf{P}$ and setting it to zero:

$$\frac{\delta\mathcal{L}}{\delta(\Delta\mathbf{P})} = \int_0^1 \mathbf{g}(s) \, \mathrm{d}s + \lambda \int_0^1 2\mathbf{M}(s)\Delta\mathbf{P} \, \mathrm{d}s = 0 \tag{27}$$

$$\Rightarrow \int_0^1 \mathbf{g}(s) \, \mathrm{d}s = -2\lambda \int_0^1 \mathbf{M}(s)\Delta\mathbf{P} \, \mathrm{d}s \tag{28}$$

$$\Rightarrow \int_0^1 \mathbf{M}(s)\Delta\mathbf{P} \, \mathrm{d}s = -\frac{1}{2\lambda} \int_0^1 \mathbf{g}(s) \, \mathrm{d}s \tag{29}$$

Now, we need to solve for $\Delta\mathbf{P}$. Since $\Delta\mathbf{P}$ is independent of the conditioning variable s, we can pull it outside the integral:

$$\int_0^1 \mathbf{M}(s)\Delta\mathbf{P} \, \mathrm{d}s = \left(\int_0^1 \mathbf{M}(s) \, \mathrm{d}s \right) \Delta\mathbf{P} \tag{30}$$

$$\Rightarrow \left(\int_0^1 \mathbf{M}(s) \, \mathrm{d}s \right) \Delta\mathbf{P} = -\frac{1}{2\lambda} \int_0^1 \mathbf{g}(s) \, \mathrm{d}s \tag{31}$$

$$\Rightarrow \Delta\mathbf{P} = -\frac{1}{2\lambda} \left(\int_0^1 \mathbf{M}(s) \, \mathrm{d}s \right)^{-1} \int_0^1 \mathbf{g}(s) \, \mathrm{d}s \tag{32}$$

The parameter λ controls the step size and can be set to satisfy the distance constraint. In practice, it serves a similar role to the learning rate in gradient descent.

B Metric tensors for common parameterizations

B.1 Elliptical Manifold

The manifold is given by:

$$M(s, \mathbf{P}) = \mathbf{c} + \mathbf{a}\cos(2\pi s) + \mathbf{b}\sin(2\pi s)$$

where $\mathbf{P} = (\mathbf{c}, \mathbf{a}, \mathbf{b})$.

The Jacobian with respect to the parameters $\mathbf{c}$, $\mathbf{a}$, and $\mathbf{b}$ is:

$$\frac{\partial\mathcal{M}}{\partial\mathbf{P}} = [\mathbf{I}, \cos(2\pi s)\mathbf{I}, \sin(2\pi s)\mathbf{I}]$$

The metric tensor is:

$$\mathbf{M}(s) = \left(\frac{\partial \mathcal{M}}{\partial \mathbf{P}}\right)^T \frac{\partial \mathcal{M}}{\partial \mathbf{P}} = \begin{bmatrix} \mathbf{I} & \cos(2\pi s)\mathbf{I} & \sin(2\pi s)\mathbf{I} \\ \cos(2\pi s)\mathbf{I} & \cos^2(2\pi s)\mathbf{I} & \cos(2\pi s)\sin(2\pi s)\mathbf{I} \\ \sin(2\pi s)\mathbf{I} & \cos(2\pi s)\sin(2\pi s)\mathbf{I} & \sin^2(2\pi s)\mathbf{I} \end{bmatrix}$$

Integrating the metric tensor over $s \in [0, 1]$ yields [2]:

$$\int_0^1 \mathbf{M}(s)\,\mathrm{d}s = \begin{bmatrix} \mathbf{I} & 0 & 0 \\ 0 & \frac{1}{2}\mathbf{I} & 0 \\ 0 & 0 & \frac{1}{2}\mathbf{I} \end{bmatrix}$$

The inverse of the integrated metric tensor is:

$$\left(\int_0^1 \mathbf{M}(s)\,\mathrm{d}s\right)^{-1} = \begin{bmatrix} \mathbf{I} & 0 & 0 \\ 0 & 2\mathbf{I} & 0 \\ 0 & 0 & 2\mathbf{I} \end{bmatrix}$$

Given the local gradient $\mathbf{g}(s) = \left(\frac{\partial \mathcal{M}}{\partial \mathbf{P}}\right)^T \nabla \ell(\mathcal{M}(s))$, the optimal update is:

$$\Delta \mathbf{P} = -\frac{1}{2\lambda}\left(\int_0^1 \mathbf{M}(s)\,\mathrm{d}s\right)^{-1} \int_0^1 \mathbf{g}(s)\,\mathrm{d}s$$

Substituting the inverse of the integrated metric tensor:

$$\Delta \mathbf{P} = -\frac{1}{2\lambda}\begin{bmatrix} \mathbf{I} & 0 & 0 \\ 0 & 2\mathbf{I} & 0 \\ 0 & 0 & 2\mathbf{I} \end{bmatrix}\begin{bmatrix} \int_0^1 \nabla \ell(\mathcal{M}(s))\,\mathrm{d}s \\ \int_0^1 \cos(2\pi s)\nabla \ell(\mathcal{M}(s))\,\mathrm{d}s \\ \int_0^1 \sin(2\pi s)\nabla \ell(\mathcal{M}(s))\,\mathrm{d}s \end{bmatrix}$$

The final update becomes:

$$\Delta \mathbf{P} = -\frac{1}{2\lambda}\begin{bmatrix} \int_0^1 \nabla \ell(\mathcal{M}(s))\,\mathrm{d}s \\ 2\int_0^1 \cos(2\pi s)\nabla \ell(\mathcal{M}(s))\,\mathrm{d}s \\ 2\int_0^1 \sin(2\pi s)\nabla \ell(\mathcal{M}(s))\,\mathrm{d}s \end{bmatrix}$$

B.2 B-Spline Parametrization

B-splines provide a flexible and numerically stable way to represent curves in weight space. A B-spline manifold is parametrized as:

$$\mathcal{M}(s, \mathbf{P}) = \sum_{i=0}^{n} \mathbf{P}_i B_i(s) \tag{33}$$

where $\mathbf{P} = (\mathbf{P}_0, \mathbf{P}_1, \ldots, \mathbf{P}_n)$ are the control points in weight space, and $B_i(s)$ are the B-spline basis functions of degree k.

Below, we will derive the metric tensor for the parameterization, assuming that the basis points are distributed strictly uniformly over the line. This assumption simplifies the metric, but would require maintaining uniformity through optimization via a constraint.

[2] a large language model was used to assist with evaluating this integral

B.2.1 Metric Tensor Derivation

To apply our manifold optimization approach, we need to compute the integrated metric tensor $\overline{\mathbf{M}} = \int_0^1 \mathbf{M}(s)ds$. The Jacobian matrix is:

$$\frac{\partial \mathcal{M}}{\partial \mathbf{P}} = (B_0(s)\mathbf{I} \quad B_1(s)\mathbf{I} \quad \cdots \quad B_n(s)\mathbf{I}) \tag{34}$$

where $\mathbf{I}$ is the identity matrix with the same dimension as the network parameters. The local metric tensor is:

$$\mathbf{M}(s) = \left(\frac{\partial \mathcal{M}}{\partial \mathbf{P}}\right)^T \frac{\partial \mathcal{M}}{\partial \mathbf{P}} = \begin{pmatrix} B_0(s)^2\mathbf{I} & B_0(s)B_1(s)\mathbf{I} & \cdots & B_0(s)B_n(s)\mathbf{I} \\ B_1(s)B_0(s)\mathbf{I} & B_1(s)^2\mathbf{I} & \cdots & B_1(s)B_n(s)\mathbf{I} \\ \vdots & \vdots & \ddots & \vdots \\ B_n(s)B_0(s)\mathbf{I} & B_n(s)B_1(s)\mathbf{I} & \cdots & B_n(s)^2\mathbf{I} \end{pmatrix} \tag{35}$$

To compute the integrated metric tensor, we need to evaluate:

$$\overline{\mathbf{M}}_{ij} = \int_0^1 B_i(s)B_j(s)\,\mathrm{d}s \cdot \mathbf{I} \tag{36}$$

B.2.2 Cubic B-Splines on Uniform Knots

For cubic B-splines ($k = 3$) on a uniform knot sequence with spacing h, each basis function $B_i(s)$ is nonzero only over four adjacent knot spans. Note that the spacing can be assumed to be uniform in s such that $h = 1/n$ for n knots.

B.2.3 Definition of Cubic B-Spline Basis Functions

For a uniform knot sequence, the cubic B-spline basis function $B_i(s)$ can be explicitly defined as:

$$B_i(s) = \frac{1}{6h^3} \begin{cases} (s - s_{i-2})^3, & s \in [s_{i-2}, s_{i-1}) \\ (s - s_{i-2})^3 - 4(s - s_{i-1})^3, & s \in [s_{i-1}, s_i) \\ (s_{i+2} - s)^3 - 4(s_{i+1} - s)^3, & s \in [s_i, s_{i+1}) \\ (s_{i+2} - s)^3, & s \in [s_{i+1}, s_{i+2}) \\ 0, & \text{otherwise} \end{cases} \tag{37}$$

A key property is that $B_i(s)B_j(s) = 0$ if $|i - j| > 3$, meaning the metric tensor has a banded structure with bandwidth 3.

B.2.4 Integration of B-Spline Products

To evaluate the integrated metric tensor, we need to compute the integrals of products of B-spline basis functions. Due to the compact support and piecewise polynomial nature of B-splines, these integrals can be computed analytically.

The analytical integration of products of B-spline basis functions yields [3]:

$$\overline{\mathbf{M}}_{ij} = \int_0^1 B_i(s)B_j(s)ds \cdot \mathbf{I} = \begin{cases} \frac{1}{140} \cdot \mathbf{I}, & \text{if } |i - j| = 3 \\ \frac{1}{60} \cdot \mathbf{I}, & \text{if } |i - j| = 2 \\ \frac{11}{140} \cdot \mathbf{I}, & \text{if } |i - j| = 1 \\ \frac{1}{20} \cdot \mathbf{I}, & \text{if } |i - j| = 0 \end{cases} \tag{38}$$

For a system with $n + 1$ control points (from $\mathbf{P}_0$ to $\mathbf{P}_n$), the integrated metric tensor $\overline{\mathbf{M}}$ has the following banded structure:

[3] a large language model was used to assist with evaluating this integral

$$\overline{\mathbf{M}} = \frac{1}{420} \cdot \begin{pmatrix} 21 & 33 & 3 & 1 & 0 & \dots & 0 \\ 33 & 21 & 33 & 3 & 1 & \dots & 0 \\ 3 & 33 & 21 & 33 & 3 & \dots & 0 \\ 1 & 3 & 33 & 21 & 33 & \dots & 0 \\ 0 & 1 & 3 & 33 & 21 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 & 3 & 33 & 21 \end{pmatrix} \cdot \mathbf{I} \tag{39}$$

The inverse of this metric tensor is required for the optimal manifold perturbation as shown in Theorem 1. This matrix is invertible as it is a Gram matrix; this inverse can be precomputed and used for all updates.

C Supplement: Algorithmic details

Below is an algorithmic specification of our update rule.

Algorithm 1 Efficient Manifold Optimization

Require: Training data $D = \{(\mathbf{x}_i, \mathbf{y}_i, s_i)\}_i^N$, manifold type $\mathcal{M}$ with basis size n, network F

Initialize manifold parameters $\mathbf{P}$ for each layer in F
for each epoch **do**
  for each batch $\{(\mathbf{x}_i, \mathbf{y}_i, s_i)\}_i^B \subset D$ **do**
    `// Forward pass`
    for each weight $\mathbf{W}(s) = \sum_k^N a_k(s)\mathbf{W}_k$ in each layer in F **do**
      Compute outputs for basis points matrices $\{\mathbf{W}_k\mathbf{x}\}_k^n$
      For each example, get outputs for the batch as $\{\mathbf{z}_i = \sum_k^n a_k(s)\mathbf{W}_k\mathbf{x}_i\}_i^B$
      Apply nonlinearities to $\mathbf{z}_i$
    end for
    Compute loss $L = \ell(\hat{\mathbf{y}}, \mathbf{y})$ using final outputs $\hat{\mathbf{y}}$
    `// Backward pass`
    Compute gradients $\nabla_{\mathbf{P}} L$ via auto-differentiation
    Apply manifold-specific rescaling: $\nabla_{\mathbf{P}} L \leftarrow \overline{\mathbf{M}}^{-\mathbf{1}} \nabla_{\mathbf{P}} L$
    Update manifold parameters using rescaled gradients
  end for
end for

D Supplement: Additional methods details

Throughout the manuscript, the term 'CNN' specifically refers to a network with 3 convolutional layers with [32, 64, 128] filters and kernel size of 3, followed by a MLP with one layer of 512 features. Each convolutional layer is followed by max pooling with a window shape and stride of 2, and all nonlinearities are ReLU.

All experiments in the main manuscript use SGD with a learning rate of 0.01 and momentum of 0.9.

For the 'concat' conditioning strategy, the conditioning information was injected into the CNN after the convolutional layers, and before the fully connected layers. For the 'embed' strategy, an embedding layer with width 32 was created which sees the conditioning information, and the embedding was concatenated the flattened output of the convolutional filters.

All experments were carried out on Nvidia H100 cards. For consistency, we report the mean and standard deviations of 20 random initialization seeds. For efficiency, these 20 networks were v-mapped in Jax and thus see the same data in the same order, i.e. share a data seed.

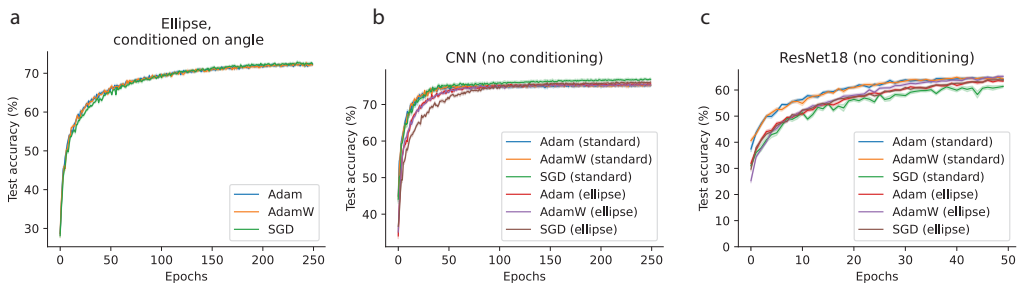

Figure 4: **a)** We train an elliptical manifold in the space of weights of the same CNN architecture in the main manuscript on rotated CIFAR-10, conditioning on rotation angle by mapping it to ellipse phase. Interestingly, we find that Adam and AdamW do not show meaningful improvements over SGD with momentum. **b)** Manifolds can also be trained without conditioning on task variables. Here, we train an ellipse on CIFAR-10 using several optimizers, randomizing for each example which network on the manifold is chosen. Convergence accuracy is identical to training a point network. **c)** Here, we train on the identical task in panel **b** but using a ResNet18 architecture with LayerNorm. Manifolds and single points (i.e. standard training) perform similarly.

E Additional experiments

Here, we extend the experiments in the main manuscript to other optimizers, architectures, and to cover the case when the entire manifold is trained on the same objective with the same data rather than conditioned on side information. The training details are as follows.

Panel a: Here, we train an elliptical manifold of CNN weights using Adam and AdamW. As with the SGD, the gradients on the basis points after metric rescaling were fed directly into standard Optax optimizers. Learning rates were tuned in the range 1e-5 to 1e-3 on each optimizer, with optimal rates at 0.01 for SGD and 0.0002 for Adam variants.

Panel b: Here, we train an identical CNN architecture with standard optimizers on the standard (not-rotated) CIFAR-10 task, and contrast this to training an elliptical architecture over weights but without conditioning. To ensure that the entire manifold is good at CIFAR-10, a random point on the manifold was used on each example.

Panel c: Similar to b, but using a ResNet-18 architecture with LayerNorm normalization layers.