Mechanistic Interpretations of Multi-Block State Space Models

Candidate Number: MBNJ4[1]

Computational Statistics and Machine Learning

Supervisors: Agostina Palmigiano, Arthur Pellegrino

Submission date: September 2025

[1]**Disclaimer:** This report is submitted as part requirement for the Computational Statistics and Machine Learning taught masters degree at UCL. It is substantially the result of my own work except where explicitly indicated in the text. The report may be freely copied and distributed provided the source is explicitly acknowledged.

Abstract

This thesis examines whether similarity metrics and linear algebraic tools can move beyond black-box evaluation to reveal the internal dynamics of Artificial Neural Network models. Both biological and artificial networks can display near-identical input–output behaviour while relying on different computational mechanisms, raising the need for principled methods to compare their latent trajectories.

We first developed controlled benchmarks using recurrent neural networks trained on memory tasks, showing that trajectory-based similarity measures can expose differences invisible at the behavioural level. Procrustes alignment provided a robust and interpretable tool for distinguishing models and inputs, whereas Dynamical Similarity Analysis proved less effective under input-driven conditions. Extending the framework to state-space models trained on primate neural data revealed functional decomposition across blocks, with later layers refining task-relevant geometry. Gated Linear Units further acted as contextual switches, modulating dimensionality and shaping the balance between shared and task-specific representations.

Together, these results demonstrate that internal dynamics are not incidental artefacts of architecture but central computational primitives. By systematically linking performance, geometry, and mechanism, the thesis advances a framework for comparing artificial and biological networks, and makes contributions in interpreting machine learning models that align with theories of neural computation.

Acknowledgements

Many thanks to my supervisors Agos and Arthur for introducing me to the field of computational neuroscience, helping me every step of the way and pushing me to produce the best research possible. Special mention to my collaborator Melina Laimon, without whom the analysis in this thesis would not be possible.

Contents

Chapter 1

Introduction

1.1 Motivation

Recent advances in machine learning have enabled the design of scalable neural network models capable of processing high-dimensional time-series data from fields as diverse as biology, physics, and language. Yet, such models are often regarded as *black boxes* [50], whose inner workings must be reverse-engineered to reveal their mechanisms.

Many physical and biological processes generate time-series data via an underlying dynamical system [48]. Therefore, a natural way to approximate these phenomena is to use a model that is itself governed by a dynamical system. Classical dynamical systems theory provides tools for analysing stability, attractors, and transient dynamics: concepts that have long been used to understand complex temporal behaviour. Historically, dynamical systems have been used to model hand-crafted physical systems, but more recently have been proven useful for larger trained neural networks in machine learning [4]. Furthermore, these tools offer a practical machinery for decomposing both trajectories and model components into modes and comparing them across systems.

Recent work in machine learning has provided methods to infer dynamical systems from data. In particular, State-Space Models (SSMs) have proven to be a scalable approach to capture model nonlinear dynamics from large-scale datasets [45]. Originally developed for control theory, SSMs have since become indispensable across domains where high-dimensional time series data must be interpreted. SSMs achieve state-of-the-art performance in several applications while being mechanistically interpretable [18]. This is afforded by the fact that SSMs are linear dynamical systems, which are well understood systems in dynamical systems theory. More recently, SSMs with structured connectivity have demonstrated scalability to longer sequences and larger datasets. Multiple recurrent SSM modules can be stacked to produce richer internal dynamics [45]. Furthermore, recent work has suggested that SSMs

can be combined with mechanisms typically used by transformer architectures in language modelling [43]. However, as such models grow in complexity, the interpretation of their internal mechanisms becomes increasingly difficult.

Data-inferred dynamical systems have proven particularly useful to studying the computations performed by the brain [50]. Indeed, understanding how neural circuits dynamically generate and transform activity to support behaviour is a central aim of systems neuroscience [6]. Neural populations do not encode information solely through the firing rates of single neurons, but through collective trajectories that evolve in time [32]. These trajectories capture the dynamical computations underlying perception, decision-making, and action. At the same time, modern machine learning has developed increasingly powerful models of temporal data. Decoding approaches form the foundation of brain–computer interfaces (BCIs), which translate neural activity into control signals for external devices [31]. Recent work has further demonstrated that structured state-space models can achieve competitive performance under hardware constraints, underscoring their potential viability for BCI applications [43]. Moreover, models from machine learning can be used not only for decoding, but as scientific tools to raise and test hypotheses about the computational mechanisms by which biological circuits operate [39]. This convergence raises a critical question: can these machine learning tools shed light on the computational mechanisms by which biological circuits operate?

This thesis addresses these challenges by examining similarity metrics as an objective framework for comparing the dynamics of artificial and biological networks. Crucially, different models trained on the same task can exhibit near-identical input–output behaviour, appearing interchangeable if one only inspects performance curves or decoded outputs. However, internally they may implement qualitatively different computational mechanisms [51]. Methods comparing the geometry of neural activation such as Procrustes Analysis [16], and methods specifically designed for dynamical systems such as Dynamical Similarity Analysis (DSA) [38], provide quantitative measures of correspondence between trajectories in state space. By interrogating latent trajectories and aligned subspaces, these tools reveal whether models implement the same computation in the same way, or whether they rely on distinct dynamical motifs [19]. Applied systematically, this makes it possible to separate trivial differences (e.g. random seed variation) from mechanistically meaningful ones (e.g difference in learned computations). Furthermore, this allows the connection of neural decoding performance with the geometry of hidden dynamics.

Our investigation proceeds in several stages. First, we establish synthetic benchmarks using continuous-time recurrent neural networks trained to produce low-dimensional time-series output, providing a controlled testbed for evaluating similarity metrics. In these models, we

establish that Procrustes alignment can reliably distinguish trajectories across inputs and architectures, while DSA fails to separate models in input-driven regimes. Second, we apply these tools to modern structured SSMs trained on large-scale experimental data from neuroscience. By comparing components within multi-layer SSMs, we uncover evidence of functional decomposition, with later blocks refining task-relevant structure. We further show that gating mechanisms, such as Gated Linear Units (GLUs), actively modulate dimensionality and act as contextual switches across tasks. Finally, we examine the alignment of network matrices using tools from linear algebra, revealing how training sculpts connectivity to coordinate dynamical motifs and alter dimensionality across stages of processing.

Together, the contributions of this thesis are twofold:

- **Systematic evaluation of similarity metrics** as tools for comparing dynamical systems, tested across synthetic and real-data settings.
- **Mechanistic analysis of structured state-space models**, highlighting functional roles of architectural components in neural decoding tasks.

The remainder of the thesis is structured as follows. The *Background* chapter reviews the theoretical tools and empirical context, including dynamical systems theory, spectral theory, state-space models, neural recording and decoding, and similarity metrics. The *Methodology* chapter introduces our framework and model selection. The *Results* chapter presents analyses across synthetic and real datasets, focusing on similarity, model components, and neural decoding. Finally, the *Conclusion* discusses implications for both machine learning and neuroscience, outlining how mechanistic insights from artificial networks can inform theories of neural computation.

In sum, this thesis advances the idea that dynamics are not merely by-products of network architecture but central computational primitives that can be quantified, compared, and understood. By treating both biological and artificial systems through the shared lens of dynamical trajectories, we take a step towards bridging the gap between performance and interpretability in models of neural data.

1.2 Research Questions

Building on the motivation above and the background to follow, our questions progress from methods, to architecture, to modulatory mechanisms:

1. **Which trajectory- and similarity-based analytical tools most effectively reveal the structure, similarity, and divergence of internal representations across trained, semi-trained, and randomised models?**

2. **How do distinct components within structured state-space models combine to contribute to the representation of task-relevant variables?**

3. **What role do auxiliary mechanisms, such as gating units, play in shaping the dimensionality, context-dependence, and performance of the model?**

Chapter 2

Background

In this chapter, we review the theoretical and experimental foundations that inform the research presented in this thesis. We begin with classical dynamical systems theory, followed by geometric techniques that underpin the analysis of high-dimensional trajectories. We then outline similarity metrics for comparing dynamical trajectories and introduce state-space models as a class of machine learning models. Finally, we describe advances in neural recording and decoding, and review applications of state-space models to neural data, which directly motivate the research questions of this thesis.

2.1 Dynamical Systems Theory

Dynamical systems theory provides a mathematical framework for characterising how the state of a system evolves over time. A general continuous-time system is

$$\frac{dx}{dt} = f(x, \theta), \tag{2.1}$$

where $x \in \mathbb{R}^n$ is the state, θ are parameters, and f is the vector field [48, Strogatz Chs. 2.0].

A *solution* to this system is a function $x(t)$ that, given an initial condition $x(0)$, satisfies the differential equation for all t in its domain. Such a curve in state space is also called a *trajectory* of the system [48, Strogatz Ch 1.2].

Fixed points. A *fixed point* (equilibrium) x^* satisfies

$$f(x^*, \theta) = 0. \tag{2.2}$$

At a fixed point, the instantaneous rate of change vanishes,

$$\frac{dx}{dt}\Big|_{x=x^*} = 0. \tag{2.3}$$

Phase portraits are used to visualise the trajectories and fixed points [48, Ch. 6.1]; see also Hirsch [21].

Notions of stability (local). An equilibrium x^* is *asymptotically stable* if trajectories that start close not only remain close for all future time but also converge to x^*. A weaker notion is *Lyapunov stability*, which only requires that trajectories remain close without necessarily converging [48, Ch. 1.2].

To test the stability of a fixed point x^*, we apply the spectral test to the eigenvalues λ_i of the Jacobian A about that point:

$$A = D_x f(x^*, \theta) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1}(x^*, \theta) & \cdots & \frac{\partial f_1}{\partial x_n}(x^*, \theta) \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1}(x^*, \theta) & \cdots & \frac{\partial f_n}{\partial x_n}(x^*, \theta) \end{bmatrix}. \tag{2.4}$$

$$\operatorname{Re}\lambda_i(A) < 0 \;\; \forall i \quad \Longrightarrow \quad \text{asymptotically stable (attractor)}, \tag{2.5}$$

$$\operatorname{Re}\lambda_i(A) > 0 \;\; \exists\, i \quad \Longrightarrow \quad \text{unstable (repeller)}. \tag{2.6}$$

Mixed signs indicate a *saddle* point with stable and unstable directions [48, Chs. 5.2]. If any eigenvalues lie exactly on the imaginary axis (i.e. $\operatorname{Re}\lambda_i(A) = 0$), the spectral test is inconclusive; such equilibria are non-hyperbolic and require higher-order analysis.

In addition, if the stability of a system changes with variation of its parameters, we say that the system bifurcates [48, Ch. 3.0].

Linearisation near a fixed point. Write the perturbation about a fixed point as

$$\delta x = x - x^*. \tag{2.7}$$

The first-order (linearised) dynamics is

$$\frac{d}{dt}\,\delta x = A\,\delta x, \tag{2.8}$$

with Jacobian

$$A = D_x f(x^*, \theta). \tag{2.9}$$

For sufficiently small $\|\delta x\|$,

$$\delta x(t) \approx e^{At}\,\delta x(0). \tag{2.10}$$

This provides a tractable, local approximation for non-linear dynamical systems [48, Ch. 6.3]; see also Hirsch, Wiggins [21, 57].

Euler discretisation. A standard way to approximate the continuous-time system

$$\frac{dx}{dt} = f(x, \theta) \tag{2.11}$$

is through the (forward) Euler method [48, Ch. 2.8]. Using a time step $\Delta t > 0$, the update rule is

$$x_{k+1} = x_k + \Delta t\, f(x_k, \theta). \tag{2.12}$$

This scheme replaces the derivative with a finite difference,

$$\frac{dx}{dt} \approx \frac{x_{k+1} - x_k}{\Delta t}, \tag{2.13}$$

and generates a discrete-time dynamical system. For linear systems,

$$\frac{dx}{dt} = Mx \quad \Longrightarrow \quad x_{k+1} = (I + \Delta t M)\, x_k. \tag{2.14}$$

2.2 Linear Algebra

Linear algebra studies vector spaces, linear mappings, and their matrix representations. Since dynamical systems evolve over real vector spaces $x \in \mathbb{R}^n$, linear algebra underpins their behaviour, and it likewise powers many machine learning models.

Vector spaces and linear maps. Dynamical systems are defined on vector spaces [47, Ch. 2]. A (real) vector space is a set of vectors V equipped with two operations: vector addition $+ : V \times V \to V$ and scalar multiplication $\cdot : \mathbb{R} \times V \to V$ [47, Strang Ch. 2.1],

Linear maps form the language of both linear dynamical systems [47, Ch. 2] and many ML architectures [3]. A linear map $T : V \to V$ satisfies [47, Strang Ch. 2.6]

$$T(\alpha v + \beta w) = \alpha T(v) + \beta T(w). \tag{2.15}$$

Represented as a matrix A in a given basis, it defines linear dynamics such as

$$\frac{dx}{dt} = Ax \tag{2.16}$$

Matrix factorizations and spectral theory. The eigenvalues λ and eigenvectors v of matrix A satisfy $Av = \lambda v$.

In linear systems,

$$x(t) = e^{At}x(0) = \sum_i e^{\lambda_i t} c_i v_i, \tag{2.17}$$

so eigenvalues indicate growth, decay, or oscillations [47, Strang Ch 6].

The Schur decomposition of A

$$A = QTQ^*, \tag{2.18}$$

finds Q a unitary orthonormal basis matrix and T an upper triangular matrix [24, Horn Ch 2.3].

Both eigendecomposition and Schur decomposition are used to analyse how a dynamical system evolves. Schur has the advantage of yielding an orthonormal basis, even when A is non-normal, whereas the eigenbasis may be non-orthogonal or even ill-conditioned in such cases (as often occurs in recurrent neural dynamics [14]).

Singular Value Decomposition. Any matrix $M \in \mathbb{R}^{m\times n}$ can be decomposed as

$$M = U\Sigma V^\top, \tag{2.19}$$

where U (left singular vectors) and V (right singular vectors) are orthogonal and Σ is diagonal with singular values $\sigma_1 \geq \sigma_2 \geq \dots$. [47, Strang Ch 6.3].

Precedence from both control theory [36] and machine learning [26] establishes that the singular vectors of system matrices can be compared to assess structural relationships.

In machine learning, SVD is the basis for dimensionality reduction via Principle Component Analysis [28, 7]. Canonical Correlation Analysis derives from related principles, uncovering shared latent modes between datasets [25, 53].

Together, these mathematical foundations connect the qualitative behaviour of systems (via eigenvalues and singular values) to practical techniques in data analysis, reflecting the shared algebraic underpinnings of dynamical systems and machine learning.

Effective Dimensionality and Participation Ratio Given a set of eigenvalues or squared singular values $\{\lambda_i\}$, the Effective Dimensionality is

$$\text{ED} = \frac{\left(\sum_i \lambda_i\right)^2}{\sum_i \lambda_i^2}. \tag{2.20}$$

This measures the effective number of dimensions contributing significantly to the variance [12]. The participation ratio (PR) normalises this quantity by the dimensionality d of the

data:

$$\mathrm{PR} = \frac{\mathrm{ED}}{d}. \tag{2.21}$$

Cosine Similarity

Cosine similarity measures the alignment between two vectors [15]. For real vectors a, b:

$$\cos\theta = \frac{a \cdot b}{\|a\|\|b\|}. \tag{2.22}$$

For complex vectors $a, b \in \mathbb{C}^n$, the Hermitian inner product is used:

$$\cos\theta = \frac{a^* b}{\|a\|\|b\|}, \tag{2.23}$$

where a^* denotes the conjugate transpose.

2.3 Similarity Metrics

Comparing trajectories across models, conditions, or biological recordings requires quantitative similarity measures. Orthogonal Procrustes analysis provides a classical method for aligning trajectories in high-dimensional space [44]. More recently, Dynamical Similarity Analysis (DSA) has been proposed as a method for measuring the similarity of system dynamics as opposed to comparing static geometry [38]. These metrics together form a toolkit for evaluating similarities and differences between model dynamics and neural activity. Previously, Guilhot et al. [19] compared Procrustes and DSA to characterise recurrent computation in RNN models of cognition.

Procrustes Alignment

The orthogonal Procrustes method [44] quantifies alignment between trajectory subspaces. Given two matrices $X, Y \in \mathbb{R}^{T \times d}$, the Procrustes problem seeks an optimal orthogonal transformation R that maps X onto Y:

$$R^* = \arg\min_{R^\top R = I} \|XR - Y\|_F^2. \tag{2.24}$$

Dynamical Similarity Analysis (DSA)

To compare the geometry of *dynamics* rather than static trajectories, Dynamical Similarity Analysis (DSA) was introduced to allow the comparison of system dynamics instead of static geometric comparisons of trajectories [38]. DSA summarises local flow with a linear transition

rule extracted from delay-embedded data, and then compares these rules across systems via a Procrustes-style alignment.

Delay-embedded Hankel matrices. Given a trajectory $x_t \in \mathbb{R}^d$ and a window length p, define the delay vector $\mathbf{w}_t = [x_t^\top, x_{t+1}^\top, \ldots, x_{t+p-1}^\top]^\top \in \mathbb{R}^{pd}$. Stacking these for $t = 1, \ldots, K$ with $K = T - p + 1$ yields the Hankel matrix $H_{\text{full}} = [\,\mathbf{w}_1, \ldots, \mathbf{w}_K\,] \in \mathbb{R}^{pd \times K}$. We then form two time-shifted views by dropping the last and the first window, respectively:

$$H \;=\; H_{\text{full}}[:,\, 1{:}K{-}1], \qquad H' \;=\; H_{\text{full}}[:,\, 2{:}K]. \tag{2.25}$$

SVD and rank selection on V, V'. Compute SVDs

$$H \;=\; U\,\Sigma\,V^\top, \qquad H' \;=\; U'\,\Sigma'\,V'^\top. \tag{2.26}$$

Select ranks r_H and $r_{H'}$ by the 95% cumulative energy rule, $\sum_{i=1}^{r} \sigma_i^2 / \sum_i \sigma_i^2 \geq 0.95$ (and similarly for Σ'), then set $r = \max(r_H, r_{H'})$. *Truncate only the right singular vectors* to obtain

$$V_r \in \mathbb{R}^{(K-1)\times r}, \qquad V_r' \in \mathbb{R}^{(K-1)\times r}. \tag{2.27}$$

Linear transition rule in temporal coordinates. Estimate the local linear evolution in the SVD temporal coordinates by least squares:

$$V_r' = A\,V_r \tag{2.28}$$

This yields a transition operator A that summarises local linear dynamics for the given system.

Cross-system alignment and DSA disparity. For two systems, compute A_x and A_y via (2.28). Their dynamical dissimilarity is then quantified using a modified Procrustes distance based on orthogonal similarity transforms:

$$d(A_x, A_y) \;=\; \min_{C \in O(r)} \left\| A_x - C A_y C^{-1} \right\|_F \tag{2.29}$$

where $O(r)$ denotes the group of $r \times r$ orthogonal matrices. The resulting Frobenius norm residual $d(A_x, A_y)$ is the DSA dissimilarity: lower values indicate more similar local dynamics.

2.4 State Space Models

State space models (SSMs) provide a general framework for describing the evolution of latent variables and their relationship to observations. Originally developed in control theory and

statistics, they have become central in both machine learning and computational neuroscience for modelling high-dimensional time series such as neural recordings. Here we review their historical development, highlighting both key innovations and limitations that motivated later advances.

Classical statistical formulations. Linear dynamical systems (LDS) and the Kalman filter form the foundational instance of state space modelling [30]. In this setting, the latent state evolves according to linear dynamics with Gaussian noise, and observations are linear projections of the state with additive Gaussian noise. This structure enabled efficient recursive inference and prediction, and provided the foundation for control-theoretic applications across engineering and neuroscience. However, the reliance on linearity and Gaussian assumptions meant that LDS could not capture the nonlinear, non-Gaussian dynamics evident in biological and real-world data, motivating extensions.

Neural network formulations. Recurrent neural networks (RNNs) emerged as a parametric, nonlinear extension of classical state space models. In these models, the latent dynamics are governed by learned recurrent weights. The addition of nonlinear activation functions also enabled the system to approximate complex transition rules directly from data rather than relying on predefined linear structure [11]. However, vanilla RNNs suffer from training instabilities, including vanishing and exploding gradients, which limit their ability to capture long-range dependencies. Architectural variants such as Long Short-Term Memory (LSTM) networks [22] and Gated Recurrent Units (GRUs) [5] were later developed to address these issues, though their increased complexity further reduces interpretability.

Structured state space models. A major recent advance has been the development of structured state space sequence models (S4), which parameterise recurrences in a way that enables efficient training and long-context modelling [18]. S4 leveraged HiPPO matrices to compress history efficiently, providing state-of-the-art performance on long-sequence modelling benchmarks. Subsequent variants, including S4D and S5, improved training stability, computational cost, and expressivity [17, 45]. These models retain the interpretability of state space formulations while significantly improving performance. Multiple S4-style layers can be stacked together end-to-end to create a deep sequence model, which can improve performance on long-range sequence tasks [45].

Hybrid attention–SSM models POYO introduces a spike-tokenisation scheme and a PerceiverIO-style cross-attention encoder so the model can ingest an arbitrary number of recorded units (variable channels, variable spikes) and compress them into a fixed latent, enabling multi-session, cross-subject training without fixing input dimensionality [2]. Building

directly on that idea, POSSM ("POYO-SSM") couples the same POYO-style input cross-attention with a recurrent SSM backbone (for example S4D) to achieve low-latency online decoding while retaining the flexible input interface; in benchmarks it matches or outperforms strong baselines (including Transformer models) at a fraction of their inference cost, and reports gains from multi-dataset pretraining [43]. In our context, this attention front-end usefully removes the fixed-channel constraint typical of plain SSM/RNN designs (e.g. S4/S4D), but the trade-off is reduced interpretability in the attention layers compared with explicitly parameterised latent dynamics.

Gating Mechanisms A further architectural innovation is the inclusion of gating mechanisms, such as the Gated Linear Unit (GLU), to the non-linearity in S4-style models. From a frequency-domain perspective, gating has been shown to broaden the effective spectrum of recurrent and state space layers, enabling richer representations of both high- and low-frequency components [56]. This highlights that beyond recurrence and attention, gating itself constitutes an important inductive bias that can shape the computational capabilities of sequential models.

Feature learning in deep SSMs. Recent work analyses how deep structured state-space models (SSMs) can learn internal representations. From a dynamical systems viewpoint, feature learning can be understood in terms of how the spectrum of the state transition matrix evolves during training. Shifts in eigenvalues correspond to changes in stability, memory depth, and the timescales that the model can represent. Vankadara et al [52] show that under naive parameter scalings, wide SSMs fail to produce meaningful movement of eigenvalues, leading to trivial dynamics.

Summary. Taken together, the historical development of state space models traces a trajectory from linear dynamical systems, through nonlinear recurrent networks, to modern deep sequence architectures that integrate structured kernels, attention mechanisms, and gating. While models such as S4 and its variants established strong baselines for long-range sequence modelling, and hybrids like POSSM introduced flexible input dimensionality via self-attention, interpretability is often sacrificed in the process. This opens a space for alternative designs: for instance, replacing the attention layer in POSSM with a linear, context-aware mechanism that maintains interpretability, systematically probing the interplay between SSMs and gating units, and analysing the learning regimes of these models. In addition, the interplay of the dynamics between stacked S4-like layers and their respective roles in performing computation is something that is relatively unknown. These underexplored directions represent promising opportunities to unify expressivity with interpretability and motivate the methodological contributions presented in this thesis.

2.5 Neural Recording and Decoding

Technological advances now allow simultaneous recording of large populations of neurons using electrophysiology and imaging methods [46]. These datasets are central to systems neuroscience, where the aim is not only to catalogue activity but to test hypotheses about how neural circuits implement perception, decision-making, and action [39].

A seminal example is the population vector model of motor cortex, where neuronal firing rates are linearly combined along preferred directions to predict movement direction [13]. This line of work inspired statistical models of neural encoding and decoding [40].

Importantly, decoding is not only of scientific interest but also of practical relevance. The same techniques form the foundation of brain–computer interfaces (BCIs), which translate neural activity into control signals for external devices [31]. These applications highlight how insights from neural recordings can be used both to probe brain function and to develop assistive technologies.

Beyond prediction, a major aim is interpretation. Dimensionality reduction techniques, such as PCA and Gaussian Process Factor Analysis (GPFA), have been applied to uncover latent low-dimensional structure underlying neural population activity [8]. These approaches provide insights into collective dynamics that are not apparent at the level of single neurons.

Together, these developments create opportunities for models that both capture high-dimensional trajectories and yield interpretable latent dynamics.

2.6 Neural Computation via Dynamics

A central aim of systems neuroscience is to understand how neural circuits implement computation. One influential perspective frames neural population activity as trajectories in a dynamical system, where the evolution of latent states over time captures both transient and sustained computations.

At the population level, Machens [32] demonstrated how prefrontal cortex circuits can flexibly separate "what" and "when" information, showing that population activity unfolds along distinct trajectories. This work helped shift the emphasis away from single-neuron tuning and towards the examination of collective neuron population geometry as regards neural computation.

Building on this population-level perspective, Sussillo [49] systematically argued that cortical circuits can be modelled as computational dynamical systems using RNNs. Instead of hand-crafting a model, the RNN could learn to implement its own internal dynamics, allowing a variety of hypotheses relating to neural computations to be tested objectively.

Expanding on this approach, Sussillo and Barak [50] showed that recurrent neural networks (RNNs) trained on cognitive tasks can be reduced to low-dimensional dynamical motifs using PCA. Their analysis revealed that fixed points, limit cycles, and manifolds emerge as

computational primitives underlying task performance. Valente et al. [51] advanced this by using low-rank RNNs to connect constrained connectivity with interpretable task-relevant dynamics, enabling partial reverse-engineering of both artificial and biological circuits.

Complementing these modelling studies, Mante et al. [34] provided a canonical empirical example of context-dependent computation in prefrontal cortex, consistent with computation emerging from task-specific recurrent dynamics. Driscoll et al. [9] further showed that recurrent networks, trained on multiple tasks with a common sub structure, repurpose existing dynamics to develop dynamical motifs that mimic the task sub structure.

In contrast to these approaches centred on fixed-point dynamics, Hennequin et al. [20] demonstrated that non-normal, transient dynamics in inhibition-stabilised recurrent networks can account for motor cortex activity and support complex movements.

Taken together, these studies establish that dynamics are not epiphenomena of recurrence but central computational primitives. They motivate a research programme in which latent trajectories are inferred from high-dimensional neural data, compared across tasks or regions, and linked to underlying connectivity and computational principles.

An under-examined area of research that may bear fruit is the interplay of the computational dynamics between blocks in a multi-layer SSM. This is another primary motivation for the contributions presented in this thesis.

Chapter 3

Methodology

Code implementation for the following methods and experiments is available here `https://github.com/walligot/don_thesis`.

3.1 Analytical Tools

In our experimental setup, we interpret the dynamics and how various models represent task-relevant variables. To achieve this, we employ a range of analytical techniques drawn from statistics, linear algebra, and dynamical systems theory. These tools are grouped here by purpose.

3.1.1 Decoding Analysis

Ridge Decoders

We train linear ridge regressors (`sklearn.linear.RidgeCV` [41]) to decode behaviour directly from neural or model activations. The decoder fits weights w by solving:

$$\hat{y} = Xw, \tag{3.1}$$

$$w = \arg\min_{w} \left\| Xw - y \right\|^2 + \alpha \|w\|^2, \tag{3.2}$$

where X is the input, y the target variable, and α a regularisation coefficient [23]. Performance is quantified using the coefficient of determination

$$R^2 = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}, \tag{3.3}$$

which measures the proportion of variance in the target explained by the input. Our analysis uses an 80:20 train/test split and candidate α values of 0.001, 0.01, 0.1, 1.

Logistic Classifier

To decode subject and task identity from neural or model activations, we train multinomial logistic regression classifiers implemented in `sklearn.linear.LogisticRegression` [41]. Given encoded labels $y \in \{1, \ldots, K\}$, the classifier estimates class probabilities using the softmax function:

$$P(y = k \mid X) = \frac{\exp(Xw_k + b_k)}{\sum_{j=1}^{K} \exp(Xw_j + b_j)}, \tag{3.4}$$

where w_k, b_k are the parameters for class k. Training proceeds by maximising the regularised log-likelihood of the data with class weights set to "balanced" to account for differing class frequencies.

Performance is reported using the *balanced accuracy* metric,

$$\text{BAcc} = \frac{1}{K} \sum_{k=1}^{K} \frac{\text{TP}_k}{\text{TP}_k + \text{FN}_k}, \tag{3.5}$$

where K is the number of classes, TP_k the number of true positives for class k, and FN_k the number of false negatives for class k. Balanced accuracy averages recall across classes and is robust to imbalance. A value of 1.0 indicates perfect classification, while the expected chance level due to random guessing of class is $1/K$. In practice, we evaluate classifiers using an 80:20 train/test split.

Bootstrap Confidence Intervals

To quantify uncertainty in decoding scores, we estimate confidence intervals using a percentile bootstrap procedure [10]. We first get 10 decoding scores using 10 separate random train/test splits. Given a set of scores $v = \{v_1, \ldots, v_n\}$ (e.g. accuracy or R^2 across trials), we resample n values with replacement B times to form bootstrap replicates $\{v^{*(1)}, \ldots, v^{*(B)}\}$. For each replicate, we compute the statistic of interest $\theta^{*(b)}$ (median score). This yields a bootstrap distribution $\{\theta^{*(1)}, \ldots, \theta^{*(B)}\}$.

The point estimate is taken as the median of the original values, and confidence bounds are given by the empirical quantiles of the bootstrap distribution. Specifically, for a desired confidence level c (95%),

$$\text{CI}_{\text{lower}} = Q_{\alpha/2}(\theta^*), \tag{3.6}$$

$$\text{CI}_{\text{upper}} = Q_{1-\alpha/2}(\theta^*), \tag{3.7}$$

where $\alpha = 1 - c/100$ and Q_p denotes the p-th quantile. This non-parametric approach makes no distributional assumptions and naturally handles skewed or heavy-tailed score distribu-

tions.

In practice, we used $B = 1000$ bootstrap replicates with a fixed random seed, and report the point estimate together with the lower and upper confidence bounds.

3.1.2 Trajectory Processing

Z-Normalisation

To ensure comparability across trajectories, we apply Z-normalisation, transforming each sequence to zero mean and unit variance:

$$\hat{x}_t = \frac{x_t - \mu}{\sigma}, \tag{3.8}$$

where μ and σ are the empirical mean and standard deviation of the trajectory. This standardisation reduces sensitivity to absolute scale and offset, and is widely used in time series analysis. When comparing different datasets across multiple trials, we apply Z-normalisation to model activations within each trial to remove differences arising from different variance structures. We use `scipy.stats.zscore` [55] for implementation.

Gaussian Smoothing

Temporal smoothing is applied to neural input sequences using a Gaussian kernel with width σ:

$$\tilde{x}_t = \sum_{\tau} x_\tau \, \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(t-\tau)^2}{2\sigma^2}\right). \tag{3.9}$$

This reduces high-frequency noise while preserving overall data structure [35]. For gaussian smoothing we use `scipy.ndimage.gaussian_filter` [55].

3.1.3 Linear Algebraic Methods

We employ a range of standard tools from numerical linear algebra to analyse weight matrices and hidden trajectories, following definitions from Golub and Van Loan [15]. To estimate the effective dimensionality of neural representations we use the participation number and participation ratio, following Gao et al. [12].

Principal Component Analysis (PCA)

PCA is performed by eigendecomposition of the covariance matrix

$$C = \frac{1}{T} X^\top X, \quad C v_i = \lambda_i v_i, \tag{3.10}$$

where $X \in \mathbb{R}^{T \times N}$ is the data matrix. The variance explained by component i is proportional to λ_i, allowing us to calculate the percentage variance explained by n PCA dimensions. PCA is a standard dimensionality-reduction technique, used to project high-dimensional data onto a 2D or 3D subspace for visualisation and quantitative analysis [29].

Singular Value Decomposition (SVD)/ Cosine Similarity/Participation Ratio

We implement these methods as described in the Linear Algebra Background section 2.2.

3.1.4 Similarity Metrics

To compare structure across models or datasets, we employ metrics that assess the similarity of subspaces or dynamical trajectories. In particular, we make use of both *Procrustes alignment* [44] and *Dynamical Similarity Analysis (DSA)* [38], two complementary approaches for assessing representational alignment.

Procrustes Alignment

We use the orthogonal Procrustes method as described in our background section 2.3. In practice, we compute Procrustes fits pairwise across sets of models trajectories using the `scipy.spatial.procrustes` implementation [54], which returns rotated/scaled matrices (M_X, M_Y) and a disparity score. To transform this Procrustes measure of dissimilarity into one of similarity, we subtract it from 1. Weakest similarity will be represented by 0 and strongest by 1.

$$ProcrustesSimilarity = 1 - Disparity \tag{3.11}$$

Dynamical Similarity Analysis (DSA)

The DSA algorithm is outlined in the background section 2.3. We use the implementation present in commit number a46c319 from Mitchell Ostrow's github repo `https://github.com/mitchellostrow/DSA`. In practice we choose a window size p = 10 and r is chosen such that `95%` of the variability is explained by the truncation to rank r.

In order to compare to Procrustes similarity, we take the angular form of DSA dissimilarity given by the above repo code, divide by pi and subtract from 1. This will normalise DSA in the same way as Procrustes, with 0 being weakest and 1 strongest similarity.

$$DSASimilarity = 1 - \frac{DSAAngular}{\pi} \tag{3.12}$$

3.2 Toy Recurrent Models for 2/3-Bit Memory

3.2.1 Model

We use a continuous-time RNN with $N = 100$ hidden units and tanh nonlinearity. Let $u_t \in \mathbb{R}^{n_{\text{bits}}}$ be the input at discrete step t, $h_t \in \mathbb{R}^N$ the hidden state, and $y_t \in \mathbb{R}^{n_{\text{bits}}}$ the output. Parameters are

$$W_{\text{in}} \in \mathbb{R}^{N \times n_{\text{bits}}}, \quad W_{\text{rec}} \in \mathbb{R}^{N \times N}, \quad W_{\text{out}} \in \mathbb{R}^{n_{\text{bits}} \times N}.$$

The continuous dynamics are Euler-integrated with step $dt = 0.15$ and time constant $\tau = 1$. Writing the noise term as η_t (i.i.d. standard Gaussian noise during training to encourage regularisation), the update is

$$\Delta h_t = \Big(-h_t + \tanh\big(W_{\text{rec}}^\top h_t + W_{\text{in}}^\top u_t\big) \Big)\, dt \tag{3.13}$$

$$h_{t+1} = h_t + \Delta h_t + \sigma_{\text{noise}} \eta_t, \qquad \sigma_{\text{noise}} = 0.015 \text{ (training only)} \tag{3.14}$$

$$y_t = W_{\text{out}}^\top h_t. \tag{3.15}$$

3.2.2 Task and Data Generation

The model emulates the classic flip–flop memory task [50]. We first describe the inputs in continuous time and then their discretisation used for training.

For each input bit b, pulses arrive as a Poisson process with rate λ. When a pulse starts, its sign is chosen uniformly from $\{-1, +1\}$; while active the input equals that sign, and otherwise it is 0. Pulse durations are exponentially distributed with rate μ.

With time step Δt, define $u_{k,b} \in \{-1, 0, +1\}$ as the value held on the interval $[k\Delta t, (k+1)\Delta t)$. For small Δt the discretisation yields

$$\Pr(\text{start in step } k) = \lambda \Delta t, \qquad \Pr(\text{end in step } k \mid \text{active}) = \mu \Delta t,$$

so $p_{\text{start}} = \lambda \Delta t$ and $p_{\text{end}} = \mu \Delta t$ per bit per step. In our experiments we set $p_{\text{start}} = 0.01$ and $p_{\text{end}} = 0.05$.

Targets are the per-bit *memory*: the most recent non-zero value of the corresponding input channel.

$$\text{Given } (u_t)_b \in \{-1, 0, +1\}, \quad (\text{target}_t)_b = \text{last non-zero value seen on bit } b. \tag{3.16}$$

3.2.3 Training Objective and Schedule

We train with Adam (weight decay 10^{-4}), mini-batches of size 32, for 4000 epochs. Training sequences use length $T_{\text{train}} = 300$; evaluation sequences use $T_{\text{eval}} = 1000$. The learning rate

follows a two-phase schedule to squeeze out extra performance.

$$\mathrm{lr} = \begin{cases} 2 \times 10^{-3}, & \text{epochs } < \frac{1}{2} \text{ total} \\ 2 \times 10^{-4}, & \text{otherwise,} \end{cases} \tag{3.17}$$

The loss combines sign-tracking and magnitude regularisation. The second term further encourages the model to output a value close to -1/1.

$$\mathcal{L} = \underbrace{\mathrm{MSE}\big(y, \mathrm{target}\big)}_{\text{sign tracking}} + \lambda_{\mathrm{abs}} \underbrace{\mathrm{MSE}\big(|y|,\ |\mathbf{target}|\big)}_{\text{unit magnitude bias}}, \quad \lambda_{\mathrm{abs}} = 0.1. \tag{3.18}$$

Random Initialisation. We use a random seed of 666 by default and 444 for alternate-seed models. The weight matrices are randomly initialised with draws from

$$w \sim \mathcal{N}\big(0,\ 0.1^2\big)\,.$$

3.2.4 Evaluation

We evaluate models on long sequences ($T_{\mathrm{eval}} = 1000$) without training noise. Performance is reported using two complementary metrics: mean squared error (MSE) and sign accuracy.

The mean squared error is

$$\mathrm{MSE} = \frac{1}{BTn_{\mathrm{bits}}} \sum_{i=1}^{B} \sum_{t=1}^{T} \sum_{b=1}^{n_{\mathrm{bits}}} \big(y_{i,t,b} - \mathrm{target}_{i,t,b}\big)^2, \tag{3.19}$$

and the sign accuracy, which measures the fraction of bits with correct ± 1 sign, is

$$\mathrm{Acc}_{\pm} = \frac{1}{BTn_{\mathrm{bits}}} \sum_{i=1}^{B} \sum_{t=1}^{T} \sum_{b=1}^{n_{\mathrm{bits}}} \mathbf{1}\big[\mathrm{sign}(y_{i,t,b}) = \mathrm{sign}(\mathrm{target}_{i,t,b})\big]\,. \tag{3.20}$$

Here B is the evaluation batch size, T the sequence length, and n_{bits} the number of memory channels.

For dynamical inspection, we extract hidden trajectories $h_{0:T}$ and visualise 2D and 3D PCA projections with quiver arrows to indicate flow.

3.2.5 Fixed Points and Local Linearisation

Given a hidden state h, the autonomous drift under zero input is

$$f(h) = \big(-h + \tanh(W_{\mathrm{rec}}^{\top} h)\big)\, dt. \tag{3.21}$$

We approximate fixed points by directly optimising h to minimise $\|f(h)\|_2$ using Adam on h with learning rate 10^{-2} for up to 500 iterations, initialised from selected trajectory states (e.g. just before/after controlled pulses and at the sequence end). Local stability is classified by the Jacobian eigenvalues of f at the fixed point:

$$J(h) = \frac{\partial f}{\partial h}(h), \quad \text{attractor if } \Re\,\lambda_i(J) < 0\ \forall i;\ \text{repeller if } \Re\,\lambda_i(J) > 0\ \forall i;\ \text{saddle otherwise.} \tag{3.22}$$

Memory labels ($\{-1, 0, +1\}^{n_{\text{bits}}}$) at fixed points are inferred by thresholding the readout $W_{\text{out}}^{\top} h$ with a small tolerance (e.g. 0.2).

3.2.6 Experimental Setup

Software: PyTorch

Hyperparameters: N=100, dt=0.15, p_{pulse}=0.01, noise σ_{noise}=0.015 (training only), batch size 32, epochs 4000, optimiser Adam with weight decay 10^{-4}, learning rate as in (3.17), loss as in (3.18).

Models Considered: For trajectory analysis we compare a baseline two-bit RNN against a set of five variants. This allows us to observe how task, architecture and random initialisation affect the models' learned representations.

- **Baseline Case**: main two-bit RNN trained with the default settings (seed = 666).
- **Case 1**: a three-bit RNN evaluated on the same input transitions as the baseline, with it's extra third input channel initially held at +1.
- **Case 2**: a two-bit RNN trained with a different random seed (444), evaluated on the same input transitions as the baseline.
- **Case 3**: same two-bit RNN trained with different random seed, evaluated on a different set of input transitions.
- **Case 4**: a two-bit RNN with all weights reinitialised randomly, evaluated on the same input transitions as the baseline.
- **Case 5**: same randomised two-bit RNN with a different set of inputs.

Input Protocols for Fixed Trajectories: We generate scripted transition loops for $n_{\text{bits}} \in \{2, 3\}$:

- **Two-bit "full square" (same input)**: $[+1, +1] \to [-1, +1] \to [-1, -1] \to [+1, -1] \to [+1, +1]$.
- **Three-bit "full square" (same input with third channel held at 1)**: $[+1, +1, +1] \to [-1, +1, +1] \to [-1, -1, +1] \to [+1, -1, +1] \to [+1, +1, +1]$.
- **Two-bit "half square" (different input)**: $[+1, +1] \to [-1, +1] \to [-1, -1] \to [-1, +1] \to [+1, +1]$.

Transitions are brief, single-bit flips embedded in longer zero-input segments; trajectories are visualised by PCA (2D/3D) with optional Gaussian smoothing and direction arrows. For fixed-point initialisation we sample states at *(i)* just before a pulse, *(ii)* just after, and *(iii)* at the segment end.

Evaluation: We plot inputs/targets/outputs per bit, PCA trajectories with fixed points coloured by memory label and shaped by stability (attractor/saddle/repeller), and we print PCA explained variance. Numerical summaries include MSE and $\text{Acc}_{\pm}$ (Eq. 3.20). We also compare the similarity of various trajectories across cases using Procrustes and DSA metrics.

3.3 Neural Datasets

Neural datasets used in this study consist of multi-unit neural recording sessions of non-human primates during reaching behaviours. Brain regions recorded were a combination of M1 (primary motor cortex), PMd (dorsal premotor cortex), and S1 (primary somatosensory cortex).

Two paradigms are common. In *Centre-Out reaching* involves movements from a central start position to one of several fixed peripheral targets. Typically, a preparatory phase begins with target presentation and ends with a go cue signalling movement onset. In *Random Target reaching*, targets appear sequentially at random positions on a grid, yielding more naturalistic, self-paced movement without a preparatory phase.

To allow direct comparison across both tasks, we analysed only the movement phase, thereby excluding preparatory activity present only in centre-out reaching. The sessions were split into trials, each corresponding to a single movement phase. Before entering any models, all spike trains were discretised into 5 ms bins and smoothed with a Gaussian kernel of width 20 ms.

Foundational Model Training/Evaluation Data

All datasets relating to the foundational model were accessed through the `BrainSets` framework, which provides a standardised interface to electrophysiological recordings across non-

human primate motor tasks [1]. For evaluation purposes, we use the trials marked 'eval' from a single session file from each dataset listed below.

Perich Miller. (2018) [42]:

- "pm_c_co" (centre-out, subject C): 352 neural units, M1/PMd regions.
- "pm_c_rt" (random-target, subject C): 87 neural units, M1/PMd regions.
- "pm_m_rt" (random-target, subject M): 164 neural units, M1/PMd regions.
- "pm_m_co" (centre-out, subject M): 158 neural units, M1/PMd regions.

O'Doherty Sabes. (2017) [37]:

- "os_i_rt" (random-target, subject I): 464 neural units, M1/S1 regions.
- "os_l_rt" (random-target, subject L): 625 neural units, M1/S1 regions.

Churchland Shenoy. (2012) [6]:

- "cs_j_co" (centre-out, subject J): 190 neural units, M1 region.
- "cs_n_co" (centre-out, subject N): 191 neural units, M1 region.

Task-Specific Model Training/Evaluation Data

For downstream analysis we also consider a task-specific model trained on the `mc_rtt` dataset from the Neural Latents Benchmark (`https://neurallatents.github.io/datasets.html`). This dataset corresponds to a trialised version of the random target tracking task originally recorded by O'Doherty et al. (2017) [33], specifically subject I.

`mc_rtt` (Neural Latents Benchmark) :

- Random target tracking, subject I (trialised release).
- 130 neural units.

3.4 Neural Encoder Models

Disclaimer. The analyses performed in this section were part of a larger research project, in collaboration with Melina Laimon. As part of this research effort, M.L. trained the SSM models, while I analysed dynamics of the trained models. Pre-trained checkpoints for these models are used as the basis for the analysis reported in later sections.

3.4.1 Overall Model Architecture

The neural encoder maps an input sequence of spikes $x_{\text{in}} \in \mathbb{R}^{L \times D_{\text{in}}}$, together with a group index g identifying the subject–task data source, into an output sequence $y_{\text{out}} \in \mathbb{R}^{L \times D_{\text{out}}}$. The architecture consists of three main stages:

$$x_{\text{enc}} = \text{Encoder}_g(x_{\text{in}}) \tag{3.23}$$

$$x_{s5} = \text{S5-Stack}(x_{\text{enc}}) \tag{3.24}$$

$$y_{\text{out}} = \text{Decoder}(x_{s5}) \tag{3.25}$$

Group-Specific Encoder

Since datasets vary in input dimensionality, the encoder first projects inputs into a shared latent space:

$$x_{\text{enc}} = x_{\text{in}} W_{\text{enc},g} + b_{\text{enc},g} \tag{3.26}$$

where $W_{\text{enc},g} \in \mathbb{R}^{D_{\text{in}} \times (H - D_{\text{ctx}})}$, $b_{\text{enc},g} \in \mathbb{R}^{H - D_{\text{ctx}}}$, and $x_{\text{enc}} \in \mathbb{R}^{L \times (H - D_{\text{ctx}})}$.

Stacked S5 Blocks

The contextualised input is processed through a stack of N S5 blocks:

$$x^{(0)} = x_{\text{ctx}} \tag{3.27}$$

$$x^{(i)} = \text{S5Block}_i(x^{(i-1)}), \quad i = 1, \ldots, N \tag{3.28}$$

with final block output $x_{s5} = x^{(N)}$.

Decoder

The decoder applies a linear projection to produce the final output:

$$y_{\text{out}} = x_{s5} W_{\text{dec}} + b_{\text{dec}} \tag{3.29}$$

where $W_{\text{dec}} \in \mathbb{R}^{H \times D_{\text{out}}}$ and $b_{\text{dec}} \in \mathbb{R}^{D_{\text{out}}}$.

—

3.4.2 The S5 Block

Each S5 block transforms $x \in \mathbb{R}^{L \times H}$ into an output sequence $y \in \mathbb{R}^{L \times H}$. The computation is:

$$x_{\text{norm}} = \text{Norm}(x) \tag{3.30}$$
$$x_{\text{ssm}} = \text{S5-SSM}(x_{\text{norm}}) \tag{3.31}$$
$$x_{\text{act}} = \text{Dropout}(\text{GELU}(x_{\text{ssm}})) \tag{3.32}$$
$$x_{\text{glu}} = \text{GLU}(x_{\text{act}}) \tag{3.33}$$
$$y = \text{Dropout}(x_{\text{glu}}) \tag{3.34}$$

The post-GLU output is passed to the next block. Note that dropout is applied during training only.

State Space Model Layer

The continuous-time linear system is:

$$\frac{dh(t)}{dt} = Ah(t) + Bu(t) \tag{3.35}$$
$$y(t) = Ch(t) + Du(t) \tag{3.36}$$

with: - $h(t) \in \mathbb{C}^P$ (latent state), - $u(t), y(t) \in \mathbb{R}^H$ (input and output), - $A = \Lambda \in \mathbb{C}^{P \times P}$ (diagonal state matrix), - $B \in \mathbb{C}^{P \times H}$, $C \in \mathbb{C}^{H \times P}$, - $D \in \mathbb{R}^H$.

The Zero-Order Hold discretisation with learnable timescale $\Delta \in \mathbb{R}^P$ gives:

$$\bar{A} = \exp(\Lambda\Delta) \tag{3.37}$$
$$\bar{B} = \Lambda^{-1}(\bar{A} - I)B \tag{3.38}$$

and the discrete recurrence at step k is:

$$h_k = (\bar{A} - \alpha I)h_{k-1} + \bar{B}u_k \tag{3.39}$$
$$y_k = \text{Re}(Ch_k) + Du_k \tag{3.40}$$

implemented efficiently as a parallel scan. This allows the continuous-time model to be applied to a discrete sequence of inputs.

GELU Activation

The Gaussian Error Linear Unit is applied element-wise:

$$\text{GELU}(x) = x \cdot \Phi(x) \tag{3.41}$$

where $\Phi(x)$ is the standard normal CDF.

Gated Linear Unit

The GLU applies multiplicative gating:

$$\text{GLU}(z) = (zW_1 + b_1) \odot \sigma(zW_2 + b_2) \tag{3.42}$$

with $z \in \mathbb{R}^H$, $W_1, W_2 \in \mathbb{R}^{H \times H}$, $b_1, b_2 \in \mathbb{R}^H$, $\sigma(\cdot)$ the sigmoid, and $\odot$ element-wise multiplication.

Final Dropout

A final dropout (training only) is applied before the block output is passed on.

3.4.3 Experimental Setup

Trained Models:

For analysis we consider two implementations of the above architecture:

1. A **Foundational Model**, trained across several neural datasets, aiming to capture shared representations across tasks. Checkpoints used were .e.g `l2_reaching_checkpoint` for the 2 block foundational model.
2. A **Task-Specific Model**, trained from scratch on the single `mc_rtt` random-target reaching dataset. Checkpoints used were .e.g `l2_scratch_all_checkpoint` for the 2 block task-specific RTT model.

Both share the same architecture but differ in training data and objectives. The checkpoints used in this research were created by Melina Laimon.

Key hyperparameters include:

- Number of S5 blocks: 2 or 4
- SSM hidden dimension P:128

- SSM input/output dimension H: 256 (2 blocks) or 512 (4 blocks)
- Learning rate and schedule: 0.002
- Optimiser: Adam
- Number of training epochs: 6000
- Training datasets: detailed in §Neural Datasets

3.4.4 Activations

Activations were generated by passing the various neural datasets into the `call_with_activations` method present in the model checkpoints.

3.4.5 Evaluation:

Our evaluation protocol consists of three complementary components:

Trajectory-level inspection: Activations were projected into low-dimensional subspaces using PCA for visualisation, and similarity metrics (Procrustes alignment) were applied to compare trajectories across models and blocks.

Decoding analysis: Linear ridge regressors were trained to decode behavioural outputs (e.g. hand velocity) from block activations at various stages of the model. Logistic regression models were trained to decode dataset subject and task. Performance was quantified with cross-validated R^2 and class accuracy respectively.

Model matrices and dynamical analyses: Weight and state-space matrices were examined using spectral analysis, cosine similarity, and participation ratios, to assess learning regimes, alignment, and effective dimensionality of block dynamics.

Chapter 4

Results

4.1 Toy RNN Analysis

4.1.1 Training

Training MSE/Accuracy The training and evaluation MSE loss and sign accuracy over 4000 epochs, for the toy rnn model trained on the two bit memory problem, is shown in Figure 4.1. For evaluation we used a fixed set of randomly generated input pulses (Training sequences are 300 timesteps and eval 1000 timesteps). As training progressed the train/eval loss fell quickly and plateaued around 0.15. The train sign accuracy also plateaued around 65%, yet the eval sign accuracy passed 90%. This indicates the model can accurately track the sign of the last input value on the longer eval sequences.

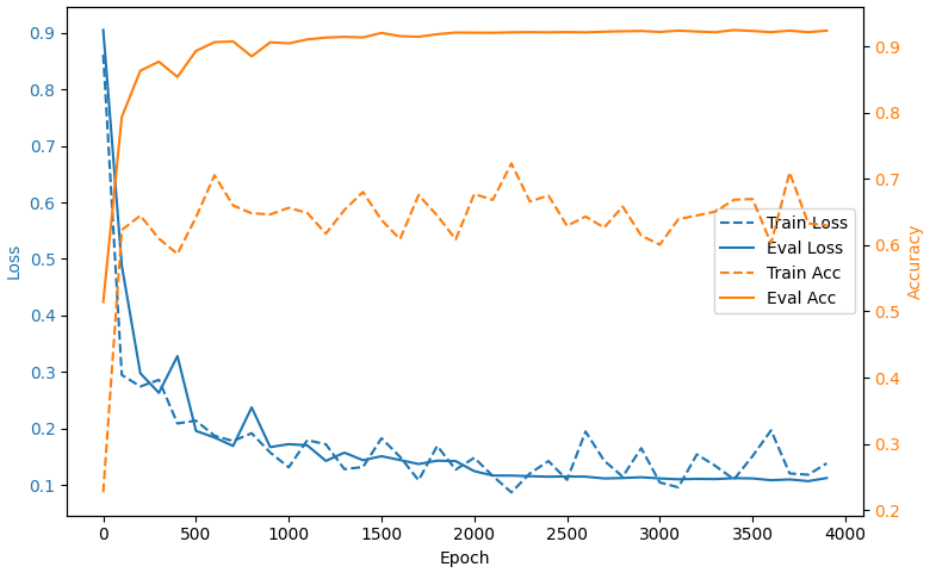

Figure 4.1: **Toy RNN has high sign accuracy after training.** Training Graph for Two-Bit Toy RNN showing train/eval MSE and sign accuracy. The Two-Bit RNN eval MSE plateaued around 0.15 but the eval sign accuracy reached 0.9

Post-Training Evaluation We took our trained two bit memory rnn and fed it a single sample randomised two bit evaluation input. In Figure 4.2 we observe that the model was able to output (green) the last randomised input pulse (orange) quite well in both bit channels.

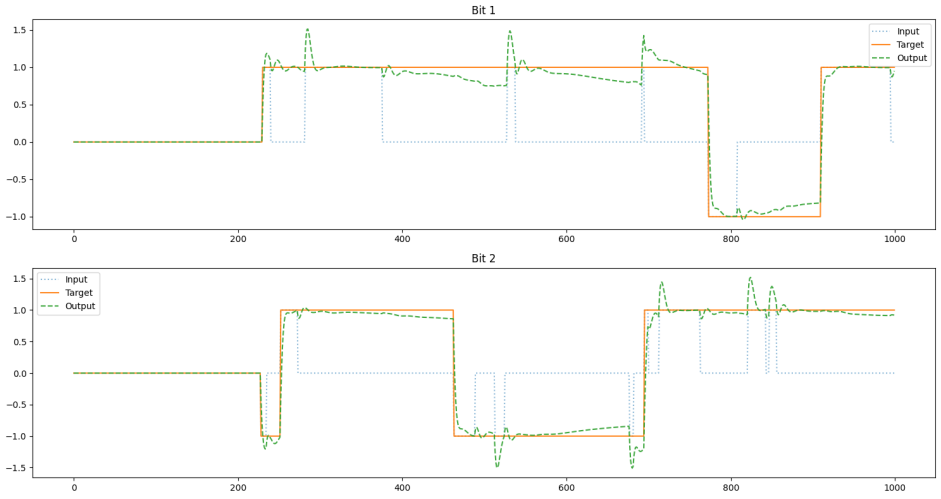

Figure 4.2: **Trained RNN tracks eval targets.** The model output (green) is able to track the eval target (orange) to a reasonable degree of accuracy.

Eigenvalue spectra We plot the eigenvalue spectra for the recurrent weights in the untrained (random) and trained two-bit memory models in Figure 4.3. In the untrained case, the random Gaussian initialisation produced eigenvalues spread fairly evenly within the unit circle, with a few outliers beyond it. After training, the real part of most eigenvalues became slightly smaller, indicating that the network had learned more stable and damped modes to support reliable memory over the input sequence. Some eigenvalues remained near (1, 0), which may have corresponded to the dominant modes solving the task. A few were pushed outside the unit circle along the imaginary axis. These outliers correspond to unstable oscillatory modes, which the network may have exploited to generate transient dynamics that push the state between fixed points. Similar spectral patterns in trained recurrent networks solving flip-flop tasks have been reported by Jarne [27].

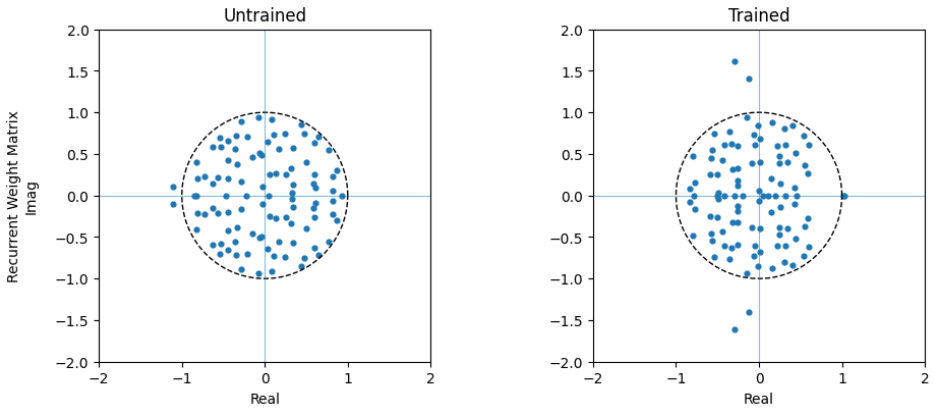

Figure 4.3: **Few eigenmodes emerge over training.** After training, the real part of some eigenvalues decreases. Few eigenvalues remain at (1,0). Some eigenvalues are pushed outside the unit circle.

4.1.2 Trajectory Plotting

Traversing the square We see in Figure 4.4 the trajectories of the various rnn models under the handcrafted inputs 3.2.6. These inputs were fixed such that one bit was flipped at a time. This produced trajectories in the Baseline Case that followed the perimeter of a square in 2D PCA space. The trajectories settled at attractor points representing memory states at each vertex. When a bit was flipped, the trajectories passed along an edge of a square, through a saddle point to a new fixed point vertex. Sussillo and Barak [50] observed a cube shape when plotting the three bit memory problem trajectories in 3D PCA space. Our two bit model matches this previous result, with our square representing one side of that cube.

The three bit model with third channel fixed (Case 1), and the two bit model with different initialisation seed but same input (Case 2) produce similar trajectories. The two bit model

with different seed and half-square input (Case 3) traverses half the square before doubling back on itself. The randomised model under either input (Cases 4/5) has a set of attractor points which it oscillates about.

For the trained models, two PCA dimensions explained >90% of the trajectory variance and ~60% for the randomised model trajectories.

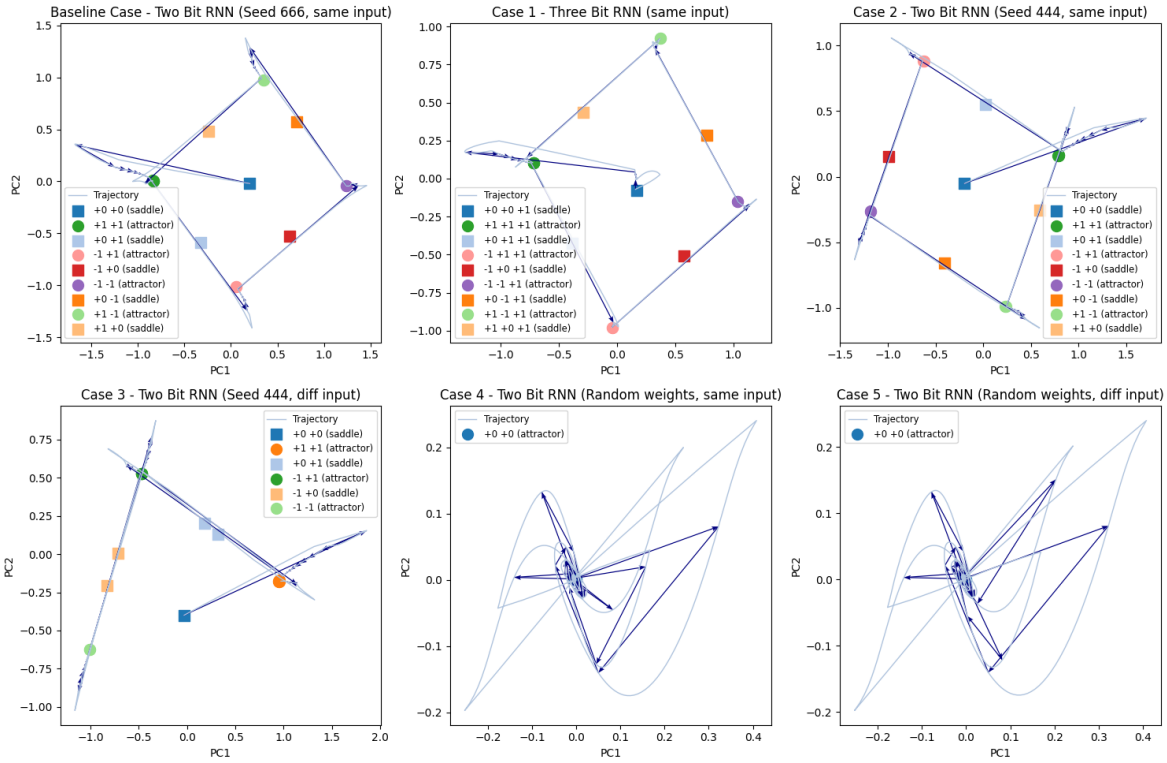

Figure 4.4: 2D PCA plots for toy RNN raw trajectories under full and half square input. □ corresponds to a saddle point and ○ an attractor.

Randomised Trajectories Next we plotted 50 random-input eval trajectories in 2D PCA in Figure 4.5, with the previous set of fixed points mapped into the same space. For the cases with different inputs another set of 50 trajectories was generated. In all cases the trajectories pass through the same set of fixed points as before. They appear to be pushed between saddle points and outside of the square by the randomly-timed input pulses.

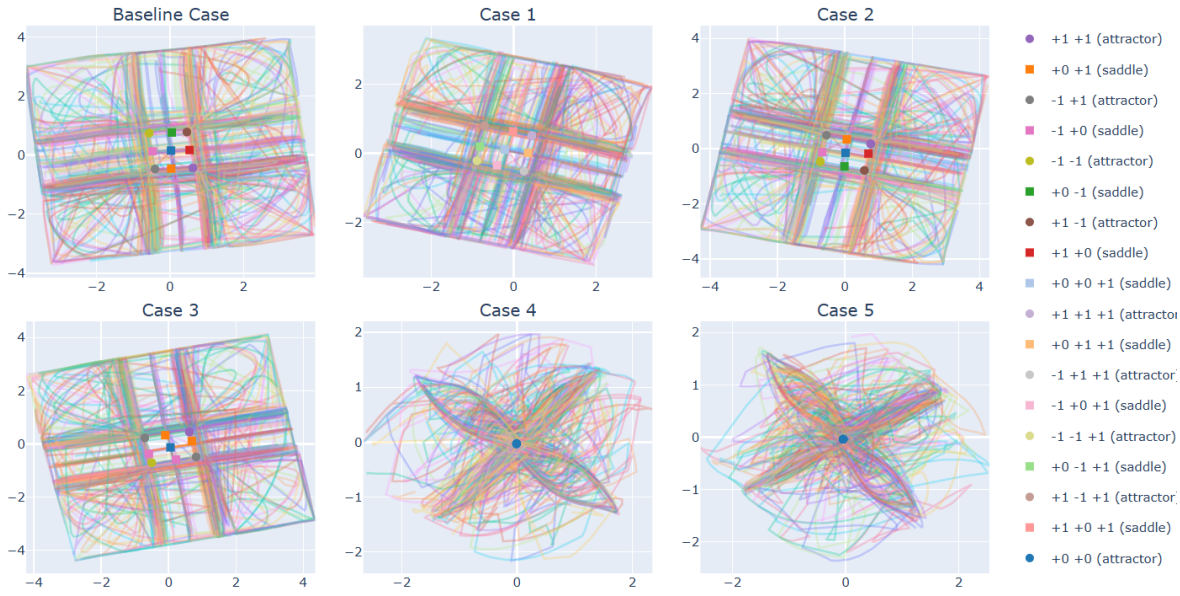

Figure 4.5: **Square trajectory shape persists for many randomised inputs.** 2D PCA plots for toy RNN 50 random eval trajectories. □ corresponds to a saddle point and ○ an attractor.

4.1.3 Similarity Metrics

Comparison of Different Noise Regimes The previous analyses suggested key differences between pairs of models, while others seemed to have identical trajectories. However, these analyses were mainly qualitative, relying on visualisation to compare models. We now turn to a quantitative assessment of model similarity using similarity metrics. To assess the suitability of using the DSA and Procrustes measures of similarity, we constructed a test to examine each metric under different levels of noise in the model dynamics i.e. the value of η_t in equation 3.14.

- No noise - models had no noise in their dynamics.
- Same noise - models were injected with the same noise sample η_t at each timestep.
- Different noise - the baseline case receives its own noise sample. Other models receive a separate set of noise samples.

For each comparison we compared the Baseline model to each other model. The same sets of 50 random-input trajectories were used for this experiment as in the plotting section. A completely randomised set of numbers was also added to the comparison as a sanity check. We expected the metrics to assign less similarity compared to the no noise case, for the same

model pair, when noise is introduced. DSA was expected to pick up similarity in dynamics whereas Procrustes the similarity in model input.

DSA was initialised with window hyperparameter p=10 and rank r chosen automatically as described here 2.3. Looking at Figure 4.6, we see that DSA assigned almost perfect similarity to all cases besides the randomised tensor, which is still realtively high at 0.5. There's a barely perceptible decrease for the noisy cases.

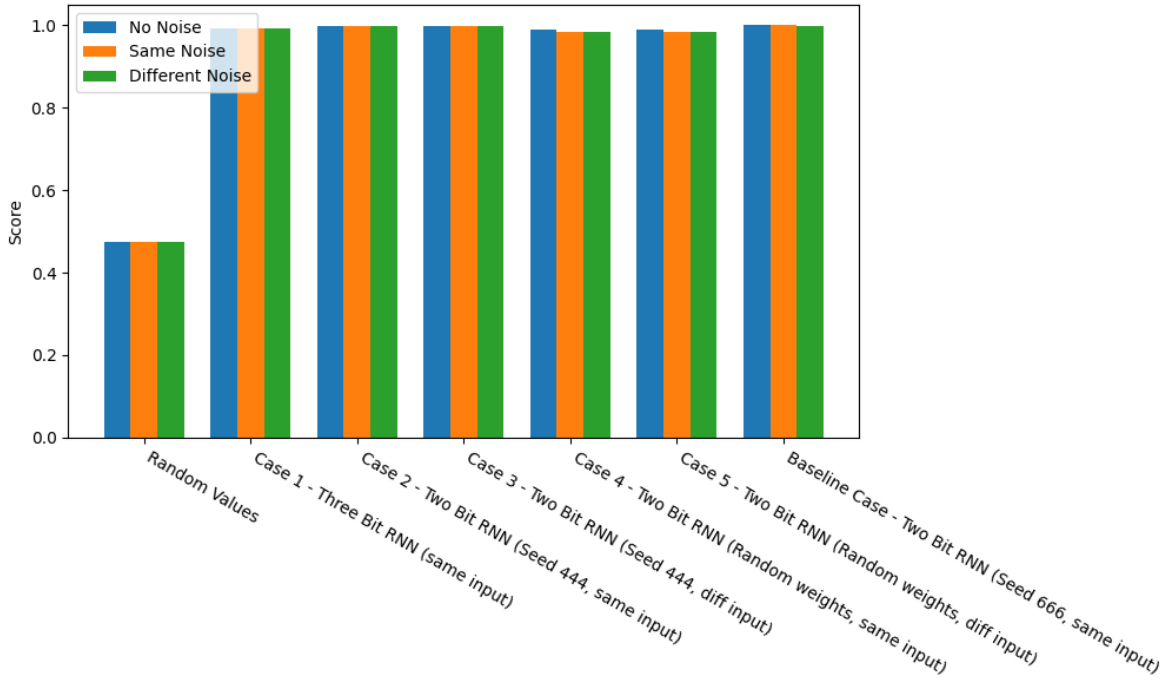

Figure 4.6: **DSA does not disentangle between model class or noise type.** DSA Similarity Scores across cases for multiple trajectories under no noise, same noise and different noise. It assigns high similarity to all cases, despite clear differences in model architecture, different input, randomised weights etc.

Procrustes, on the other hand, produced much cleaner results in Figure 4.7. It assigned zero similarity for models under different input. For models under same input, it assigned the most similarity to itself, then the same model with different random initialisation, followed by the threebit model and finally a steep drop for the randomised model. The tensor of Random Values was assigned zero similarity. The similarity dropped slightly for models under same and different noise, albeit not by much.

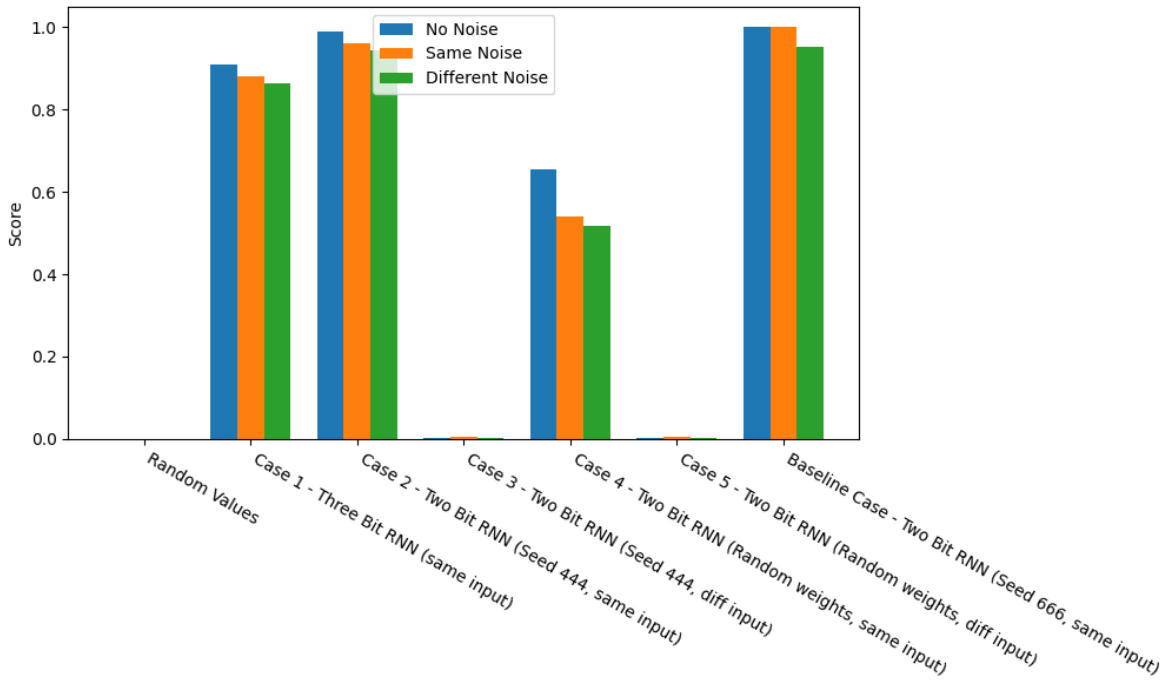

Figure 4.7: **Procrustes disentangles differences in model inputs and architectures.** Procrustes Similarity Scores across cases for multiple trajectories under no noise, same noise and different noise

Finally we examined the matrix of similarities in Figure 4.8. The same pattern emerged, where DSA showed high similarity between all models. There was a slight block pattern, where models with same dynamics had slightly higher values, but the difference is negligible. Procrustes again showed higher similarity for models under same input, and near zero similarity for those under different input.

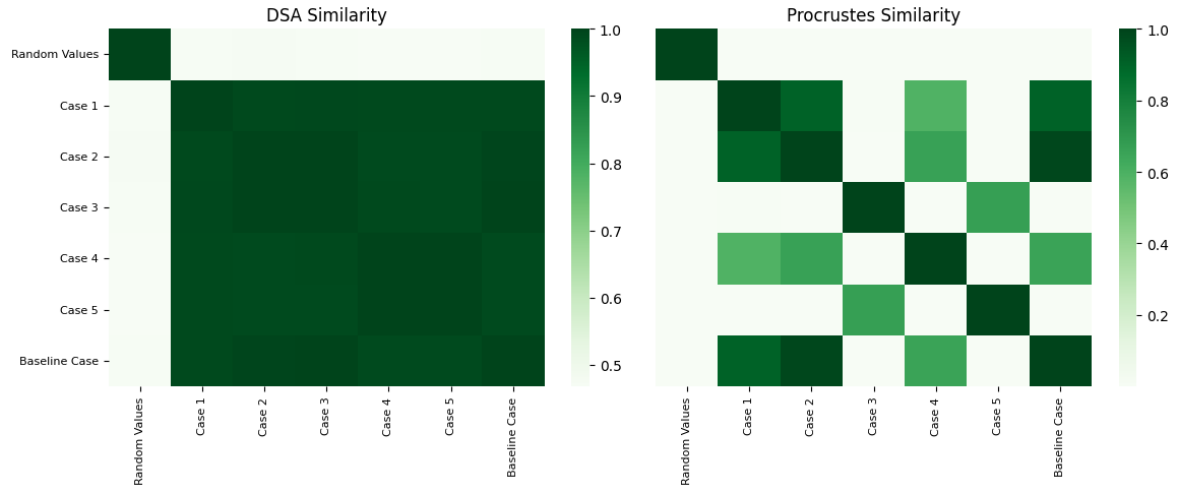

Figure 4.8: **DSA fails to highlight similarities in model input, Procrustes succeeds.** .DSA/Procrustes Similarity Matrices across cases for multiple random-input trajectories

Overall, our experiments show that DSA did not meaningfully differentiate between model classes, noise regimes, or input conditions: it consistently reported high similarity values, with only minimal decreases when noise was added. This suggests that DSA is largely insensitive to the dynamics we aimed to probe, and is therefore ill-suited for comparing RNN models in this specific experimental setup. By contrast, Procrustes produced clearer separation across conditions. It distinguished models driven by the same versus different inputs, ranked related architectures in a sensible order, and assigned near-zero similarity to tensors of random values.

One possible explanation for DSA's poor performance is that the N-bit memory task is strongly input-driven, with trajectories largely determined by input pulses rather than autonomous dynamics. In this case, DSA's reliance on local trajectory structure offers little discriminative power, while Procrustes, by directly aligning state trajectories, remains sensitive to input differences.

Future work could test whether DSA performs better in tasks with richer autonomous dynamics or longer memory dependence, inputs that produce continuous trajectories. Another direction would be to explore hybrid similarity measures that combine sensitivity to input structure with trajectory alignment.

4.2 Neural Encoding Models

4.2.1 Feature Learning

In the toy RNN we saw that few large eigenvalues possibly implemented task computation. We therefore asked whether structured SSM also followed this pattern.

In Figure 4.9 we observed the movement of eigenvalues of the discretised A dynamics matrix (A bar in equation 3.37). We noticed that in block 0, the eigenvalues became more clustered after training on the positive real end of the unit circle, with some eigenvalues fanning out around the perimeter. In the second block, there is stronger clustering around the positive real end of the circle. Two smaller clusters appear further inside the circle. However, we did not see any dramatic shift in the structure of the eigenvalues, with very few being pushed away from the unit circle and none end up close to the origin. It is unclear whether feature has not occurred or not in this SSM.

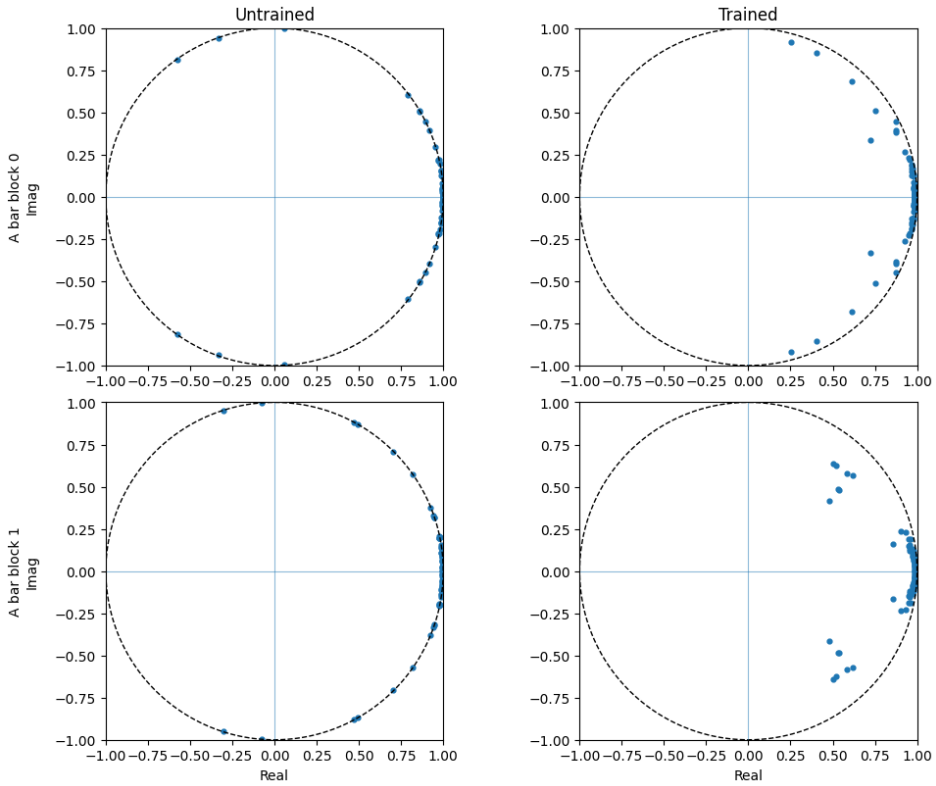

Figure 4.9: **Recurrent matrix eigenvalues cluster after training.** Eigenvalue plots for the A bar matrices in Foundational 2-Block Model

4.2.2 Trajectory Plotting

In this section we have plotted and analysed activations and modifications of activations from various stages of the Task-Specific RTT and Foundational Reaching neural encoder models. Both two block/layer and four block variants are considered. With reference to the model description section 3.4.1,the activation stages are as follows:

- Neural Input Raw - the raw dataset input
- Neural Input - the dataset input with Gaussian Smoothing, this is what the models were trained on.
- Encoded Input Raw - The model's encoder layer applied to Neural Input Raw. This allows concatenation across datasets with different input dimensions.
- Encoded Input - model encoder applied to Neural Input
- Hidden State/x block N - x_{ssm}, the complex valued hidden state from the SSM block/layer N
- y block N - y_{out}, the real valued convolution of the hidden state that is passed to the GELU/GLU layers in block N
- Post GELU block N - GLU(GELU(y_{out})), the activation passed through the GELU and GLU non-linearities. The final activation of block N before entering block N=1.

Task-Specific RTT 2-Block Model In Figure 4.10 we have plotted y and post-glu activations in 2D PCA space for RTT two-block model. The RTT trial metadata allowed us to group by reaching angle, so the trajectories are averaged by buckets of 45°. We observed separation by reaching angle bucket in the final post-glu activations, with the angles arranged in correct circular order. Weaker separation can be seen in prior activations. It was apparent that this model relied on its final GELU/GLU layers to solidify its prediction for hand velocity.

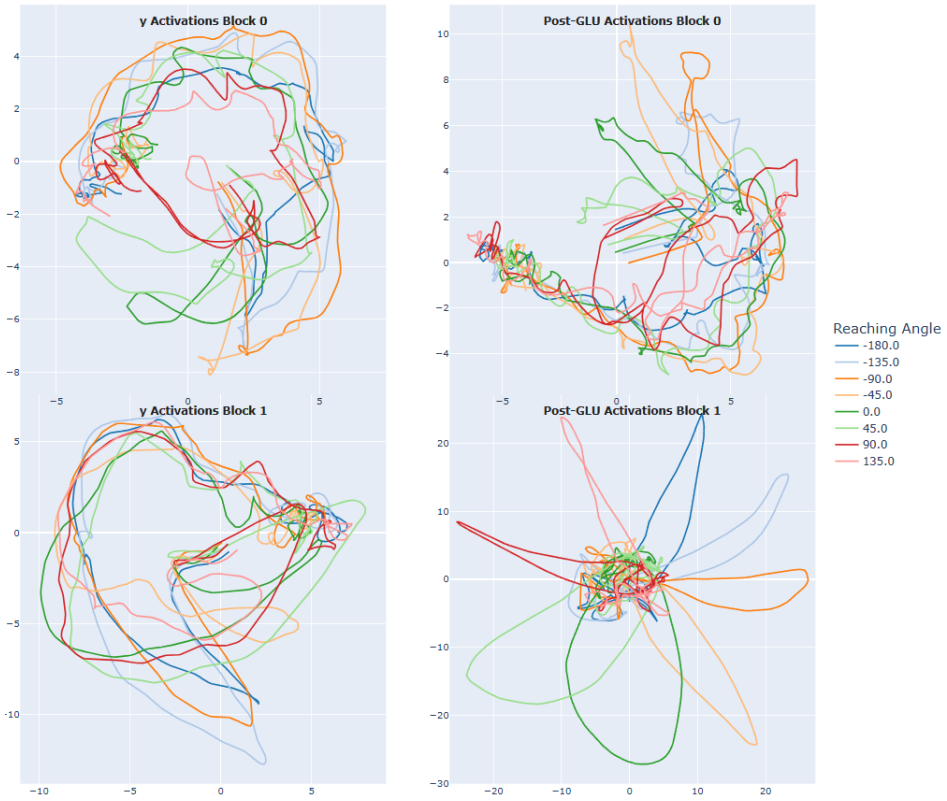

Figure 4.10: **RTT trajectory separation by condition in the final GLU layer.** Condition-Averaged Trajectory plots for various activations in the RTT two-block model, projected into 2D PCA space

Foundational Reaching 2-Block Model As an example, we chose 10 trials from the `os_i_rt` and plotted them in 2D PCA space in Figure 4.11. The explained variance from two PCA dimensions is `~15%` in block 0 and `~30%` in block 1. We observed a similar separation in the final Post GELU/GLU activations. It cannot be determined whether these correspond to particular angles due to the lack of `os_i_rt` reach angle metadata for labelling. Nonetheless the similarity with the RTT was clear. The shapes of the earlier layer activations is harder to ascertain, but clearly have less distinctive shape than the final layer activations.

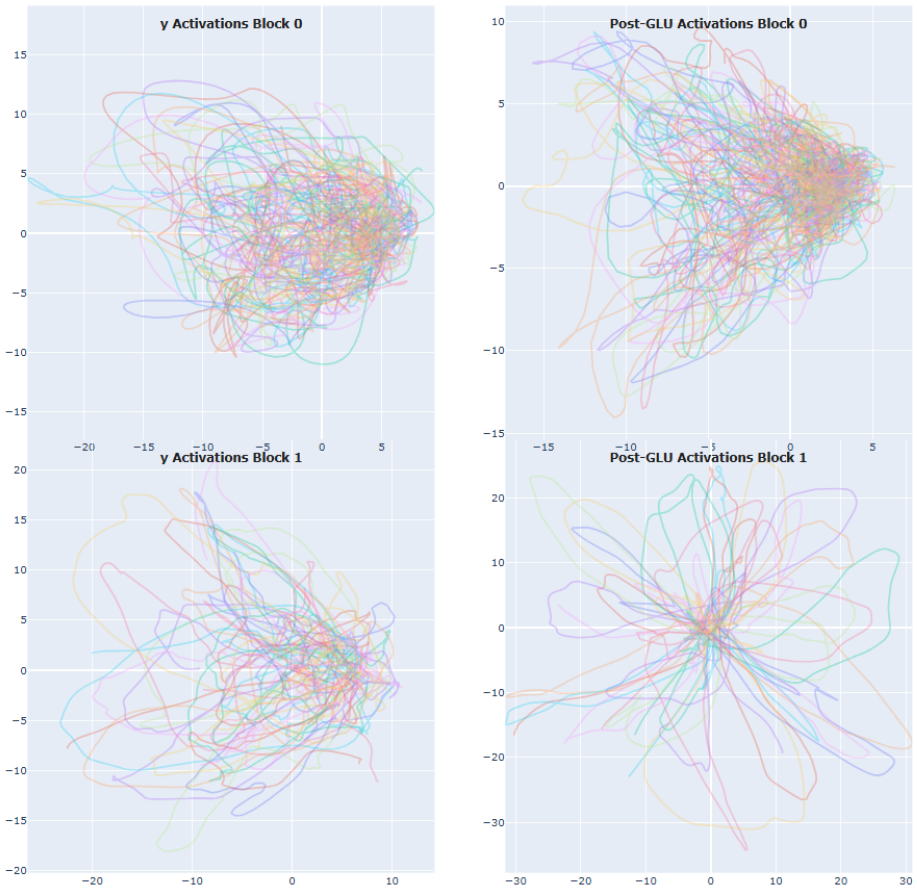

Figure 4.11: **Foundational Model show some weaker angle separation in the final GLU layer.** Condition-Averaged Trajectory plots for various activations in the foundational two-block model, projected into 2D PCA space

4.2.3 Procrustes recognises Center-Out similarity across datasets

In order to compare activations between different foundational datasets with differing magnitudes, number of trials and trial lengths, we z-normalised activations per trial. Since Procrustes works best when trajectories are temporally aligned, we calculated the average activation value for 100 timesteps after bucketing trajectories into 45 degree reaching angle bins, and concatenating bins in the same order for each dataset. We then applied Procrustes similarity all pairs of datasets. We chose to compare activations from the Encoded Neural Input, first post-GLU and last post-GLU stages from the Foundational Reaching 2 and 4 block variants.

After plotting the matrices of similarity in Figure 4.12, we noticed the strongest similarities were between in the final layer of the 2-Block model (bottom left), for the Center-Out (CO)

tasks. There was also mild similarity among Perich-Miller Random-Target tasks. The rest of the components in the 2 or 4 block models did not exhibit much strong similarity among datasets. A possible weakness of this assessment was that Procrustes relies on trajectories being temporally aligned, which may not be strictly possible between datasets, even with our efforts to mitigate this. It is encouraging however, that the Center-Out tasks formed a clear block in the heatmap.

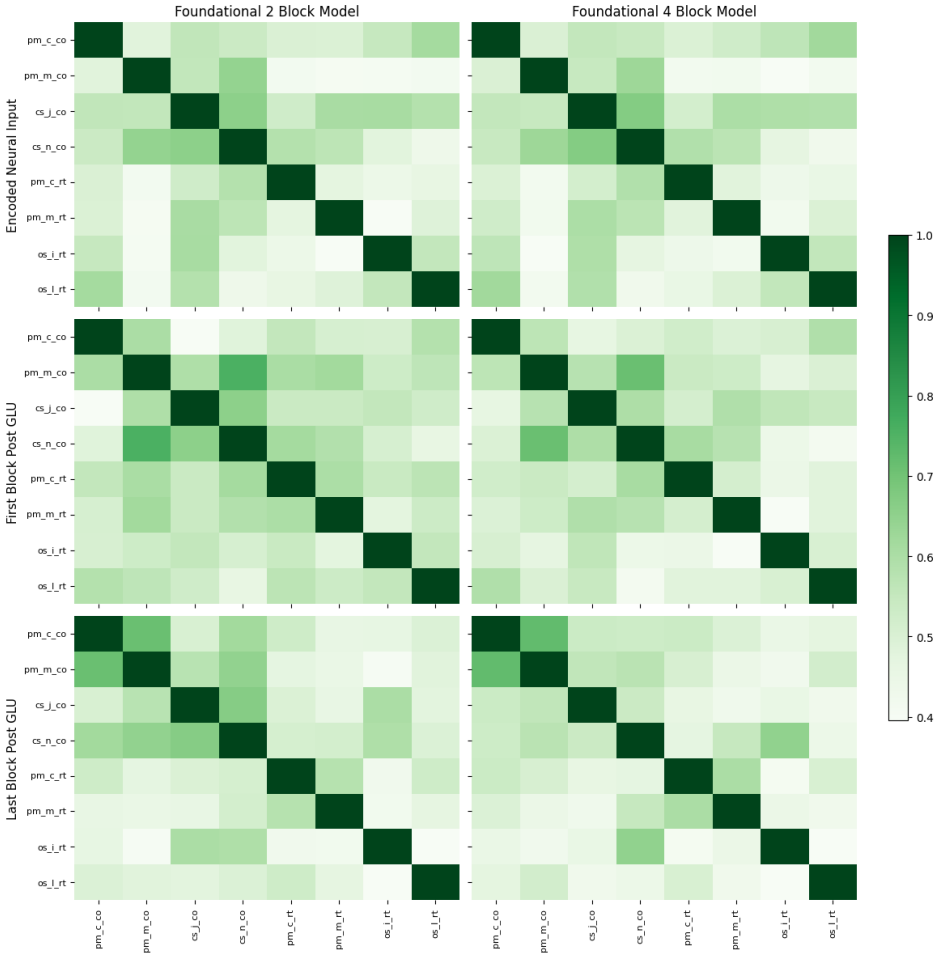

Figure 4.12: **Procrustes shows some similarity among center-out datasets** Procrustes Similarity Matrices across datasets for 2 and 4 block models, encoded neural input, first and last post-glu activations

4.2.4 Task-Subject Encoders converge to modes after training

For this analysis we compared the alignment between the task-subject specific input encoders to the SSM dynamics matrix A. The neural input data is passed through the encoder for task-subject group g and subsequently through the common input matrix B. In then contributes additively to the hidden state h update.

$$x_{\text{enc}} = \text{Encoder}_g(x_{\text{in}}) \tag{4.1}$$

$$\frac{dh(t)}{dt} = Ah(t) + Bx_{\text{enc}} \tag{4.2}$$

We calculated the cosine similarity of the top 6 left singular vectors of the Encoder*B weights vs the top 6 right singular vectors of the A matrix. This gave a metric of alignment between the dominant modes of the input space with the dynamical modes within the SSM block. Each similarity was multiplied by the two singular values from each pair, scaled by the max singular value such that the final metric has max value 1. This ensured that we would avoid highlighting alignment between weaker modes.

In Figure 4.13, we plotted the alignment of the top 6 modes for each task-subject specific group for the Foundational 2-Block model, along with an untrained encoder as a baseline. The untrained encoder did not appear to have much alignment in the dominant modes. After training, each group encoder formed stronger alignment in certain modes. There did seem to be some variability among datasets, suggesting that the model projects a given dataset into the recurrent dynamical space in slightly different ways.

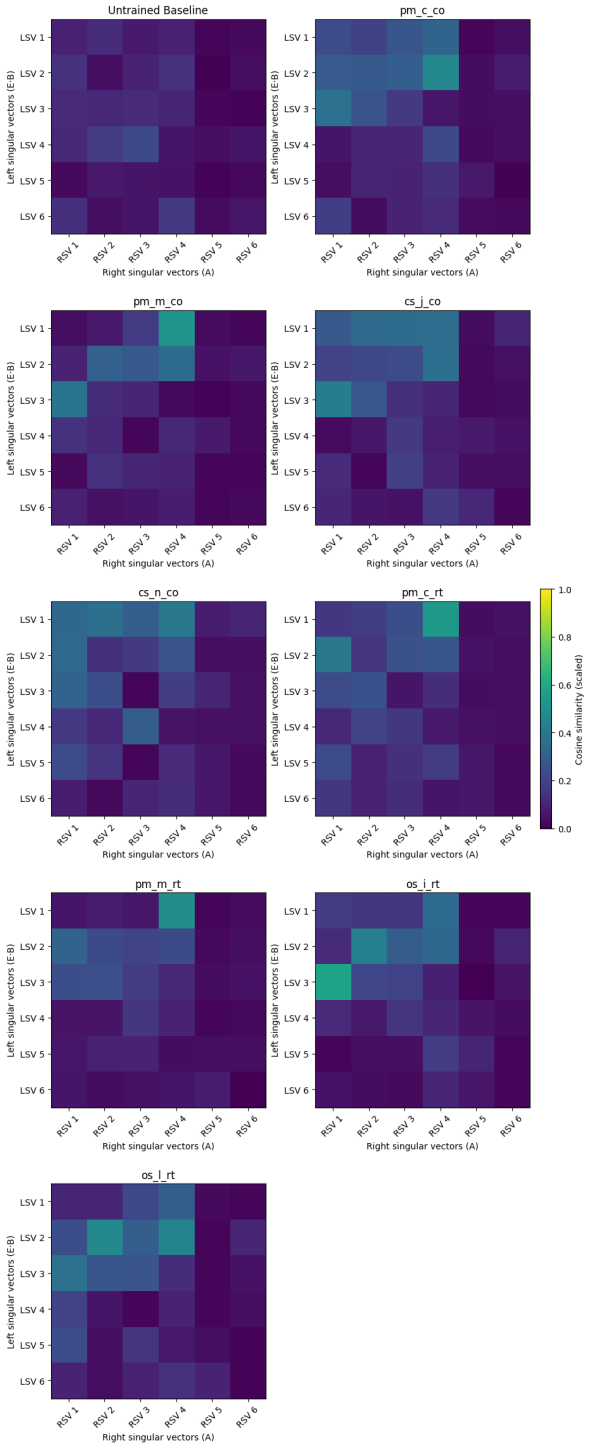

Figure 4.13: **Task-Subject specific decoders show alignment with recurrent modes after training.** Cosine Similarity between Encoders and A matrix for Foundational Two-Block Model

In order to quantify the similarity of alignment between datasets, we computed another

cosine similarity of the flattened similarity matrix values above, represented in Figure 4.14. This did not paint a clear picture as to how well aligned these similarity matrices are between subjects and tasks, as they all had a high similarity score. However, it's clear that the untrained encoder is different from any trained encoder, supporting the idea that training changes the way in which the model encodes input data.

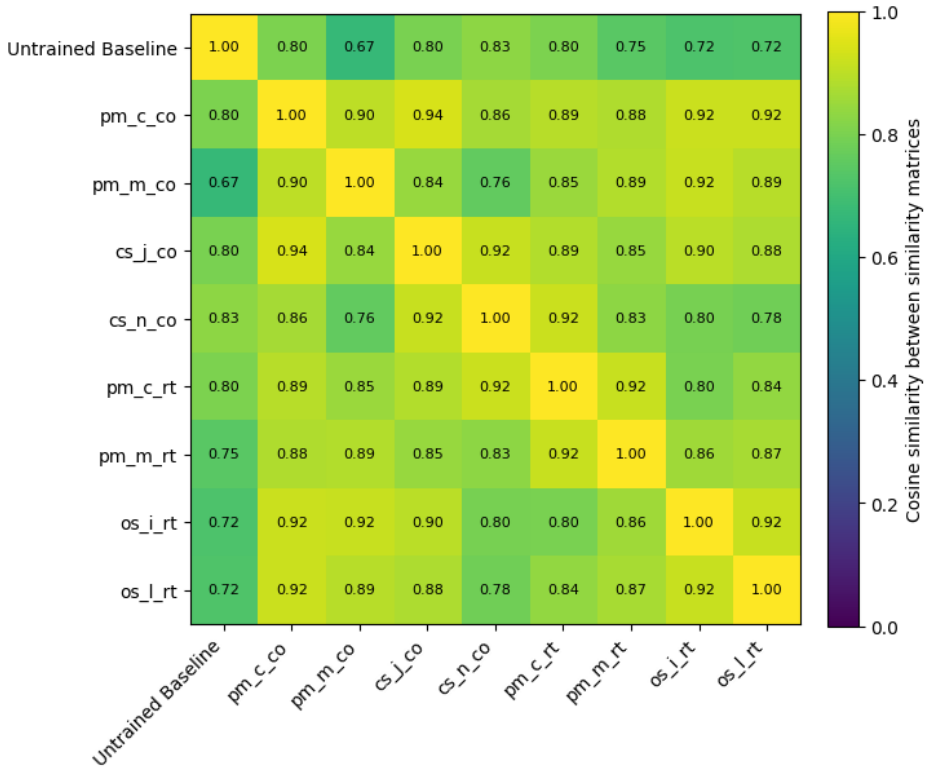

Figure 4.14: **Trained encoders are relatively similar across task types and subjects.** Cosine Similarity of Similarities between Encoders and A matrix for Foundational Two-Block Model

4.2.5 Behaviour decoding improves throughout model

We fitted both ridge regression models, for decoding hand trajectory behaviour, and logistic regression models, for decoding subject and task class derived from the dataset labels, to activations z-normalised by trial. We concatenated activations from all datasets to produce global behaviour, task and subject decoding scores per model component. Behaviour was predicted for each timestep in each trial. Subject and task decoding were predicted from concatenating 100 timesteps together. Individual behaviour decoding per dataset was also considered. Ten different random train-test splits were performed and 5/95% percentile intervals were bootstrapped. R^2 quantified the performance of the ridge models and % accuracy for the classification models. Median score was reported in all cases.

Foundational Reaching 2-Block Decoding Referring to Figure 4.15, we observed that global and dataset-specific behaviour decoding R^2 improved as we progressed further into the model. In general, components from the final block provided a major improvement in R^2. The center-out tasks outperformed the random-target tasks to a large degree in behaviour decoding. The GLU only provided slight improvement in decoding in the final layer. For decoding behaviour in general, the error bands produced from bootstrapping were barely visible.

Global task class decoding accuracy was consistently high throughout the model, not falling below 80% accuracy, indicating that the activation profile over a larger number of timesteps gives an indicative profile of whether the model is performing center-out or random-target reaching.

Global subject decoding was again constant at around 60%, but still well above the baseline of randomly guessing the subject. Given the model was trained to output hand velocity behaviour, it makes sense that within the model, keeping track of what task is being performed is more important than what subject produced the hand movement.

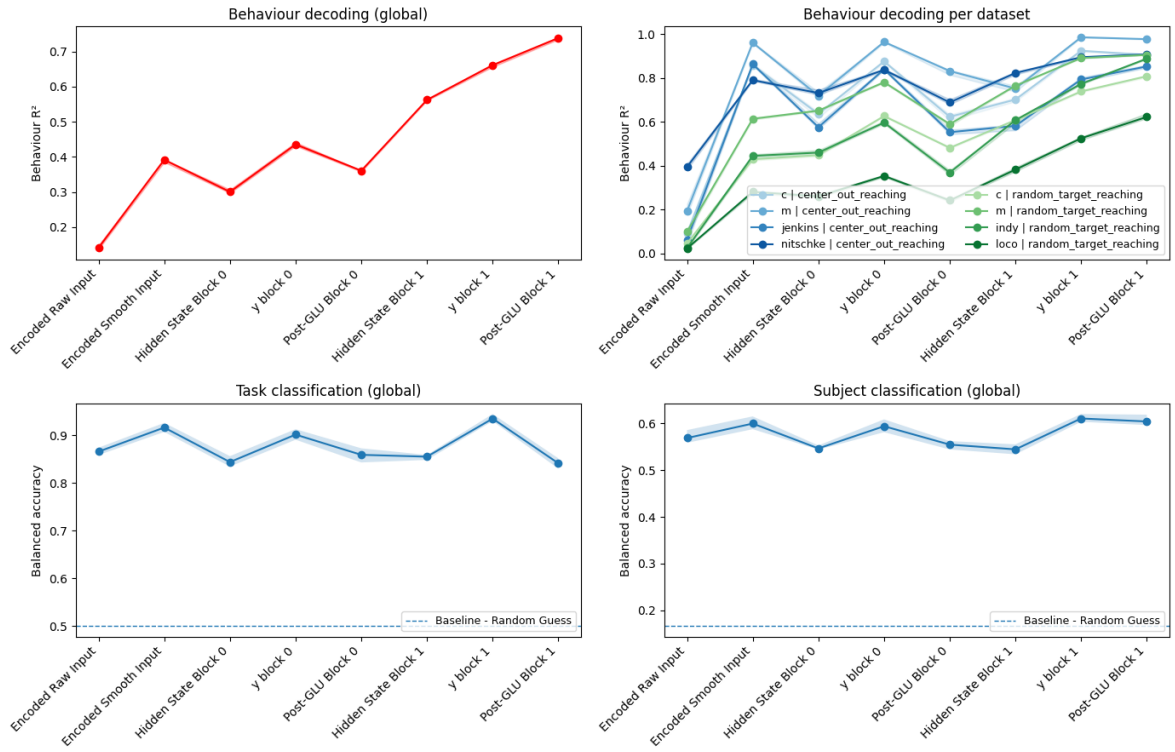

Figure 4.15: **Foundational model shows strong decoding from activations to behaviour and task, weaker for subject decoding.** Top left: Decoding to behaviour from concatenated activations improves from model input to final activation. Top right: Center-out tasks generally decode better than Random-Task datasets. Bottom left: Task decoding is high throughout the model. Bottom right: Subject decoding is constant but lower than task decoding.

Task Specific RTT 2-Block Decoding Given this model was trained from scratch on a single `mc_rtt` dataset, we could only perform global behaviour decoding. In Figure 4.16 we observed that this task specific model decoded better from its x_{ssm} layers but otherwise a similar pattern emerges where behaviour decoding improved as you progress through the layers.

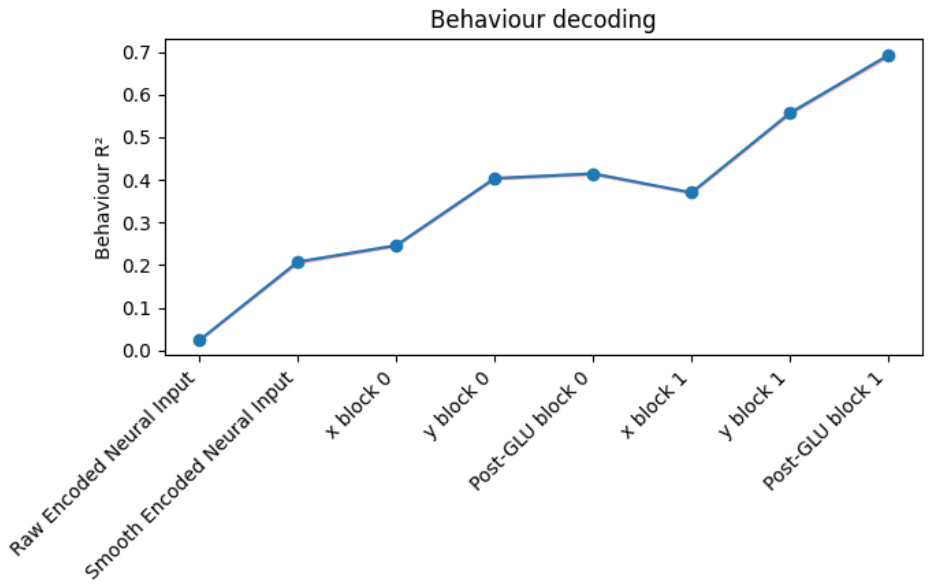

Figure 4.16: **RTT Model decoding to behaviour improves throughout the model.** Decoding to behaviour for RTT Two-Block Model across various stages of model activations

4.2.6 Foundational Model exhibits shared dimensionality

In this section we calculated Effective Dimensionality and Participation Ratio of model activations at various stages of the 2 block models. This helped determine the relative compression and expansion of the data, from input to output.

Foundational Reaching 2-Block Model Given the activations from the various datasets had different number of trials and timesteps, we again flattened trials and timesteps into one dimension and took the largest common number of flattened timesteps before concatenating into a single set of activations for the whole model. We then computed ED and PR for the concatenated and single dataset activations throughout the model.

In Figure 4.17 we observed that in the foundational model, the ED/PR generally fluctuated throughout the model, with the complex hidden state x_{ssm} generally expanding the data and real values y compressing the data. In the final layers, the dimensionality across datasets was compressed into a final lower effective dimension, as is prepares to be decoded to final behavioural output. The reaching tasks generally required more effective dimensions for their computations, which may be due to that task type being less structured than the center-out tasks. The concatenated activations had lower PR/ED compared to the highest-dimensional individual dataset (`os_l_rt`). This implies there is shared structure among tasks and that the dimensionality needed to compute behaviour for all datasets is less than the sum of the

individual tasks. Referring back to the decoding results 4.15, we observed that the lower the effective dimensionality for a given dataset, the higher the ability to decode from that models activations.

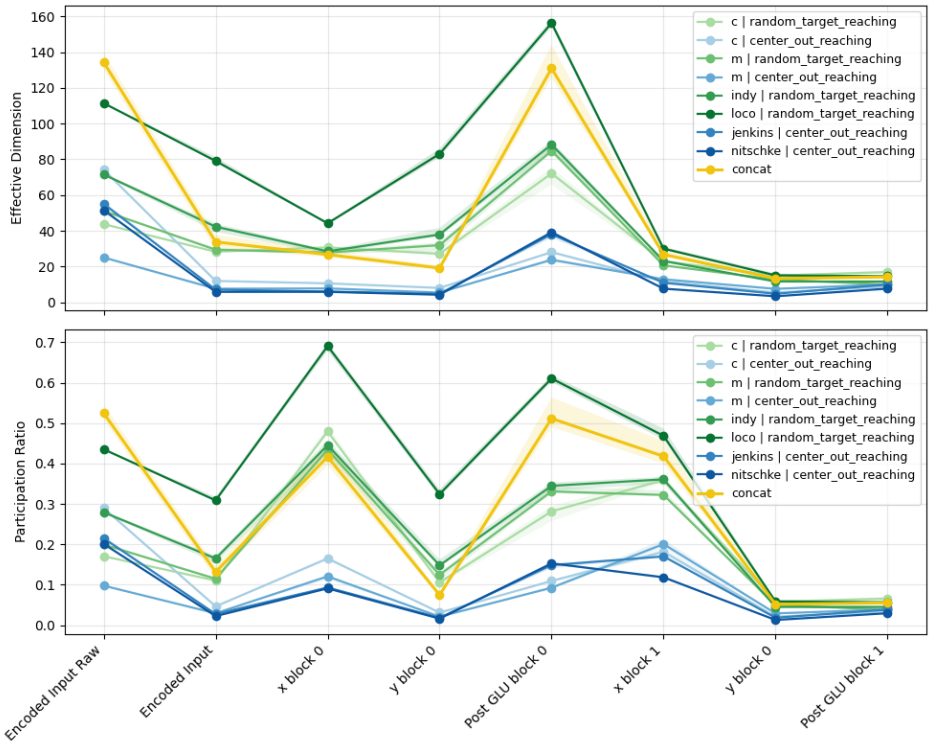

Figure 4.17: **Effective Dimensionality higher for Random-Target than Center-out datasets. Concatenated activations use lower dimensionality than sum of individual datasets.** Effective Dimensionality and Participation Ratio for model component activations in Foundational Reaching Two-Block Model

Task-Specific RTT 2-Block Model According to Figure 4.18, the effective dimensionality declined throughout the model. There are sharper spikes in PR in the hidden state x_{ssm} layers, which indicated that it used a higher percent of its true dimensions for computation. The GELU/GLU lowered dimensionality, in contrast to the foundational model. The single task model needed less dimensions to decode to behaviour in the final layer compared to the multi task foundational model.

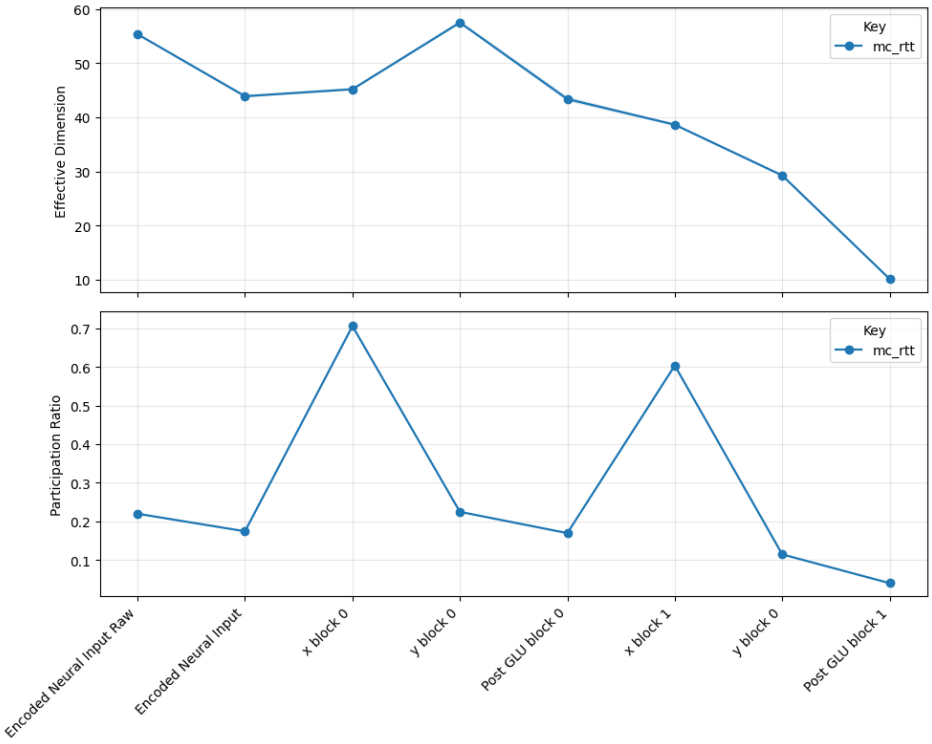

Figure 4.18: RTT model's effective dimensionality declines throughout the model. GLU lowers dimensionality. Effective Dimensionality/Participation Ratio for model component activations in RTT Two-Block Model

4.2.7 Gated Linear Units appear to modulate context

In order to ascertain the function of gating in the Gated Linear Unit across datasets, we passed the Foundational 2-Block y activations first through the GELU and then through the second sigmoid gating term in the GLU equation 3.42. The gated values were averaged across trials and timepoints per unit/channel to see if certain units remain consistently open or closed per dataset.

We plotted these values as a heatmap in Figure 4.19, choosing the top 25 units by inter-dataset variance. The average gate values showed strong on/off gating in certain dataset such as the Churchland-Shenoy (`cs_*`) in both blocks. For all datasets, there did appear to be a small amount of units which remained open or shut uniquely for a given dataset in both blocks.

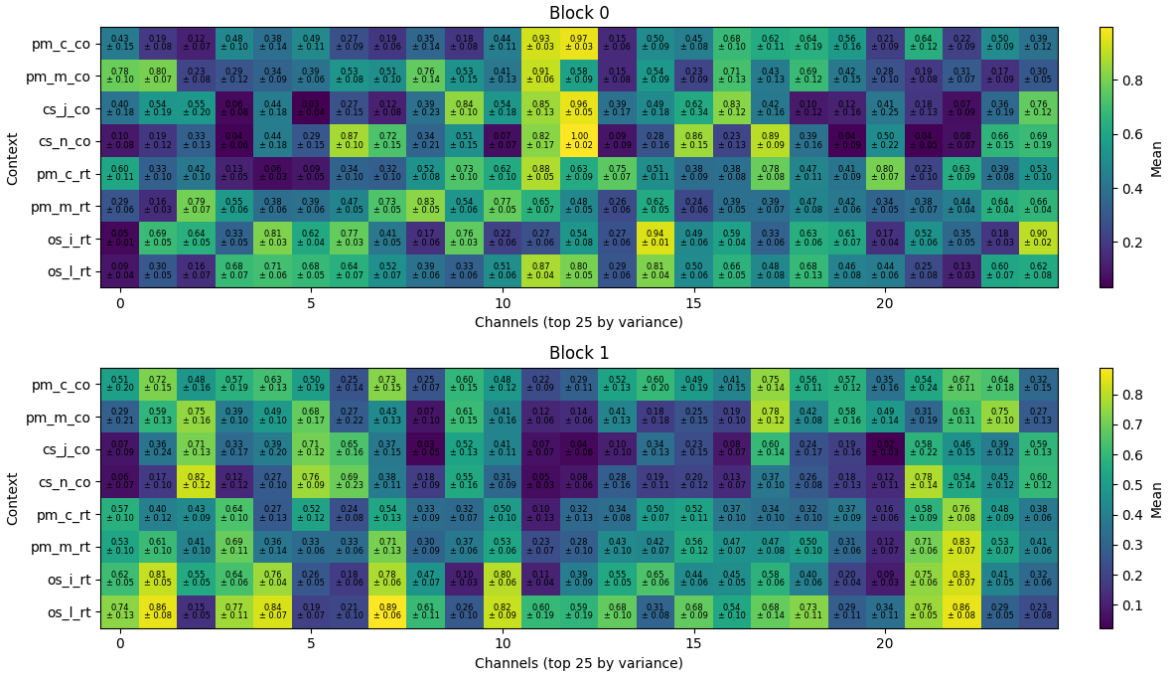

Figure 4.19: **GLU appears to hold certain channels open or shut for certain datasets.** Heatmap of average GLU gate values per dataset for Foundational Two-Block Model

Next we looked at the cosine similarity between full average gate vectors in Figure 4.20. There was strong similarity between the gate vectors for all datasets, suggesting a large amount of shared functionality in the GLU units. We observed slightly stronger similarity among random-target tasks in general, Perich-Miller Center-Out, and Churchland-Shenoy Center-Out tasks in block 1. The GLU gate does appear to moderate dataset and task identity to a degree.

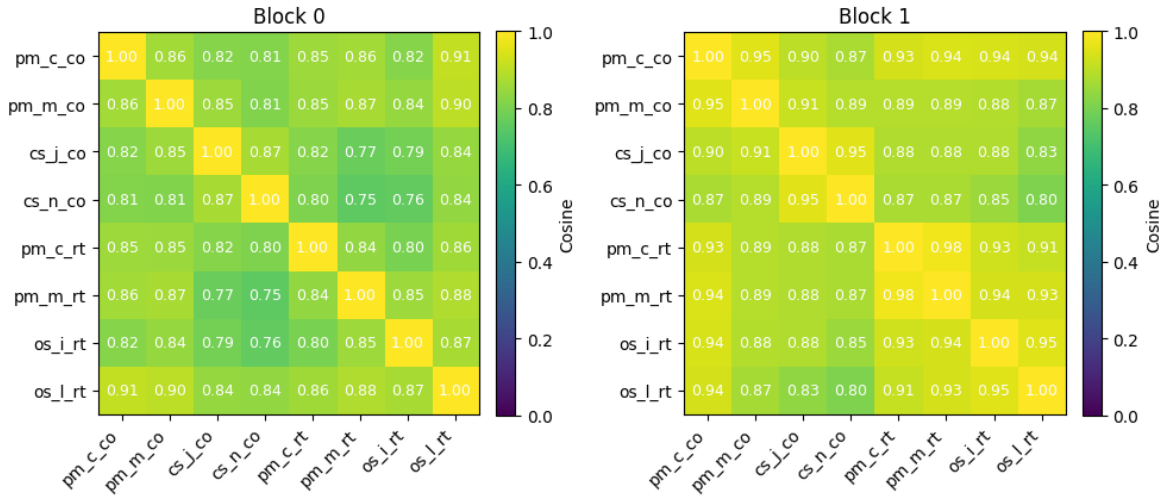

Figure 4.20: **GLU gate vectors overall similar across datasets, slightly stronger similarity among random-target datasets.** Cosine similarity of average GLU gate values per dataset for Foundational Two-Block Model

Chapter 5

Conclusion

This thesis began from the observation that both biological and artificial neural systems can be viewed as dynamical systems, yet their internal mechanisms are often opaque. The central aim was to examine whether similarity metrics and linear algebraic tools can reveal meaningful structure in models that otherwise appear functionally equivalent when judged by input–output behaviour alone. We approached this through two main avenues: controlled synthetic experiments on toy recurrent networks, and analyses of modern structured state-space models trained on neural datasets.

Discussion

The synthetic toy RNN experiments confirmed that even simple recurrent models trained on memory tasks develop hidden dynamics organised around attractor–saddle geometries, consistent with classical observations in dynamical systems neuroscience by Sussillo [50]. Training improved the model's ability to recall the last non-zero input and produced trajectories with low-dimensional structure, echoing prior reports of cube-shaped trajectories in higher-bit tasks. Importantly, the similarity metrics diverged in their effectiveness.

Procrustes alignment successfully detected distinctions between models with different seeds, inputs, and architectures, while DSA largely failed to discriminate and reported uniformly high similarity. This contrast highlights that input-driven dynamics may undermine trajectory-local metrics such as DSA, while trajectory alignment methods like Procrustes remain useful under noisy or misaligned conditions. Given a tensor of random values was used as a control, and DSA did indeed pick up on this by assigning lower similarity, there was robustness in this experimental setup. However, these results contradict prior work such as Guilhot et al [19], who found DSA to be superior to Procrustes in discriminating between different model architectures for other memory tasks such as Delay/Anti-Delay. Future work could involve replicating prior Procrustes and DSA results under identical experimental conditions. Further tuning of DSA hyperparameters could assess whether its sensitivity improves

in tasks that are less dominated by input and develop stronger autonomous model dynamics.

Extending the framework to structured state-space models (SSMs) trained on neural recordings provided a richer picture. Eigenvalue spectra revealed only modest restructuring, suggesting that these SSMs stabilise but do not radically reshape their eigen-modes during training, echoing findings that stability is preserved even in highly expressive recurrent architectures [27]. Trajectory analysis showed that task-specific RTT models developed clear separation of reach angles in their final GLU layers, while foundational multi-task models exhibited weaker but still structured separation. Procrustes similarity further revealed that centre-out tasks aligned more closely than random-target tasks, indicating shared representational structure across datasets despite differences in recording conditions. Together, these results suggest that deeper SSM blocks refine task-specific geometry in latent trajectories, building on more generic transformations in earlier blocks.

Decoder analyses added behavioural relevance to these observations. Behaviour decoding improved consistently across depth, peaking in final blocks, whereas task decoding remained uniformly high and subject decoding more modest. This indicates that the models strongly prioritise encoding of task structure over subject identity, consistent with the literature on invariant neural representations of task versus individual differences [6]. Dimensionality measures reinforced this interpretation: random-target tasks required more effective dimensions than centre-out tasks, reflecting their greater behavioural variability, while gating mechanisms (GLUs) modulated dimensionality in task-dependent ways. Analysis of gate values further suggested that gating serves as a contextual switch, opening or closing channels in a dataset-specific manner, reminiscent of context-dependent gating reported in biological circuits [32]. These results highlight how architectural components like GLUs do not merely stabilise training, but actively sculpt the dynamical landscape in ways that parallel neural computation.

Several limitations qualify these conclusions. Procrustes relies on temporal alignment and may miss shared structure across tasks with differing time courses. Our analysis was confined to S5-style SSMs and a narrow set of motor cortex datasets, raising questions of generalisability across tasks, architectures, and brain areas. Future experiments could extend to hybrid attention–SSM models [43], or integrate analyses with explicit neuroscientific hypotheses about motor preparation and execution [34]. A more robust experimental setup could include testing similarity metrics on datasets with stronger temporal alignment and richer metadata, where trial structure and task labels are more clearly defined, allowing more reliable cross-dataset comparisons. A natural next step would be to apply these tools to datasets with richer cognitive structure, to examine whether gating and dimensionality modulation generalise to decision-making and memory tasks.

Closing Thoughts

In addressing the research questions posed at the outset, this thesis demonstrated that trajectory- and similarity-based analytical tools can provide meaningful insight into the hidden dynamics of recurrent models. Procrustes analysis, in particular, proved robust and interpretable in distinguishing models that shared inputs or architectures from those that did not, showing that geometry-based alignment can reveal important differences even when input–output behaviour appears identical. By contrast, DSA was less effective under the input-driven conditions we studied, highlighting the importance of choosing tools suited to the dynamical regime under investigation.

We also showed that different components within structured state-space models play complementary roles in shaping computation. Early blocks provided broad transformations, while later blocks refined task-relevant structure, a form of functional decomposition that links to longstanding questions about hierarchical organisation in neural circuits. These findings suggest that similarity metrics can do more than compare whole models: they can also expose the contribution of subcomponents to the overall computation.

Finally, we found that auxiliary mechanisms such as GLU gating are not passive architectural choices but active modulators of dynamics. By altering dimensionality and acting as contextual switches, gating units shaped whether the model emphasised shared versus task-specific representations. This result resonates with evidence from neuroscience on context-dependent gating in prefrontal circuits, underscoring the potential of these architectural motifs to capture biologically relevant computations.

Together, these results demonstrate that internal dynamics are not incidental side effects of training but central computational primitives. More broadly, they support a shared dynamical framework for comparing artificial and biological systems, moving us beyond black-box evaluation toward mechanistic insight into how both kinds of networks implement behaviour.

Bibliography

[1] Mehdi Azabou, Vinam Arora, Venkataramana Ganesh, Ximeng Mao, Santosh Nachimuthu, Michael Mendelson, Blake Richards, Matthew Perich, Guillaume Lajoie, and Eva L. Dyer. A unified, scalable framework for neural population decoding. In *Proceedings of the 37th Conference on Neural Information Processing Systems (NeurIPS) Dataset and Benchmark Track*, 2023.

[2] Mehdi Azabou, Vinam Arora, Venkataramana Ganesh, Ximeng Mao, Santosh Nachimuthu, Michael J. Mendelson, Blake Richards, Matthew G. Perich, Guillaume Lajoie, and Eva L. Dyer. A unified, scalable framework for neural population decoding. In *NeurIPS*, 2023. Project page: poyo-brain.github.io; arXiv:2310.16046.

[3] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, New York, 2006.

[4] Bo Chang, Minmin Chen, Eldad Haber, and Ed H. Chi. Antisymmetricrnn: A dynamical system view on recurrent neural networks, 2019.

[5] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 1724–1734, 2014.

[6] Mark M. Churchland, John P. Cunningham, Matthew T. Kaufman, Stephen I. Ryu, and Krishna V. Shenoy. Neural population dynamics during reaching. *Nature*, 487(7405):51–56, 2012.

[7] John P. Cunningham and Byron M. Yu. Dimensionality reduction for large-scale neural recordings. *Nature Neuroscience*, 17:1500–1509, 2014.

[8] John P. Cunningham and Byron M. Yu. Dimensionality reduction for large-scale neural recordings. *Nature Neuroscience*, 17(11):1500–1509, 2014.

[9] Laura N. Driscoll, Krishna V. Shenoy, and David Sussillo. Flexible multitask computation

in recurrent networks utilizes shared dynamical motifs. *Nature Neuroscience*, 27:1349–1363, 2024.

[10] Bradley Efron and Robert J. Tibshirani. *An Introduction to the Bootstrap*. Monographs on Statistics and Applied Probability. Chapman & Hall/CRC, New York, 1994.

[11] Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990.

[12] Peiran Gao, Eric Trautmann, Byron M. Yu, Gopal Santhanam, Stephen I. Ryu, Krishna V. Shenoy, and Surya Ganguli. A theory of multineuronal dimensionality, dynamics and measurement. *bioRxiv*, page 214262, 2017.

[13] Apostolos P. Georgopoulos, Andrew B. Schwartz, and Ronald E. Kettner. Neuronal population coding of movement direction. *Science*, 233(4771):1416–1419, 1986.

[14] Mark S. Goldman. Memory without feedback in a neural network. *Neuron*, 61(4):621–634, 2009.

[15] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, 4 edition, 2013.

[16] John C. Gower. Generalized procrustes analysis. *Psychometrika*, 40(1):33–51, 1975.

[17] Albert Gu, Tri Dao, Stefano Ermon, Atri Rudra, and Christopher Ré. Hippo-s4d: Simplified state space layers for sequence modeling. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

[18] Albert Gu, Karan Goel, and Christopher Ré. Efficiently modeling long sequences with structured state spaces. In *International Conference on Learning Representations*, 2021.

[19] Romain Guilhot, Leo Kozachkov, and Ila Fiete. Dynamical similarity analysis uniquely captures how computations develop in rnns. *bioRxiv*, 2024.

[20] Guillaume Hennequin, Tim P. Vogels, and Wulfram Gerstner. Optimal control of transient dynamics in balanced networks supports generation of complex movements. *Neuron*, 82(6):1394–1406, 2014.

[21] Morris W. Hirsch, Stephen Smale, and Robert L. Devaney. *Differential Equations, Dynamical Systems, and an Introduction to Chaos*. Academic Press, 2004.

[22] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[23] Arthur E. Hoerl and Robert W. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.

[24] Roger A. Horn and Charles R. Johnson. *Matrix Analysis*. Cambridge University Press, 2nd edition, 2012.

[25] Harold Hotelling. Relations between two sets of variates. *Biometrika*, 28(3/4):321–377, 1936.

[26] Ehsan Imani, Wei Hu, and Martha White. Representation alignment in neural networks. *Transactions on Machine Learning Research*, 2022. Published in TMLR—showing neural network representations align their top singular vectors to the targets.

[27] Cecilia Jarne. Different eigenvalue distributions encode the same temporal tasks in recurrent neural networks. *arXiv preprint arXiv:2005.13074*, 2020.

[28] Ian T. Jolliffe. *Principal Component Analysis*. Springer, 2 edition, 2002.

[29] Ian T. Jolliffe and Jorge Cadima. Principal component analysis: a review and recent developments. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 374(2065):20150202, 2016.

[30] Rudolf E. Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(1):35–45, 1960.

[31] Mikhail A. Lebedev and Miguel A. L. Nicolelis. Brain–machine interfaces: past, present and future. *Trends in Neurosciences*, 29(9):536–546, 2006.

[32] Christian K. Machens, Ranulfo Romo, and Carlos D. Brody. Functional, but not anatomical, separation of "what" and "when" in prefrontal cortex. *Journal of Neuroscience*, 30(1):350–360, 2010.

[33] Joseph G. Makin, Joseph E. O'Doherty, Mariana M. B. Cardoso, and Philip N. Sabes. Superior arm-movement decoding from cortex with a new, unsupervised-learning algorithm. *Journal of Neural Engineering*, 15(2):026010, 2018.

[34] Valerio Mante, David Sussillo, Krishna V. Shenoy, and William T. Newsome. Context-dependent computation by recurrent dynamics in prefrontal cortex. *Nature*, 503:78–84, 2013.

[35] David Marr and Ellen Hildreth. Theory of edge detection. *Proceedings of the Royal Society of London. Series B. Biological Sciences*, 207(1167):187–217, 1980.

[36] Brian C. Moore. Principal component analysis in linear systems: Controllability, observability, and model reduction. *IEEE Transactions on Automatic Control*, 26(1):17–32, 1981.

[37] Joseph E. O'Doherty, Maria M. B. Cardoso, John G. Makin, and Paul N. Sabes. Nonhuman primate reaching with multichannel sensorimotor cortex electrophysiology. *Zenodo*, 2017.

[38] Mitchell Ostrow, Adam Eisen, Leo Kozachkov, and Ila Fiete. Beyond geometry: Comparing the temporal structure of computation in neural circuits with dynamical similarity analysis. *arXiv*, abs/2306.10168, 2023.

[39] Liam Paninski and John P. Cunningham. Neural data science: accelerating the experiment–analysis–theory cycle in large-scale neuroscience. *Neuron*, 99(2):199–201, 2018.

[40] Liam Paninski, Jonathan W. Pillow, and Eero P. Simoncelli. Statistical models for neural encoding, decoding, and optimal stimulus design. *Progress in Brain Research*, 165:493–507, 2007.

[41] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[42] Matthew G. Perich, Juan A. Gallego, and Lee E. Miller. A neural population mechanism for rapid learning. *Neuron*, 100(4):964–976.e7, 2018.

[43] Avery Hee-Woon Ryoo, Nanda H. Krishna, Ximeng Mao, Mehdi Azabou, Eva L. Dyer, Matthew G. Perich, and Guillaume Lajoie. Generalizable, real-time neural decoding with hybrid state-space models (possm), 2025.

[44] Peter H. Schönemann. A generalized solution of the orthogonal procrustes problem. *Psychometrika*, 31(1):1–10, 1966.

[45] Jimmy T.H. Smith, Andrew Warrington, and Scott W. Linderman. Simplified state space layers for sequence modeling. arXiv preprint arXiv:2208.04933, 2023. Last revised 3 March 2023.

[46] Ian H. Stevenson and Konrad P. Kording. How advances in neural recording affect data analysis. *Nature Neuroscience*, 14(2):139–142, 2011.

[47] Gilbert Strang. *Linear Algebra and Its Applications*. Brooks Cole, 4th edition, 2006.

[48] Steven H. Strogatz. *Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry, and Engineering*. Addison-Wesley, 1994.

[49] David Sussillo. Neural circuits as computational dynamical systems. *Current Opinion in Neurobiology*, 25:156–163, 2014.

[50] David Sussillo and Omri Barak. Opening the black box: Low-dimensional dynamics in high-dimensional recurrent neural networks. *Neural Computation*, 25(3):626–649, 2013.

[51] Adrian Valente, Jonathan W. Pillow, and Srdjan Ostojic. Extracting computational mechanisms from neural data using low-rank rnns. In *Advances in Neural Information Processing Systems*, volume 35, pages 9877–9890, 2022.

[52] Leena Vankadara, Zeyu Liao, Kamyar Azizzadenesheli, and Anima Anandkumar. On feature learning in structured state space models. In *Proceedings of the 38th International Conference on Neural Information Processing Systems (NeurIPS)*, 2024.

[53] T. L. Veuthey, K. Derosier, S. Kondapavulur, and K. Ganguly. Single-trial cross-area neural population dynamics during long-term skill learning. *Nature Communications*, 11(1):4057, 2020.

[54] Pauli Virtanen et al. Scipy 1.0: Fundamental algorithms for scientific computing in python. *Nature Methods*, 17:261–272, 2020.

[55] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, CJ Carey, Ilhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Richard K. Archibald, Antonio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental algorithms for scientific computing in python. *Nature Methods*, 17:261–272, 2020.

[56] Yifan Wang, Xu Ma, Yitian Zhang, Zhongruo Wang, Sung-Cheol Kim, Vahid Mirjalili, Vidya Renganathan, and Yun Fu. Gmnet: Revisiting gating mechanisms from a frequency view. *arXiv preprint arXiv:2503.22841*, 2025. Under review.

[57] Stephen Wiggins. *Introduction to Applied Nonlinear Dynamical Systems and Chaos*. Springer, 2003.

Appendix A

Use of AI

ChatGPT was used to produce docstrings, function comments and to refactor otherwise original code into a repo structure. All other output was the work of the author.

Appendix B

Other Figures

B.1 Toy RNN Experiments

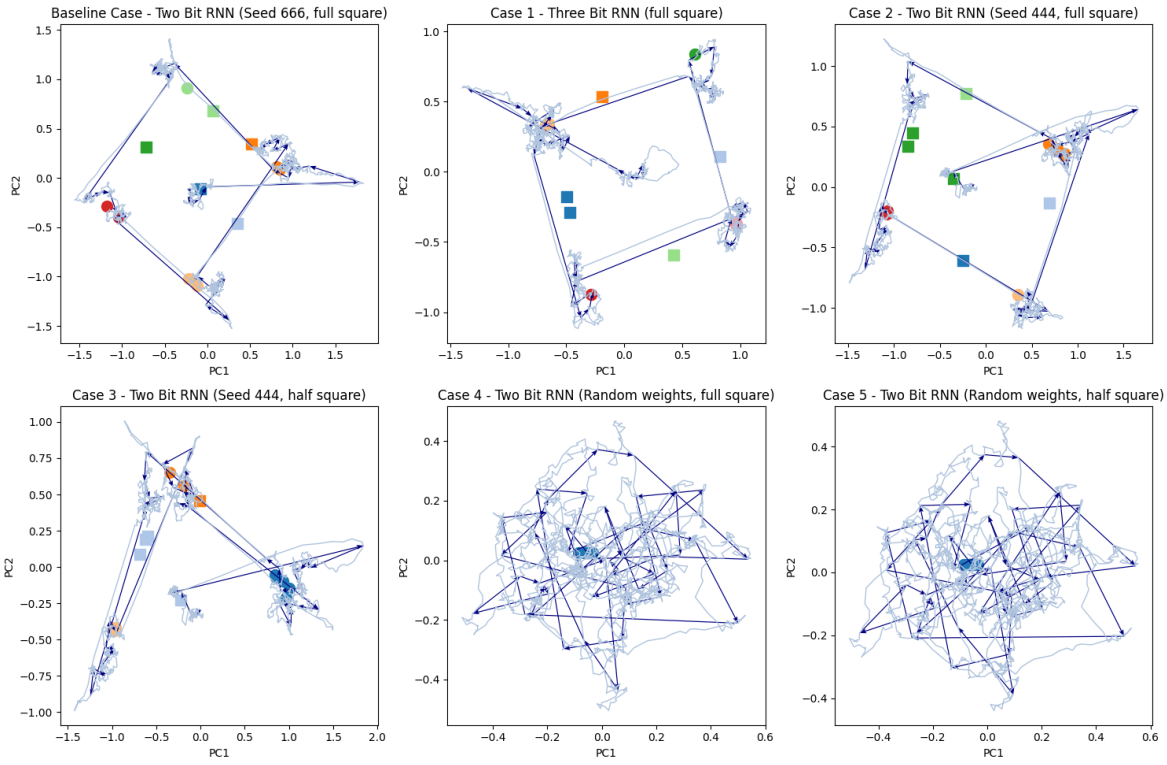

Figure B.1: 2D PCA plots for toy RNN noisy trajectories under full and half square input. □ corresponds to a saddle point and ○ an attractor.

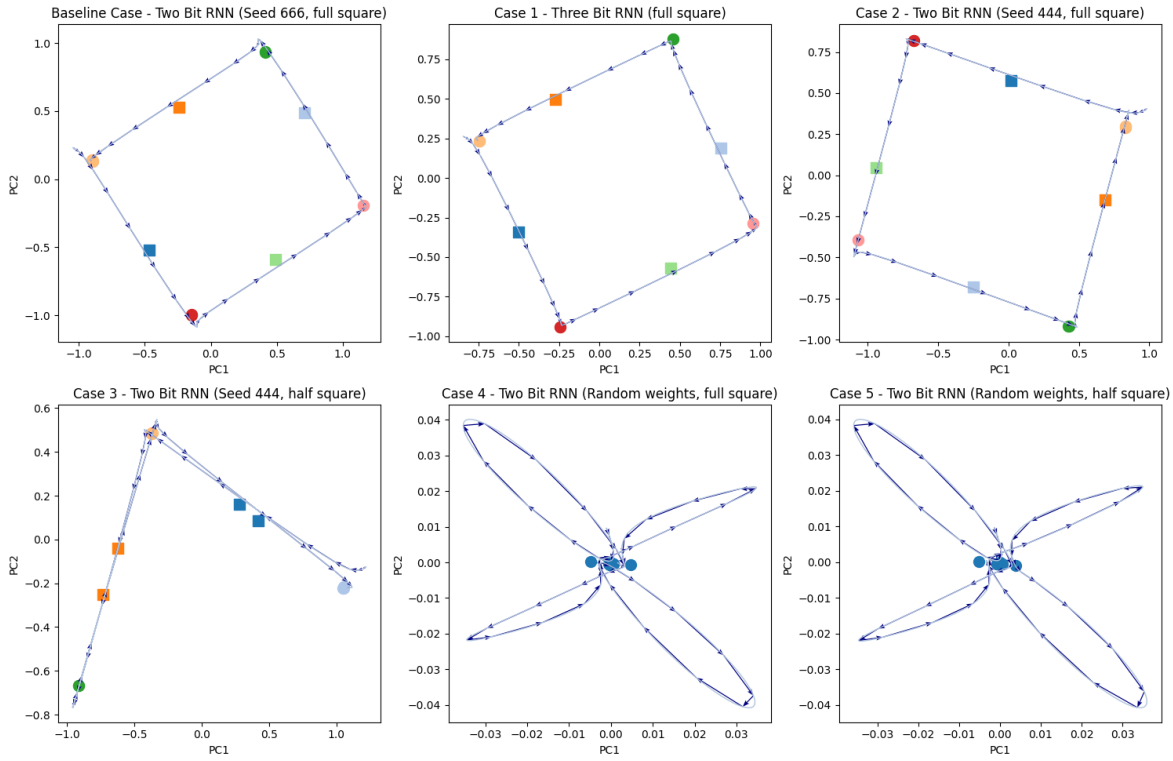

Figure B.2: 2D PCA plots for toy RNN trajectories under full and half square input, with gaussian smoothing applied. □ corresponds to a saddle point and ○ an attractor.

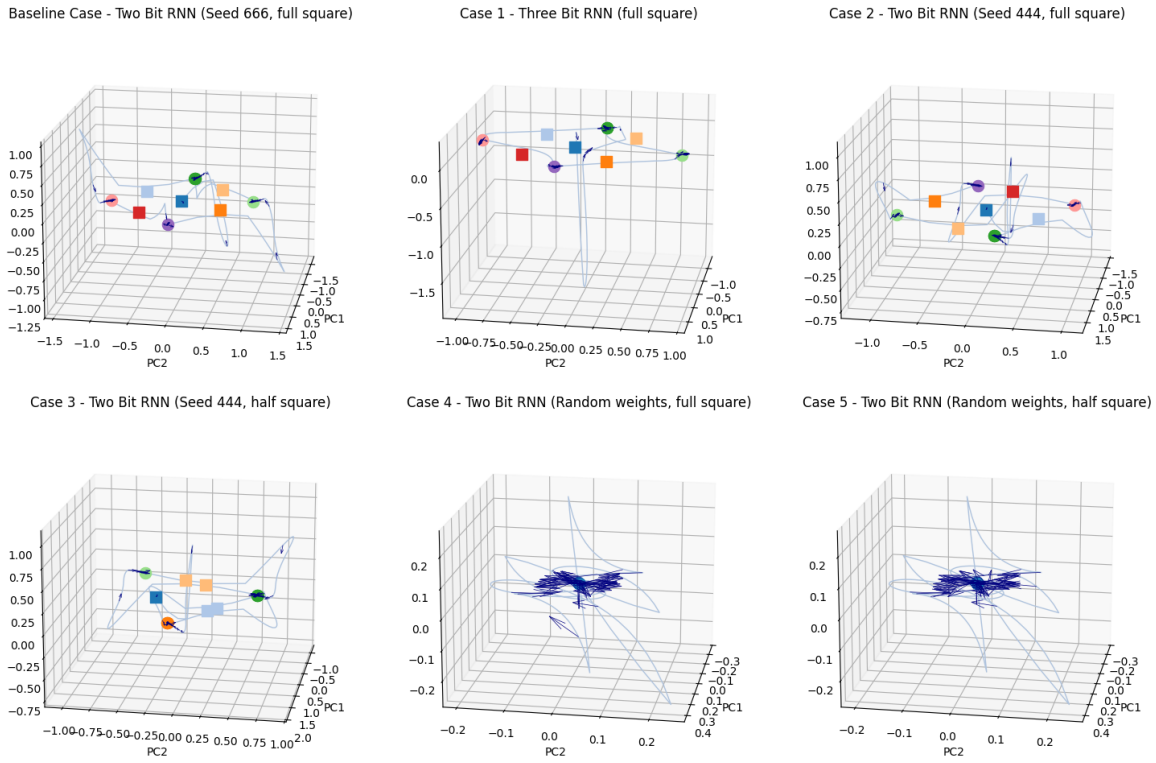

Figure B.3: 3D PCA plots for toy RNN trajectories under full and half square input. □ corresponds to a saddle point and ○ an attractor.

B.2 Neural Encoder Experiments

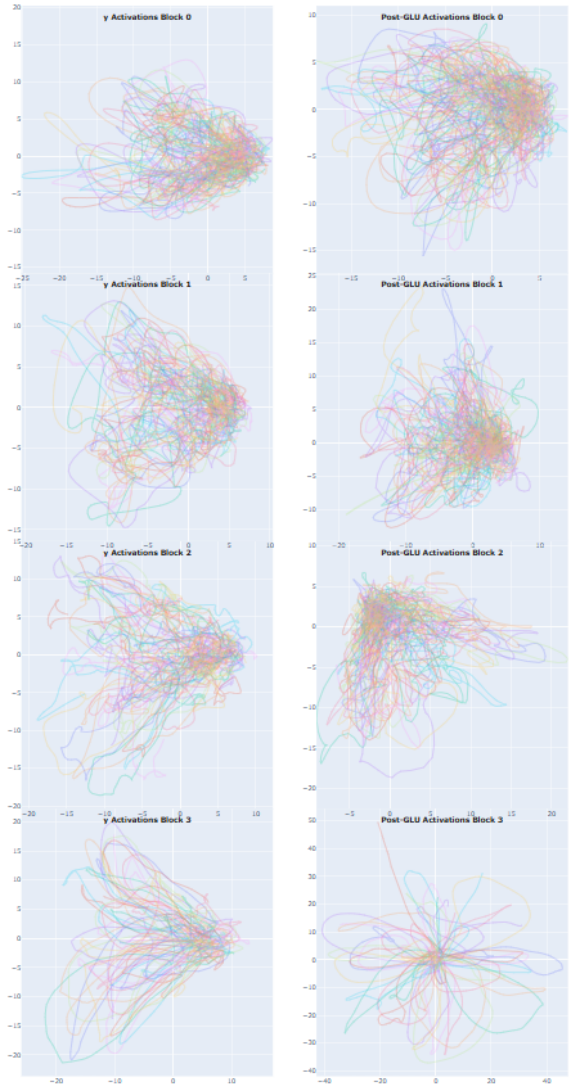

Figure B.4: Condition-Averaged Trajectory plots for various activations in the foundational four-block model, projected into 2D PCA space

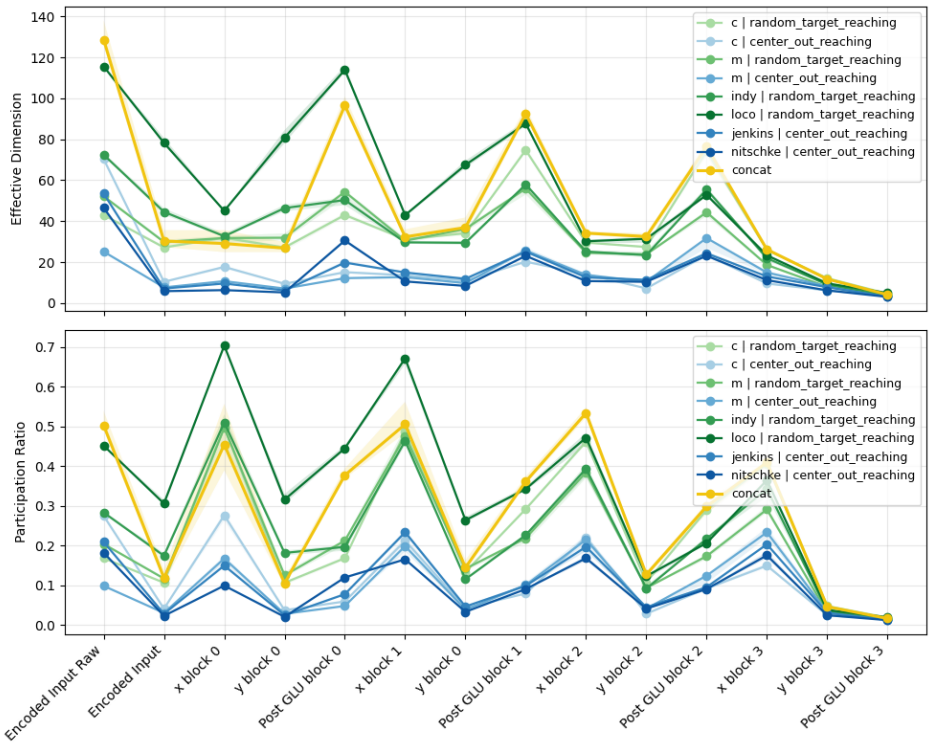

Figure B.5: **Effective Dimensionality higher for Random-Target than Center-out datasets. Concatenated activations use lower dimensionality than sum of individual datasets.** Effective Dimensionality and Participation Ratio for model component activations in Foundational Reaching Four-Block Model