Proceedings Track

The representational geometry of discrete computations on continuous manifolds

Abstract

Many tasks require mapping continuous input manifolds to discrete task outputs. Yet, how neural networks perform such *discrete computations on continuous manifolds* remains poorly understood. In parallel, recent work has argued that studying the representational geometry of neural networks could provide insights into task computation. However, the way neural representations of continuous inputs are warped in order to perform discrete tasks remains elusive. Here, we show that signatures of such discrete computations on continuous manifolds can be found in the representational geometry of neural networks. By analysing the Riemannian metric across layers of a neural network, we find that network computation can be decomposed into two stages: discretising continuous input features and performing logical operations on these discretised variables. We further demonstrate how different learning regimes (rich vs. lazy) have contrasting metric and curvature structures, affecting the ability of the networks to generalise, and how input noise during training smooths the hidden layer geometry. Overall, our work provides a geometric framework for understanding discrete computation on continuous manifolds in neural networks.

Keywords: Riemannian geometry, neural manifolds, representations, logic gates

1. Introduction

Studying the geometry of manifolds of neural activation can provide insights into the computations performed by neural networks (Chung and Abbott, 2021). Empirically, dimensionality reduction methods have been used to infer low-dimensional structure in the high-dimensional activity of biological (Pellegrino et al., 2024) and artificial neural networks (Aubry and Russell, 2015; Li et al., 2023). More recently, analytical methods stemming from Riemannian geometry have related these low-dimensional manifolds of neural network activation to task computations (Hauser and Ray, 2017). These works share a common basis: the manifold hypothesis, which posits that the input data to a network lie on low-dimensional continuous manifolds.

In contrast, seminal work has studied how neural networks can perform discrete computations, such as logical operations (Minsky and Papert, 1969). This idea has grown to eventually be at the heart of neurosymbolic AI, which tackles how networks can perform discrete tasks on raw input data such as images or text (Mao et al., 2019). Yet, the relationship between continuous representational geometry and the discrete computations neural networks perform remains elusive. For instance, in artificial neural networks, image data is thought to lie on a low-dimensional submanifolds of pixel space, but vision models are often trained to map this continuous manifold onto a small set of discrete classes (Cohen et al., 2020). Other models combine continuous inputs with discrete contextual cues — such as large language models responding to a manifold of possible inputs conditioned on a discrete prompt (Li et al., 2023), or diffusion models generating images from discrete textual descriptions (Dhariwal and Nichol, 2021). Similar patterns appear in neuroscience: animals may be trained to map images at different orientations on a continuous circular manifold to discrete categories (Reinert et al., 2021), or to integrate discrete contextual

evidence with continuous sensory input in decision-making (Mante et al., 2013), working memory (Panichello and Buschman, 2021), and motor control (Zimnik and Churchland, 2021). Across these domains, biological and artificial neural networks must integrate continuous data with discrete rules — whether learned through training or provided as explicit cues — to perform their computations.

Here, we argue that formally studying the representational geometry of neural networks using Riemannian geometry can help understand how they perform such discrete computations on continuous manifolds. First, we provide a simple example of a multi-layer perceptron trained to implement logical gates (XOR, AND) on a manifold of inputs. The Riemannian metric and Gaussian curvature reveal the discretisation of the continuous input manifold in the early layers of the network and the discrete logical operation in subsequent layers. Thus, the computation is decomposed into input data manifold-specific and logical task-specific discrete computation. Furthermore, rich and lazy learning regimes can lead to structured and random representational geometries respectively, linked to the ability of the network to generalise. Next, we show how input noise during training affects the smoothness of the metric tensor across decision boundaries and causes the curvature of the manifold to decrease monotonically with the noise level, corresponding to the network learning the posterior distribution of the output.Overall, our work links continuous input data manifolds to discrete task output by studying the Riemannian geometry of hidden layer activation.

2. Contributions

1. **Riemannian geometry of discrete computation on continuous manifolds.** We introduce a framework anchored in Riemannian geometry to study discrete computations on continuous manifolds.

2. **The metric tensor reveals how neural networks implement logical gates on non-linear input manifolds.** In multi-layer perceptrons trained on logical gates tasks we show that a highly localised metric tensor close to decision boundaries reflects the discretisation of the continuous input manifold.

3. **Different learning regimes have different representational geometries.** In the same model we show that rich learning produces a network that partitions the computation into a binarisation corresponding to the collapse of the embedding space of the input manifold and a degenerate metric tensor, followed by the discrete logical computations. In comparison, lazy learning learns a random projection of the inputs. Furthermore, the rich, but not the lazy, geometry generalises to unseen inputs on the manifold.

4. **Noise smooths the geometry to implement Bayesian computation.** We show that the curvature of the manifold decreases with input noise level, which corresponds to a flatter posterior distribution of the output of the network. Furthermore, the curvature can switch from positive to negative past a level of noise.

3. Related works

Representational geometry of neural networks. A long line of work has argued that studying how variables are encoded can help understand how neural networks solve tasks,

with applications in vision (Aubry and Russell, 2015), large language models (Li et al., 2023), and reinforcement learning (Tennenholtz and Mannor, 2022). In particular, tools from Riemannian geometry have been used to characterise the exact intrinsic geometry neural representation of networks receiving low-dimensional inputs (Hauser and Ray, 2017; Benfenati and Marta, 2023). Here, we instead characterise how discrete target task output — which are discontinuous maps from the continuous input data manifold to the discrete target output — affects the geometry of the hidden layer activation.

Neural networks trained on discrete tasks. A long line of work dating back to the inception of the perceptron has studied whether and how neural networks can implement discrete and logical tasks (McCulloch and Pitts, 1943; Minsky and Papert, 1969), which culminated in proving that neural networks can simulate any Turing machine (Siegelmann and Sontag, 1992). More recent work on neurosymbolic AI and geometric deep learning has tackled how neural networks can use continuous input data to perform logical computations (Mao et al., 2019; Petersen et al., 2022). Here, we ask more specifically how neural networks represent these continuous input variables in their hidden layer activations as they perform discrete tasks. Furthermore, we extend our framework to non-linear manifolds and show that their topology affects the learned computation and representation.

4. Riemannian geometry provides the tools to study the intrinsic geometry of neural network representations

In this section we briefly review concepts from Riemannian geometry that we will use to study neural network representations. In particular, we will exactly characterise the intrinsic geometry of the hidden layer activation. We work under the manifold hypothesis, and the overall neural network can be summarised as:

$$\mathcal{M} \xrightarrow{\psi} \mathbb{R}^{n_{\text{in}}} \xrightarrow{\varphi} \mathbb{R}^{n} \xrightarrow{\zeta} \mathbb{R}^{n_{\text{out}}}$$

The input data manifold is $\mathcal{M}$, which is embedded into the input space of the neural network $\mathbb{R}^{n_{\text{in}}}$ via ψ. Assuming that we are interested in studying the representational geometry of a particular layer, we can decompose the neural network into two functions: φ, which maps the embedded inputs to the hidden layer of interest, and ζ mapping this hidden layer to the output. The activation of the network in response to all possible inputs $\varphi(\mathcal{M})$ will itself be a manifold of the same topology as the input. We are interested in characterising the geometry of this manifold as it sits in the hidden-layer state-space.

To characterise the intrinsic geometry of the hidden layer activation, the pullback of the metric can be computed:

$$g : T_p\mathcal{M} \times T_p\mathcal{M} \xrightarrow{\mathrm{d}\psi,\mathrm{d}\psi} \mathbb{R}^{n_{\text{in}}} \times \mathbb{R}^{n_{\text{in}}} \xrightarrow{\mathrm{d}\varphi,\mathrm{d}\varphi} \mathbb{R}^{n} \times \mathbb{R}^{n} \xrightarrow{\langle\cdot,\cdot\rangle} \mathbb{R}$$

where $T_p\mathcal{M}$ is the tangent space at a point $p \in \mathcal{M}$. Intuitively, tangent vectors of the input manifold can be mapped to vectors tangent to the manifold in the input embedding space $\mathbb{R}^{n_{\text{in}}}$ via the pushforward of the input embedding $\mathrm{d}\psi$, and then to vectors tangent to the manifold in the hidden layer state-space $\mathbb{R}^{n}$ via the Jacobian of the neural network $\mathrm{d}\varphi$. Thus, the inner product between two tangent vectors of the input manifold can be measured via the standard Euclidean dot product $\langle\cdot,\cdot\rangle$ by pushing them forward to the relevant Euclidean state-space.

This definition is independent of the choice of local coordinates on the manifold. However, in many cases, there will be a basis that represents the particular task variables at hand — e.g. the orientation and translation of an object in image space — whose neural representation we seek to understand. In this case, the metric can be summarised by how it acts on this basis: if we call $\mathbf{z}(p_1, p_2) \in \mathbb{R}^n$ the response of the neural population to an input depending on two task variables $p_1, p_2 \in \mathbb{R}$, we can characterise the encoding of these task variables by asking how the neural representation changes in response to small changes in the first $\partial_{p_1}\mathbf{z} = \frac{d\mathbf{z}(p_1,p_2)}{dp_1}$ or second $\partial_{p_2}\mathbf{z} = \frac{d\mathbf{z}(p_1,p_2)}{dp_2}$ variable. If variables have correlated representations then changing them individually will cause similar changes in representation, whereas uncorrelated representations cause changes in orthogonal directions in neural state-space. For example, in neuroscience, it has been suggested that sequentially presented images are represented orthogonally by the brain (Xie et al., 2022). Mathematically, this is equivalent to saying that $\partial_{p_1}\mathbf{z} \cdot \partial_{p_2}\mathbf{z} = 0$. However, variables such as the level of evidence and the choice of a subject (Koay et al., 2022) have correlated representations, meaning that $\partial_{p_1}\mathbf{z} \cdot \partial_{p_2}\mathbf{z} \neq 0$. In a task involving k variables, the pullback metric summarises such correlations:

$$G = \begin{bmatrix} \partial_{p_1}\mathbf{z} \cdot \partial_{p_1}\mathbf{z} & \dots & \partial_{p_1}\mathbf{z} \cdot \partial_{p_k}\mathbf{z} \\ \vdots & \ddots & \vdots \\ \partial_{p_k}\mathbf{z} \cdot \partial_{p_1}\mathbf{z} & \dots & \partial_{p_k}\mathbf{z} \cdot \partial_{p_k}\mathbf{z} \end{bmatrix}$$

These are meaningful insights, as the computation necessary to solve these tasks require these correlations — memories are encoded in orthogonal spaces because they are unrelated, whereas the evidence level and choice of an animal are strongly related.

5. Neural network models can perform discrete computations on continuous manifolds of inputs

Neural networks can approximate arbitrary logic gates (Siegelmann and Sontag, 1992). For example, the XOR task has long been regarded as a canonical example of a non-linearly separable problem requiring neural networks to perform discrete computations (Minsky and Papert, 1969; Goodfellow et al., 2016), and has been widely used as a simple benchmark across machine learning (Afzal and Wani, 2014; Mehta et al., 2018) and neuroscience (Stöckel and Eliasmith, 2022). In this section we consider a continuous extension of the XOR task, allowing inputs to lie on a non-linear manifold. In particular, we start with a simple example where the input manifold is a flat torus, consisting of the Cartesian product of two circles which each encode a different input to the gate. This provides a minimal yet rich setting for analysing how representational geometry supports discrete computations on continuous manifolds.

To generate the input manifold, we define two angular task variables, $\theta_1, \theta_2 \in [0, 2\pi)$, which form a torus embedded in $\mathbb{R}^4$:

$$\mathbf{x} = [\cos(\theta_1)\ \ \sin(\theta_1)\ \ \cos(\theta_2)\ \ \sin(\theta_2)]^\top.$$

We define a decision boundary on each input circle at an angle α, such that $0 \leq \theta - \alpha < \pi$ is mapped to $+1$, and $\pi \leq \theta - \alpha < 2\pi$ is mapped to 0. The network is then trained to perform XOR on the discretised inputs, creating four quadrants (with periodic boundaries) on the input manifold (Fig. 1a). Although inputs to the network are 4-dimensional, an efficient

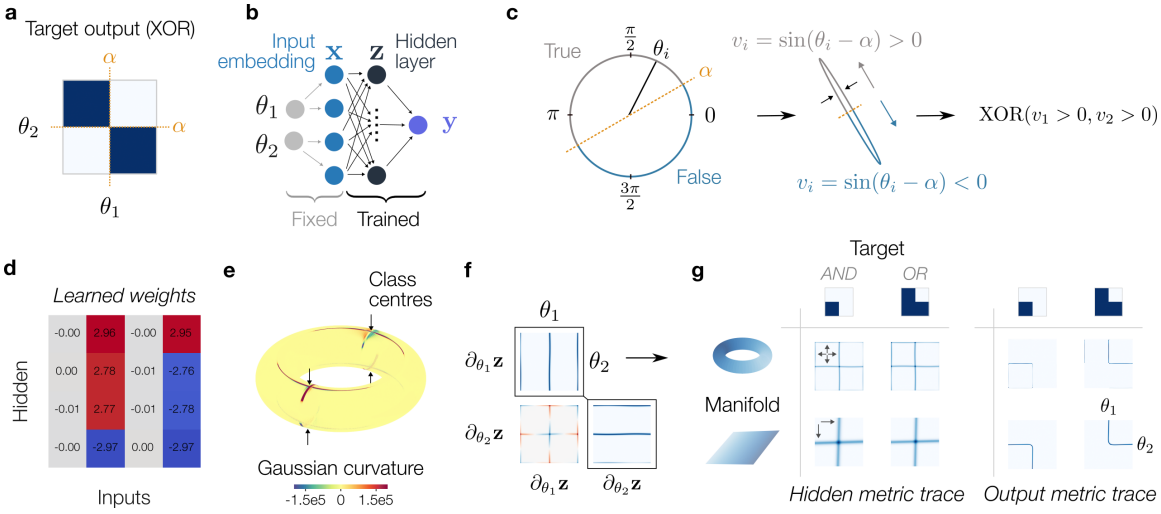

Figure 1: **Neural network geometry reflects discrete computations on manifolds.** **a.** The target outputs of the XOR task with two angular inputs $\theta_1, \theta_2 \in [0, 2\pi)$. **b.** Diagram of the network architecture used to solve the task. **c.** *Left:* Schematic showing an input variable θ_i embedded on a unit circle, $\mathbf{x} = [\cos(\theta_i), \sin(\theta_i)]^\top$, with decision boundaries at α and $\pi + \alpha$. *Right:* The optimal solution compresses the irrelevant dimensions and performs the logic operation on the 1D representation. **d.** Input weight matrix of a trained network (for $\alpha = 0$). Weights corresponding to $\mathbf{x}_1 = \cos(\theta_1)$ and $\mathbf{x}_3 = \cos(\theta_2)$ are close to zero. **e.** The Gaussian curvature of the hidden layer manifold visualised on a torus. The curvature diverges if $\theta_i \in \{\frac{\pi}{2}, \frac{3\pi}{2}\}$. **f.** The components of the metric tensors of the hidden layer activation of the network (lower-triangular part of the metric, each entry varies over the manifold). **g.** Trace of the metric for networks trained on different combination of input manifolds (torus, plane) and target output (AND, OR).

representation is sensitive only to the sign of the following latent variables:

$$v_i(\theta_i, \alpha) = \sin(\theta_i - \alpha)$$

such that the target output is $y = \mathrm{XOR}[v_1(\theta_1, \alpha) > 0, v_2(\theta_2, \alpha) > 0]$ (Fig. 1c). Without loss of generality, we can set $\alpha = 0$ and recover solutions for all other boundary rotations with a rotation of the input weights. To solve the task, we used a feedforward network with four hidden neurons (Fig. 1b), trained with a tanh hidden activation, SGD optimiser and binary cross-entropy loss.

5.1. The metric and curvature of the neural representation ties continuous inputs to discrete task outputs

In networks trained on the XOR task on the flat torus, the input weights compress task-irrelevant dimensions. For example, when $\alpha = 0$, the task output is independent of x_1 and x_3: the cosines of the two input angles (Fig. 1c-d). For $\alpha \neq 0$, different linear combinations

of the x_i's will encode the task-relevant information. Thus, two dimensions of the input torus are collapsed, with the remaining inputs (e.g. $x_2, x_4 \in [-1, 1]$ for $\alpha = 0$) defining a patch of the $\mathbb{R}^2$ plane. This is reflected by the intrinsic curvature of the torus in the hidden layer, which for collapsed x_1 and x_3 directions is approximately given by

$$K \simeq \frac{M(\theta_1, \theta_2)}{\cos^4(\theta_1)\cos^4(\theta_2)}$$

where $M(\theta_1, \theta_2)$ is a term depending on the first and second derivatives of the metric tensor. When $\cos(\theta_i) = 0$ for either i the curvature diverges. This happens near the class centres $\theta_i \in \{\frac{\pi}{2}, \frac{3\pi}{2}\}$ (Fig. 1e), where the circles are "folded" (Fig. 1c).

From the standpoint of Riemannian geometry, the hidden layer metric tensor encodes this effect. Visualising the entries of the metric tensor in the input coordinates highlighted that space near decision boundaries is stretched, while space far from them is compressed (Fig. 1f). This effectively discretises the inputs, with transitions between the discretised variables only occurring across boundaries. In comparison, the metric pulled back from the output layer reflected the logical gate (Fig. 1g). To see how this result depended on the choice of task and input manifold, we repeated this analysis in the AND and OR tasks, with inputs either on a torus or a plane. Specifically, we looked at the trace of the metric tensor, which reflects the overall stretching and compression of space at any point on the manifold. We found that the hidden layer metric's trace showed stretching of space near the inputs class boundaries, while the output metric showed stretching of space near the target output decision boundaries (Fig. 1g). Thus, the metric and curvature of the hidden layer reflect computations that are tied to the choice of input manifold and its embedding, while the output layer geometry reflects logic gate-specific computation.

Finally, we explored what happens when continuous and discrete computations cannot be partitioned across layers. To do that, we went back to the XOR task and compared the geometry of a 1-layer network to that of a 2-layer network. Classic work has shown that, with binary inputs to the network, the XOR task requires at least one hidden layer (Minsky and Papert, 1969). Thus, our 1-layer network cannot use its hidden layer to discretise the input manifold, while the 2-layer network can in principle discretise the input manifold in the first hidden layer before performing the classic solution on the binarised variables in the second layer. Despite this discrepancy, both 1- and 2-layer networks can perform the XOR task to a similar performance level (Fig. S1a). Interestingly, the two networks used different geometries: the 2-layer network employed a similar representation of the discretised variable in the first layer, while the 1-layer network showed a rich geometry which combined the discretisation and logic computation (Fig. S1b).

Overall, this suggests that computations in these simple networks are partitioned across layers into the discretisation of inputs manifolds and the performance of discrete logical computations.

5.2. Rich and lazy learning of input manifold features result in different intrinsic geometries

Next, we tried to better understand how different learning regimes affect the hidden layer representation and the overall discrete computations on the continuous manifold. Previous work has shown that varying the variance of the weights at initialisation can qualitatively

affect the learned representation (Saxe et al., 2013). More recent work has studied these different learning regimes in the context of the discrete tasks (Flesch et al., 2022). Here we replicate these analyses and additional logic gates (Fig. S2) with our model of discrete computation on continuous non-linear manifolds.

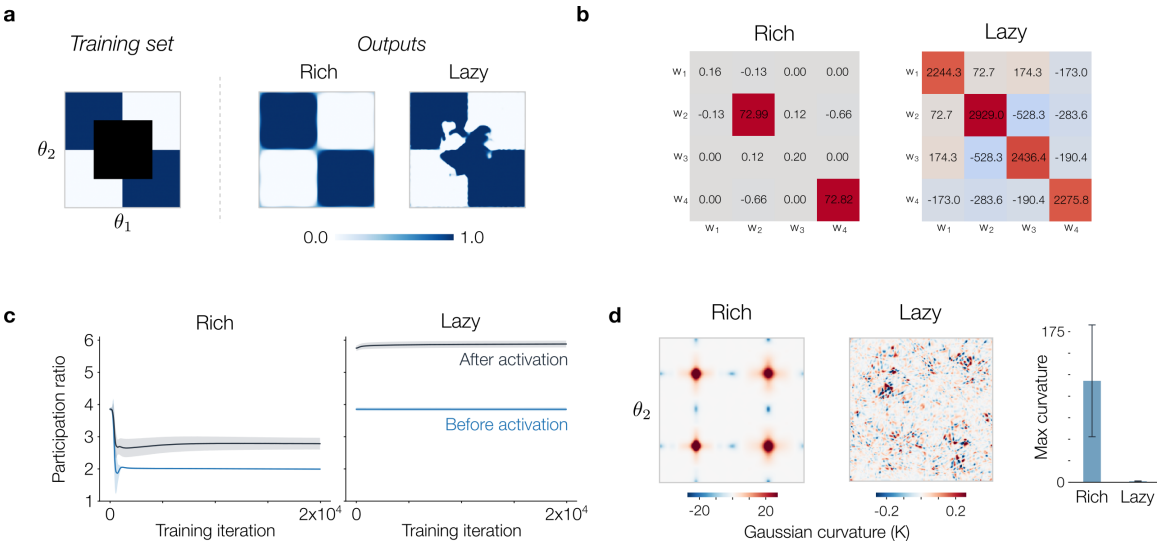

Figure 2: **Rich networks generalise and have different geometries to lazy networks. a.** Output of rich and lazy networks trained on XOR with a portion of the input manifold held out during training, represented by the black square. Rich networks are able to generalise to unseen inputs whereas lazy networks overfit to the training data. **b.** The Gram matrices of the input weights for rich and lazy networks. Rich networks ignore irrelevant inputs (corresponding to columns). Lazy networks randomly project each input into the high dimensional hidden space. **c.** The participation ratios over training in rich and lazy networks, with error bars calculated as the standard deviation across 10 different seeds of weight initialisations. Rich networks learn low-dimensional representations corresponding to low-dimensional input manifold. Lazy networks learn high-dimensional representations determined by weight matrix rank at initialisation. **d.** Curvature over the hidden layer manifold in rich and lazy networks, with a bar plot showing the average maximum curvature over the manifold (error bars show standard deviation). Curvature in rich networks is highly structured and has large magnitude, corresponding to large bending away from class centres. Lazy networks are mostly flat, and have random curvature patterns. Rich networks have much larger maximum curvatures with higher variability compared to lazy networks.

As in previous work, we continuously vary the strength of the initial weights of the network $W_{ij} \sim \mathcal{N}(0, \sigma^2)$. Previous work has found that small initial weights, corresponding to the rich learning regime, tend to generalise better (Saxe et al., 2013). To test that in our setting, we held out for testing the points on the input manifold corresponding to $(\theta_1, \theta_2) \in [-\frac{\pi}{2}, \frac{3\pi}{2}] \times [-\frac{\pi}{2}, \frac{3\pi}{2}]$. We found that the rich, but not the lazy learning regime

generalised to the unseen points on the manifold (Fig. 2a). The high-dimensional ($n = 100$) rich network learned the optimal low-dimensional projection of the input found by the small-width ($n = 4$) network of the previous section, as reflected by its Gram matrix $W^T W$ having sparse low-rank structure (Fig. 2b, *left*). Instead the lazy network learned random, near-orthogonal projections of the inputs (Fig. 2b, *right*).

To better understand the geometry underpinning this generalisation ability, we studied the embedding dimensionality and intrinsic curvature of the representation. First, consistent with previous work, rich learning led to lower-dimensional representations as reflected by the participation ratio $\left(\sum_{i=1}^{n} \sigma_i\right)^2 / \left(\sum_{i=1}^{n} \sigma_i^2\right)$ of the hidden layer representation of the torus (Fig. 2c). The discrepancy in the representation was also reflected in the intrinsic geometry. The rich network representation had peaks in positive curvature near the class centres (Fig. 2d, *left*). These points are where small changes in either input angle leads to a change in hidden layer representation in the same direction, towards switching the value of the discretised input variable. Since the peaks are symmetrical, only being trained on the points on the manifold on one of their halves is sufficient to generalise to the computation to the unseen inputs. In comparison, the lazy network had a mostly flat curvature, with small, randomly spread out, peaks in positive an negative curvature, which did not seem to encode a general computation (Fig. 2d, *right*). Furthermore, the rich network was more robust to noise in the state-space, likely linked to its highly structured representational geometry, while noise on the manifold had similar effects in both networks (Fig. S2).

Thus, our results suggest that different learning regimes can have fundamentally different effects on the representational geometry of discrete computations on continuous manifolds. Furthermore, these geometries are tied to different generalisation abilities of the network.

5.3. Noise-robust neural representations have low curvature

Next we asked how injecting noise tangent to the input manifold during training affects the learned representational geometry. We implemented this by a perturbation of the angle $\mathbf{x} = [\cos(\theta_1 + \eta_1), \sin(\theta_1 + \eta_1), \cos(\theta_2 + \eta_2), \sin(\theta_2 + \eta_2)]^T$ where η_1, η_2 followed a wrapped normal distribution of variance σ^2.

The network output on the original un-noised inputs smoothed out near the class boundaries for networks trained with larger noise levels (Fig. 3a, *top*). Interestingly, the curvature near the class centres decreased, even below zero past a certain noise level (Fig. 3a, *bottom*; Fig. 3b). This suggests that after a certain noise level, small changes in either input angles near the class centres do not lead to similar changes in the representation. Moreover, the value of the metric decreased, near the class boundaries, and so did its change over the manifold, suggesting less warping (Fig. 3c). This effect was correlated with the model output learning the flatter posterior distribution of the output (Fig. 3d).

Thus, noise during training affects representational geometry of discrete computations on continuous manifolds. In particular, the continuous decrease in warping with noise reflects the reduced confidence of the network in the output, as quantified by a flatter posterior distribution. Furthermore, this is accompanied by a qualitative change from positive to negative curvature past a certain noise threshold.

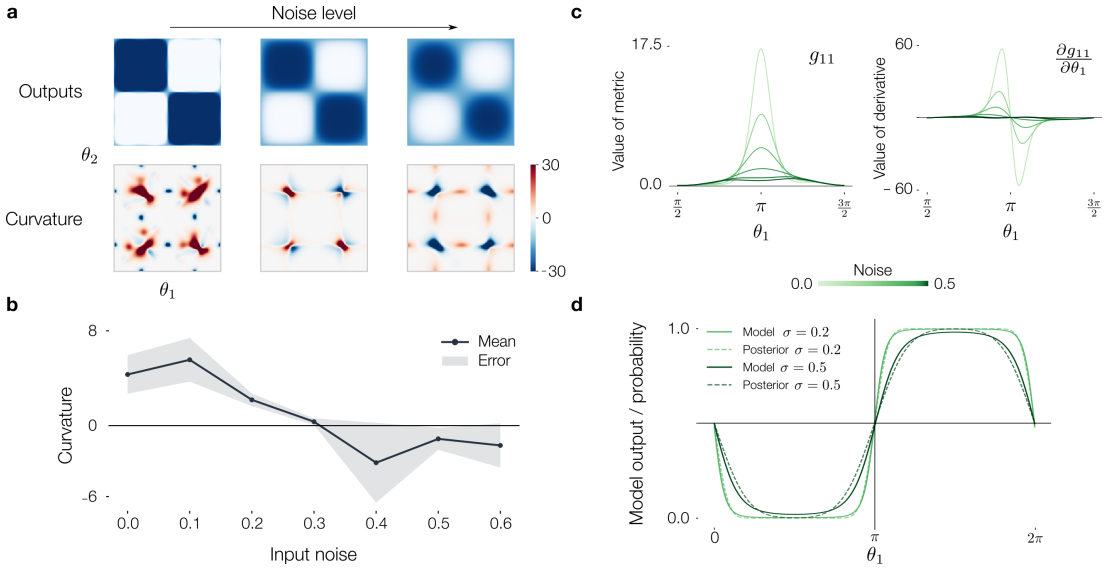

Figure 3: **The metric and curvature encode the output posterior distribution. a.** Outputs and curvatures of rich networks trained on XOR with different amounts of input noise during training. Increasing noise increases uncertainty near the boundaries, leading to larger regions of outputs close to 0.5. Curvature decreases with noise. **b.** Mean curvature vs. input noise. Curvature becomes more negative for larger input noises. **c.** Metric and change in metric in the θ_1 direction for fixed $\theta_2 = \frac{\pi}{4}$ for different noises. In noisy models, the metric changes less quickly across the boundaries. **d.** Outputs learned by models trained with different levels of noise for fixed θ_2 and analytic predictions across the boundary. The learned distribution closely matches the expected posterior distribution and the slope of the posterior is less steep for larger values of noise.

6. Conclusion

The manifold hypothesis posits that data lie on low-dimensional manifolds. Yet, neural networks must often perform fundamentally discrete computations on these continuous input manifolds, whether classifying samples from a manifold of images in vision tasks or making discrete actions over continuous state spaces in RL tasks. Here, we showed that the hidden layer geometry reflected a sequential partitioning of these computations into i) discretisation of continuous variables and ii) logical computations on the discretised variables. Furthermore, we demonstrated a link between these effects and different learning regimes, as well as Bayesian computations.

While we focused our analyses on simple networks trained on hand-crafted input manifolds, future work could attempt to extend these analyses by considering more complex architectures trained on real-world data. Indeed, the tools leveraged here scale to high-dimensional networks and inputs. Additionally, while this study was mostly descriptive, future work could take a prescriptive approach by manipulating representational geometry via constraints on the metric during training, in order to design more performant models.

References

Saduf Afzal and M. Wani. Comparative study of high speed back- propagation learning algorithms. 6:34–40, 2014. doi: 10.5815/ijmecs.2014.12.05.

Mathieu Aubry and Bryan C Russell. Understanding deep features with computer-generated imagery. In *Proceedings of the IEEE international conference on computer vision*, pages 2875–2883, 2015.

Alessandro Benfenati and Alessio Marta. A singular riemannian geometry approach to deep neural networks i. theoretical foundations. *Neural Networks*, 158:331–343, 2023. ISSN 0893-6080. doi: https://doi.org/10.1016/j.neunet.2022.11.022.

SueYeon Chung and Larry F Abbott. Neural population geometry: An approach for understanding biological and artificial neural networks. *Current opinion in neurobiology*, 70: 137–144, 2021.

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

Prafulla Dhariwal and Alexander Nichol. Diffusion models beat gans on image synthesis. *Advances in neural information processing systems*, 34:8780–8794, 2021.

Timo Flesch, Keno Juechems, Tsvetomira Dumbalska, Andrew Saxe, and Christopher Summerfield. Orthogonal representations for robust context-dependent task performance in brains and neural networks. *Neuron*, 110(7):1258–1270, 2022.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. Adaptive Computation and Machine Learning series. MIT Press, 2016. ISBN 978-0-262-03561-3.

Michael Hauser and Asok Ray. Principles of riemannian geometry in neural networks. *Advances in neural information processing systems*, 30, 2017.

Sue Ann Koay, Adam S Charles, Stephan Y Thiberge, Carlos D Brody, and David W Tank. Sequential and efficient neural-population coding of complex task information. *Neuron*, 110(2):328–349, 2022.

Kenneth Li, Aspen K Hopkins, David Bau, Fernanda Viégas, Hanspeter Pfister, and Martin Wattenberg. Emergent world representations: Exploring a sequence model trained on a synthetic task. *ICLR*, 2023.

Valerio Mante, David Sussillo, Krishna V Shenoy, and William T Newsome. Context-dependent computation by recurrent dynamics in prefrontal cortex. *nature*, 503(7474): 78–84, 2013.

Jiayuan Mao, Chuang Gan, Pushmeet Kohli, Joshua B Tenenbaum, and Jiajun Wu. The neuro-symbolic concept learner: Interpreting scenes, words, and sentences from natural supervision. *International Conference on Learning Representations*, 2019.

Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

Dhagash Mehta, Xiaojun Zhao, Edgar A. Bernal, and David J. Wales. Loss surface of XOR artificial neural networks. 97(5):052307, 2018. doi: 10.1103/PhysRevE.97.052307. Publisher: American Physical Society.

Marvin Minsky and Seymour Papert. An introduction to computational geometry. *Cambridge tiass., HIT*, 479(480):104, 1969.

Jonas Paccolat, Leonardo Petrini, Mario Geiger, Kevin Tyloo, and Matthieu Wyart. Geometric compression of invariant manifolds in neural networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2021(4):044001, April 2021. ISSN 1742-5468. doi: 10.1088/1742-5468/abf1f3.

Matthew F Panichello and Timothy J Buschman. Shared mechanisms underlie the control of working memory and attention. *Nature*, 592(7855):601–605, 2021.

Arthur Pellegrino, Heike Stein, and N Alex Cayco-Gajic. Dimensionality reduction beyond neural subspaces with slice tensor component analysis. *Nature Neuroscience*, 27(6):1199–1210, 2024.

Felix Petersen, Christian Borgelt, Hilde Kuehne, and Oliver Deussen. Deep differentiable logic gate networks. *Advances in Neural Information Processing Systems*, 35:2006–2018, 2022.

Sandra Reinert, Mark Hübener, Tobias Bonhoeffer, and Pieter M Goltstein. Mouse prefrontal cortex represents learned rules for categorization. *Nature*, 593(7859):411–417, 2021.

Andrew M Saxe, James L McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*, 2013.

Hava T Siegelmann and Eduardo D Sontag. On the computational power of neural nets. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 440–449, 1992.

Andreas Stöckel and Chris Eliasmith. Computational properties of multi-compartment LIF neurons with passive dendrites. 2(2):024011, 2022. ISSN 2634-4386. doi: 10.1088/2634-4386/ac724c. Publisher: IOP Publishing.

Guy Tennenholtz and Shie Mannor. Uncertainty estimation using riemannian model dynamics for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 35:19008–19021, 2022.

Yang Xie, Peiyao Hu, Junru Li, Jingwen Chen, Weibin Song, Xiao-Jing Wang, Tianming Yang, Stanislas Dehaene, Shiming Tang, Bin Min, et al. Geometry of sequence working memory in macaque prefrontal cortex. *Science*, 375(6581):632–639, 2022.

Andrew J Zimnik and Mark M Churchland. Independent generation of sequence elements by motor cortex. *Nature neuroscience*, 24(3):412–424, 2021.

Appendix A. Geometry of Computations in Logic Gate Tasks

A.1. 1 Layer XOR vs 2 Layer XOR

We trained networks on the XOR task with 1 and 2 layers and a flat torus input manifold. The 2 layer network (Fig. A.1, *right*) decomposed input-manifold specific computations and task-specific computations across layers, with the first hidden layer metric showing the characteristic discretisation pattern found in AND and OR networks (Fig. 1e). The 1-layer hidden metric shows additional structure with sensitivity oscillating across the discretisation boundaries, suggesting a more complex computation is occurring. The XOR task with discrete inputs requires at least one hidden layer to solve (Minsky and Papert, 1969), so the 1-layer network must discretise and solve the logic gate in one "step".

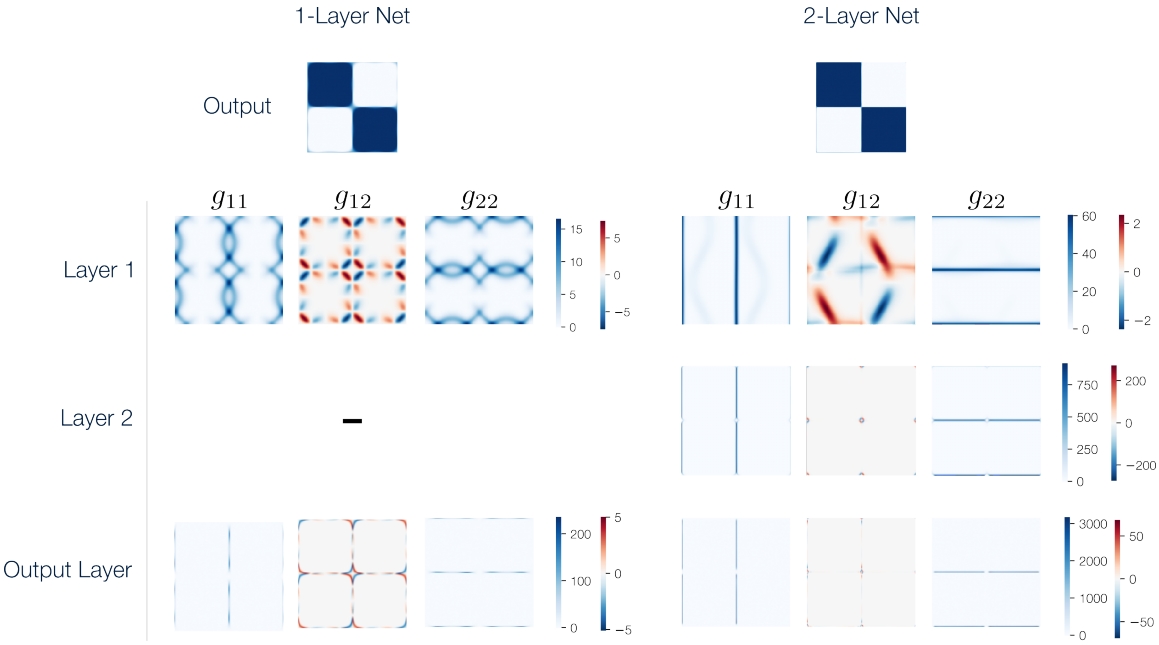

Figure S1: **Varying depth of XOR networks** *Left*: Hidden and output metrics for the 1 hidden layer XOR network. *Right*: Hidden and output metrics for the 2 hidden layer XOR network.

A.2. Analytic Expression for Hidden Metric in XOR Task

Consider a network trained on the XOR task with a tanh hidden non-linearity and input weights $W \in \mathbb{R}^{K\times 4}$ with components $w_{a,b}$ for the weight between the b^{th} input and the a^{th} hidden neuron. Inputs θ_1 and θ_2 are embedded on a flat torus

$$\mathbf{x} = [\cos(\theta_1), \sin(\theta_1), \cos(\theta_2), \sin(\theta_2)]^\top, \tag{1}$$

such that the target outputs are $\text{XOR}(0 \le \theta_1 < \pi, 0 \le \theta_2 < \pi)$. The hidden layer activations are $\mathbf{z} = \tanh(\mathbf{u}) = \tanh(W\mathbf{x}) \in \mathbb{R}^K$. We first calculate the Jacobian of the hidden

activations with respect to the inputs:

$$\begin{aligned} J_{\mathbf{z}} &= \begin{bmatrix} \frac{\partial \mathbf{z}}{\partial \theta_1} & \frac{\partial \mathbf{z}}{\partial \theta_2} \end{bmatrix} \\ &= \begin{bmatrix} \frac{d\mathbf{z}}{d\mathbf{u}}\frac{\partial \mathbf{u}}{\partial \theta_1} & \frac{d\mathbf{z}}{d\mathbf{u}}\frac{\partial \mathbf{u}}{\partial \theta_2} \end{bmatrix} \\ &= \operatorname{diag}(\operatorname{sech}^2(\mathbf{u})) \begin{bmatrix} \frac{d\mathbf{u}}{d\mathbf{x}}\frac{\partial \mathbf{x}}{\partial \theta_1} & \frac{d\mathbf{u}}{d\mathbf{x}}\frac{\partial \mathbf{x}}{\partial \theta_2} \end{bmatrix} \\ &= \operatorname{diag}(\operatorname{sech}^2(W\mathbf{x}))WJ_{\mathbf{x}} \end{aligned}$$

where from Equation 1 we calculate:

$$J_{\mathbf{x}} = \begin{bmatrix} -\sin(\theta_1) & 0 \\ \cos(\theta_1) & 0 \\ 0 & -\sin(\theta_2) \\ 0 & \cos(\theta_2) \end{bmatrix}. \tag{2}$$

The metric tensor of the hidden layer is thus:

$$\begin{aligned} G_{\mathbf{z}} &= J_{\mathbf{z}}^\top J_{\mathbf{z}} \\ &= J_{\mathbf{x}}^\top W^\top \operatorname{diag}(\operatorname{sech}^4(W\mathbf{x}))WJ_{\mathbf{x}}. \end{aligned} \tag{3}$$

Which can be written in component form as:

$$\begin{aligned} G_{ij} &= \left(J_{\mathbf{x}}^\top W^\top DWJ_{\mathbf{x}}\right)_{ij} \\ &= \sum_{k=1}^{K} (WJ_{\mathbf{x}})_{ki}(WJ_{\mathbf{x}})_{kj}D_{kk} \end{aligned} \tag{4}$$

where $D = \operatorname{diag}(\operatorname{sech}^4(W\mathbf{x}))$. Assume $w_{k1} = w_{k3} = 0$, then

$$(WJ_{\mathbf{x}})_{k1} = w_{k2}\cos(\theta_1), \quad (WJ_{\mathbf{x}})_{k2} = w_{k4}\cos(\theta_2) \tag{5}$$

and the pre-activations are

$$u_k = (W\mathbf{x})_k = w_{k2}\sin(\theta_1) + w_{k4}\sin(\theta_2). \tag{6}$$

Substitute (5)-(6) into (4):

$$\begin{aligned} G_{ij} &= \sum_{k=1}^{K} (WJ_{\mathbf{x}})_{ki}(WJ_{\mathbf{x}})_{kj}\operatorname{sech}^4(u_k) \\ &= \sum_{k=1}^{K} (w_{k,2i}\cos(\theta_i))(w_{k,2j}\cos(\theta_j))\operatorname{sech}^4[w_{k2}\sin(\theta_1) + w_{k4}\sin(\theta_2)] \\ &= \cos(\theta_i)\cos(\theta_j)\sum_{k=1}^{K} w_{k,2i}w_{k,2j}\operatorname{sech}^4[w_{k2}\sin(\theta_1) + w_{k4}\sin(\theta_2)] \\ &= \cos(\theta_i)\cos(\theta_j)g^{(ij)}(\theta_1, \theta_2) \end{aligned} \tag{7}$$

where $2i \in \{2, 4\}$ corresponds to the input index of θ_i ($i = 1 \to 2, i = 2 \to 4$).

To observe how the Gaussian curvature depends on this expression, we use the Brioschi formula which has the form:

$$K = \frac{M(\theta_1, \theta_2)}{(\det(\mathrm{G}))^2}. \tag{8}$$

Substituting (7) into (8) yields:

$$\begin{aligned} K &= \frac{M(\theta_1, \theta_2)}{\left[\cos^2(\theta_1)\cos^2(\theta_2) g^{(11)}(\theta_1, \theta_2) g^{(22)}(\theta_1, \theta_2) - \cos^2(\theta_1)\cos^2(\theta_2) g^{(12)}(\theta_1, \theta_2)^2\right]^2} \\ &= \frac{1}{\cos^4(\theta_1)\cos^4(\theta_2)} \frac{M(\theta_1, \theta_2)}{\left[g^{(11)} g^{(22)} - (g^{(12)})^2\right]^2} \\ &= \frac{M'(\theta_1, \theta_2)}{\cos^4(\theta_1)\cos^4(\theta_2)} \end{aligned} \tag{9}$$

Appendix B. Rich and lazy learning geometry

Rich networks trained on XOR had low-dimensional hidden manifolds, and structured Gram matrices, whereas lazy networks had high-dimensional hidden manifolds and near-orthogonal projections of inputs. This difference was also captured by the hidden layer metrics. Rich networks had localised sensitivity to inputs near the boundaries (Fig. S2a), converging to the small width solutions (Fig. S1, *left*), whereas lazy networks had random metrics with large magnitude across the manifold (Fig S2b). Output metrics in both networks showed sensitivity close to task boundaries, indicating that a good solution was found. The loss curves of the rich network show a slight initial delay in learning compared to the lazy network (Fig.S2) consistent with the prediction of a slower convergence in rich networks (Saxe et al., 2013), although this effect is small.

We also analysed the geometry of rich and lazy networks in the AND task, observing better generalisation (Fig S2d), a lower-dimensional hidden manifold (Fig S2e), and more structured and larger curvature (Fig S2f), consistent with the XOR results. The rich network's curvature was negative in one quadrant and positive in the other three, mirroring the target outputs. This suggests the curvature pulls apart unlike classification regions on the hidden manifold.

We tested the robustness to noise of XOR networks for a range of weight initialisations (Fig S2g). Noise at the level of the task variables θ_1, θ_2 showed no difference in robustness between learning regimes (S2h), because the mapping from task variables to outputs was approximately identical irrespective of internal geometry. Isotropic noise at the level of embedded inputs $\mathbf{x}(\theta_1, \theta_2)$ showed differences in robustness (Fig S2g). Here, task were embedded first on a 4-D flat torus, and then by an orthogonal projection to a higher-dimensional space. The difference in robustness increased with embedding dimension. High-dimensional spaces containing low-dimensional manifolds have a large number of directions orthogonal (task-irrelevant) to the manifold, which increases with embedding dimension. Rich networks are known to compress task-irrelevant dimensions (Paccolat et al., 2021) and so become increasingly robust to noise off the manifold as the number of orthogonal dimensions increases.

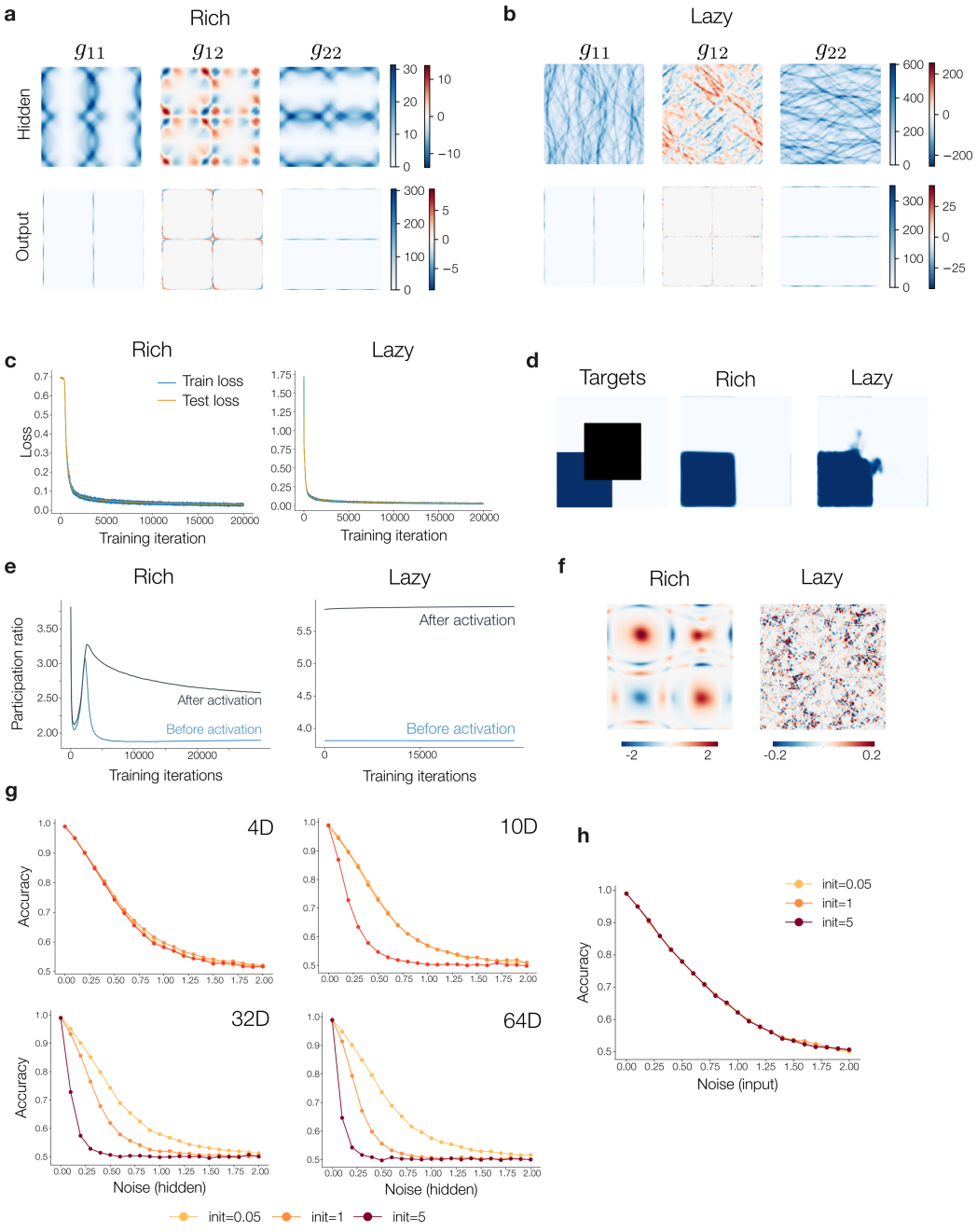

Figure S2: **Rich and lazy geometry a.** The hidden and output metric components for the XOR network in the rich regime. **b.** Metric components for the lazy XOR network. **c.** Loss curves for rich and lazy XOR networks. **d.** Outputs of rich and lazy AND networks trained with a portion of the input space held out during training. Rich networks generalise better to unseen inputs than lazy networks. **e.** Participation ratios of rich and lazy AND networks across training. **f.** The hidden layer Gaussian curvature across the manifold in rich and lazy networks. **g.** Accuracy as a function of variance of input embedding noise for networks trained with varying weight initialisations and increasing input embedding dimension. **h.** Accuracy as a function of variance of task variable noise for varying weight initialisations.

Appendix C. A Bayesian Model of Noise

To study whether networks learn approximately Bayesian inference under training noise, we computed the posterior for a simpler 1-dimensional classification problem on a circle.

We consider a random variable δ uniformly distributed on a circle, representing an angle in $[-\pi, \pi)$. We observe a noisy measurement $c = \delta + \eta$, where the noise η is drawn from a zero-mean Gaussian distribution, $\eta \sim \mathcal{N}(0, \sigma^2)$.

Our goal is to determine the probability that δ is in the upper semi-circle, $[0, \pi)$, given the measurement c. We define a binary variable A:

$$A = \begin{cases} +1 & \text{if } 0 \le \delta < \pi \\ -1 & \text{if } -\pi \le \delta < 0 \end{cases}$$

We want to find the posterior probability $P(A = 1|c)$. From Bayes' theorem, and given that the priors $P(A = 1)$ and $P(A = -1)$ are both 0.5, the posterior is:

$$P(A = 1|c) = \frac{P(c|A = 1)}{P(c|A = 1) + P(c|A = -1)}$$

Likelihoods For a circular variable, the Gaussian noise wraps around the circle. The conditional probability of observing c given δ is given by the wrapped normal distribution:

$$P(c|\delta) = \sum_{k=-\infty}^{\infty} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(c - \delta - 2\pi k)^2}{2\sigma^2}\right)$$

Likelihood for A=1 We find $P(c|A = 1)$ by marginalizing over $\delta \in [0, \pi)$. Given $A = 1$, the PDF $p(\delta|A = 1) = 1/\pi$.

$$\begin{aligned} P(c|A = 1) &= \int_0^{\pi} P(c|\delta) p(\delta|A = 1) d\delta \\ &= \frac{1}{\pi} \int_0^{\pi} \sum_{k=-\infty}^{\infty} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(c - \delta - 2\pi k)^2}{2\sigma^2}\right) d\delta \\ &= \frac{1}{\pi\sqrt{2\pi\sigma^2}} \sum_{k=-\infty}^{\infty} \int_0^{\pi} \exp\left(-\frac{(c - \delta - 2\pi k)^2}{2\sigma^2}\right) d\delta \end{aligned}$$

The integral of a Gaussian is related to the error function, $\mathrm{erf}(z)$. Evaluating the integral gives:

$$\begin{aligned} P(c|A = 1) &= \frac{1}{2\pi} \sum_{k=-\infty}^{\infty} \left[\mathrm{erf}\left(\frac{\delta - c + 2\pi k}{\sqrt{2}\sigma}\right)\right]_0^{\pi} \\ &= \frac{1}{2\pi} \sum_{k=-\infty}^{\infty} \left[\mathrm{erf}\left(\frac{\pi - c + 2\pi k}{\sqrt{2}\sigma}\right) - \mathrm{erf}\left(\frac{-c + 2\pi k}{\sqrt{2}\sigma}\right)\right] \\ &= \frac{1}{2\pi} \sum_{k=-\infty}^{\infty} \left[\mathrm{erf}\left(\frac{\pi - c + 2\pi k}{\sqrt{2}\sigma}\right) + \mathrm{erf}\left(\frac{c - 2\pi k}{\sqrt{2}\sigma}\right)\right] \end{aligned}$$

Likelihood for A=-1 Similarly, for $A = -1$, we marginalize over $\delta \in [-\pi, 0)$, where $p(\delta|A = -1) = 1/\pi$.

$$\begin{aligned}
P(c|A=-1) &= \frac{1}{\pi}\int_{-\pi}^{0}\sum_{k=-\infty}^{\infty}\frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{(c-\delta-2\pi k)^2}{2\sigma^2}\right)d\delta \\
&= \frac{1}{2\pi}\sum_{k=-\infty}^{\infty}\left[\operatorname{erf}\left(\frac{\delta-c+2\pi k}{\sqrt{2}\sigma}\right)\right]_{-\pi}^{0} \\
&= \frac{1}{2\pi}\sum_{k=-\infty}^{\infty}\left[\operatorname{erf}\left(\frac{-c+2\pi k}{\sqrt{2}\sigma}\right)-\operatorname{erf}\left(\frac{-\pi-c+2\pi k}{\sqrt{2}\sigma}\right)\right] \\
&= \frac{1}{2\pi}\sum_{k=-\infty}^{\infty}\left[\operatorname{erf}\left(\frac{\pi+c-2\pi k}{\sqrt{2}\sigma}\right)-\operatorname{erf}\left(\frac{c-2\pi k}{\sqrt{2}\sigma}\right)\right]
\end{aligned}$$

Final Posterior Probability Substituting the likelihoods into the posterior formula, the $\frac{1}{2\pi}$ factor cancels. The denominator is the sum of the numerators of the two likelihoods.

$$\begin{aligned}
\text{Numerator} &= \sum_{k=-\infty}^{\infty}\left[\operatorname{erf}\left(\frac{\pi-c+2\pi k}{\sigma\sqrt{2}}\right)+\operatorname{erf}\left(\frac{c-2\pi k}{\sigma\sqrt{2}}\right)\right] \\
\text{Denominator} &= \sum_{k=-\infty}^{\infty}\left[\left(\operatorname{erf}\left(\frac{\pi-c+2\pi k}{\sigma\sqrt{2}}\right)+\operatorname{erf}\left(\frac{c-2\pi k}{\sigma\sqrt{2}}\right)\right)\right. \\
&\qquad \left.+\left(\operatorname{erf}\left(\frac{\pi+c-2\pi k}{\sigma\sqrt{2}}\right)-\operatorname{erf}\left(\frac{c-2\pi k}{\sigma\sqrt{2}}\right)\right)\right] \\
&= \sum_{k=-\infty}^{\infty}\left[\operatorname{erf}\left(\frac{\pi-c+2\pi k}{\sigma\sqrt{2}}\right)+\operatorname{erf}\left(\frac{\pi+c-2\pi k}{\sigma\sqrt{2}}\right)\right]
\end{aligned}$$

The final expression for the posterior probability is:

$$P(A=1|c) = \frac{\sum_{k=-\infty}^{\infty}\left[\operatorname{erf}\left(\frac{\pi-c+2\pi k}{\sigma\sqrt{2}}\right)+\operatorname{erf}\left(\frac{c-2\pi k}{\sigma\sqrt{2}}\right)\right]}{\sum_{k=-\infty}^{\infty}\left[\operatorname{erf}\left(\frac{\pi-c+2\pi k}{\sigma\sqrt{2}}\right)+\operatorname{erf}\left(\frac{\pi+c-2\pi k}{\sigma\sqrt{2}}\right)\right]}$$

where range of the index k controls how many "wrap-arounds" of the circle are considered. The term in the sum corresponding to $k = 0$ is equivalent to the posterior without periodic boundaries (i.e. a line). The circular posterior for $k = \{-1, 0, 1\}$, alongside the distribution learned by the network is plotted in Figure 3d.