

Supplementary Materials

S7 The representational metric of dynamical systems

In this section we introduce the mathematical framework used throughout the main manuscript. In order for this work to be self-contained, we first briefly review the relevant concepts of differential and Riemannian geometry. To provide intuition in a mathematically simplified setting, we study deep neural networks as a series of geometric transformations, showing that this view can provide insights into the computations performed by the network. We then provide our main contribution: a full derivation of the geometry of dynamical systems’ manifolds. Finally, we show a concrete application of these mathematical results for continuous-time RNNs.

S7.1 Differential geometry of deep neural networks

The results of this subsection mostly follow from elementary differential geometry (e.g. we refer the reader to¹), and we therefore state them without derivation.

The topological and differentiable structure of deep neural networks. In this section we consider a generic depth- l neural network of the form:

$$\varphi_l = f_l \circ f_{l-1} \circ \dots \circ f_1 : \mathbb{R}^{n_0} \rightarrow \mathbb{R}^{n_l}$$

where f_k is the map from hidden layer $k - 1$ to hidden layer k . For example, $f_k(\mathbf{z}_{k-1}) = \tanh(W_k \mathbf{z}_{k-1} + \mathbf{b}) = \mathbf{z}_k$ where $\mathbf{z}_k \in \mathbb{R}^{n_k}$ is the vector of activations of the neurons in the k th hidden layer. Following previous work,² we consider the input to this network as living on a low-dimensional manifold:

$$\varphi_k(\psi(p)) = \varphi_k(\mathbf{x}) = \mathbf{z}_k, \quad p \in \mathcal{M}$$

where ψ is an embedding of \mathcal{M} in \mathbb{R}^{n_0} . The following proposition gives a clear characterisation of the topology of the manifold over which \mathbf{z} lies. In particular, if $\varphi_k : \mathbb{R}^{n_0} \rightarrow \mathbb{R}^{n_k}$ is i) continuous ii) invertible and iii) its inverse is continuous then it is a *homeomorphism*.

Proposition S7.1. *Let $\mathbf{x} \in \mathbb{R}^{n_0}$ be constrained to $\psi(\mathcal{M})$, an embedding of \mathcal{M} in \mathbb{R}^{n_0} , and let φ_k be as defined above. If φ_k is a homeomorphism then $\mathbf{z}_k = \varphi_k(\mathbf{x})$ is also constrained to \mathcal{M} .*

Importantly, φ_k is a simple real function, and its invertibility and continuity can be verified via basic calculus. Thus, \mathbf{x} and \mathbf{z} are on the same manifold, simply represented differently in \mathbb{R}^{n_0} and \mathbb{R}^{n_k} .

In general, the invertibility and continuity of the network transformation are mostly independent of the parameters of the network. For example, if the network is of the form $f_k = \phi(W_k \mathbf{x} + \mathbf{b})$ where ϕ is an element-wise nonlinearity, then if ϕ is (element-wise) homeomorphic (e.g. \tanh , softplus , leaky-ReLU , but not ReLU), and W_k is full rank, and the network doesn’t have a bottleneck layer, then these conditions are satisfied. In particular, W_k is full rank with probability 1 if it is initialised with entry-wise i.i.d. Gaussian or uniform weights. Thus, provided the rank of W_k does not change over training, the topology of a trained network will match that at initialisation, and hence does not reflect the computations learned by the network. However, in some instances the topology may indeed change: recent work has highlighted the ubiquity of low-rank networks,³ and depending on the objective, the learned weights may be of a rank lower than the minimum embedding dimensionality of the manifold — trivially one could think of a network trained to map all its inputs to 0. Nevertheless, in general it is the geometry of the neural representation that provides most insight into the computations learned by the network, as this geometry depends continuously on the parameters of the network whereas the topology is invariant to large classes of parameter changes.

There are multiple routes to characterising the geometry of a manifold. Here we will specifically consider *smooth manifolds*. In particular, such smooth manifolds have a tangent space $T_p \mathcal{M}$ at any point $p \in \mathcal{M}$.

Proposition S7.2. *Let $\varphi_k : \mathbb{R}^{n_0} \rightarrow \mathbb{R}^{n_k}$ differentiable with Jacobian $J_k \in \mathbb{R}^{n_k \times n_0}$. If J_k is full matrix rank and ψ is an embedding, then $d(\varphi_k \circ \psi) = J_k d\psi(\mathbf{v}) : T_p \mathcal{M} \rightarrow T_{\varphi_k(\psi(p))}$ is surjective and \mathcal{M} is an immersed submanifold of \mathbb{R}^{n_k} .*

We note here that the differentiability and homeomorphicity conditions are overlapping but not restrictions of one another. In particular, even when φ_k is not bijective $d\varphi_k$ can still be surjective

and therefore its Jacobian full rank. Similarly, φ_k bijective and differentiable does not imply $d\varphi_k$ surjective.

Riemannian geometry provides insights into network computations. So far the structure we've explored had no geometry, in that we did not define a notion of distance between points on the manifold. For this, we must add a *metric* to the manifold.

Definition S7.1. A *metric* g is a function defining the inner product between vectors of the tangent space of a manifold. The metric is Riemannian if it satisfies: $g : T_p\mathcal{M} \times T_p\mathcal{M} \rightarrow \mathbb{R}$ such that $g(\mathbf{v}, \mathbf{v}) > 0$ if $\mathbf{v} \neq \mathbf{0} \in T_p\mathcal{M}$ and $g(\mathbf{v}_1, \mathbf{v}_2) = g(\mathbf{v}_2, \mathbf{v}_1)$

Importantly such a metric allows defining the geodesic distance between points:

Definition S7.2. Let $\gamma : [0, T] \rightarrow \mathcal{M}$ a smooth curve on \mathcal{M} where $T \in \mathbb{R}$. In particular, this implies that $\dot{\gamma}(t) \in T_{\gamma(t)}\mathcal{M}$, and we can define the *length* of γ as:

$$L(\gamma) = \int_0^T g(\dot{\gamma}(t), \dot{\gamma}(t)) dt.$$

Furthermore, the geodesic distance between $p_1 \in \mathcal{M}$ and $p_2 \in \mathcal{M}$ is defined as:

$$d(p_1, p_2) = \inf_{\gamma} L(\gamma), \quad \text{constraining } \gamma(0) = p_1, \gamma(T) = p_2$$

This definition was central to our study of dynamical systems in the main text.

In figure 2 the manifold considered was the circle $\mathcal{M} = S^1$ and representing different points on the circle via their angle ($p = \theta$), the embedding of the circle in the 2-d input space was $\mathbf{x} = \psi(\theta) = [\cos(\theta), \sin(\theta)]$. However, we could have chosen a different embedding, say $\mathbf{x} = [(1 + 0.1 \sin(\theta)) \cos(\theta), \sin(\theta)]$, which in general will affect the geometry of the hidden representation \mathbf{z}_k . Indeed, suppose the target output is independent of the input geometry — as is the case in figure 2 where it is only the point on the circle that determines the output — then the network must perform different computations on these different input representations to perform the same output.

To characterise these transformations, we can *pull back* the usual metric defined on \mathbb{R}^{n_k} , which is the standard dot product, to the manifold \mathcal{M} .

Definition S7.3. The *pullback metric* is defined as:

$$g(\mathbf{v}_1, \mathbf{v}_2) = (d(\varphi_k \circ \psi)(\mathbf{v}_1)) \cdot (d(\varphi_k \circ \psi)(\mathbf{v}_2)), \quad \mathbf{v}_1, \mathbf{v}_2 \in T_p\mathcal{M}$$

in particular $d(\varphi_k \circ \psi)(\mathbf{v}_i) \in \mathbb{R}^{n_k}$ and \cdot is the standard Euclidean dot product.

Thus, we can define a metric on \mathcal{M} by taking tangent vectors in $T_p\mathcal{M}$, pushing them forward through the network, where at the k th layer they become vectors of $T_p\mathbb{R}^{n_k} = \mathbb{R}^{n_k}$, such that we can then simply take their dot product. The value of this dot product is our definition of the inner product between the original tangent vectors in \mathcal{M} . The pullback metric represents how the tangent spaces of the manifold are transformed through the layers of the network and therefore the computations of the network. In particular we can more explicitly write the pullback metric as:

$$g(\mathbf{v}_1, \mathbf{v}_2) = (J_k d\psi(\mathbf{v}_1)) \cdot (J_k d\psi(\mathbf{v}_2))$$

with

$$J_k = W_k \text{diag}(\phi'(\mathbf{z}_{k-1})) \dots W_1 \text{diag}(\phi'(\mathbf{x})) d\psi(\mathbf{v}_1)$$

where $J_k \in \mathbb{R}^{n_k \times n_0}$ is the Jacobian of φ_k . This highlights the earlier claim that the geometry, as defined via the pullback metric, is directly dependent on the parameters W_1, \dots, W_l of the network.

Geodesic gridlines and computations. In our networks the input metric was uniform, but in general data of the world maybe be unevenly distributed on the manifold, which will affect the learned parameters, and therefore the learned representation. In the case of our simple network the loss function was more formally defined as:

$$l = \int_{\mathcal{M}} \|\varphi_l(\psi(\theta)) - \mathbf{y}_{\text{target}}\|^2 \sqrt{\det(h)} d\theta$$

where $h : T_p\mathcal{M} \times T_p\mathcal{M}$ is the metric on the input manifold. That is, some inputs are more heavily weighted in the loss if space is more spread around them. For example, in a probabilistic setting where

instead of integrating over all possible inputs only a finite sample of them is drawn at each iterations,
 $\sqrt{\det(h)}$ can be seen as a probability distribution (up to normalisation) on the input manifold — so
that some input-output pairs are drawn more frequently than others. In our case all angles were equal,
so the input metric was uniform:

$$l = c \int_{\mathcal{M}} \|\varphi_l(\psi(\theta)) - \mathbf{y}_{target}\|^2 d\theta$$

for some $c \in \mathbb{R}_+$. But more generally non-uniform metrics on the manifold of input will lead to
different learned representations since the loss is different. For example, a class being more frequent
in a binary classification task may change the representation: if in figure 2 we had sampled more
frequently angles $[0, \pi)$ than $(\pi, 2\pi)$, it would have led to an asymmetric stretching of space on each
side of the class boundaries.

Thus, the pullback metric g gives insights into the learned representation, but the learned representa-
tion is dependent on the metric of the input manifold h . Using the definition of geodesic distance,
both g and h can be used to measure the distance between points on the manifold. But what points to
choose to measure the distance between? In 2, 4 and 6 we showed that an insightful reflection of
the computations done by the network was provided by the computing the geodesic distance using
 g between points that were evenly apart under the h metric. That is define points p_1, p_2 such that
 $d^h(p_1, p_2) = 1$ and study $d^g(p_1, p_2)$ where d^h and d^g are the geodesic distance functions under the
 h and g metric respectively.

Coordinates on the input manifold provide a natural input metric. So far our treatment of the
topology, smooth structure and geometry of deep neural networks has been relatively free of a choice
of a coordinate system for \mathcal{M} . An m -dimensional manifold is defined as a collection of pieces of
 \mathbb{R}^m glued together via the so-called transition functions. However, different choices of pieces of \mathbb{R}^m
can give rise to the same manifold. A trivial example is that of a line manifold: the open interval
 $(0, 3)$ and $(10, 16)$ are the same manifold, but endowed with different coordinates; there is an explicit
diffeomorphism between them given by $F(p) = 10 + 2p$.

In many cases, there is a meaningful coordinate system — at least around a point — on the input
manifold. A manifold of images may for example be characterised by the position of an object in the
image or its orientation.⁴ It is insightful to understand how a neural network represents these different
features. When a specific coordinate system is given, a natural matrix representation of the metric
can be defined.

Definition S7.4. A given coordinate system² $\mathbf{x}(p) = (x_1, x_2, \dots, x_m)$ where $\mathbf{x} : \mathcal{M} \rightarrow \mathbb{R}^m$ induces
a natural basis of the tangent space such that $\mathbf{v} \in T_p \mathcal{M}$ can be written as a linear combination of
them:³

$$\mathbf{v} = \sum_{i=1}^m v^i \frac{\partial}{\partial x_i}$$

In this coordinate system inner products are given by:

$$g(\mathbf{v}_1, \mathbf{v}_2) = \sum_{i,j=1}^m v_1^i G_{i,j} v_2^j$$

This highlights that the metric in a particular coordinate system can also be written as a matrix
 $G \in \mathbb{R}^{m \times m}$ and $G_{i,j} = g(\frac{\partial}{\partial x_i}, \frac{\partial}{\partial x_j})$ are its entries. Furthermore, if the manifold is embedded in the
Euclidean space:

$$G = J^T J \in \mathbb{R}^{m \times m}, \quad J = \left[\frac{\partial p}{\partial p_1}, \dots, \frac{\partial p}{\partial p_m} \right] \in \mathbb{R}^{m \times m}$$

Back to our deep neural network, the pullback metric changes from layer to layer as:

$$G_k = J_k^T d\psi(\mathbf{v})^T d\psi(\mathbf{v}) J_k = J_{f_k}^T \dots J_{f_1}^T d\psi(\mathbf{v})^T d\psi(\mathbf{v}) J_{f_1} \dots J_{f_k} = J_{f_k}^T G_{k-1} J_{f_k} \in \mathbb{R}^{m \times m}$$

²The reader may be more familiar with the notation (U, φ) for a chart, here we've simply written $\mathbf{x} = \varphi$.

³This notation defines $\frac{\partial f}{\partial x_i} = (f \circ \gamma)|_{t=1}$ for the smooth curve $\gamma(t)$ defined such that $x_i(\gamma) = x_i(p)$, ..., $x_i(p) + t$, ..., $x_m(p)$ and a smooth function $f : \mathcal{M} \rightarrow \mathbb{R}$.

where $J_{f_i} \in \mathbb{R}^{n_i \times n_{i-1}}$ is the Jacobian of f_i . This means that:

$$G_k = d(\varphi \circ \psi)(\mathbf{v})^T d(\varphi \circ \psi)(\mathbf{v}) \in \mathbb{R}^{m \times m}$$

Integrating along a curve on the manifold under the pullback metric in coordinates thus gives:

$$L(\gamma) = \int_0^T g(\dot{\gamma}(t), \dot{\gamma}(t)) dt = \int_0^T d(\varphi \circ \psi)(\dot{\gamma}(t))^T G_k d(\varphi \circ \psi)(\dot{\gamma}(t)) dt$$

Where now $d(\varphi \circ \psi)(\dot{\gamma}(t))$ is a usual derivative of a real function, and G_k a real matrix, which can both be obtained using standard calculus tools (e.g. auto-differentiation).

S7.2 Dynamical systems' topology

We now generalise the theory of the previous subsection to dynamical systems, where the inputs are time-varying functions. As we shall see, a manifold of functions naturally gives rise to a manifold of functions that are solutions to the dynamical system. Much of the work will lie in viewing this manifold of solutions as a submanifold of the state space.

The topology of the input manifold constrains the topology of the solution to a dynamical system. Here we consider a more general setting than in the main manuscript where the dynamical system is defined on a manifold. That is:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t)), \quad \mathbf{x}(t) \in \mathcal{R}^n, \mathbf{u}(t) \in \mathcal{U}^d$$

Furthermore, consider a manifold \mathcal{M} of input functions. That is $\mathbf{u} \in \mathcal{M}$ and $\mathbf{u}(t) \in \mathcal{U}$ where \mathcal{U} is the ambient space in which the input is embedded at any particular time point. To define such a *manifold of functions* (or more precisely of curves) we rely on the following construction:

Definition S7.5. A *manifold of real curves* is a manifold \mathcal{M} such that $(\mathbf{u}, t) \in \mathcal{M} \times \mathbb{R}$ for $\mathbf{u}(t) \in \mathcal{U}$, and $\mathbf{u} \in \mathcal{M}$ is $\mathbf{u} = \pi(\mathbf{u}, t)$ for π the projection map.

On a particular chart, the function can be parametrised:

$$\mathbf{u}_{\kappa}(t), \kappa \in \mathbb{R}^m$$

For example, in section 4 of the main manuscript we've considered $\mathbf{u}_{\kappa}(t) = [\kappa_1, \kappa_2, \mathbb{1}_{\text{ctx}=1}, \mathbb{1}_{\text{ctx}=2}]$ where κ_1, κ_2 were the levels motion and colour evidence. This input, fixing one of the two discrete contexts, indeed lies on a 2-dimensional manifold with planar topology as $\kappa \in \mathbb{R}^2$. We formulate a stronger version of the theorem of the main text:

Theorem S7.3. Consider the dynamical system:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t)), \quad \mathbf{x}(t) \in \mathcal{R}^n, \mathbf{u}(t) \in \mathcal{U}^d, \mathbf{u} \in \mathcal{M}^m$$

where \mathbf{f} is at least once differentiable in its arguments, then $\mathbf{x} \in \mathcal{N}^{m+1}$ (the curve) and \mathcal{N}^{m+1} has the topology of $\mathcal{M}^m \times \mathbb{R}$.

where the superscripts indicate the dimensionality of the manifold. Theorem 3.1 of the main text is therefore just a special case of this theorem as the dimensionality of the dynamical system is directly defined by the manifold topology. To prove this theorem we need the following lemma:

Lemma S7.4. ¹ Let $\dot{\mathbf{y}} = \mathbf{g}(\mathbf{y})$, $\mathbf{y}(0) = \mathbf{y}_0$ with $\varphi(t, \mathbf{y}_0) = \mathbf{y}_0 + \int_0^t \mathbf{g}(\mathbf{y}(\tau)) d\tau$ the solution of the dynamical system at time t starting at \mathbf{y}_0 , then if \mathbf{g} is smooth, φ is a diffeomorphism.

We can now prove the main theorem:

Proof of theorem S7.3. We prove this in a particular coordinate chart of \mathcal{M} and \mathcal{R} . Then, the dynamical system can be reformulated as (by abuse of notation):

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}_{\kappa}(t)), \quad \mathbf{x}(t) \in \mathbb{R}^n, \kappa \in \mathbb{R}^m$$

We can augment this dynamical system by introducing the variable $\mathbf{y} \in \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}$ defined as:

$$\varphi(t, \kappa) = \mathbf{y}(t) = [\mathbf{x}(t), t, \kappa]$$

then:

$$\dot{\mathbf{y}} = [\dot{\mathbf{x}}, 1, \mathbf{0}] = [\mathbf{f}(\mathbf{y}_{1:n}, \mathbf{u}_{\mathbf{y}_{n+2:n+m+1}}(y_{n+1})), 1, \mathbf{0}], \quad \mathbf{y}(0) = [\mathbf{x}_0, 0, \kappa]$$

which is an autonomous dynamical system. By the theorem on solution of ODE, if the r.h.s. is at least once differentiable in its argument (which it is since \mathbf{f} is), there exists a unique solution to this dynamical system. Furthermore, by lemma S7.4, φ is a diffeomorphism and since κ is on an m -dimensional space and t in a 1-dimensional, $\mathbf{y}(t)$ is in an $m + 1$ -dimensional space. More generally \mathbf{y} is an $m + 1$ -dimensional space of curves of topology $\mathbb{R}^m \times \mathbb{R}$. Finally, \mathbf{x} and \mathbf{y} are simply related by a projection $\mathbf{x}(t) = P(\mathbf{y}(t)) = \mathbf{y}_{1:n}$.

For the sake of completeness we also show that this holds not only over a single chart but over the whole manifold. Consider a smooth transition function $F_{i,j} : \mathbb{R}^m \rightarrow \mathbb{R}^m$ from coordinates of chart i to coordinates of chart j of \mathcal{M} , and $\varphi^i(t, \kappa^i)$ the solution to the dynamical system on chart i . Then $\varphi^j(t, \kappa^j) = \varphi^i(t, J_{F_{i,j}} \kappa^i)$ where $J_{F_{i,j}}$ is the Jacobian of $F_{i,j}$. Since φ is a composition of diffeomorphisms it is a diffeomorphism. \square

Different conditions on the projection P will give different properties to \mathbf{x} , for example if P is homeomorphic then it preserves the topology of the manifold, if dP is surjective then \mathbf{x} is on $\mathcal{M} \times \mathbb{R}$ an immersed submanifold of \mathcal{R}^n . So far — and in the main text — we have mainly considered fixed initial conditions of the dynamical system but the previous proof naturally gives the following corollary.

Corollary S7.5. *If $\mathbf{x}_0 \in \mathcal{Q}^q$ and $\mathbf{u} \in \mathcal{M}^m$ then $\mathbf{y}(t) \in \mathcal{N}^{m+q+1} = \mathcal{M}^m \times \mathbb{R} \times \mathcal{Q}^q$ with $\mathbf{x}(t) = P(\mathbf{y}(t))$ so that varying the initial state only adds extra dimensionalities.*

Proof. The proof follows mutatis mutandis from that of theorem S7.3 additionally noticing that if $\mathbf{x}_0 \in \mathcal{Q}^q$ then $\mathbf{y}_0 \in \mathcal{Q}^q \times \{0\} \times \mathbb{R}^m$. \square

The effect of noise can also be studied from theorem S7.3. In general, noise processes driving differential equations are sampled from an infinite-dimensional manifold of functions (e.g. L^2). Thus, the manifold of states taken by the system under *all* realisations of the noise may in general be the entirety of the state space. However, the following corollary states that fixing the noise realisation generates a manifold:

Corollary S7.6. *Consider the stochastic differential equation $d\mathbf{x} = \mathbf{f}(\mathbf{x}, \mathbf{u}(t))dt + \mathbf{g}(\mathbf{x})d\mathcal{W}$, $\mathbf{x}_0 = \mathbf{x}(0)$ where $\mathbf{g}(\mathbf{x}) \in \mathbb{R}^{n \times q}$ is at least once differentiable in \mathbf{x} and $d\mathcal{W} \in \mathbb{R}^q$ are the increments of a Wiener process. Then if $\mathbf{u} \in \mathcal{M}$ then $\mathbf{x}(t) = P(\mathbf{y}(t))$ where P is a projection matrix and $\mathbf{y}(t) \in \mathcal{M} \times \mathbb{R}_+$.*

Proof. The proof follows a similar construction as the deterministic case. As in S7.3 let $\mathbf{y}(t) = [\mathbf{x}(t), t, \kappa]$, now $d\mathbf{y} = \bar{\mathbf{f}}(\mathbf{x}, \mathbf{u}_\kappa(t))dt + \bar{\mathbf{g}}(\mathbf{x}, \mathbf{u}_\kappa(t))d\mathcal{W}$ where $\bar{\mathbf{f}} = [\mathbf{f}, 1, \mathbf{0}]$, $\bar{\mathbf{g}} = [\mathbf{g}^T, 0]^T$. Thus \mathbf{y} follows an SDE driven by the same noise realisation. Then, using the smoothness of the solution of an SDE with respect to its initial state,⁵ the rest of the proof follow the same steps as that of S7.3. \square

This shows that under a fixed noise realisation the state of the system lies on a manifold. In general, for different noise realisations P may yield a submanifold an immersion or an embedding of $\mathcal{M} \times \mathbb{R}_+$. Future work could study the distribution of such objects under the noise, for example characterising the measure of the solutions for which it is a proper embedding or an immersion.

S7.3 Geometry as input-manifold transformations

Having characterised the geometry and differentiable structure of the manifold of dynamical systems, we next turn to their Riemannian geometry. The main insight will be that the metric on the manifold can be defined in terms of the solution to an adjoint dynamical system. This will provide both theoretical insights and a practical way to obtain the matrix representation of a metric.

Proof of theorem 3.3. The Jacobian of \mathbf{y} is given by (ignoring time dependencies):

$$J_{\mathbf{y}} = \begin{bmatrix} \frac{\partial \mathbf{x}}{\partial t} & \frac{\partial \mathbf{x}}{\partial \kappa} \\ \frac{\partial t}{\partial t} & \frac{\partial t}{\partial \kappa} \\ \frac{\partial \kappa}{\partial t} & \frac{\partial \kappa}{\partial \kappa} \end{bmatrix} = \begin{bmatrix} \mathbf{f} & \frac{\partial \mathbf{x}}{\partial \kappa} \\ 1 & \mathbf{0} \\ \mathbf{0} & I \end{bmatrix}$$

where the submatrices are concatenated along their appropriate dimensions. The term \mathbf{f} is simply the r.h.s. of the dynamical system, so it remains to compute the term in blue. Each column of this term is

of the form $\frac{\partial \mathbf{x}}{\partial \kappa_i}$, which describes how the state of the system at a particular time point changes for an infinitesimal change in one of the input variables on a particular chart. These are given by:

$$\begin{aligned}\frac{\partial \mathbf{x}}{\partial \kappa_i} &= \frac{\partial}{\partial \kappa_i} \left(\mathbf{x}_0 + \int_0^t \mathbf{f}(\mathbf{x}(\tau), \tau, \mathbf{u}_\kappa(\tau)) d\tau \right) \\ &= \int_0^t \frac{\partial}{\partial \kappa_i} \mathbf{f}(\mathbf{x}(\tau), \tau, \mathbf{u}_\kappa(\tau)) d\tau\end{aligned}$$

Now:

$$\frac{\partial \mathbf{x}}{\partial \kappa_i} = \int_0^t \frac{\partial \mathbf{f}}{\partial \mathbf{u}} \frac{\partial \mathbf{u}}{\partial \kappa_i} + \frac{d\mathbf{f}}{d\mathbf{x}} \frac{\partial \mathbf{x}}{\partial \kappa_i} d\tau$$

where we have omitted the arguments of each function for notational clarity. Taking the time derivative on both sides and noticing that $\frac{\partial \mathbf{x}}{\partial \kappa_i} \Big|_{t=0} = \mathbf{0}$ this defines a new dynamical system:

$$\frac{d}{dt} \frac{\partial \mathbf{x}}{\partial \kappa_i} = \frac{d\mathbf{f}}{d\mathbf{x}} \frac{\partial \mathbf{x}}{\partial \kappa_i} + \frac{\partial \mathbf{f}}{\partial \mathbf{u}} \frac{\partial \mathbf{u}}{\partial \kappa_i} \quad \frac{\partial \mathbf{x}}{\partial \kappa_i} \Big|_{t=0} = \mathbf{0}$$

note that this is a form of adjoint dynamics. Indeed if we define $\mathbf{a}_i(t) = \frac{\partial \mathbf{x}}{\partial \kappa_i}(t)$ and $\mathbf{h}(\mathbf{x}, t, \kappa) = \frac{\partial \mathbf{f}}{\partial \mathbf{u}} \frac{\partial \mathbf{u}}{\partial \kappa_i}$, then we obtain the final equation:

$$\dot{\mathbf{a}}_i = J_{\mathbf{f}} \mathbf{a}_i + \mathbf{h}(\mathbf{x}, t, \kappa)$$

This is a controlled dynamical system, where the control term \mathbf{h} depends on the solution of the original dynamical system $\mathbf{x}(t)$ and the input $\mathbf{u}(t)$.

□

S7.4 Some practical considerations

In this section we briefly discuss how this framework can be concretely implemented in an auto-differentiation framework such as Jax or Pytorch. The numerical solution of a dynamical system obtained with, say, an Euler solver, is differentiable with respect to the input to the dynamical system at any time point.⁶ In our work, we compute these derivatives in different ways that differ in their computational cost and accuracy:

	Adjoint dynamics	Auto-differentiation	Finite difference
Time complexity	$\mathcal{O}(n^2 t)$	$\mathcal{O}(nt)$	$\mathcal{O}(nt)$
Memory complexity	$\mathcal{O}(n)$	$\mathcal{O}(nt)$	$\mathcal{O}(nt)$
Easy implementation		✓	✓
Accurate	✓	✓	

653

In figure 3 we studied the adjoint analytically, in figures 4 and 6d–h we used auto-differentiation and in figures 6c and S9a,b finite difference.

S8 Models

In this section we provide the detailed description of the models used throughout the main manuscript. Jupyter notebooks implementing all analyses performed here are also provided.

S8.1 Two-layer deep neural network

In section 2 we considered a one-hidden-layer deep neural network given by:

$$y = \tanh(D\mathbf{z}) \in \mathbb{R}^1, \quad \mathbf{z} = \tanh(W\mathbf{x} + \mathbf{b}), \quad \mathbf{x} \in \mathbb{R}^2, W \in \mathbb{R}^{3 \times 2}, D \in \mathbb{R}^{1 \times 3}$$

The inputs \mathbf{x} are constrained to a circle plus some noise during training:

$$\mathbf{x}(\theta) = [\cos(\theta), \sin(\theta)] + \xi, \quad \theta \in [0, 2\pi), \xi \sim \mathcal{N}(\mathbf{0}, I)$$

where \mathcal{N} is the Gaussian distribution. The loss was given by:

$$\int_0^\pi \|y(\theta) - 1\|^2 d\theta + \int_\pi^{2\pi} \|y(\theta) - (-1)\|^2 d\theta$$

We evaluated these integrals by discretising $[0, 2\pi)$ into 200 even bins. The parameters were initialised as $w_{i,j}, b_i \sim \mathcal{U}(-1/2, 1/2)$, $d_{i,j} \sim \mathcal{U}(1/3, 1/3)$ where \mathcal{U} is the uniform distribution and optimised using Adam⁷ with learning rate 0.01 over 1000 iterations.

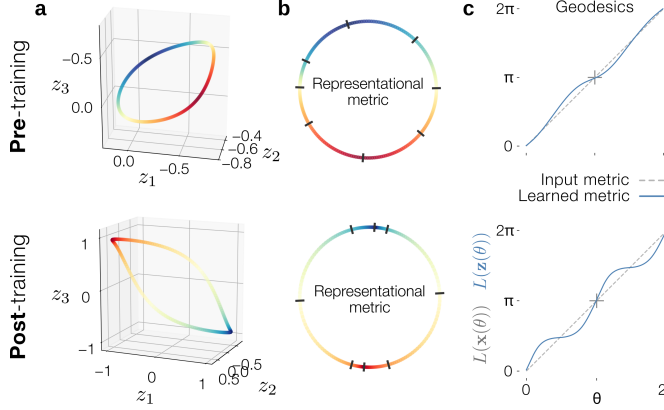


Figure S7: The representational metric changes over training in deep neural networks. Top: randomly initialised network. Bottom: network after training. **a.** Activation of the hidden layer in response all inputs. **b.** Learned representational metric visualised as gridlines on the input manifold. **c.** Arclength under either the uniform input metric (dashed line) or learned metric (blue line) integrating from $\theta = 0$ to $\theta = 2\pi$.

S8.2 E-I network

In section 3 we considered a three-neuron continuous-time RNN following Dales' law, also called an Excitatory-Inhibitory (E-I) network. The dynamics were given by:

$$\dot{\mathbf{x}} = W\phi(\mathbf{x}) - \mathbf{x} + B\mathbf{u}$$

The inputs were constant in time and lied on a 1-d line manifold and the weights were defined according to:⁸

$$W = \begin{bmatrix} a & b & -c \\ b & a & -c \\ d & d & -e \end{bmatrix}, \quad \mathbf{u} = [u, 1 - u]$$

with $a, b, c, d \in \mathbb{R}_+$ and $u \in [0, 1]$. We discretized $u \in [0, 1]$ into 200 points and evaluated the corresponding dynamical for each point. To evaluate the 200 dynamical systems we used the Heun second order numerical method⁹ until they reached steady-state, saved at 100 time points. This gave a $200 \times 100 \times 3$ mesh that is visualised in panel c of figure 3.

S8.3 Contextual decision-making under noisy inputs

In section 4 we used a classic model of contextual evidence integration.¹⁰ The model is described in the main text. The weights were initialised as $w_{ij} \sim \mathcal{N}(0, g/\sqrt{n})$ the other parameters were initialised as $b_{i,j}, c_{i,j} = \mathcal{N}(0, 1/\sqrt{2})$ and $d_{i,j} \sim \mathcal{N}(0, 1/\sqrt{n})$. We used a differentiable SDE solver with the Heun method and an adaptive step solver.⁹ To optimised the system we used Adam in Optax.¹¹ The hyper-parameters were:

n	g	σ_{input}	σ_{neuron}	lr	Iterations	Optimiser	SDE solver ⁹	PID ⁹ rtol, atol
100	0.5	0.5	0.05	5×10^{-5}	3000	Adam ⁷	Heun	$10^{-2}, 10^{-3}$

680

681 We next briefly describe the novel analyses performed here.

The basis of the tangent space. The basis of the tangent space was constructed using auto-differentiation through the ODE solver. This can be simply achieved in Jax by defining fully differentiable function $\varphi : \mathbb{R}_+ \times \mathcal{M} \rightarrow \mathbb{R}^n$ taking points on the manifold of evidences \mathcal{M} and a particular time point in \mathbb{R}_+ to return the state of the system at that time point. This function can then

be differentiated w.r.t. each of its arguments to obtain the basis of the tangent space at the state of the system at that time point. This function is a composition of i) an embedding of the manifold \mathcal{M} in \mathbb{R}^2 , which here simply consists of $\mathbf{u}(t) = [u_1, u_2]$ for $u_1, u_2 \in [-0.2, 0.2]$ ii) the evaluation of the ODE under these inputs.

Variability over noise. In figure 4a (right) and d we show the variability of the output of the network to variations to its input. For this we fix a particular Wiener process realisation \mathcal{W} and evaluate the dynamical system by either fixing $u_2 = 0$ and varying $u_1 \in [-0.2, 0.2]$ (for panel a) or fixing $u_1 = 0.2$ and varying $u_2 \in [-0.2, 0.2]$ (for panel d). The rest of our analyses were performed under a fixed realisation of the noise: the one where $d\mathcal{W}(t) = 0$ for all t . Although we note that future work could explore the distribution of the analyses presented here under the measure of this noise process.

Manifold plotting. In figure 4b and e we show slices of the manifold. In b they are evidence slices, where u_1 and u_2 are fixed and we evaluate φ varying t , in order to obtain a trajectory of the dynamical system. In e we take 5 time slices of the manifold by fixing t and evaluating φ for all $u_1, u_2 \in [-0.2, 0.2]$. The attractor planes were simply computed by letting t be large enough that the r.h.s. of the dynamical system became close to zero. We note that this is not a “plane attractor” in the sense that the state of the system would tend to those values in the absence of inputs. They are a manifold of attractor fixed points parameterised by u_1, u_2 .

The metric. Under a particular choice of coordinates — here the $[t, u_1, u_2]$ coordinates — the metric at each point on the manifold is a 3×3 positive (semi-)definite symmetric matrix. In figure 4e we plot the metric at two time points on the manifold $t = 0.2$ and $t = 10$ and for $u_1, u_2 = 0$. The results are nevertheless consistent over different values of u (Fig. S8).

Warping. In figure 4f and g we analyse the warping of the manifold across time and conditions. In panel f we compute the squared geodesic distance along either the u_1 (top) or u_2 (bottom) grid lines. For example for the top panels $\int_0^\mu \|\partial_{u_1} \varphi(u_1, u_2)\|^2 du_1$ evaluated at $\mu \in \{-0.2, \dots, -0.04, 0, 0.04, \dots, 0.2\}$. We divided this quantity by the total length of the geodesic to visualise only the effect of warping across values of the input as opposed to the overall stretch or compression of the manifold. Figure S8 shows the same analysis without this division, and integrated from -0.2 to μ (instead of 0 to μ), which further highlights the strength and context dependence of the warping. This is the same analysis performed on the deep neural network of figure 2, except repeated for the different values of the u that is not integrated over. In panel g we compute the eigenvalues of the metric, averaged over the irrelevant input, and then normalised by the top eigenvalue at $u_1 = 0$ in context 1 and $u_2 = 0$ in context 2. The error bars are the eigenvalues over the irrelevant input. The fact that one eigenvalue is much larger at late time points suggests that space is strongly warped into a seemingly 1-dimensional manifold. We note that this effect is less strong away from the decision boundary (Fig. S8), so the warping is not uniform over all relevant input values (as also shown in panel f).

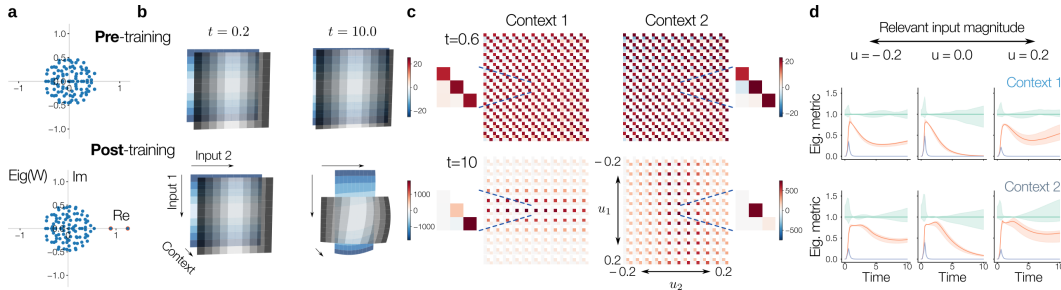


Figure S8: The context-dependent warping emerges over training and is consistent across inputs. **a.** Eigenspectrum of the weights illustrating that the network has few large (real) eigenvalues (circled in red). **b.** Geodesic gridlines from $u_i = -0.2$ in context i . The gridlines from the two contexts are overlapping to illustrate that the stretch/compression of the irrelevant input over time is context-dependent. **c.** Metric over all inputs at early and late time points. **d.** Eigenvalues of the metric over different relevant inputs. The error bars are over the irrelevant input.

722 S8.4 Sequential working memory

723 In section 5 we used a common model of working memory^{12,13} that is described in the main text.
 724 We used the same parameters initialisation as in the contextual evidence integration RNN. The hyper-parameters were:

n	g	lr	Iterations	Optimiser	ODE solver ⁹	PID ⁹ rtol, atol
100	0.5	5×10^{-3}	3000	Adam ⁷	Heun	$10^{-2}, 10^{-3}$

725

726 **Gaussian curvature.** In figure 6c we showed the Gaussian curvature of the torus at a particular
 727 time point. In principle the curvature could be computed in a similar way to the metric using adjoint
 728 dynamics or automatic differentiation. However such methods would be computationally inefficient,
 729 especially to compute the curvature over a fine mesh on the torus. To efficiently compute the Gaussian
 730 curvature we used a finite difference method. We evaluated the torus using the ODE solver and fixed
 731 step sizes $h = 0.01$ and a mesh of angles $\theta_i \in \{0, \Delta\theta, \dots, 2\pi - \Delta\theta\}$ where $\Delta\theta = 0.01\pi$ (i.e. a mesh
 732 size of 200) and computed first order accuracy finite difference estimates of the terms of Gauss's
 733 Theorem Egregium formula:

$$\begin{aligned}
 \partial_{\theta_1} \mathbf{x} &= (\mathbf{x}(t, \theta_1 + \Delta\theta, \theta_2)) / \Delta\theta \\
 \partial_{\theta_2} \mathbf{x} &= (\mathbf{x}(t, \theta_1, \theta_2 + \Delta\theta)) / \Delta\theta \\
 V &= [\partial_{\theta_1} \mathbf{x}, \partial_{\theta_2} \mathbf{x}] && \text{Basis of tangent space} \\
 G &= V^T V && \text{Metric} \\
 \partial_{\theta_1} G &= (G(t, \theta_1 + \Delta\theta, \theta_2) - G(t, \theta_1, \theta_2)) / \Delta\theta \\
 \partial_{\theta_2} G &= (G(t, \theta_1, \theta_2 + \Delta\theta) - G(t, \theta_1, \theta_2)) / \Delta\theta \\
 k_1 &= (\partial_{\theta_2} G_{\theta_1, \theta_2} - \partial_{\theta_1} G_{\theta_2, \theta_2}) / 2\sqrt{\det G} \\
 k_2 &= (\partial_{\theta_1} G_{\theta_1, \theta_2} - \partial_{\theta_2} G_{\theta_1, \theta_1}) / 2\sqrt{\det G} \\
 \partial_{\theta_1} k_1 &= (k(\theta_1 + \Delta\theta, \theta_2) - k(\theta_1, \theta_2)) / \Delta\theta \\
 \partial_{\theta_2} k_2 &= (k(\theta_1, \theta_2 + \Delta\theta) - k(\theta_1, \theta_2)) / \Delta\theta \\
 K &= (\partial_{\theta_1} k_1 + \partial_{\theta_2} k_2) / \sqrt{\det G} && \text{Gaussian curvature}^{14}
 \end{aligned}$$

734 To illustrate the curvature, we found the points θ_1^*, θ_2^* where the curvature was highest (resp. lowest)
 735 and took a piece of the torus $\theta_i \in [\theta_i^* - 0.19\pi : \theta_i^* + 0.19\pi]$, defined the matrix $X = [\mathbf{x}(t, \theta_1, \theta_2)]$
 736 for these values of θ_1 and θ_2 , and projected $X - \langle X \rangle_{\theta_1, \theta_2}$ on the PC space.

737 **Plotting the tori.** In figure 6b we plot tori at different time points. They are all color coded according
 738 to θ_1 . We computed the principal components after mean-centring. That is on $X = [\mathbf{x}(t, \theta_1 = 0, \theta_2 =$
 739 $0), \mathbf{x}(t, \Delta\theta, 0), \mathbf{x}(t, 0, \Delta\theta), \dots, \mathbf{x}(t, 2\pi, 2\pi)]$ where $\Delta\theta = 0.01\pi$, and then projected $X - \langle X \rangle_{\theta_1, \theta_2}$.

740 **The metric.** In figure 6d we show the metric over the whole torus where the colormap is normalised
 741 to the 10th and 90th quantile per entry of the metric. In figure 6e (top) we averaged the metric over
 742 the torus, while in figure 6e (bottom) we first computed $G_{\theta_2\theta_2} / (G_{\theta_1\theta_1} + G_{\theta_2\theta_2})$ before averaging.
 743 The error bars showed the 10th and 90th quantile of that value across the points on the torus.

744 **Subspace similarity.** In figure 6g we computed the similarity between the 2D PC spaces computed
 745 at any pairs of time points. The similarity is defined via the nuclear norm $\|V(t_1)V(t_2)^T\|_* / 2$ where
 746 $V(t_i)$ is the $2 \times n$ matrix of the two right singular vectors with largest singular values. This defines a
 747 notion of average angle between the subspaces.¹⁵

748 **Higher-dimensional torus.** In figure 6h,i we trained a new RNN on the same task but now with three
 749 targets at three angles $\theta_1, \theta_2, \theta_3$. In figure 6i we show the squared geodesic distance along the gridlines
 750 defined by θ_i for all i . They are computed by choosing reference angles (here $(\theta_1, \theta_2, \theta_3) = (\pi, \pi, \pi)$)
 751 and computing $L_{\text{right}}^{\theta_i} = \int_{\pi}^{2\pi} G_{\theta_i\theta_i}(\theta_1, \theta_2, \theta_3) d\theta_i$ (and similarly $L_{\text{left}}^{\theta_i} = \int_0^{\pi} G_{\theta_i\theta_i}(\theta_1, \theta_2, \theta_3) d\theta_i$). The
 752 θ_i side of the prism shown is of length $\langle L_{\text{left}}^{\theta_i} + L_{\text{right}}^{\theta_i} \rangle_{\theta_j, \theta_k}$. The grid shown on the prism is obtained
 753 by varying the bound of (say the first) integral and normalising by the total integral $L_{\text{right}}^{\theta_i}$.

754 **The topology of working memory.** In theorem S7.3 we show that the state of the RNN at a particular
 755 time point must lie on the same manifold as the inputs. However, this manifold can be immersed in

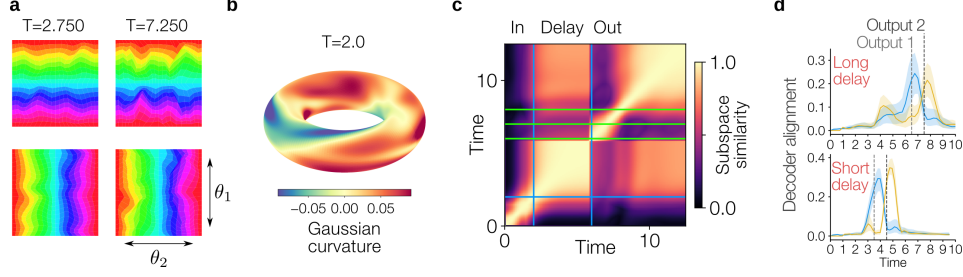


Figure S9: **Working memory dynamically changes representational geometry.** **a.** Geodesic gridlines at two different time points (mid-delay and end of output) from $\theta_i = \pi$ to 0 and 2π . The top row are the θ_1 geodesics. Unlike the decision task, there is no stronger warping around a particular point on the manifold, because there is no decision boundary. **b.** Gaussian curvature at the beginning of the delay period. The curvature is more uniform than late in the trial (Fig. 6), but non-zero. **c.** Subspace (2D PC space) similarity during the whole task. **d.** Decoder alignment to the basis of the tangent space under two different delays. It suggests that it is the go signal that cues this alignment.

the neural state space (e.g. intersecting itself), or quotiented (e.g. if there is periodicity). Here we show that it must be exactly a torus at any time point after the second input presentation, and that is embedded in the state space. For this it suffices to prove that $\mathbf{x}(t, \theta_1, \theta_2)$ is a homeomorphism in its arguments.

Proposition S8.1. *Let $\dot{\mathbf{x}}$ a working memory RNN defined as in section 5, and suppose the RNN performs the task correctly, meaning that $D\phi(\mathbf{x}(t_1, \theta_1, \theta_2)) = [\cos(\theta_1), \sin(\theta_1)]$ where t_1 is the first output time (and similarly for the second output). Then throughout the delay and output period the state of the RNN lies on a (hyper-)torus embedded in the RNN states-space \mathbb{R}^n at any time point.*

We will prove this for a 2-torus (i.e. 2 targets), the proof generalises naturally to an arbitrary number of targets.

Proof. After the second input has been presented the system is a composition of autonomous flows: $(\varphi_{t_3}^3 \circ \varphi_{t_2}^2 \circ \varphi_{t_1}^1)(\mathbf{x}(t_0, \theta_1, \theta_2))$ where t_0 is the time point where the second input is removed, t_1 is the delay, t_2 the duration of the go cue and t_3 the time until the end of the trial. Each flow is autonomous and therefore a time- and initial-state diffeomorphism by lemma S7.4. Furthermore, a composition of diffeomorphic flows is diffeomorphic — and in particular homeomorphic. Therefore, the manifold will be an embedded torus if it is an embedded torus at a particular time point.

In theorem S7.3 we have shown that $\mathbf{x}(t, \theta_1, \theta_2) = P(\mathbf{y}(t, \theta_1, \theta_2))$ where P is a projection map and $\mathbf{y}(t, \theta_1, \theta_2)$ lies on a manifold of the same topology as its input, here a torus. It therefore remains to prove that P is a bijection from the torus to its image (i.e. injective).

Let $(\theta_1^1, \theta_2^1), (\theta_1^2, \theta_2^2)$ such that $\mathbf{x}(t, \theta_1^1, \theta_2^1) = \mathbf{x}(t, \theta_1^2, \theta_2^2)$. Then, by time-diffeomorphism, $\mathbf{x}(t_1, \theta_1^1, \theta_2^1) = \mathbf{x}(t_1, \theta_1^2, \theta_2^2)$ where t_1 is the first output time. We can now apply the decoder on each side: $D\phi(\mathbf{x}(t_1, \theta_1^1, \theta_2^1)) = D\phi(\mathbf{x}(t_1, \theta_1^2, \theta_2^2))$. By assumption, the RNN performs the task correctly, so this equality holds if and only if $\theta_1^1 = \theta_1^2$. By the same argument for the time of the second output $\theta_2^1 = \theta_2^2$. Thus by the time diffeomorphism, for all t , $\mathbf{x}(t, \theta_1^1, \theta_2^1) = \mathbf{x}(t, \theta_1^2, \theta_2^2)$ if and only if $(\theta_1^1, \theta_2^1) = (\theta_1^2, \theta_2^2)$. This shows that \mathbf{x} is a topological embedding.

□

Supplementary References

1. Lee, J. M. and Lee, J. M., *Smooth manifolds*, Springer, 2003.
2. Hauser, M. and Ray, A. 2017, *Principles of Riemannian Geometry in Neural Networks*, *Advances in Neural Information Processing Systems*, ed. by Guyon, I. et al., vol. 30.
3. Zangrando, E. et al. 2024, *Neural rank collapse: Weight decay and small within-class variability yield low-rank bias*, *arXiv preprint arXiv:2402.03991*.
4. Aubry, M. and Russell, B. C. 2015, *Understanding deep features with computer-generated imagery*, *Proceedings of the IEEE international conference on computer vision*, pp. 2875–2883.

- 790 5. Kunita, H. and Ghosh, M., Lectures on stochastic flows and applications, vol. 78, Tata Institute of Funda-
791 mental Research Bombay, 1986.
- 792 6. Chen, R. T. et al. 2018, Neural ordinary differential equations, Advances in neural information processing
793 systems **31**.
- 794 7. Kingma, D. P. 2014, Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980.
- 795 8. Wong, K.-F. and Wang, X.-J. 2006, A recurrent network mechanism of time integration in perceptual
796 decisions, Journal of Neuroscience **26**(4), 1314–1328.
- 797 9. Kidger, P. 2022, On neural differential equations, arXiv preprint arXiv:2202.02435.
- 798 10. Mante, V. et al. 2013, Context-dependent computation by recurrent dynamics in prefrontal cortex, nature
799 **503**(7474), 78–84.
- 800 11. DeepMind, The DeepMind JAX Ecosystem, 2020.
- 801 12. Stroud, J. P. et al. 2023, Optimal information loading into working memory explains dynamic coding in the
802 prefrontal cortex, Proceedings of the National Academy of Sciences **120**(48), e2307991120.
- 803 13. Cueva, C. J. et al. 2021, Recurrent neural network models for working memory of continuous variables:
804 activity manifolds, connectivity patterns, and dynamic codes, arXiv preprint arXiv:2111.01275.
- 805 14. Pressley, A. N., Elementary differential geometry, Springer Science & Business Media, 2010.
- 806 15. Hitzer, E. 2013, Angles between subspaces, arXiv preprint arXiv:1306.1629.