Bridging Neural Activity and Computation with Recurrent Neural Networks

Dissertation

der Mathematisch-Naturwissenschaftlichen Fakultät

der Eberhard Karls Universität Tübingen

zur Erlangung des Grades eines

Doktors der Naturwissenschaften

(Dr. rer. nat.)

vorgelegt von

Matthijs Pals

aus Emmen/den Niederlande

Tübingen

2026

Gedruckt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der Eberhard Karls Universität Tübingen.

Tag der mündlichen Qualifikation:	15.06.2026
Dekan:	Prof. Dr. Thilo Stehle
1. Berichterstatter:	Prof. Dr. Jakob Macke
2. Berichterstatterin:	Prof. Dr. Anna Levina

ABSTRACT

How does cognition arise from the coordinated activity of billions of neurons? The field of computational neuroscience aims to answer such questions through mathematical modeling and simulation. While early computational efforts often consisted of interpretable, hand-designed models based on recordings of a handful of neurons, current experimental techniques enable capturing the activity of hundreds to thousands of units. These large-scale recordings reveal complex, heterogeneous, time-varying dynamics. This necessitates the use of models that are expressive enough to capture the high-dimensional neural activity, while still being able to provide understanding. We pursue this dual objective by using recurrent neural networks (RNNs).

RNNs are an expressive, biologically inspired model class that has emerged as a powerful tool for generating hypotheses about neural computation. Yet, it remains challenging to fit these models to neural data in a way that accurately captures variability and scales to modern datasets. In this thesis, we introduce a method for fitting stochastic RNNs to multi-session neural recordings at single-trial resolution. To facilitate interpretability of large networks, we derive a bound on the number of fixed points in low-rank, piecewise-linear RNNs.

We apply RNNs to investigate two neuroscientific questions related to working memory. First, using task-trained RNNs, we show how low-dimensional attractor dynamics support phase-based coding of working memory. We identify a connectivity structure that couples the phase of intrinsic oscillations to an external oscillatory input, in a multi-stable manner. Second, using large-scale recordings from macaques performing sequence working memory tasks, we test for the presence of either attractor or transient dynamics. With our method for fitting stochastic RNNs, we obtain a nuanced picture in which stable attractors guide memory maintenance alongside substantial temporal variability.

Overall, this thesis contributes a framework that uses machine learning to bridge low-dimensional RNN dynamics with high-dimensional neural activity, synaptic connectivity, and cognitive functions.

ZUSAMMENFASSUNG

Wie entsteht Wahrnehmung aus der koordinierten Aktivität von Milliarden von Neuronen? Das Forschungsgebiet Computational Neuroscience versucht, solche Fragen mithilfe mathematischer Modellierung und Simulationen zu beantworten. Während frühe computergestützte Ansätze oft auf interpretierbaren, manuell entwickelten Modellen basierten, die auf Aufnahmen von wenigen Neuronen beruhten, ermöglichen moderne experimentelle Techniken heute das Aufnehmen der Aktivität von Hunderten bis Tausenden von Neuronen. Diese riesigen Datensätze offenbaren komplexe, heterogene und zeitlich variierende neuronale Dynamiken. Daraus ergeben sich neue Herausforderungen für Modelle, die einerseits ausdrucksstark genug sein müssen, um hochdimensionale neuronale Aktivität zu erfassen, und andererseits interpretierbar bleiben sollen. Wir adressieren diese Herausforderungen mithilfe von Recurrent Neural Networks (RNNs).

RNNs sind eine ausdrucksstarke, biologisch inspirierte Modellklasse, die sich als leistungsfähiges Hilfsmittel zur Generierung von Hypothesen über neuronale Informationsverarbeitung etabliert hat. Dennoch bleibt es herausfordernd, diese Modelle so an neuronale Daten anzupassen, dass sie sowohl die Variabilität korrekt erfassen als auch auf aktuelle Datensätze skalieren. In dieser Arbeit stellen wir eine Methode vor, um stochastische RNNs an neuronalen Daten von 'multi-session'-Aufnahmen mit einer Auflösung von einzelnen Versuchen anzupassen. Um die Interpretierbarkeit großer Netzwerke zu verbessern, leiten wir zudem eine obere Schranke für die Anzahl von Fixpunkten in 'low-rank, piecewise-linear' RNNs her.

Wir wenden RNNs an, um zwei neurowissenschaftliche Fragestellungen im Bereich Working Memory (Arbeitsgedächtnis) zu untersuchen. Erstens zeigen wir anhand von RNNs, die trainiert wurden um Aufgaben zu lösen, wie niedrigdimensionale Attraktordynamiken eine phasenbasierte Kodierung im Working Memory unterstützen. Dabei identifizieren wir eine Konnektivitätsstruktur, die die Phase intrinsischer Oszillationen multistabil mit einem externen oszillatorischen Input koppelt. Zweitens untersuchen wir anhand großskaliger Aufnahmen aus Makaken, die 'Sequence Working Memory' Aufgaben ausführen, ob eher Attraktor- oder transiente Dynamiken vorliegen. Mithilfe unserer Methode zum Fitten stochastischer RNNs erhalten wir ein differenziertes Bild, in dem stabile Attraktoren die Gedächtniserhaltung übernehmen, während gleichzeitig erhebliche zeitliche Variabilität besteht.

Insgesamt trägt diese Arbeit zu einem Rahmenwerk bei, das Maschinelles Lernen nutzt, um niedrigdimensionale RNN-Dynamiken mit hochdimensionaler neuronaler Aktivität, synaptischer Konnektivität und kognitiven Funktionen zu verknüpfen.

ACKNOWLEDGEMENTS

I first and foremost want to thank my principal investigator, Jakob Macke. From my initial internship, to a master thesis to the PhD, it has been an amazing journey. Thanks for the scientific freedom that I have had over the last years and enabling me to follow my curiosity. Thanks for facilitating the right environment and giving me the guidance that meant this freedom resulted into productive research. I am also very grateful for all the conferences I have been able to attend, and the research visits I have made, and the many fun interactions with the amazing computational neuroscience community that resulted.

I want to thank Omri Barak, whose lab I had the pleasure of visiting for two months during my PhD, resulting in the third chapter of thesis. Omri's innate curiosity and endless source of ideas for tackling scientific questions are to me an inspiration for how science should be conducted.

I also want to thank everyone in the Mackelab for such an amazing environment, the shared lunch breaks are one of the things that made me look forward to cycling up the hill every morning. In particular thanks to Guy Moss, Julius Vetter, Jai Kapoor, Sebastian Bisschof, Zina Stefanidi, Auguste Schulz, Richard Gao, Cornelius Schroeder and Michael Deistler, and especially my co-authors, A Erdem Sağtekin, Felix Pei and Manuel Gloeckler, without whom the fourth chapter would not have come out anywhere near as nicely as it did. It was insightful good fun working with you guys! I also want to thank the students that worked with me during their internships, and our amazing admin Franzi Weiler.

I would not have started a PhD if it was not for the people that I was able to do research with during my Bachelor and Master degree. In particular, I want give a big thanks to Jelmer Borst for the support and long weekly meetings during my Bachelor thesis, which led to my first scientific paper, and by no doubt catalyzed the scientific path that I set out on.

I want to thank my parents John and Liesbeth, my siblings Remco and Maaike and my friends (you now who you are) for supporting me, and giving me the best distraction during the holidays. Finally, I want to thank my fiancée, Imy, without your incredible understanding when I had to work at inconvenient times, your ability to put challenging and stressful situations into perspective, and giving me a feeling of having a home and family in Tübingen, I could not have done this.

CONTENTS

1 Introduction **1**
- 1.1 Making sense of large scale neuroscience data 1
- 1.2 Towards expressive and interpretable models 2
- 1.3 Recurrent neural networks 2
- 1.4 Outline of this thesis 4
  - 1.4.1 Publications included in this thesis 4
  - 1.4.2 Publications not included in this thesis 4

2 Background **5**
- 2.1 Recurrent neural networks 5
  - 2.1.1 Definition and motivation 5
  - 2.1.2 Low-rank RNNs 7
  - 2.1.3 Dynamical systems analysis 9
  - 2.1.4 Dynamics of linear recurrent neural networks 9
  - 2.1.5 Interpreting RNNs 1. Piecewise linear 12
  - 2.1.6 Interpreting RNNs 2. Reduced systems and Mean-field 13
- 2.2 Fitting stochastic RNNs to experimental data 16
  - 2.2.1 RNNs as state space models 16
  - 2.2.2 Inference 17
  - 2.2.3 Importance sampling 17
  - 2.2.4 Sequential Importance Sampling and particle filtering 18
  - 2.2.5 Parameter learning 19
  - 2.2.6 Connection to standard ELBO 20
  - 2.2.7 Relations to teacher forcing 22

3 Trained recurrent neural networks develop phase-locked limit cycles in a working memory task **25**
- 3.1 Declaration 26
  - 3.1.1 Author contributions 26
- 3.2 Introduction 27
- 3.3 Results 28
  - 3.3.1 Tractable, oscillating recurrent neural networks perform a working memory task 28
  - 3.3.2 Phase-coded memories correspond to limit cycle attractors 30
  - 3.3.3 Low-rank RNNs as phase-coupled oscillators 33
- 3.4 Discussion 36
- 3.5 Models and methods 38
  - 3.5.1 Code availability 38

3.5.2 Training RNNs on a phase-coding task 38
3.5.3 Analysing dynamics of oscillating low-rank RNNs 40
3.5.4 Coupled oscillators and population structure 43
3.6 Acknowledgments 46

4 Inferring stochastic low-rank recurrent neural networks from neural data 47
4.1 Declaration 48
4.2 Introduction 49
4.3 Theory and methods 50
4.3.1 Low-rank RNNs 50
4.3.2 Model learning with variational sequential Monte Carlo 51
4.3.3 Finding fixed points in piecewise-linear low-rank RNNs 54
4.4 Empirical Results 55
4.4.1 RNNs recover ground truth dynamics in student-teacher setups 55
4.4.2 Stochasticity allows recovering low-dimensional latents underlying EEG data 55
4.4.3 Interpretable latent dynamics underlying spikes recorded from rat hippocampus 57
4.4.4 Extracting stimulus-conditioned dynamics in monkey reaching task 59
4.4.5 Searching for fixed points 61
4.5 Discussion 63

5 Attractor dynamics underlie sequence working memory in data-constrained recurrent neural networks 65
5.1 Declaration 66
5.2 Introduction 67
5.3 Results 68
5.3.1 Dynamical mechanisms underlying stable working memory representations 68
5.3.2 Data simulated with multi-session RNNs match key properties of recordings 69
5.3.3 Analysis of RNN indicates attractor-guided dynamics 71
5.4 Discussion 75
5.5 Acknowledgments 76
5.6 Methods 76
5.6.1 Details of the data 76
5.6.2 Recurrent neural networks 77
5.6.3 Inference and training 79
5.6.4 Stimuli and time basis directions 80
5.6.5 Conditional fixed points 82
5.6.6 Perturbation analysis 83

5.6.7 Behavioral decoding 84
5.6.8 Hand-constructed models 84
5.6.9 Dataset for student-teacher setups 86
5.6.10 Session-consistency loss for student-teacher setups 86

6 Discussion and Future Directions 87
6.1 Summary 87
6.2 Future Directions 87

References 91

Supplement 109
.1 Supplementary information Background 111
.1.1 Derivation of low-rank RNN mean-field equations 111
.2 Supplementary information Chapter 3 114
.2.1 Loss curves 114
.2.2 Dynamics of unconstrained networks are similar to low-rank RNNs 115
.2.3 Influence of the training setup on the network solution 116
.2.4 Rate-coding solution in detail 118
.2.5 Converged RNN dynamics match those of coupled oscillators 120
.2.6 Connectivity of trained models 121
.2.7 Connectivity of mean-field model 122
.2.8 Geometry is preserved in a model coding for 4 stimuli, but more coupling populations are required. 123
.2.9 Phase precession through translation of the coupling function 125
.3 Supplementary information Chapter 4 127
.3.1 Proof of preposition 1 127
.3.2 Additional figures & tables 137
.3.3 Additional details of low-rank RNNs 145
.3.4 Details of empirical experiments 147
.4 Supplementary information Chapter 5 155
.4.1 Ring attractor model 155
.4.2 Transient dynamics model 156
.4.3 Student-teacher setup with attractor dynamics 156
.4.4 No spurious attractors in student-teacher setup with transient teacher 157

1

INTRODUCTION

1.1 MAKING SENSE OF LARGE SCALE NEUROSCIENCE DATA

Our cognition: Our memory, ideas, and plans all arise from the collective activity of 86 billion neurons, operating alongside a comparable number of glial (non-neural) cells [1–3]. Each cortical neuron has on average seven thousand synaptic connections with which it communicates [4]. Activity unfolds across a wide range of temporal and spatial scales, from millisecond spiking events to lifelong learning, and from local synaptic events to brain-wide networks. How can we understand a system as complex as the human brain?

Perhaps surprisingly given the massive scale of the system of interest, the field of neuroscience has been able to obtain remarkable insights by studying the activity of only a fraction of all neurons recorded during simple tasks. As an example, in the late 1960s Fuster & Alexander started recording from the prefrontal cortex and thalamus of macaques doing a working memory task [5]. Their seminal 1971 paper showed that a substantial number of recorded units (around 100 out of 170, recorded from 5 macaques) have increased firing rates when information is maintained in working memory. A finding that was independently reproduced in the same year by Kubota & Niki [6].

These studies suggested that working memory is coded by the increased firing of prefrontal neurons. States of elevated firing during memory maintenance were conceptualized as static attractor states in recurrent circuits [7–9], drawing on foundational works on attractor networks [10, 11]. However, as recording technologies have scaled, the underlying picture has proven to be more complex. Neural responses in prefrontal cortex tend to be more heterogeneous and time-varying than they initially appeared [12–16], leading to a shift towards population level views [17–19].

The scale of recordings is still rapidly increasing, producing datasets of unprecedented sizes and finer resolutions [20, 21]. While in the past, the field has been able to get insight from small datasets and hand-picked single neurons, with the data that is currently available, many processes in the brain might appear more complex, or nuanced, than initially thought. To fully understand the neural processes that give rise to cognition, models need to embrace this complexity. For such models, we can posit two central challenges.

1. Expressivity. Models should match the large amount of neurons that are recorded, and capture the heterogeneity and variability without relying on (trial) averages.

2. Interpretability. A framework is needed to understand these large-scale models; How does high-dimensional neural activity relate to cognitive processes, and how do these arise from their mechanistic underpinnings?

1.2 TOWARDS EXPRESSIVE AND INTERPRETABLE MODELS WITH MACHINE LEARNING AND DYNAMICAL SYSTEMS

Machine learning [22, 23] enables us to design computer programs that learn from data and experience. This field has made huge strides in recent years. Algorithms such as diffusion models [24] and transformers [25] are now capable of generating realistic images and videos, as well as allowing us to engage in dialogues with chatbots about a wide range of topics in technical depth.

The ability of modern machine learning algorithms to learn how to represent complex, high-dimensional data positions them to be well suited for addressing the challenge of *expressivity*. Recent examples include diffusion models for realistic neural data generation [26, 27], and transformers for learning representations from neural data [28, 29], allowing scaling to hundreds of datasets, spanning tens of thousands of units [30].

However, a central question remains: How can we use these models to facilitate understanding? One promising avenue is including some form of dimensionality reduction [31–33]. If high-dimensional data can be represented through low-dimensional latent trajectories, we, as humans, might be able to get more intuition into how the underlying model operates. Ideally, the mapping between the data and the latent space is such that the we can attribute the contribution of specific units to each latent (and vice versa). Yet, even if we can successfully reduce dimensionality of large datasets, it is not immediately obvious how we could understand trajectories in a space of e.g., ten dimensions.

A promising approach to the second challenge, *interpretability*, is provided by dynamical systems theory [34]. This rich theoretical framework offers a principled way to characterize complex, nonlinear dynamics in terms of attractors, invariant sets, and the geometry of trajectories, which enables reducing a high-dimensional system to interpretable constituents. The idea that neural activity can be understand through a (usually low-dimensional) dynamical system is called *computation through dynamics* [33, 35–38], or more generally *dynamical systems reconstruction* [39].

In line with this idea, a growing class of methods in neurosciences uses machine learning to infer latent dynamical systems directly from neural data [40–46]. However it is not always immediately obvious how these latent dynamical systems relate back to biological mechanisms, such as synaptic connectivity and population structure.

1.3 RECURRENT NEURAL NETWORKS

There exists a tension between expressivity and interpretability: Increasing model complexity, realism and capacity typically improves the ability to capture rich data structure, yet can also make the resulting model harder to understand. A potentially useful comprise is made by recurrent neural networks (RNNs) [47].

RNNs are a common model class in neuroscience that are inspired by biological neural networks, and are by construction dynamical systems. At the same time,

they are highly expressive: they can approximate finite-time trajectories of any dynamical system [48, 49]. RNNs can be trained to perform cognitive tasks using modern machine learning techniques [50–58], designed to implement putative mechanisms [59–61], as well as fit to neural data [62–69]. Once trained, RNNs can be analyzed (or reverse engineered) to form hypotheses about the recurrent circuits and dynamics that underlie computations [47, 51, 70–72].

Despite these successes, challenges remain. With regard to fitting to data, much work has focused on deterministic RNNs [62–64, 66–69], using modest amounts of data. To accurately model the variability and complexity in neural data we need principled methods for fitting RNNs with stochastic transitions. To get the most out of modern datasets we also need to scale the size of our models accordingly. In Chapter 4, we present a method that can accurately fit stochastic RNNs to neural recordings. In Chapter 5 we extend this method, allowing us to fit RNNs to over thousand units recorded over 15 sessions.

In terms of analysis and reverse engineering, the field has so far largely relied on approximate algorithms for finding fixed points [51, 70] and post-hoc dimensionality reduction methods [31]. It is unclear how well these approaches scale to systems that are intrinsically higher than three-dimensional (where visualization is less straightforward), or to systems that exhibit strongly time-varying trajectories, rather than being guided by fixed-point dynamics. As models become more realistic and operate at finer time scales [73], both model fitting and subsequent analysis can be expected to become more challenging.

While these challenges are not completely solved at the time of writing this thesis, we make the following contributions: For the case were RNNs are piece-wise linear and have low-rank connectivity, we show in Chapter 4 how to efficiently identify all fixed points with polynomial complexity in number of units. As previous methods had an exponential complexity [64, 65], our result facilitates exact analysis of the dynamics of large (low-rank) RNNs. Additionally, we outline methods for studying oscillatory (Chapter 3) and other non-static attractors (Chapter 5).

Nevertheless, the field has reached a stage where RNNs can perform cognitive tasks to a degree that enables the extraction of meaningful insights and the generation of experimentally testable predictions. In this thesis, RNNs are used throughout to investigate open questions in cognition. Motivated by the observation that the timing (or phase) of neural activity can encode information, Chapter 3 uses RNNs to study what types of neural dynamics and connectivity support phase-based coding of working memory. In Chapter 5, we apply our newly developed method to fit stochastic RNNs to neural data of macaques performing sequence working memory tasks [74]. Our data-constrained RNNs reveal a nuanced picture in which attractor dynamics coexist with temporal variability during working memory.

1.4 OUTLINE OF THIS THESIS

We start with a background on RNNs, including different approaches for analyzing and interpreting them. This is followed by a background section on fitting RNNs to neural data, including an overview of variational methods for probabilistic state space models and their connection to teacher forcing methods. The next two chapters are based on our published work. In Chapter 3, we use task-trained RNNs to investigate how oscillatory dynamics can support working memory through phase coding of information. In Chapter 4, we develop a method for fitting stochastic recurrent neural networks to neural data. In the final results chapter (Chapter 5), we extend the fitting method such that we can fit RNNs to multiple recording sessions, and aim to provide insight into a long-standing question regarding attractor dynamics versus transients in (sequence-based) working memory. Finally, we summarize our findings in Chapter 6 and discuss avenues for further research.

1.4.1 *Publications included in this thesis*

Chapter 3: **Matthijs Pals**, Jakob H Macke, Omri Barak. Trained recurrent neural networks develop phase-locked limit cycles in a working memory task. *PLOS Computational Biology* 20, 2 (2024).

Chapter 4: **Matthijs Pals**, A Erdem Sağtekin, Felix Pei, Manuel Gloeckler, Jakob H Macke. Inferring stochastic low-rank recurrent neural networks from neural data. *The Thirty-eighth Annual Conference on Neural Information Processing Systems* (2024).

1.4.2 *Publications not included in this thesis*

During my PhD I have also contributed to the following papers, which are not part of this thesis:

Michael Deistler, Kyra L Kadhim, **Matthijs Pals**, Jonas Beck, Ziwei Huang, Manuel Gloeckler, Janne K Lappalainen, Cornelius Schröder, Philipp Berens, Pedro J Gonçalves, Jakob H Macke. Differentiable simulation enables large-scale training of detailed biophysical models of neural dynamics. *Nature Methods* 22 (2025).

Sebastian Bischoff, Alana Darcher, Michael Deistler, Richard Gao, Franziska Gerken, Manuel Gloeckler, Lisa Haxel, Jaivardhan Kapoor, Janne K Lappalainen, Jakob H Macke, Guy Moss, **Matthijs Pals**, Felix C Pei, Rachel Rapp, A Erdem Sağtekin, Cornelius Schröder, Auguste Schulz, Zinovia Stefanidi, Shoji Toyota, Linda Ulmer, Julius Vetter. A Practical Guide to Sample-based Statistical Distances for Evaluating Generative Models in Science. *Transactions on Machine Learning Research* (2024).

2

BACKGROUND

The first section of this chapters introduces recurrent neural networks (RNNs), including various ways to train and analyse them. The second section focuses on fitting RNNs to neural data. It introduces variational methods for probabilistic state space models and their connection to teacher forcing methods.

2.1 RECURRENT NEURAL NETWORKS

2.1.1 *Definition and motivation*

Recurrent neural networks (RNNs) are a widely used model class in neuroscience [47, 50–58, 62, 64–69, 75, 76]. They strike a balance between expressivity (being universal approximators of dynamical systems [48, 49]) and (partial) biological plausibility, while also being trainable using modern machine learning techniques. Besides being trained to perform cognitive tasks or match neural recordings, RNNs can be hand-designed to implement putative mechanisms [59–61], analyzed using tools from statistical physics [77, 78], or reverse-engineered in terms of underlying attractors [47, 51, 70–72].

Recurrent neural networks with N neurons can be defined as the following continuous time system of differential equations:

$$\tau \frac{d\mathbf{x}(t)}{dt} = -\mathbf{x}(t) + \mathbf{J}\phi(\mathbf{x}) + \mathbf{H}\mathbf{u}(t) + \sigma\boldsymbol{\xi}(t), \tag{2.1}$$

with neural activity (currents) $\mathbf{x} \in \mathbb{R}^N$ at time t, time-constant $\tau \in \mathbb{R}^+$, recurrent weight matrix $\mathbf{J} \in \mathbb{R}^{N \times N}$, element-wise (current-to-rate) non-linearity ϕ, external input $\mathbf{u} \in \mathbb{R}^{N_{inp}}$ reaching the neurons with weights $\mathbf{H} \in \mathbf{R}^{N \times N_{inp}}$ and N-dimensional white noise process $\boldsymbol{\xi}(t)$, and standard-deviation $\sigma \in \mathbb{R}^+$. If desired the weights can be constrained to be excitatory-inhibitory (Dale's law) [54, 79].

2.1.1.1 *To spike or not to spike*

The rate-based model makes many simplifications compared to biological neural networks. Perhaps the most striking is the use of smooth, continuous firing rates, whereas real neurons communicate via discrete spikes [80]. Although spikes are physically continuous, they are typically modeled as delta-pulses. There are, however, settings where an equivalences between spiking and rate-based RNNs exists. Typical arguments rely on temporal averaging, or spatial averaging (over duplicates); As a simple heuristic motivation, if the input to a unit consists of N independent

Poisson samples, the mean grows linearly with N while the standard deviation scales as $\sqrt{N}$, causing relative fluctuations to vanish in the large-N limit [81]. Other equivalences can be formulated in terms of dynamical regimes [82] (altough there is some nuance [83]). Recent work showed equivalences between spiking and rate models in large heterogeneous networks without spatial duplicates [84], or by simple a scaling of the weights [85]. Smooth low-dimensional latent dynamics can also appear in high-dimensional spiking networks [60, 86, 87]. These latents can be defined similarly to those in the low-rank rate networks below, i.e., as weighted sums of neural activity, that again heuristically average out spiking variability.

As a point of nuance, we can of course not exclude that there are spike-timing based codes that are challenging to model with a purely rate based network. There recent work on directly training spike-based networks (e.g., [88–90]), might offer a solution.

2.1.1.2 *Training*

For training and simulation Eq. 2.1 is discretized, typically with Euler-Maruyama:

$$\mathbf{x}_{t+\Delta t} = (1 - \frac{\Delta t}{\tau})\mathbf{x}_t + \frac{\Delta t}{\tau}\left(\mathbf{J}\phi(\mathbf{x}_t) + \mathbf{H}\mathbf{u}_t\right) + \frac{\sqrt{\Delta t}}{\tau}\sigma\,\boldsymbol{\epsilon}_t, \tag{2.2}$$

with $\boldsymbol{\epsilon}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. When training RNN to perform tasks, it is typical to take a readout of the RNNs firing rates or currents:

$$\begin{aligned} \mathbf{y}_t &= \mathbf{W}\mathbf{x}_t, \\ \textit{or}\quad \mathbf{y}_t &= \mathbf{W}\phi(\mathbf{x}_t), \end{aligned}$$

and optimize parameters $\boldsymbol{\theta} = (\mathbf{J}, \mathbf{H})$ (and optionally τ [75, 91]) by minimizing a (e.g., mean-squared error or cross-entropy) loss to targets $\hat{\mathbf{y}}$ with gradient descent:

$$\mathcal{L} = \sum_{t=1}^{T} \ell(\hat{\mathbf{y}}_t, \mathbf{y}_t)$$

Backpropagation through time can be used to calculate the gradients with respect to $\boldsymbol{\theta}$ [92]. The contribution of the loss at each timestep t depends on all states up till that timestep, using the total derivative and the chain-rule we have:

$$\begin{aligned} \frac{\partial \mathcal{L}_t}{\partial \boldsymbol{\theta}} &= \sum_{r=1}^{t} \frac{\partial \mathcal{L}_t}{\partial \mathbf{y}_t}\frac{\partial \mathbf{y}_t}{\partial \mathbf{x}_t}\frac{\partial \mathbf{x}_t}{\partial \mathbf{x}_r}\frac{\partial \mathbf{x}_r}{\partial \boldsymbol{\theta}}^{+} \\ &= \sum_{r=1}^{t} \frac{\partial \mathcal{L}_t}{\partial \mathbf{y}_t}\frac{\partial \mathbf{y}_t}{\partial \mathbf{x}_t}\left(\prod_{k=0}^{t-r-1} \mathbf{J}_{\text{eff}}(\mathbf{x}_{t-1-k})\right)\frac{\partial \mathbf{x}_r}{\partial \boldsymbol{\theta}}^{+}. \end{aligned}$$

Here, $\frac{\partial \mathbf{x}_r}{\partial \boldsymbol{\theta}}^{+}$ denotes the explicit derivative of the state update equation at time r with respect to $\boldsymbol{\theta}$. The matrix $\mathbf{J}_{\text{eff}}(\mathbf{x}_{t-1-k})$ is the Jacobian evaluated along the trajectory:

$$\mathbf{J}_{\text{eff}}(\mathbf{x}_{t-1-k}) = \frac{\partial \mathbf{x}_{t-k}}{\partial \mathbf{x}_{t-1-k}}\bigg|_{\mathbf{x}_{t-1-k}} = (1 - \frac{\Delta t}{\tau})\mathbf{I} + \frac{\Delta t}{\tau}\mathbf{J}\ \text{diag}\left(\phi'(\mathbf{x}_{t-1-k})\right). \tag{2.3}$$

The repeated multiplication of Jacobians, can lead to *vanishing* and *exploding gradients* [93]. To illustrate this, if the network was linear ($\phi(\mathbf{x}) = \mathbf{x}$), the product would simply be $\prod_{k=r}^{t-1} \mathbf{J}_{\text{eff}}$, with $\mathbf{J}_{\text{eff}} = (1 - \frac{\Delta t}{\tau})\mathbf{I} + \frac{\Delta t}{\tau}\mathbf{J}$, i.e., the state of the RNN is determined by the repeated multiplication of $\mathbf{J}_{\text{eff}}$. The long-term behavior of this product is governed by the matrix's *spectral radius* (its largest absolute eigenvalue). If the spectral radius is greater than 1, the network's autonomous dynamics will eventually explode; if it is smaller than 1, the activity will decay to zero (see Section 2.1.4). During backpropagation the maximum amplification or decay of gradients are bounded by the matrix's *spectral norm* (its largest singular value); If the largest singular value of $\mathbf{J}_{\text{eff}}$ is greater than 1, the gradients can explode, whereas if it is smaller than 1, they vanish. The discrepancy between the spectral radius and the spectral norm arises because the effective Jacobian is typically a *non-normal* matrix, which allows for transient amplification (see Section 2.1.4.2).

Various strategies help to mitigate these issues, including gradient clipping and regularization [93, 94]. Weight initialization often also has a large impact on stable learning in recurrent networks. In RNNs with tanh activations and weights sampled i.i.d., there is a transition from decaying to chaotic dynamics (that is sharp as $N \to \infty$) when the effective gain (i.e., spectral radius in case of linearized dynamics) becomes 1 [77] (or $\sqrt{2}$ for ReLU [78]). Initializing just near this 'edge of chaos' [95] is often a good choice [96], as it provides long time scales and rich dynamics [97] without overly exploding gradients. The 'rich basis' can also be exploited in so-called echo-state networks [98, 99], where backpropagation through time can be avoided alltogether and only readout weights are trained.

When fitting RNNs to data, strategies such as teacher forcing can help [64, 100], which will be discussed in more detail in section 2.2.7.

2.1.2 *Low-rank RNNs*

2.1.2.1 *Linking high-D neural activity with low-D dynamics*

Understanding an N-dimensional system (typically hundreds to thousands) is hard, understanding a lower-dimensional system is typically easier. RNNs with low-rank structure in their connectivity [49, 56, 59, 60, 76, 101, 102] admit a direct mapping between high-dimensional neural activity and the underlying low-dimensional dynamics. Additionally the low-rank structure facilitates linking computation to population structure [56], analytic analysis [49, 59, 76] and inference (Chapter 4 and Valente, Ostojic & Pillow [103]).

For ease of exposition, we will drop the additive noise term (see Chapter 4 and Valente, Ostojic & Pillow [103]) for ways to include it). The key assumption is that

the recurrent weight matrix $\mathbf{J}$ can be decomposed as a sum of R outer products of left and right connectivity vectors $\mathbf{m}^{(r)}$ and $\mathbf{n}^{(r)}$'s:

$$\begin{aligned} \mathbf{J} &= \sum_{r}^{R} \mathbf{m}^{(r)} \mathbf{n}^{(r)\mathsf{T}} = \mathbf{M}\mathbf{N}^{\mathsf{T}}, \\ \tau \frac{d\mathbf{x}(t)}{dt} &= -\mathbf{x}(t) + \mathbf{M}\mathbf{N}^{\mathsf{T}} \phi(\mathbf{x}(t)) + \mathbf{H}\mathbf{u}(t). \end{aligned} \tag{2.4}$$

The advantage of this, is that (given $\mathbf{x}(0)$ in the span of $\mathbf{M}$ and $\mathbf{H}$, or alternatively after initial transients have decayed), the recurrent activity is confined to a lower-dimensional linear subspace: We can see that in Eq. 2.4 new activity comes in only along (in the span of) $\mathbf{M}$ and $\mathbf{H}$:

$$\mathbf{M} \underbrace{\mathbf{N}^{\mathsf{T}} \phi(\mathbf{x}(t))}_{R-\text{dim}}, \quad \mathbf{H} \underbrace{\mathbf{u}(t)}_{N_{inp}-\text{dim}}.$$

We can obtain a set of $R + N_{inp}$ equations that directly describe the dynamics in this lower-dimensional space. Using:

$$\begin{aligned} \mathbf{x}(t) &= \mathbf{M}\boldsymbol{\kappa}(t) + \mathbf{H}\mathbf{v}(t), \\ \boldsymbol{\kappa}(t), \mathbf{v}(t) &= [\mathbf{M}, \mathbf{H}]^{\dagger} \mathbf{x}(t), \end{aligned}$$

and using the property of the left pseudoinverse, $[\mathbf{M}, \mathbf{H}]^{\dagger}[\mathbf{M}, \mathbf{H}] = \mathbf{I}$, we can write:

$$\begin{aligned} \tau \frac{d\boldsymbol{\kappa}(t)}{dt} &= -\boldsymbol{\kappa}(t) + \mathbf{N}^{\mathsf{T}} \phi(\mathbf{M}\boldsymbol{\kappa}(t) + \mathbf{H}\mathbf{v}(t)), \\ \tau \frac{d\mathbf{v}(t)}{dt} &= -\mathbf{v}(t) + \mathbf{u}(t). \end{aligned} \tag{2.5}$$

Thus we obtained an equivalent low-dimensional system that still fully describes the dynamics. Note that Eq. 2.5 can be seen a a one one-hidden layer neural ordinary differential equation [104].

2.1.2.2 *Orthogonalized basis*

To get a cleaner separation between the latent variable and inputs, we can move to an orthogonalized basis.

In particular we will take $\mathbf{M}$ to be orthonormal ($\mathbf{M}^{\mathsf{T}}\mathbf{M} = \mathbf{I}$) and $\mathbf{N}$ to have orthogonal columns. Even when training RNNs without this strict constraint, we can always obtain such a decomposition after training via the singular value decomposition:

$$\mathbf{M}\mathbf{N}^{\mathsf{T}} = \mathbf{J} = \mathbf{U}\mathbf{S}\mathbf{V}^{\mathsf{T}},$$

and setting $\mathbf{M} \leftarrow \mathbf{U}$ and $\mathbf{N} \leftarrow \mathbf{V}\mathbf{S}$ (where we take only the first R singular vectors and values). This basis is unique up to sign flips and permutations of singular vectors with the same singular values.

Additionally, to cleanly separate the contributions of the input, we decompose the input weights into components parallel and orthogonal to the recurrent subspace:

$$\mathbf{H} = \mathbf{M}\mathbf{A}_{\parallel} + \mathbf{H}_{\perp}\mathbf{A}_{\perp}.$$

Here, $\mathbf{A}_{\parallel}$ and $\mathbf{A}_{\perp}$ encode the magnitudes (i.e., projections / lengths) of the input directions onto the corresponding subspaces.

We first define the parallel component, and project it out:

$$\mathbf{A}_{\parallel} = \mathbf{M}^{\mathsf{T}}\mathbf{H},$$
$$\mathbf{H}_{\text{res}} = \mathbf{H} - \mathbf{M}\mathbf{A}_{\parallel}.$$

Next, we construct an orthonormal basis for the orthogonal component via QR decomposition:

$$\mathbf{H}_{\text{res}} = \mathbf{Q}\mathbf{R},$$

and set $\mathbf{H}_{\perp} \leftarrow \mathbf{Q}$, $\mathbf{A}_{\perp} \leftarrow \mathbf{R}$. Now, we can project the dynamics into the orthogonalized basis (of orthonormal $\mathbf{M}$ and $\mathbf{H}_{\perp}$) using:

$$\mathbf{x}(t) = \mathbf{M}\boldsymbol{\kappa}(t) + \mathbf{H}_{\perp}\mathbf{v}(t),$$
$$\boldsymbol{\kappa}(t) = \mathbf{M}^{\mathsf{T}}\mathbf{x}(t), \quad \mathbf{v}(t) = \mathbf{H}_{\perp}^{\mathsf{T}}\mathbf{x}(t),$$

and obtain the reduced dynamics:

$$\begin{aligned} \tau\frac{d\boldsymbol{\kappa}(t)}{dt} &= -\boldsymbol{\kappa}(t) + \mathbf{N}^{\mathsf{T}}\phi\big(\mathbf{M}\boldsymbol{\kappa}(t) + \mathbf{H}_{\perp}\mathbf{v}(t)\big) + \mathbf{A}_{\parallel}\mathbf{u}(t), \\ \tau\frac{d\mathbf{v}(t)}{dt} &= -\mathbf{v}(t) + \mathbf{A}_{\perp}\mathbf{u}(t). \end{aligned} \tag{2.6}$$

All figures in the following chapters that show low-rank RNN dynamics make use of this orthogonalised basis. Note that, if we are just interested in visualisation we can simulate the original (non-orthogonal) coordinate system and then change the trajectories to an orthogonal basis with matrix $\boldsymbol{\kappa} \leftarrow \mathbf{U}^{\mathsf{T}}\mathbf{M}\boldsymbol{\kappa}$, where $\mathbf{U}$ are the first R left-singular vectors of $\mathbf{J}$.

2.1.3 *Dynamical systems analysis*

The field of dynamical systems studies systems that evolve over time, and tries to characterize their long-term behavior, will the system converge to a stable state, will it oscillate or display chaotic behavior [34]? This provides a framework for studying activity in recurrent networks.

2.1.4 *Dynamics of linear recurrent neural networks*

Consider a linear RNN without input or noise. Assuming a linear activation function (i.e., $\phi(\mathbf{x}) = \mathbf{x}$), the continuous-time dynamics are given by:

$$\tau \frac{d\mathbf{x}(t)}{dt} = -\mathbf{x}(t) + \mathbf{J}\mathbf{x}(t),$$
$$\frac{d\mathbf{x}(t)}{dt} = \mathbf{C}\mathbf{x}(t), \quad \text{where } \mathbf{C} = \frac{1}{\tau}(\mathbf{J} - \mathbf{I}).$$

This system has a fixed point at $\mathbf{0}$ (or a subspace in the case of degenerate eigenvalues). Fixed points are defined as states where the rate of change is zero, i.e., $\frac{d\mathbf{x}(t)}{dt} = \mathbf{0}$.

The behavior given an initial state $\mathbf{x}(0)$ can be expressed using the matrix exponential $\mathbf{x}(t) = e^{\mathbf{C}t}\mathbf{x}(0)$. However, more insightful might be the solution obtained by diagonalizing the system [34]. Let $\mathbf{v}_i$ be the eigenvectors and λ_i the corresponding eigenvalues of $\mathbf{C}$:

$$\mathbf{x}(t) = \sum_{i=1}^{N} c_i e^{\lambda_i t}\mathbf{v}_i,$$

where we assume the eigenvectors are linearly independent, and the c_i's are set by the initial condition. This expression shows that the dynamics are a sum of activity patterns, each evolving along its respective eigenvector. Considering a single eigenmode, the term $c_i e^{\lambda_i t}\mathbf{v}_i$ captures the system's trajectory along the direction $\mathbf{v}_i$.

If $\lambda_i \in \mathbb{R}$, the amplitude of this component grows or decays exponentially depending on the sign of λ_i: if $\lambda_i < 0$, this direction is *stable*, whereas if $\lambda_i > 0$, it is *unstable*. If a fixed point has both stable and unstable directions, it is termed a *saddle point*.

If instead if eigenvalues are complex: $\lambda_i = \mathrm{Re}(\lambda_i) + i\mathrm{Im}(\lambda_i)$, the dynamics exhibit oscillations. Using Euler's formula: $e^{\lambda_i t} = e^{\mathrm{Re}(\lambda_i)t}\left(\cos(\mathrm{Im}(\lambda_i)t) + i\sin(\mathrm{Im}(\lambda_i)t)\right)$, we see that $\mathrm{Re}(\lambda_i)$ controls the exponential growth or decay of the amplitude, while $\mathrm{Im}(\lambda_i)$ determines the oscillation frequency (with period is $\frac{2\pi}{\mathrm{Im}(\lambda_i)}$). Since $\mathbf{C}$ is real-valued, complex eigenvalues occur in conjugate pairs, which implies that the resulting trajectories are also real-valued and correspond to rotations (oscillations) in a two-dimensional subspace.

2.1.4.1 *Discretized linear dynamics.*

As stated before, in practice we often consider a discrete-time version of the system. Using a time step Δt, a forward Euler discretization of the linear dynamics results in:

$$\mathbf{x}_{t+\Delta t} = \mathbf{x}_t + \Delta t\, \mathbf{C}\mathbf{x}_t = \tilde{\mathbf{C}}\mathbf{x}_t,$$

where $\tilde{\mathbf{C}} = \mathbf{I} + \Delta t\, \mathbf{C} = (1 - \frac{\Delta t}{\tau})\mathbf{I} + \frac{\Delta t}{\tau}\mathbf{J}$. The solution at time t (t steps of duration Δt) can again be written in terms of eigenvalues and eigenvectors [34] (now of $\tilde{\mathbf{C}}$):

$$\mathbf{x}_t = \tilde{\mathbf{C}}^t \mathbf{x}_0.$$
$$\mathbf{x}_t = \sum_{i=1}^{N} c_i \lambda_i^t \mathbf{v}_i.$$

Now, stability is determined by the *magnitude* of the eigenvalues: the fixed point at $\mathbf{0}$ is stable if $|\lambda_i| < 1$ for all i, unstable if $|\lambda_i| > 1$ for all i, and a saddle if both cases are present. Complex eigenvalues $\lambda_i = re^{i\theta}$ give rise to oscillatory dynamics with frequency $\theta/(2\pi)$ and growth or decay as r^t.

2.1.4.2 *Non-normal Dynamics*

The eigenvectors of the effective connectivity matrix $\mathbf{C}$ (or $\tilde{\mathbf{C}}$) might not be orthogonal to each other. This kind of matrix is called *non-normal* (defined as not commuting with its transpose: $\mathbf{C}\mathbf{C}^\top \neq \mathbf{C}^\top\mathbf{C}$).

This can give rise to counter-intuitive transient dynamics. Even if all eigenvalues have negative real parts and the system is thus stable, the non-orthogonality means that activities along eigenvectors can feed into each other, which can lead to transient increases in activity (before the eventual decay). This phenomenon is relevant in neuroscience, and underlies theories of coding in working-memory (e.g., [105], see also Chapter 5). To get insight into non-normal structure one can apply a Schur decomposition to $\mathbf{C}$ [106].

2.1.4.3 *Non-linear Dynamics and Linearization*

For a non-linear RNN defined by:

$$\frac{d\mathbf{x}(t)}{dt} = \mathbf{F}(\mathbf{x}(t)) = \mathbf{C}\phi(\mathbf{x}(t)),$$

with $\mathbf{C}$ as before, we can not directly use the same strategy as before to get insight into global stability of the system. What we can analyze, in a similar fashion to the linear system, is the system's behavior near a fixed point $\mathbf{x}^*$. Define the perturbation $\boldsymbol{\eta}(t) = \mathbf{x}(t) - \mathbf{x}^*$ [34]. The perturbation evolves as $\frac{d\boldsymbol{\eta}(t)}{dt} = \frac{d}{dt}(\mathbf{x}(t) - \mathbf{x}^*) = \frac{d\mathbf{x}(t)}{dt} = \mathbf{F}(\mathbf{x}(t)) = \mathbf{F}(\mathbf{x}^* + \boldsymbol{\eta}(t))$. A Taylor expansion at $\mathbf{x}^*$ gives:

$$\frac{d\boldsymbol{\eta}(t)}{dt} = \mathbf{F}(\mathbf{x}^* + \boldsymbol{\eta}(t)) = \underbrace{\mathbf{F}(\mathbf{x}^*)}_{=0} + \frac{\partial \mathbf{F}}{\partial \mathbf{x}}\Big|_{\mathbf{x}^*} \boldsymbol{\eta}(t) + h.o.t.,$$

For sufficiently small $\boldsymbol{\eta}$, the higher-order terms ($h.o.t.$) become negligible, allowing us to approximate the local dynamics as a linear system ($\frac{d\boldsymbol{\eta}(t)}{dt} = \frac{\partial \mathbf{F}}{\partial \mathbf{x}}|_{\mathbf{x}^*}\boldsymbol{\eta}(t)$), with stability given by the eigenvalues of the Jacobian $\frac{\partial \mathbf{F}}{\partial \mathbf{x}}|_{\mathbf{x}^*}$.

To find the fixed points of non-linear systems we might have to resolve to numerical optimization schemes. For high-dimensional systems such as RNNs, specific methods have been proposed [51, 70, 107], e.g., by numerically finding minima of $|\mathbf{F}(\mathbf{x})|^2$.

In non-linear systems visualizations also become more important [34]. If the system is two-dimensional we can plot a *phase plane*, which shows the Jacobian evaluated at various points. For higher-dimensional systems, plotting the fixed points and example trajectories in the space spanned by the principal components of the trajectories can be illuminating [51].

Next to having multiple fixed points, non-linear dynamics also exhibit other phenomena absent in linear systems. One example is attractive (or repelling) periodic trajectories, called *limit cycles*, which we study in the context of RNNs in Chapter 3 (see also Sato & Gohara [108]). Another example is *chaos*, which can be defined as a deterministic system that displays sensitive dependence on initial conditions (i.e., exponential divergence), while being globally stable [34], a regime in which the brain has been hypothesized to be in [109].

2.1.5 *Interpreting RNNs 1. Piecewise linear*

While we generally need to resort to numerical optimization to find fixed points in non-linear systems, RNNs with piecewise-linear activations have fixed points and cycles that can, in principle, be accessed analytically [64, 65, 70, 110–113]. Whether this is practically feasible depends on the network size N and, as we demonstrate in Chapter 4, the rank of the connectivity matrix.

The key insight is that the state space of a piece-wise linear RNN can be partitioned into distinct regions in which the dynamics are linear. To illustrate this, consider the piecewise-linear ReLU activation function with thresholds $\mathbf{h}_i$: $\phi(\mathbf{x}_i) = \max(\mathbf{x}_i - \mathbf{h}_i, 0)$. We can rewrite this activation as a (state-dependent) linear operation by introducing a diagonal indicator matrix [65]:

$$\mathbf{D}_\Omega = \begin{bmatrix} d_1 & & & \\ & d_2 & & \\ & & \ddots & \\ & & & d_N \end{bmatrix},$$

where the diagonal entries are defined as:

$$d_i = \begin{cases} 1, & \text{if } \mathbf{x}_i > \mathbf{h}_i \\ 0, & \text{otherwise} \end{cases}.$$

For each region Ω determined by $\mathbf{D}_\Omega$, the RNN dynamics become linear. For a given state vector $\mathbf{x}(t)$ and corresponding $\mathbf{D}_\Omega$, the RNN equation reads:

$$\tau \frac{d\mathbf{x}}{dt} = -\mathbf{x}(t) + \mathbf{J}\mathbf{D}_\Omega \mathbf{x}(t) - \mathbf{J}\mathbf{D}_\Omega \mathbf{h} + \mathbf{H}\mathbf{u}(t). \tag{2.7}$$

Because each of the N neurons can be either active ($\mathbf{x}_i > \mathbf{h}_i$) or inactive ($\mathbf{x}_i \leq \mathbf{h}_i$), there are up to 2^N possible configurations for $\mathbf{D}_\Omega$, each corresponding to a distinct region in the phase space ruled by linear dynamics.

Consequently, for each of these 2^N regions, we can analytically solve for its fixed point by setting $\frac{d\mathbf{x}}{dt} = 0$ conditioned on a specific configuration of $\mathbf{D}_\Omega$ [65]. Solving the linear system results in a candidate fixed point $\mathbf{x}^*$. After finding it, a self-consistency check has to be performed: does the obtained $\mathbf{x}^*$ actually lie in the

assumed region Ω? If it does, a true fixed point of the full, non-linear RNN has been identified.

Similarly, in piecewise-linear framework we can find cycles, which is most easily shown when switching to discrete maps. Let $\mathbf{x}_{t+1} = \mathbf{F}_\Omega[\mathbf{x}_t]$ be the linear map conditioned on a region Ω, corresponding to discretizing Eq. 2.7. To find cycles with period 2 (2-cycles), construct the map $\mathbf{x}_{t+2} = \mathbf{F}_{\Omega'}[\mathbf{F}_\Omega[\mathbf{x}_t]]$, and solve for the fixed point of all possible 2^{2N} sets of linear equations (all configurations of pairs of regions (Ω, Ω'). Similarly, for k-order cycles we can search through 2^{kN} combinations. While this relies on discretization, recent work has investigated finding cycles in continuous systems, where one can analytically integrate the continuous system until it cross a boundary between regions [113].

While attractive on paper, searching through 2^N regions is unfeasible for RNNs with hundreds of units given typical compute. Recent work has shown that a heuristic algorithm can navigate this combinatorial space efficiently [70]. Furthermore, if the network is both piecewise-linear and low-rank (with rank R), we show in Chapter 4 that a vast majority of the 2^N regions can be ignored when searching for fixed points. In low-rank RNNs the (long-term) dynamics live in a R-dimensional subspace of the full N-dimensional neural space. This R-dimensional subspace only intersects a small subset of the 2^N total regions, namely:

$$\sum_{r=0}^{R} \binom{N}{r},$$

of them, which as a consequence, also bounds the maximum number of fixed points. In Chapter 4, we prove this result, and also generalize it to piece-wise linear activation functions constructed using a linear combination of D ReLUs: $\phi(\mathbf{x}_i) = \sum_d^D \mathbf{b}_i^{(d)} \max(\mathbf{x} - \mathbf{h_i}^{(d)})$, as for instance used in Dendritic RNNs [111], or the clipped ReLUs used in Hess *et al.* [64] and Chapter 4.

2.1.6 *Interpreting RNNs 2. Reduced systems and mean-field equations*

So far, we have reduced the N-dimensional RNN dynamics to an effective R-dimensional system, and looked at analysis of fixed points. However, despite evolving in a low-dimensional subspace, low-rank RNNs still do contain $\mathcal{O}(NR)$ parameters (still a significant reduction from $\mathcal{O}(N^2)$ in standard RNNs). To gain more insight into how connectivity and population structure shape the dynamics, we can obtain a statistical description in which the connectivity vectors are drawn from a probability distribution. This allows us to go from $\mathcal{O}(NR)$ parameters to a description in terms of $\mathcal{O}(R^2)$ first- and second-order statistics (means and covariances) of the probability distribution. This builds on ideas from dynamical mean-field theory in statistical physics, introduced in the context of random recurrent neural networks by Sompolinsky, Crisanti & Sommers [77]. In the case of low-rank RNNs, this framework leads to a particularly succinct description of the dynamics [49, 56, 59, 102].

Alternative approaches to further reduce the system include descriptions in terms of graphs [52, 57, 114], that highlight how the RNN switches between fixed points and other attractors.

The background presented here largely follows work by Beiran *et al.* [49], Dubreuil *et al.* [56], and Schuessler *et al.* [102], which build on Mastrogiuseppe & Ostojic [59], yet we use an alternative approximation of the resulting integrals, which results in a potentially more transparent expression for the dynamics. We apply this framework in Chapter 3 to study low-rank RNNs driven by oscillatory inputs.

Here, we will consider a rank-1 RNN with 1 input vector (and we redefine $\mathbf{n} \leftarrow N\mathbf{n}$). For simplicity we will assume $\mathbf{h}$ orthogonal to $\mathbf{m}$. The approach here can be straightforwardly extended to higher ranks. The rank-1 RNN is defined as:

$$\tau \frac{d\kappa(t)}{dt} = -\kappa(t) + \frac{1}{N}\sum_{i}^{N} \mathbf{n}_i^{\mathsf{T}} \phi\left(\kappa(t)\mathbf{m}_i + v(t)\mathbf{h}\right). \tag{2.8}$$

Assuming the triplets $(\mathbf{n}_i, \mathbf{m}_i, \mathbf{h}_i)$ are all independently sampled per neuron from a joint probability distribution $p(\mathbf{y})$; $\mathbf{y} = (n, m, h)$, Eq. 2.8 is a sampling estimator for the recurrent input. This approaches per the law of large numbers, as $N \to \infty$:

$$\tau \frac{d\kappa(t)}{dt} \overset{N\to\infty}{=} -\kappa(t) + \mathbb{E}_{p(\mathbf{y})}\left[n\phi\left(\kappa(t)m + v(t)h\right)\right]. \tag{2.9}$$

We assume $p(\mathbf{y})$ is a mixture of L zero-mean Gaussians [49, 56]. This gives each component an intuitive interpretation as a separate subpopulation in the network.

$$p(\mathbf{y}) = \sum_{l}^{L} w_l \mathcal{N}(0, \Sigma_l) = \sum_{l}^{L} w_l p(\mathbf{y}^{(l)}), \quad \mathbf{y}^{(l)} = (n^{(l)}, m^{(l)}, h^{(l)}).$$

As derived in Beiran *et al.* [49] and Dubreuil *et al.* [56] and in all detail here in Supplementary .1.1, this implies that Eq. 2.9 can be rewritten as:

$$\tau \frac{d\kappa(t)}{dt} = -\kappa(t) + \sum_{l}^{L} w_l \left(\kappa(t)\sigma_{mn}^{(l)} + v(t)\sigma_{hn}^{(l)}\right) \mathbb{E}_{p(z)}\left[\phi'\left(\sqrt{\Delta^{(l)}(t)}z\right)\right], \tag{2.10}$$

with $\Delta^{(l)}(t) = (\sigma_m^{(l)}\kappa(t))^2 + (\sigma_h^{(l)}v(t))^2$ and $z \sim \mathcal{N}(0,1)$.

This equation can be interpreted as an effective linear recurrent system, in which the coupling is determined by the covariances between the recurrent and input weight vectors (i.e., $\sigma_{mn}^{(l)}$ and $\sigma_{hn}^{(l)}$). These couplings are modulated by a state-dependent nonlinear gain term given by the expectation $\mathbb{E}_{p(z)}\left[\phi'\left(\sqrt{\Delta^{(l)}(t)}z\right)\right]$. For typical sigmoidal activation functions, this results in a stabilizing effect: as the variance of the input, $\Delta^{(l)}(t)$, increases, the nonlinearity ϕ enters a saturated regime in which ϕ' tends toward zero, thereby weakening the effective recurrent interactions again.

2.1.6.1 *A solvable example*

To further understand how the covariances shape the dynamics, we consider the case of a single population ($L = 1$) and no external input ($v(t) = 0$). The mean-field dynamics reduce to:

$$\tau \frac{d\kappa(t)}{dt} = -\kappa(t) + \sigma_{mn}\,\kappa(t)\,\mathbb{E}_{p(z)}\left[\phi'\left(\sqrt{\Delta(t)}\,z\right)\right], \tag{2.11}$$

where $\Delta(t) = \sigma_m^2 \kappa(t)^2$, $z \sim \mathcal{N}(0,1)$.

Previous work numerically approximated this expected [49, 56], however we can obtain a closed-form expression by choosing a convenient activation function ϕ. As we show in Chapter 3, using sigmoidal activation

$$\phi(x) = \operatorname{erf}\left(\frac{\sqrt{\pi}}{2}x\right),$$

which closely approximates the commonly used $\tanh(x)$, allows us to compute the gain term analytically:

$$\mathbb{E}_{p(z)}\left[\phi'\left(\sqrt{\Delta}\,z\right)\right] = \frac{1}{\sqrt{1+\frac{\pi}{2}\Delta}}.$$

This has recently been generalized in Schmutz *et al.* [76], where closed-form expressions are also derived for other nonlinearities such as ReLU.

Substituting the analytical expression for the gain, we obtain:

$$\tau \frac{d\kappa(t)}{dt} = -\kappa(t) + \frac{\sigma_{mn}\,\kappa(t)}{\sqrt{1+\frac{\pi}{2}\sigma_m^2\kappa(t)^2}}. \tag{2.12}$$

To understand this system, we next analyze its fixed points. Setting $\frac{d\kappa}{dt} = 0$, we obtain:

$$\kappa = \frac{\sigma_{mn}\,\kappa}{\sqrt{1+\frac{\pi}{2}\sigma_m^2\kappa^2}}. \tag{2.13}$$

One solution is $\kappa = 0$. To assess its stability, we linearize Eq. (2.12) around $\kappa = 0$. The Jacobian (derivative) evaluated at $\kappa = 0$ is:

$$\frac{d}{d\kappa}_{|k=0}\frac{d\kappa}{dt} = -1 + \sigma_{mn}.$$

Therefore, the fixed point $\kappa = 0$ is stable if $\sigma_{mn} < 1$, and becomes unstable when $\sigma_{mn} > 1$.

We can solve for additional fixed points by dividing both sides of the fixed-point equation (Eq. 2.13) by κ, and then solving for κ, to obtain:

$$\kappa^2 = \frac{2}{\pi}\frac{\sigma_{mn}^2 - 1}{\sigma_m^2}.$$

Additional fixed points therefore exist only when $\sigma_{mn} > 1$, in which case two symmetric solutions emerge:

$$\kappa = \pm\sqrt{\frac{2}{\pi}\frac{\sigma_{mn}^2 - 1}{\sigma_m^2}}.$$

We can thus describe the system as follows: For $\sigma_{mn} < 1$, the only stable fixed point is $\kappa = 0$. As σ_{mn} increases, the system undergoes a pitchfork bifurcation at $\sigma_{mn} = 1$. The fixed point at 0 becomes unstable and two stable non-zero fixed points appear, where the recurrent drive along σ_{mn} balances with the stabilizing effect of the sigmoidal non-linearity.

This also shows that there is are at most 2 (symmetric) stable fixed-points in a one-Gaussian-population rank-1 RNN with the chosen activation function. To generate additional fixed-points, one can add additional subpopulations that saturate / stabilize the dynamics at different regions in phase-space as is explained in Beiran *et al.* [49].

2.2 FITTING STOCHASTIC RNNS TO EXPERIMENTAL DATA

In this section we will give background for the methods used in this thesis to fit RNNs to experimental data. Neural responses can be highly variable. Recordings are (generally) noisy recordings, and contain of only a subset of the neurons responsible for the cognitive process we are interested in. To model this variability in neural data, in this section we consider recurrent neural networks (RNNs) formulated as stochastic state space models. Our goal in this section is twofold:
Inference: Estimate latent trajectories, i.e., RNN states $\mathbf{z}_{1:T}$, given observed data $\mathbf{y}_{1:T}$ at times $t = 1, \ldots, T$.
Learning: Optimize parameters $\boldsymbol{\theta}$ of the RNNs that maximize the marginal likelihood of the data:

$$\arg\max_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(\mathbf{y}_{1:T}) = \arg\max_{\boldsymbol{\theta}} \int p_{\boldsymbol{\theta}}(\mathbf{y}_{1:T} \mid \mathbf{z}_{1:T}) p_{\boldsymbol{\theta}}(\mathbf{z}_{1:T})\, d\mathbf{z}_{1:T}. \tag{2.14}$$

While directly computing the likelihood is generally intractable, we can work towards these two goals parsimoniously using *variational inference*, where we jointly maximize a lower bound to the marginal likelihood $p(\mathbf{y}_{1:T})$ and minimize a divergence between $q(\mathbf{z}_{1:T} \mid \mathbf{y}_{1:T})$ and the true, usually intractable, posterior $p(\mathbf{z}_{1:T} \mid \mathbf{y}_{1:T})$. In particular we will use a method called variational particle filtering or variational sequential Monte Carlo [115–117], as detailed below, which is specifically suitable for time-series data.

2.2.1 *RNNs as state space models*

The joint probability of a state space model can be decomposed as [118]:

$$p(\mathbf{z}_{1:T}, \mathbf{y}_{1:T}) = p(\mathbf{z}_1) \prod_{t=2}^{T} p(\mathbf{z}_t \mid \mathbf{z}_{t-1}) \prod_{t=1}^{T} p(\mathbf{y}_t \mid \mathbf{z}_t). \tag{2.15}$$

We will assume the latent dynamics are Gaussian:

$$p(\mathbf{z}_t \mid \mathbf{z}_{t-1}) = \mathcal{N}(F(\mathbf{z}_{t-1}), \Sigma_{\mathbf{z}}),\ p(\mathbf{z}_1) = \mathcal{N}(\mu_{\mathbf{z}_1}, \Sigma_{\mathbf{z}_1}),$$

where $F(\mathbf{z}_{t-1})$ is the non-linear transition function parameterized by an RNN. The observation model $p(\mathbf{y}_t \mid \mathbf{z}_t)$ is adapted based on the data-modality; in Chapter 4 we use a linear Gaussian model for continuous observations such as EEG data, and in Chapters 4 and 5 we use a non-linear Poisson observation model for spiking data.

2.2.2 *Inference*

Irrespective of the formulation (non-linear or linear), in a state space model, the filtering posterior $p(\mathbf{z}_t \mid \mathbf{y}_{1:t})$ satisfies the following recursion [118]:
Predict step: The dynamics are applied to the posterior and $\mathbf{z}_{t-1}$ is marginalized out:

$$p(\mathbf{z}_t \mid \mathbf{y}_{1:t-1}) = \mathbb{E}_{p(\mathbf{z}_{t-1} \mid \mathbf{y}_{1:t-1})}[p(\mathbf{z}_t \mid \mathbf{z}_{t-1})] \tag{2.16}$$

Update step: The observations are used to update:

$$p(\mathbf{z}_t \mid \mathbf{y}_{1:t}) \propto p(\mathbf{y}_t \mid \mathbf{z}_t) p(\mathbf{z}_t \mid \mathbf{y}_{1:t-1}) \tag{2.17}$$

For a linear Gaussian model these are closed-form (via the Kalman Filter [119]). For an RNN with non-linear state transitions the predict step becomes intractable. If observations are also not linear Gaussian (e.g., Poisson), the update step also needs to be approximated.

2.2.3 *Importance sampling*

For non-linear models, a Monte Carlo approximation can be used for the intractable posterior, meaning $p(\mathbf{z}_{1:t} \mid \mathbf{y}_{1:t})$ is approximated by a set of K weighted samples, called *particles*:

$$p(\mathbf{z}_{1:t} \mid \mathbf{y}_{1:t}) \approx \frac{1}{K} \sum_{k=1}^{K} \delta(\mathbf{z}_{1:t}^k),$$

where $\delta(\mathbf{z}_{1:t}^k)$ denotes the k-th particle Dirac delta at $\mathbf{z}_{1:t}^k$.

While ideally this approximation would be constructed by directly sampling from $p(\mathbf{z}_{1:t} \mid \mathbf{y}_{1:t})$ (Eq. 2.16), this is usually not possible. Instead, *importance sampling* can be used [120]: draw samples from a proposal distribution $r(\mathbf{z}_{1:t} \mid \cdot)$, chosen such

that it has support wherever the target $p(\mathbf{z}_{1:t} \mid \mathbf{y}_{1:t})$ is non-zero, and reweight the samples accordingly. This only requires evaluating a quantity proportional to the target density. Here $p(\mathbf{z}_{1:t} \mid \mathbf{y}_{1:t}) \propto p(\mathbf{y}_{1:t}, \mathbf{z}_{1:t})$ can be used since the observed data $p(\mathbf{y}_{1:t})$ is constant with respect to $\mathbf{z}_{1:t}$:

$$p(\mathbf{z}_{1:t} \mid \mathbf{y}_{1:t}) \propto p(\mathbf{y}_{1:t}, \mathbf{z}_{1:t}) \frac{r(\mathbf{z}_{1:t} \mid \cdot)}{r(\mathbf{z}_{1:t} \mid \cdot)},$$
$$p(\mathbf{z}_{1:t} \mid \mathbf{y}_{1:t}) \propto w(\mathbf{z}_{1:t})\, r(\mathbf{z}_{1:t} \mid \cdot),$$

where the (unnormalized) importance weight function is $w(\mathbf{z}_{1:t}) = \frac{p(\mathbf{y}_{1:t}, \mathbf{z}_{1:t})}{r(\mathbf{z}_{1:t} \mid \cdot)}$. Next, drawing K samples, $\mathbf{z}^k_{1:t} \sim r$, the Monte Carlo approximation of $r(\mathbf{z}_{1:t} \mid \cdot) \approx \frac{1}{K} \sum_{k=1}^{K} \delta(\mathbf{z}^k_{1:t})$ is inserted

$$p(\mathbf{z}_{1:t} \mid \mathbf{y}_{1:t}) \underset{\sim}{\propto} \sum_{k=1}^{K} w^k_{1:t}\, \delta(\mathbf{z}^k_{1:t}),$$

with (unnormalised) particle weights $w^k_{1:t} = \frac{p(\mathbf{y}_{1:t}, \mathbf{z}^k_{1:t})}{r(\mathbf{z}^k_{1:t} \mid \cdot)}$. After normalizing the weights $\overline{w}^k_{1:t} = \frac{w^k_{1:t}}{\sum_{j=1}^{K} w^j_{1:t}}$, the final particle approximation is obtained:

$$p(\mathbf{z}_{1:t} \mid \mathbf{y}_{1:t}) \approx \sum_{k=1}^{K} \overline{w}^k_{1:t}\, \delta(\mathbf{z}^k_{1:t}). \tag{2.18}$$

2.2.4 *Sequential Importance Sampling and particle filtering*

Constructing a proposal that allows directly sampling $\mathbf{z}_{1:T}$ jointly is hard. However we can design a proposal that successively samples $\mathbf{z}_t$ only conditioned on previous particles $\mathbf{z}^k_{1:t-1}$ and/or observations $\mathbf{y}_{1:T}$. Then, the Markov structure of the model in Eq. 2.15 can be used to write the importance weights recursively as [120]:

$$w^k_{1:t} \propto w^k_{1:t-1} \frac{p(\mathbf{y}_t \mid \mathbf{z}^k_t)\, p(\mathbf{z}^k_t \mid \mathbf{z}^k_{t-1})}{r(\mathbf{z}^k_t \mid \cdot)}. \tag{2.19}$$

Over time however, this suffers from weight degeneracy, where the variance of the importance weights diverges until only a few particles have non-negligible weights. To mitigate this, a resampling step is introduced [120]. Sample:

$$a^k_t \sim \mathsf{Discrete}(a^k_t \mid \overline{w}^k_{1:t}),$$

And take the corresponding particles as the new posterior approximation:

$$p(\mathbf{z}_t \mid \mathbf{y}_{1:t}) \approx \frac{1}{K} \sum_{k=1}^{K} \delta(\mathbf{z}^{a^k_t}_t).$$

Note that after a resampling step all weights were reset to $\overline{w}^k_{1:t} = 1/K$.

2.2.4.1 *Summary of the particle filtering algorithm*

Given initial samples $\mathbf{z}_1^{1:K} \sim r$ and corresponding importance weights $\overline{w}_1^{1:K}$ repeatedly execute the following steps:

$$\begin{aligned} &\textit{resample} && a_{t-1}^k \sim \mathsf{Discrete}(a_{t-1}^k \mid \overline{w}_{t-1}^k), \\ &\textit{propose} && \mathbf{z}_t^k \sim r(\mathbf{z}_t^k \mid \cdot), \\ &\textit{reweight} && w_t^k = \frac{p(\mathbf{y}_t \mid \mathbf{z}_t^k) p(\mathbf{z}_t^k \mid \mathbf{z}_{t-1}^{a_{t-1}^k})}{r(\mathbf{z}_t^k \mid \cdot)}, \end{aligned}$$

Note that resampling is applied every step, but it might be advantageous to resample only when the *effective sample size* drops below a certain threshold [121]. Following the presentation in Doucet & Johansen [120], here $p(\mathbf{z}_{1:t} \mid \mathbf{y}_{1:t})$ is directly approximated instead of the separate update and predict steps (Eq. 2.16 and Eq. 2.17). The predict step ($p(\mathbf{z}_t \mid \mathbf{y}_{1:t-1})$) can however be approximated by plugging the particle filtering approximation for $p(\mathbf{z}_{t-1} \mid \mathbf{y}_{1:t-1})$ (Eq. 2.18), into Eq. 2.16. This results in the following mixture distribution:

$$p(\mathbf{z}_t \mid \mathbf{y}_{1:t-1}) \approx \sum_{k=1}^{K} \overline{w}_{t-1}^k p(\mathbf{z}_t \mid \mathbf{z}_{t-1}^k). \tag{2.20}$$

where we used that $\int f(x)\delta(a)dx = f(a)$.

2.2.4.2 *A note on proposal distributions*

The efficiency of the particle filter relies heavily on the choice of the proposal distribution r. In this thesis we will make use of the following designs:
Bootstrap proposal: The simplest choice uses the transition dynamics as the proposal: $r = p(\mathbf{z}_t \mid \mathbf{z}_{t-1}^k)$. Plugging this into the weight update equation leads to a massive simplification: $w_t^k = p(\mathbf{y}_t \mid \mathbf{z}_t^k)$. Conditioning on observations only takes place through resampling.
Optimal proposal: If observations are linear and Gaussian: $r \propto p(\mathbf{y}_t \mid \mathbf{z}_t) p(\mathbf{z}_t \mid \mathbf{z}_{t-1}^k)$, can be computed in closed form. This is called the optimal proposal as it minimizes the variance of sampling (in fact the weights become independent of the newly sampled $\mathbf{z}_t^k$) [120].
Encoder $\times$ Transition: When observations are non-linear (as in Chapter 4), the transition distribution can be combined with a learned encoder network $e_\phi(\mathbf{z}_t \mid \mathbf{y}_t)$, i.e., we take $r \propto e_\phi(\mathbf{z}_t \mid \mathbf{y}_t) p(\mathbf{z}_t \mid \mathbf{z}_{t-1}^k)$.

2.2.5 *Parameter learning*

When fitting models to experimental data, one wants to find the model parameters $\boldsymbol{\theta}$ that maximize the marginal likelihood of the observations (Eq. 2.14). Note that,

although, not explicitly written, in the following section we assume both q and p can have learnable parameters, which are potentially (partly) shared.

2.2.5.1 *Variational filtering objective*

Particle filters provide an unbiased estimate of the marginal likelihood which is (assuming resampling every time-step) [120]:

$$\hat{p}(\mathbf{y}_{1:T}) = \prod_{t=1}^{T} \frac{1}{K} \sum_{k=1}^{K} w_t^k.$$

As proposed in three works published concurrently [115–117], to optimise parameters of p and q we do gradient descent on a lower bound to this estimate of the marginal log-likelihood $\log p(\mathbf{y}_{1:T})$.

Denoting with q the approximate posterior distribution:

$$q(\mathbf{z}_{1:T}^{1:K}, a_{1:T-1}^{1:K} \mid \mathbf{y}_{1:T}) = \prod_{k=1}^{K} r(\mathbf{z}_1^k \mid \mathbf{y}_1) \prod_{k=1}^{K} \prod_{t=2}^{T} r(\mathbf{z}_t^k \mid \mathbf{z}_{t-1}^{a_{t-1}^k} \mathbf{y}_t) \mathsf{Discrete}(a_{t-1}^k \mid \overline{w}_{t-1}^k).$$

We have

$$\log p = \log \mathbb{E}_q[\hat{p}] \geq \mathbb{E}_q[\log \hat{p}], \tag{2.21}$$

per *Jensen's inequality*, so we maximize:

$$\mathcal{L}_{SMC} = \mathbb{E}_q[\log \hat{p}] = \sum_{t=1}^{T} \mathbb{E}_q[\log \frac{1}{K} \sum_{k=1}^{K} w_t^k]. \tag{2.22}$$

with backpropagation (through time). As suggested in previous studies [115–117, 122], the high-variance terms arising from the resampling can be dropped, at the cost of inducing a slight bias in the gradients (but not in the likelihood estimate).

2.2.6 *Connection to standard ELBO*

We next show that by first dropping the resampling step, and then setting the number of particles to $K = 1$, we recover the standard evidence lower bound (ELBO) used in variational inference, e.g., in variational autoencoders [123, 124]:

$$\begin{aligned} \mathcal{L}_{ELBO} &= \mathbb{E}_q \left[\log p(\mathbf{y}_{1:T}, \mathbf{z}_{1:T}) - \log q(\mathbf{z}_{1:T})\right] \\ &= \underbrace{\mathbb{E}_q[\log p(\mathbf{y}_{1:T} \mid \mathbf{z}_{1:T})]}_{\text{Reconstruction}} - \underbrace{\mathbb{D}_{\mathrm{KL}}(q(\mathbf{z}_{1:T}) \parallel p(\mathbf{z}_{1:T}))}_{\text{Regularization}}. \end{aligned} \tag{2.23}$$

2.2.6.1 *Dropping resampling: IWAE*

Removing the resampling step results in Sequential Importance Sampling, as previously described. In this case, particle trajectories evolve independently and the likelihood estimator becomes

$$\hat{p}(\mathbf{y}_{1:T}) = \frac{1}{K}\sum_{k=1}^{K} w_{1:T}^k = \frac{1}{K}\sum_{k=1}^{K}\prod_{t=1}^{T} w_t^k$$

and the corresponding objective

$$\mathcal{L}_{IWAE} = \mathbb{E}_q\left[\log \frac{1}{K}\sum_{k=1}^{K}\prod_{t=1}^{T} w_t^k\right],$$

which is the Importance Weighted Autoencoder (IWAE) [125–127] objective. Note that because we are not resampling, the weights are not reset, so we have to multiply them within the summation inside the log.

2.2.6.2 *Recovering the ELBO ($K = 1$)*

Consider now the case $K = 1$. The objective reduces to

$$\mathcal{L} = \mathbb{E}_q\left[\log \prod_{t=1}^{T} w_t\right].$$

Now we can again pull the product out of the log:

$$\mathcal{L} = \mathbb{E}_q\left[\sum_{t=1}^{T} \log w_t\right] = \sum_{t=1}^{T} \mathbb{E}\left[\log w_t\right].$$

Substituting the definition of w_t gives

$$\mathcal{L} = \sum_{t=1}^{T} \mathbb{E}_q\left[\log p(\mathbf{y}_t \mid \mathbf{z}_t) + \log p(\mathbf{z}_t \mid \mathbf{z}_{t-1}) - \log r(\mathbf{z}_t \mid \cdot)\right].$$

Since, without resampling and with $K = 1$, the approximate posterior is simply $q(\mathbf{z}_t \mid \cdot) = r(\mathbf{z}_t \mid \cdot)$, we obtain exactly the ELBO (Eq. 2.23).

2.2.6.3 *On the ELBO in sequential VAEs*

Similarly as to the sequential importance sampling approaches above, we can also use the Markov structure of the model (Eq. 2.15) to decompose the ELBO (Eq. 2.23), resulting in:

$$\mathcal{L}_{SVAE} = \sum_{t=1}^{T} \mathbb{E}_{q_t}\left[\log p(\mathbf{y}_t \mid \mathbf{z}_t)\right] - \sum_{t=2}^{T} \mathbb{E}_{q_{t-1}}\left[\mathbb{D}_{\mathrm{KL}}\big(q(\mathbf{z}_t \mid \cdot) \,\|\, p(\mathbf{z}_t \mid \mathbf{z}_{t-1})\big)\right]. \quad (2.24)$$

This is used in various recent works (e.g., [44, 45, 111, 128, 129]), with different choices of $q(\mathbf{z}_t \mid \cdot)$. Brenner *et al.* [111] and Mi *et al.* [129] make so-called mean-field assumptions, which explicitly assumes that the latent variables are independent of each other, when we condition on the data $q(\mathbf{z}_{1:T} \mid \mathbf{y}_{1:T}) = \prod_{t=1}^{T} q(\mathbf{z}_t \mid \mathbf{y}_{1:T})$. This factorization has the advantage that the SVAE loss (Eq. 2.24) can be evaluated in parallel over time steps, and no backpropagation through time takes place. This is computationally efficient, but might imply the RNN struggles to learn stable long-term dynamics [23, 100]. Dowling, Zhao & Park [45] use approximate posteriors of the form $q(\mathbf{z}_t \mid \mathbf{z}_{t-1}, \mathbf{y}_{t:T})$, where they combine the transition distribution $p(\mathbf{z}_t \mid \mathbf{z}_{t-1})$ with an encoder $e(\mathbf{z}_t \mid \mathbf{y}_{t:T})$, similar to the proposals we use in Chapter 4. Instead of a particle-based approximation they use Gaussian approximation, obtained by sampling from $p(\mathbf{z}_t \mid \mathbf{y}_{1:t-1})$ (not unlike an ensemble Kalman Filter [130] or an unscented Kalman Filter [131] without *fixed* sample points), after which the update is equivalently $q(\mathbf{z}_t \mid \mathbf{y}_{1:T}) \propto p(\mathbf{z}_t \mid \mathbf{y}_{1:t-1}) e(\mathbf{z}_t \mid \mathbf{y}_{t:T})$.

2.2.7 *Relations to teacher forcing*

Another common approach for training (generally deterministic) RNNs to reproduce sequences of data is *teacher forcing* [23, 100]. We will start with the fully observed setting where we model the sequence $\mathbf{y}_{1:T}$ directly with a recurrent neural network, $\mathbf{y}_t = F_{\boldsymbol{\theta}}(\mathbf{y}_{t-1})$. Here we can minimize the mean-squared error (MSE) of one-step-ahead predictions:

$$\mathcal{L}_{\text{MSE}} = \sum_{t=2}^{T} \left\| \mathbf{y}_t - F_{\boldsymbol{\theta}}(\mathbf{y}_{t-1}) \right\|^2,$$

where $\mathbf{y}_t$ and $\mathbf{y}_{t-1}$ are successive samples from the data. This objective has a probabilistic interpretation as maximum likelihood estimation given a Gaussian observation model, i.e., $\arg\max_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(\mathbf{y}_{2:T} \mid \mathbf{y}_1) = \prod_{t=2}^{T} p_{\boldsymbol{\theta}}(\mathbf{y}_t \mid \mathbf{y}_{t-1})$, with $p_{\boldsymbol{\theta}}(\mathbf{y}_t \mid \mathbf{y}_{t-1}) = \mathcal{N}\big(F_{\boldsymbol{\theta}}(\mathbf{y}_{t-1}), \sigma^2 I\big)$, for which the negative log-likelihood equals the MSE up to scaling and constants.

Similarly, if we consider an RNN with latent dynamics $\mathbf{z}_t = F_{\boldsymbol{\theta}}(\mathbf{z}_{t-1})$ and linear observations $\mathbf{y}_t = B\mathbf{z}_t$, we could optimize

$$\mathcal{L}_{\text{MSE}} = \sum_{t=2}^{T} \left\| \mathbf{B}^{\dagger} \mathbf{y}_t - F_{\boldsymbol{\theta}}(\mathbf{z}_{t-1}) \right\|^2, \tag{2.25}$$

which can be interpreted as teacher forcing in latent space, where the targets are inferred deterministically from observerd data using the (pseudo-) inverse of the observation matrix.

For stochastic transitions, $p_{\boldsymbol{\theta}}(\mathbf{z}_t \mid \mathbf{z}_{t-1}) = \mathcal{N}\big(F_{\boldsymbol{\theta}}(\mathbf{z}_{t-1}), \sigma^2 I\big)$, we get a direct relation to the ELBO in Eq. 2.24. In particular, consider a mean-field approximation of the posterior of the form:

$$q(\mathbf{z}_t \mid \mathbf{y}_t) = \mathcal{N}\big(B^{\dagger} \mathbf{y}_t, \boldsymbol{\Sigma}_q\big).$$

If we plug this into the $\mathbb{D}_{\mathrm{KL}}$ term of the ELBO we get:

$$\mathbb{D}_{\mathrm{KL}}\big(q(\mathbf{z}_t \mid \mathbf{y}_t) \,\|\, p_{\boldsymbol{\theta}}(\mathbf{z}_t \mid \mathbf{z}_{t-1})\big) = \frac{1}{2\sigma^2}\left\|\mathbf{B}^\dagger \mathbf{y}_t - F_{\boldsymbol{\theta}}(\mathbf{z}_{t-1})\right\|^2 + \frac{1}{2\sigma^2}\mathrm{tr}(\boldsymbol{\Sigma}_q) - \frac{R}{2} + \frac{1}{2}\log\frac{\sigma^{2R}}{\det \boldsymbol{\Sigma}_q},$$

where R is the dimensionality of $\mathbf{z}$. Thus, up to additional terms involving the covariances of q and p, the $\mathbb{D}_{\mathrm{KL}}$ term results in the same MSE objective as above. Variational inference naturally extends this framework to more complex (nonlinear) observation models by learning an encoder $q_{\boldsymbol{\phi}}(\mathbf{z}_t \mid \mathbf{y}_t)$ (or $q_{\boldsymbol{\phi}}(\mathbf{z}_t \mid \mathbf{y}_{1:T})$) that probabilistically inverts the observations.

The forms of teacher forcing discussed so far have the computational advantage that loss terms can be evaluated in parallel over time, avoiding backpropagation through time. As noted before this is also true for Eq. 2.24 with a mean-field assumption on the posterior. However, during simulation (roll-out), the model must generate latents $\mathbf{z}_t$ based on its own previous samples, i.e., by successively sampling from $p(\mathbf{z}_t \mid \mathbf{z}_{t-1})$, rather than from $q(\mathbf{z}_t \mid \mathbf{y}_t)$. This mismatch can lead to error accumulation and instability when generating long time-series, similar to classical limitations of teacher forcing in deterministic RNNs [23, 100].

Recent work has attempted to mitigate this issue by training on n-step-ahead predictions instead of one-step ahead predictions (i.e., only 'forcing' the RNN every n timesteps), so-called *sparse teacher forcing* [132]. Alternatively, one can apply *generalized teacher forcing* [64, 100], where the model is softly forced using a convex combination of targets and predictions:

$$\mathbf{z}_t = (1-\alpha)F_{\boldsymbol{\theta}}(\mathbf{z}_{t-1}) + \alpha\mathbf{B}^\dagger \mathbf{y}_t.$$

Theses approaches strike a balance between full backpropagation through time (with its associated vanishing/exploding gradient issues [93]) and standard teacher forcing.

Both variants of teacher forcing can also be interpreted through the lens of variational inference [69]. In Chapter 4, we will sample latents from distributions like: $q(\mathbf{z}_t \mid \mathbf{y}_t, \mathbf{z}_{t-1})$. For linear observation models, $p(\mathbf{z}_t \mid \mathbf{y}_t, \mathbf{z}_{t-1})$ is available in closed form (as in Kalman filtering [119]), and corresponds to a linear combination between RNN-predicted and data-inferred states, with weights determined by their respective uncertainties. We have for the mean of q:

$$\boldsymbol{\mu}_t = (\mathbf{I} - \mathbf{K}_t\mathbf{B})F_{\boldsymbol{\theta}}(\mathbf{z}_{t-1}) + \mathbf{K}_t\mathbf{y}_t,$$

with $\mathbf{K}_t$ the Kalman Gain. This resembles the interpolation in generalized teacher forcing. Similarly, sparse teacher forcing could potentially be interpreted as alternating between sampling from $q = p(\mathbf{z}_t \mid \mathbf{z}_{t-1})$ and $q = q(\mathbf{z}_t \mid \mathbf{y}_t)$, which bears resemblance to encoder dropout regularization schemes used in sequential VAE models [45, 133].

3

TRAINED RECURRENT NEURAL NETWORKS DEVELOP PHASE-LOCKED LIMIT CYCLES IN A WORKING MEMORY TASK

Neural oscillations are ubiquitously observed in many brain areas. One proposed functional role of these oscillations is that they serve as an internal clock, or 'frame of reference'. Information can be encoded by the timing of neural activity relative to the *phase* of such oscillations. In line with this hypothesis, there have been multiple empirical observations of such *phase codes* in the brain. Here we ask: What kind of neural dynamics support phase coding of information with neural oscillations? We tackled this question by analyzing recurrent neural networks (RNNs) that were trained on a working memory task. The networks were given access to an external reference oscillation and tasked to produce an oscillation, such that the phase difference between the reference and output oscillation maintains the identity of transient stimuli. We found that networks converged to stable oscillatory dynamics. Reverse engineering these networks revealed that each phase-coded memory corresponds to a separate limit cycle attractor. We characterized how the stability of the attractor dynamics depends on both reference oscillation amplitude and frequency, properties that can be experimentally observed. To understand the connectivity structures that underlie these dynamics, we showed that trained networks can be described as two phase-coupled oscillators. Using this insight, we condensed our trained networks to a reduced model consisting of two functional modules: One that generates an oscillation and one that implements a coupling function between the internal oscillation and external reference.

In summary, by reverse engineering the dynamics and connectivity of trained RNNs, we propose a mechanism by which neural networks can harness reference oscillations for working memory. Specifically, we propose that a phase-coding network generates autonomous oscillations which it couples to an external reference oscillation in a multi-stable fashion.

3.1 DECLARATION

This chapter is based on: Matthijs Pals, Jakob H Macke, Omri Barak. Trained recurrent neural networks develop phase-locked limit cycles in a working memory task. *PLOS Computational Biology* 20, 2 (2024).

3.1.1 *Author contributions*

The project was conceptualised by me with input from Jakob H Macke and Omri Barak. The RNN task and analysis were primarily developed by me, with the reverse engineering analysis being partly thought up by Omri Barak. I implemented all experiments, performed all mathematical analysis and generated all figures. The paper was written by me with help from Omri Barak and feedback from Jakob H Macke.

CRediT

Conceptualization: Matthijs Pals, Jakob H Macke, Omri Barak
Formal analysis: Matthijs Pals
Funding acquisition: Jakob H Macke, Omri Barak
Software: Matthijs Pals
Supervision: Jakob H Macke, Omri Barak
Writing – original draft: Matthijs Pals, Omri Barak
Writing – review & editing: Jakob H Macke

3.2 INTRODUCTION

Rhythmic neural activity is an abundant phenomenon in nervous systems. Neural oscillations naturally underlie behavior with an observable oscillatory component, such as walking and digesting [134, 135]. Oscillating neural activity is also widely observed in brain regions implicated in higher cognitive functions, where there is no obvious correlate to oscillatory behavior [136]. Such rhythmic neural activity has been suggested to support short-term memory maintenance (among other functions), by serving as an internal clock for the brain [137–142]. This makes *phase coding* possible: Information can be encoded by spikes that are systematically timed with respect to the phase of ongoing oscillations. Empirical observations of a *phase code* in the brain were first described in the hippocampus of moving rats [143]. Since then, phase coding has also been associated with the representation of discrete object categories [144–146] for short-term maintenance of stimuli [147–150] and goals [151]. Such observations have been reported in a wide range of brain regions, including the human medial temporal lobe as well as primate prefrontal and sensory cortices.

Oscillations are a dynamic phenomenon, and it is therefore a natural question to ask: How do ongoing oscillations in the brain interact with the neural dynamics that support cognitive functions? Specifically, we seek to characterize dynamical systems that could underlie working memory relying on phase coding with neural oscillations. We do so by assuming that cognitive functions can be described by a low-dimensional dynamical system, implemented through populations of neurons (computation through dynamics) [36, 37, 39, 46, 47, 56, 152], in line with empirical observations [32, 33].

We make use of artificial recurrent neural networks (RNNs). RNNs are universal dynamical systems, in the sense that they can approximate finite time trajectories of any dynamical system [48, 49]. We are thus able to use these networks as hypothesis generators — first training them on a cognitive task, and then reverse engineering the resulting networks [39, 47, 51–58, 153].

Concretely, we provide RNNs with both a reference oscillation, and stimuli. Experimental observations show that the phase of spikes relative to ongoing oscillations contains information about external stimuli [145, 147, 150, 151]. Analogous to this, we ask the RNN to produce an oscillatory signal, which has to code the identity of stimuli by its phase relative to the reference oscillation (Fig. 3.1). Depending on the training setup, we find two different solutions, of which one corresponds to network units coding for information by their phase and one solution in which units code with their average firing rates (Fig. 3.2). We show that phase-coded memories correspond to stable limit cycles and demonstrate how empirically observable quantities control the stability of these attractors (Fig. 3.3). Having detailed the dynamics, we study the connectivity of our trained RNNs (Fig. 3.4). Finally, we show that the system can be well approximated by two coupled oscillators. Based on these analyses, we propose that phase-coded memories reside in stable limit cycles, resulting from the coupling of an oscillation generated by the recurrent network to an external reference oscillation.

3.3 RESULTS

3.3.1 *Tractable, oscillating recurrent neural networks perform a working memory task*

In order to study the dynamics of phase coding during working memory, we defined a task in which an RNN receives transient stimuli, and has to encode their identity using the relative phase of oscillations (Fig. 3.1a). The network consists of N units, with activation $\mathbf{x}(t) \in \mathcal{R}^N$, recurrently connected via a connectivity matrix $\mathbf{J} \in \mathcal{R}^{N\times N}$, and receiving external oscillatory input $u(t) \in \mathcal{R}$ [140, 150, 154], as well as stimuli $\mathbf{s}(t) \in \mathcal{R}^2$,

$$\tau \frac{d\mathbf{x}}{dt} = -\mathbf{x}(t) + \mathbf{J}\tanh(\mathbf{x}(t)) + \mathbf{I}^{(osc)}u(t) + \mathbf{I}^{(s)}\mathbf{s}(t) + \boldsymbol{\xi}(t), \tag{3.1}$$

where τ represents the time constant of the units, tanh is an elementwise nonlinearity, $\mathbf{I}^{(osc)} \in \mathcal{R}^N, \mathbf{I}^{(s)} \in \mathcal{R}^{N\times 2}$ represent the input weights, and $\boldsymbol{\xi}(t) \in \mathcal{R}^N$ independent noise for each unit. Motivated by experiments observing phase coding relative to local field potential oscillations (LFPs) [145, 147, 148, 150, 151], we used filtered rat CA1 local field potentials [155, 156] as input $u(t)$. By using the LFP as input, we make the assumption that the neurons in our model have an overall negligble effect on the LFP, which is valid if they are only a small subset of all contributing neurons, or because the LFP largely reflects input coming an upstream population.

During a given trial, one of two stimuli a or b, was transiently presented to the network at a randomized onset-time, with amplitude s_a or s_b, respectively, representing the working-memory to be stored (e.g. a navigational goal [151], or sequence list position [150]). To define the desired output, we were inspired by experimental studies showing that the relative timing of spikes to an ongoing oscillation is informative about sensory stimuli. Similarly to these observations, we wanted RNNs to encode information about external events relative to ongoing oscillations. To do this, the task for the network was to produce an oscillation, that is either in-phase with respect to the reference signal, following stimulus a, or anti-phase, following stimulus b. Networks were trained with backpropagation through time (Fig. S.1) and successfully learned the task (Fig. 3.1b).

As we were interested in reverse engineering the networks, we made two simplifications. After training, we replaced the LFP reference signal with a pure sine wave with phase θ. During training, we chose a constraint on the connectivity that reduces the complexity of our analysis while still allowing for expressive networks. Specifically, we constrained the recurrent weight matrix to be of rank two, by decomposing it as an outer product of two pairs of vectors (Fig. 3.1c; see Fig. S.2 for unconstrained networks) [49, 56, 59, 102],

$$\mathbf{J} = \frac{1}{N}(\mathbf{m}^{(1)}\mathbf{n}^{(1)\mathsf{T}} + \mathbf{m}^{(2)}\mathbf{n}^{(2)\mathsf{T}}). \tag{3.2}$$

By constraining the weight matrix, we directly constrain the dynamics. Specifically, the projections of network activity on the two vectors $\mathbf{m}^{(1)}, \mathbf{m}^{(2)}$, which we term

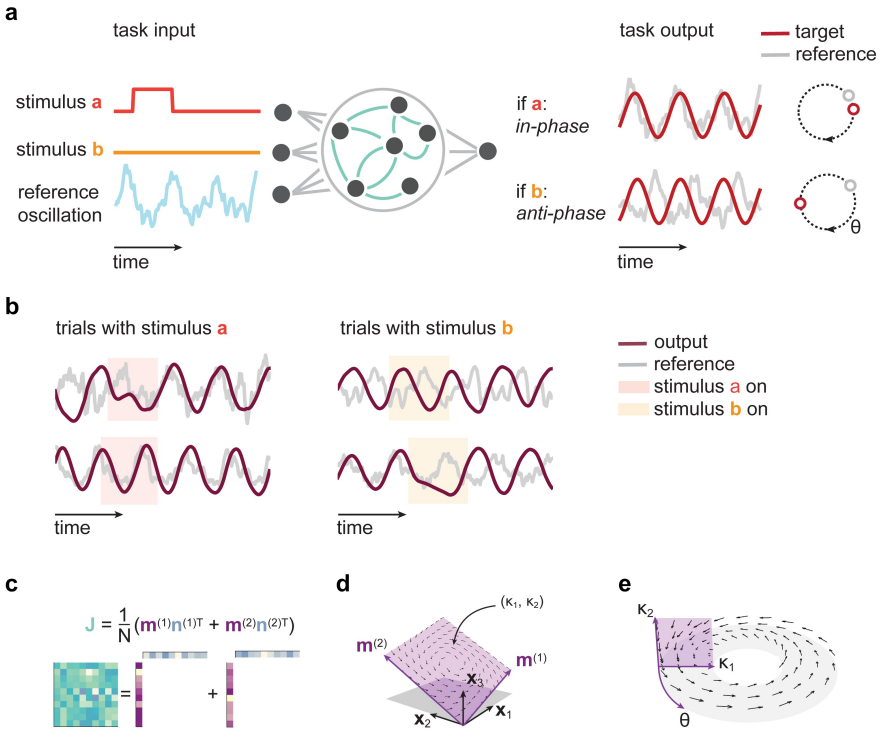

FIGURE 3.1: **Trained RNNs encode stimuli in oscillation phase. a)** RNNs receive transient stimuli as input, along with a reference oscillation. Networks are trained to produce an oscillation, such that the phase of the produced oscillation (relative to the reference oscillation), maintains the identity of transient stimuli. **b)** Example output of trained networks. Transient presentation of stimulus a, results in an in-phase output oscillation (left), regardless of the initial phase (top or bottom). Similarly, the b stimulus results in an anti-phase oscillation, again irrespective of its initial phase (right). **c)** To obtain a tractable model, we apply a low-rank constraint to the recurrent weight matrix of the RNN, i.e., we require that the weight matrix can be written as the outer product of two sets of vectors $\mathbf{m}^{(1)}, \mathbf{m}^{(2)}$ and $\mathbf{n}^{(1)}, \mathbf{n}^{(2)}$. **d)** Low-rank connectivity leads to low dimensional dynamics. In the absence of any input, the recurrent dynamics, described by coordinates κ_1, κ_2, lie in a linear subspace spanned by $\mathbf{m}^{(1)}$ and $\mathbf{m}^{(2)}$ (purple). **e)** When probing the model with sinusoidal oscillations, we can rewrite the system as a dynamical system in a three-dimensional phase space, where the additional axis, θ, is the phase of the input oscillation. We can visualize this phase space as a toroid such that the horizontal circle represents θ, and the vertical cross-section is the κ_1, κ_2 plane.

κ_1, κ_2 respectively, are sufficient to describe the network dynamics in the absence of inputs (Fig. 3.1d).

In the presence of sinusoidal input, the κs are not sufficient to describe the dynamics, and we also need to know the current phase θ of the reference oscillation.

These three numbers constitute the complete phase space $\mathcal{M}$ for our dynamical system (Fig. 3.1e),

$$\mathcal{M} = \{(\kappa_1, \kappa_2, \theta) \in \mathcal{R}^2 \times \mathcal{S}^1\}.$$

3.3.2 *Phase-coded memories correspond to limit cycle attractors*

We reverse-engineered the dynamics of our trained networks in order to understand how they solve the task [51]. We found that networks converge to one of two solutions, characterized by their activity following the transient stimuli. The solution that the network ends up choosing is dependent on both the training setup and the initialisation used (See Fig. S.3 for an investigation into when each solution arises).

In the first solution, individual network units code for stimuli by using their phase of oscillation relative to the reference, as illustrated by rate traces of example units (Fig. 3.2a), as well as statistics for all units (Fig. S.3). In κ space, the population activity corresponds to two cycles that roughly lie on the same area, but have a different phase relation to the reference oscillation (Fig. 3.2b). In the full phase space $\mathcal{M}$ introduced above, this solution corresponds to the network activity residing in one of two linked cycles (Fig. 3.2c).

In the second solution, single units code stimulus identity by their average activation (Fig. 3.2d). Units with activity far away from 0 saturate their (i.e. are at the flat part of their) tanh nonlinearity, and as a consequence have little effect on the phase of the output oscillation. A different subset of units saturates after after either stimulus, leading to different units determining the output oscillation. These units either directly transmit the input oscillation, creating an in-phase oscillation, or flip its sign to create an anti-phase oscillation. In κ space, activity lies on two non-overlapping cycles (Fig. 3.2e), which are likewise completely separated in the full phase space (Fig. 3.2f).

Given the that the phase-coding solution matches the phenomenon we set out to investigate, whereas the rate-coding solution shows similarities to previous work on fixed point dynamics in RNNs [49, 56, 58], we focus here on the analysis of the 'phase-coding' solution (Fig. 3.2a-c). The rate-coding and its similarities to existing work is detailed in the Supplementary (Fig. S.4).

Single trials have a limited duration, and hence the cycles we observed might arise either from a transient dynamical phenomenon or from stable attractors. These would lead to different experimental observations of residual dynamics (trial-to-trial variability in neural population responses) [157], and responses to perturbations [153]. To study stability of limit cycles, we used discrete-time iterative maps, also known as Poincaré maps [34, 108]. To obtain such a map, we start by making a cross-section Q of the phase space ($Q = \{(\kappa_1, \kappa_2, \theta) : \mod \theta = 2\pi\}$), such that trajectories go through Q once every cycle of the reference oscillation (Fig. 3.3a). We can then define an iterative map, corresponding to following a trajectory starting in Q till its next intersection with Q. We can denote this mapping ($Q_0 \to Q_{2\pi}$) as

$$\boldsymbol{\kappa}_{c+1} = \mathbf{P}(\boldsymbol{\kappa}_c),$$

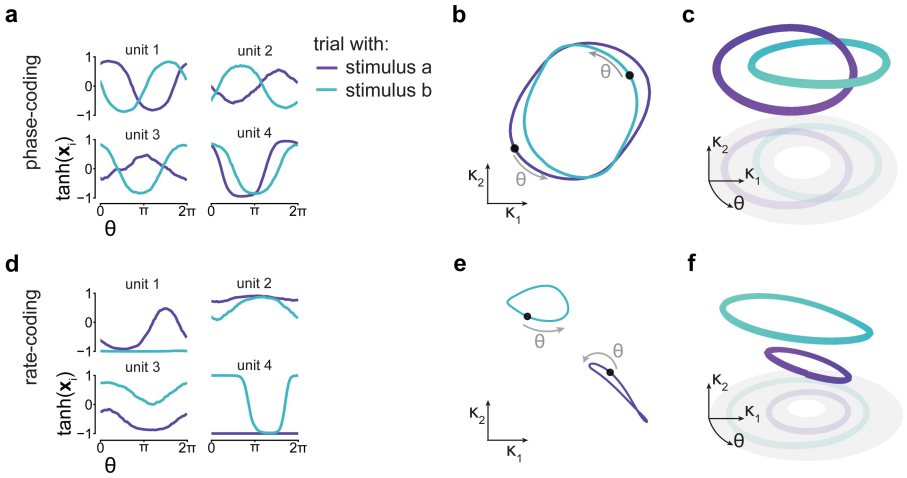

FIGURE 3.2: **Both phase coding and rate coding can solve the working memory task.** We found two qualitatively different solutions. **a)** In the first solution (panels A to C), we find that single unit activity codes for stimulus information by relative phase: We plot the rates $\tanh(\mathbf{x}_i)$ of 4 example units i, as a function of the reference oscillation phase θ. We find that single unit activity oscillates, with the phase relative to the reference oscillation depending on stimulus identity (colors). **b)** Projecting $\mathbf{x}$ to the κ_1, κ_2 plane reveals that population activity lies on overlapping cycles in this plane. Here, the black dots denote $\theta = 0$. **c)** In the full phase space $\mathcal{M}$, the trajectories are non-overlapping, but the cycles are linked. **d)** In the second solution (Panels d to f), we find that stimulus identity does not modulate the phase of single units, but rather their mean activity. **e)** This rate-code corresponds to two cycles separated in the κ_1, κ_2 plane. **f**.) These cycles are also separated in the full phase space.

where $\boldsymbol{\kappa}_c = [\kappa_{1,c}, \kappa_{2,c}]^\mathsf{T}$ corresponds to the c'th intersection of trajectory with Q. Practically, we obtain the map by integrating the dynamics for one period of the input oscillation (Eq 3.9). Once we find a fixed point of $\mathbf{P}$, i.e. a trajectory comes back exactly where it started, we obtain a limit cycle of the original system.

Using the Poincaré map, we found that trajectories starting from many initial conditions quickly converged to one of two fixed points of $\mathbf{P}$ (Fig. 3.3a), corresponding to the cycles observed during the working memory task (Fig. 3.2c). To confirm their stability, we performed linear stability analysis by calculating the Floquet multipliers (λ), i.e., the eigenvalues of the linearized Poincaré map. Limit cycles are stable if these have a magnitude less than one, i.e., $|\lambda| < 1$.

This analysis allows us to study how the two inputs direct activity to the corresponding limit cycle. Without inputs, both cycles are stable (Fig. 3.3b, leftmost cartoon). We can then tonically provide input corresponding to a scaled version of one of the stimuli, and recalculate the maximal Floquet multiplier for the resulting limit cycle. We found that this procedure gradually destabilizes the other limit cycle, so that eventually only the limit cycle corresponding to the correct input remains (Fig. 3.3b, right cartoons). Thus, for both stimuli, once a sufficient amplitude is

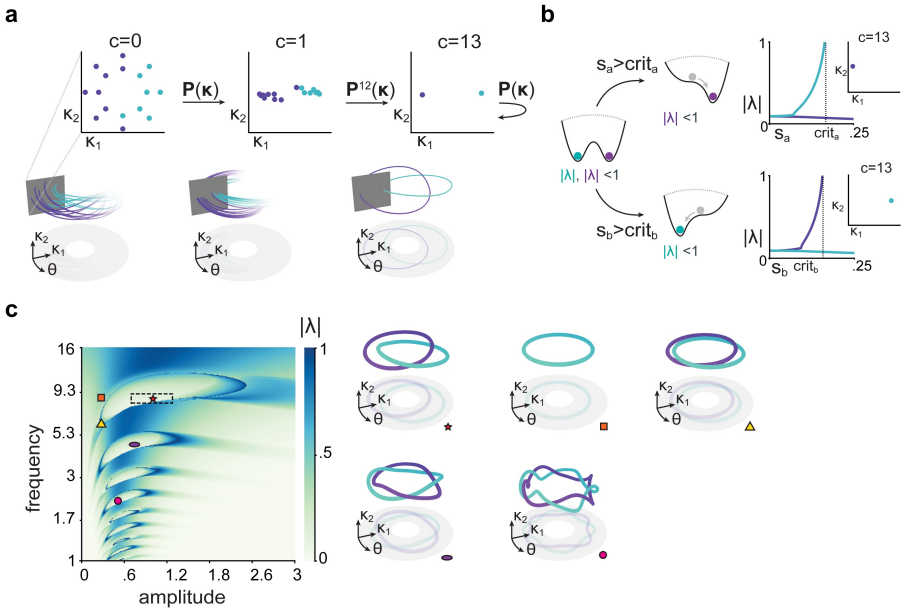

FIGURE 3.3: **Input controls the stability of the attractors in which phase-coded memories reside. a)** Poincaré maps illustrate that the cycles of Fig. 3.2 correspond to attractive limit cycles. Sixteen trajectories with different initial conditions in three different snapshots (after 0, 1 and 13 cycles). Color marks the cycle they end up in. Bottom part shows how trajectories intersect the Poincaré plane, which is shown on top. **b)** Linear stability analysis of the Poincaré maps shows that stimuli of sufficient magnitude ($s_i > crit_i$) lead to a bifurcation, such that only one of two limit cycles remains stable. Left: cartoon of the bistable dynamics (without stimuli) and bifurcation during stimulus presentation illustrated as a potential well. Right: Floquet multiplier norm (a measure of stability) as a function of stimulus amplitude. Insets show the Poincaré map after 13 iterations with the stimulus presented at amplitude $s_i = 1$. **c)** Stability analysis for a range of amplitudes and frequencies of the reference oscillation (quantities that naturally vary in the brain). The box with dashed lines indicates the 5'th and 95'th percentiles of the amplitudes and frequencies used during training trials. The 'islands' correspond to regions where bistable dynamics persist. If the reference oscillation and amplitude are within such a bistable region (e.g. the red star), there are two stable cycles, and the model can maintain memory of a stimulus. For lower amplitudes (orange square and yellow triangle), the model only retains bistability if the frequency is also lower (yellow triangle). The additional 'islands' correspond to regions with bistable $m : n$ phase locking, where the reference oscillation is an integer divisor of the oscillation generated by the RNN (e.g. purple oval, bistable $1 : 2$ phase locking; pink circle, bistable $1 : 4$ phaselocking). Trajectories on the right correspond to the different markers in the parameter space on the left.

crossed ($s_i > crit_i$; Fig. 3.3b), a bifurcation occurs and only one limit cycle remains stable. When the transient stimulus is removed, both limit cycles are stable again, but the network has already been directed to one of them.

As realistic neural oscillations exhibit drifts in both frequency and amplitude, we next assessed the robustness of our network to variations of the reference oscillation, going substantially beyond the range encountered during training. This analysis also makes testable predictions, as changes in oscillation amplitude and frequency can be experimentally observed, and potentially controlled [158].

We calculated the norm of the maximum Floquet multiplier for varying frequencies and amplitudes of the reference oscillation (Fig. 3.3c), in order to determine when bifurcations occur (i.e., maximum multiplier norm exceeding one). The resulting diagram allows us to draw two conclusions: First, there are multiple regions with two stable oscillations, but with an $m : n$ phase coupling, i.e., where the internal oscillation frequency is an integer multiplier of the reference frequency. This kind of cross-frequency phase-phase coupling is an integral part of previously proposed theories of phase coding [140, 142] (however, interpretation of experimental observations can be challenging [159]). Second, based on the shape and location of the bistable regions (the 'islands' in Fig. 3.3c), we predict that being able to phase-code stimuli is possible for reference oscillations with low frequency, only if the amplitude is also low.

3.3.3 *Low-rank RNNs as phase-coupled oscillators*

Having described the dynamics, we next aimed to understand the population structure that gives rise to these dynamics. To proceed, we observe that, following the stimulus, the internal dynamics of the RNN quickly converge to a trajectory in the (κ_1, κ_2) plane, which can be approximately described by its phase (Fig 3.2b $\phi = \arctan(\frac{\kappa_2}{\kappa_1})$). Using this insight, we can rewrite our model as two phase-coupled oscillators (Fig. 3.4a): one that describes the external reference oscillation phase (θ) and one that characterizes the internal RNN phase (ϕ). The dynamics of these oscillators are then determined by the oscillation frequency ω, and a coupling function $g(\phi, \theta)$ [160]:

$$\begin{aligned} \dot{\theta} &= \omega, \\ \dot{\phi} &= \omega + g(\phi, \theta). \end{aligned} \tag{3.3}$$

To extract the coupling function g from our trained networks, we rewrote the model equations in polar coordinates, and approximated the radius in κ-space as constant (Fig. 3.4b, Methods). We then simulated the oscillators of Eq 3.3 using this function, and observed two stable trajectories (superimposed lines in Fig. 3.4b), showing that the coupling function is sufficient to induce bistable dynamics. We found that the convergent trajectories are close to those of the RNN when projected to the same phase space (θ, ϕ). We thus verified that our approximate description captures the dynamics of the full RNN (Fig. S.5). Furthermore, the observed stimulus-induced dynamics (Fig. 3.3c) can now be explained by the effect of input on the coupling function. In the presence of input, the coupling function only admits a single stable trajectory, resulting from one stable and one unstable trajectory colliding and annihilating in a saddle-node bifurcation (Fig. 3.4c).

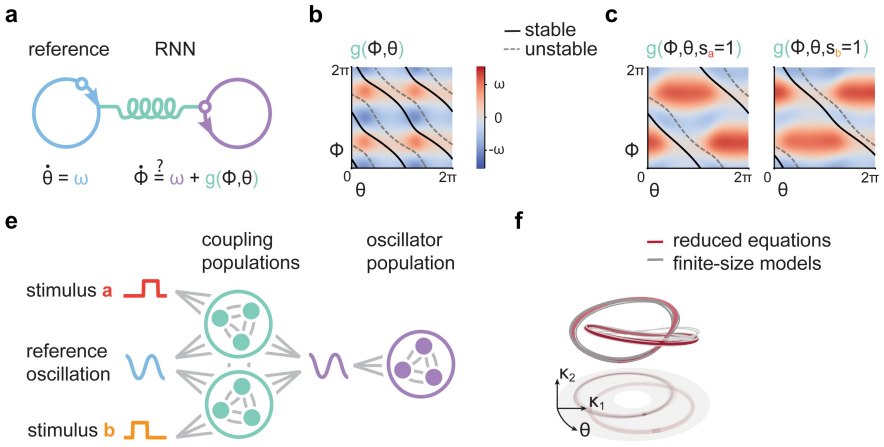

FIGURE 3.4: **Phase-coding RNNs are oscillators. a)** We hypothesized that the model functions as two coupled oscillators, where one represents the external reference oscillation phase and one the RNNs' internal oscillation phase. **b)** We extracted the coupling functions from our trained RNNs. This coupling function induces bistable dynamics when it couples two oscillators, as apparent from the superimposed stable and unstable trajectories. **c)** Input stimuli transiently modify the coupling function, resulting in the bifurcation, previously observed in Fig. 3.3b. **d)** To formalize the coupled oscillator description, we created a reduced model where weights are drawn from a mixture of Gaussians. This model consists of a population that generates oscillations, and two populations that together implement the coupling function between internal and external oscillations. **e)** Simulating a reduced set of equations that describe the idealized dynamics of RNNs with connectivity in terms of a Gaussian mixture distribution, as well as 10 finite-size models with weights sampled from this distribution, all result in trajectories similar to our original system, validating our reduced description.

Finally, we wanted to know how the rank-two connectivity leads to these coupled oscillators. To this end, we approximated the weights in the connectivity vectors of the RNN using a mixture of Gaussians [49, 56, 59]. Fitting a mixture of Gaussians to the connectivity of trained networks [161], and sampling weights from this mixture, as in previous work [56], did not reliably lead to functioning networks (Fig. S.6 a). We were, however, able to manually design a reduced model, with weights drawn from a mixture of Gaussians, based on the reverse-engineered dynamics of trained networks, as well as the structure in their connectivity as revealed through the clustering analysis (Fig. S.6 b). The reduced model consisted of three mixture components (or subpopulations; Fig. 3.4d, Fig. S.7). One component is not connected to the reference oscillation and autonomously generates its own oscillation. The other two components implement the required coupling function. The two coupling components differ only in their connectivity with the input; one population saturates (i.e. is inhibited) by stimulus *a*, and one by stimulus *b*. By adding additional coupling

populations, one can obtain a model that stores more than two stimuli, each in a separate limit cycle (Fig. S.8).

The description in terms of a mixture of Gaussians enables us to derive a reduced set of three equations, which describe the dynamics of the RNN, in the limit of infinite units (3.5.4.2). We confirmed that finite-size networks are appropriately described by this reduced description, by simulating trajectories of 10 RNNs (with N=4096 units) sampled from the reduced connectivity (Fig. 3.4e). Thus, we show that a sufficient connectivity for working memory through phase-coding entails two modules: an oscillator and a coupling function that induces bistable dynamics.

3.4 DISCUSSION

In this study, we raised a hypothesis about a potential dynamical mechanism underlying phase-coding with neural oscillations. Namely, that phase coding in recurrent networks can be implemented through multi-stable coupling of two oscillations, one internal and one external to the network. We arrived at the hypothesis by training RNNs on a working memory task, while supplying them with oscillatory reference input. The networks had to encode transient stimuli by producing an oscillation with a persistent phase shift with respect to the reference input.

Through reverse engineering the dynamics of trained RNNs, we found that phase-coded memories correspond to stable periodic attractors. These materialized in our models as linked cycles in phase space. The presence of attractive oscillatory dynamics, as opposed to marginally stable or transient trajectories, can be detected by analyzing residual dynamics from data [157] or through perturbation studies [153]. We showed how LFP frequency and amplitude jointly control the stability of the attractors. As LFP oscillation frequency and amplitude vary naturally in the brain, and can potentially be steered [158], this relationship could be probed directly from neural data.

Beyond characterizing the dynamics, we also revealed the effective underlying connectivity. We showed that our trained RNNs are analogous to coupled oscillators and that a sufficient connectivity for phase coding entails two modules: one generating an oscillation and one implementing a coupling function between this oscillation and an external reference one. Two independently generated oscillations that couple during memory are found in the medial temporal lobe (i.e. theta and gamma oscillations) [136, 138, 149]. Our model would predict that coupling functions extracted from neural data [160, 162], recorded from subjects performing working memory tasks, should have a structure that induces multi-stability.

We used trained recurrent neural networks [37, 39, 47]. RNNs can be trained to reproduce both neural activity and behavior [46, 55, 58, 153]. The resulting networks can be understood in terms of their dynamical objects, i.e. the set of attractors [51, 163], and in terms of the trajectories [57, 114] flowing between them. In particular, recurrent models that implement discrete memory, as in this study, often have a separate *static* attractor, or fixed point, for each memory [10, 49, 52, 53], although oscillatory models of memory have also been proposed [164–166]. Here we complement previous work by showing that stable *dynamic* attractors naturally emerge when tasking networks to store memories in the relative phase of oscillations.

Our findings support the notion that rhythmic neural activity can play a supporting role in cognitive phenomena. Brain waves during working memory are widely observed in neural systems of rats, primates, and birds [167]. Phase coding of information relative to theta oscillations has been proposed to be a general coding scheme in the brain [138]. Here, we focused on the coding of two distinct stimuli in phase, which suffices to highlight the dynamics underlying the coding of discrete pieces of information. Both our model and findings can also be extended to coding

for more than two items. We show that a rank-2 model with additional coupling subpopulations can memorise 4 stimuli in Fig. S.8. Phase coding has also been observed during memory of sequences of information, as well as for coding of position [143]. Linking these findings to our model requires further investigation. We note that phase precession, in the sense of a continuous variable (e.g., position in a place field) changing the phase to which a group of neurons locks could be explained by our model through input translating the coupling function, as is tentatively shown in Fig. S.9.

In summary, we used RNNs as trainable dynamical systems to form a hypothesis on how oscillations can be harnessed for neural computation. We proposed that phase-coded memories in recurrent networks reside in stable limit cycles resulting from the coupling of internal and external oscillations.

3.5 MODELS AND METHODS

3.5.1 *Code availability*

All code for obtaining the results and generating the figures is available at: https://github.com/mackelab/phase-limit-cycle-RNNs.

3.5.2 *Training RNNs on a phase-coding task*

3.5.2.1 *Model definition*

As stated in Eq 3.1, we used a continuous time RNN with N units [54],

$$\tau \frac{d\mathbf{x}}{dt} = -\mathbf{x}(t) + \mathbf{J}\tanh(\mathbf{x}(t)) + \mathbf{I}^{(osc)}u(t) + \mathbf{I}^{(s_b)}s_b(t) + \mathbf{I}^{(s_a)}s_a(t) + \boldsymbol{\xi}(t). \tag{3.4}$$

The input weights $\mathbf{I}$ of Eq 3.1 are split into three vectors in $\mathcal{R}^N$: The oscillatory reference input $u(t)$ with input weights $\mathbf{I}^{(osc)}$ and transient stimuli $s_a(t)$ and $s_b(t)$ with weights $\mathbf{I}^{(s_a)}$ and $\mathbf{I}^{(s_b)}$, respectively. To obtain networks with tractable dynamics, we implemented a low-rank constraint according to Eq 3.2. Trained networks consisted of $N = 512$ units, with $\tau = 20$ ms.

3.5.2.2 *Task*

We defined a task in which stimulus identity is to be encoded in the phase of an oscillation. During each trial, of duration $T = 0.8s$, a reference oscillation $u(t)$ with random initial phase θ was provided to the RNN. Trials started with an initial period, with duration drawn uniformly from $[0.125s, 0.25s]$. After the initial period, either stimulus a, $s_a(t)$, or stimulus b, $s_b(t)$, was shown, consisting of a 'pulse' with a constant amplitude of 1 (see Fig. 3.1a). The duration of stimuli was drawn uniformly from $[0.125s, 0.175s]$. For the remainder of the trial, the target $\hat{r}(t)$ was either $\sin(\theta(t) - 0.2\pi)$, when stimulus a was shown ('in-phase' trial), or else $\sin(\theta(t) - 1.2\pi)$ when stimulus b was shown ('anti-phase' trial).

3.5.2.3 *LFP processing*

We used a publicly available data set containing LFP recordings from 3 Long-Evans rats, to obtain a reference oscillation that captures the statistics of ongoing oscillations in biological neural systems [155, 156]. Rats were chasing randomly placed drops of water or food, and neural activity was recorded using multichannel silicon probes inserted in area CA1 of the right dorsal hippocampus. The data contains LFP from 31 to 64 channels, recorded over multiple sessions.

We read the data using Neo [168]. We first re-sampled the data from 1250 Hz to 500 Hz and then high-pass filtered at 7 Hz using a FIR filter with Hamming window and 511 taps [169]. We normalised the signal by dividing the signal channel-wise

with $\sqrt{2}\sigma_{LFP}$, where σ_{LFP} is the channel-wise standard deviation, resulting in the signals having a root mean square equal to a sine wave with amplitude 1.

In order to obtain a unique reference signal for each training trial, we split the recordings for each rat in chunks. In particular, we first split the data into 4 second segments, and picked a random channel for each segment. To extract instantaneous phase of the LFP oscillation for creating training targets, we convolved the signal with complex Morelet wavelets, consisting of the product of a complex sinusoid with a Gaussian with standard deviation $c/2\pi f$. We picked frequencies f from 7 to 9 in steps of 0.2, and set c ('cycles') to 7. For each trial we took the phase (angle) corresponding to the frequency with the highest power. The first second of the signal was discarded to avoid boundary effects. We also discarded a small fraction of trials with artifacts, i.e. trials where the maximum absolute value of the signal was larger than 4 (after normalisation). We ended up with 5708 trials from rat 1, 5669 from rat 2, and 5632 from rat 3.

3.5.2.4 *Training*

We approximated Eq 3.4 using the Euler–Maruyama method with timestep h, giving us for a rank-two network,

$$\begin{aligned}\mathbf{x}_{n+1} =&(1-\frac{h}{\tau})\mathbf{x}_n \\ &+\frac{h}{\tau}(\frac{1}{N}\sum_{i=1}^{2}[\mathbf{m}^{(i)}(\mathbf{n}^{(i)\mathsf{T}}\tanh(\mathbf{x}_n))]+\mathbf{I}^{(osc)}u_n+\mathbf{I}^{(s_b)}s_{b,n}+\mathbf{I}^{(s_a)}s_{a,n}) \\ &+\sqrt{\frac{h}{\tau}}\mathcal{N}(0,2\sigma^2_{noise}).\end{aligned}$$

We use $h = 2$ for training, whereas for the stability analysis and Figures, we take $h = 0.5$. We defined a mean-squared error loss $\mathcal{L}$ between the targets $\hat{r}(t)$ and a linear readout $r(t)$ of the model's activity at time t,

$$\begin{aligned}r(t) &= \frac{1}{N}\mathbf{w}^{\mathsf{T}}\mathbf{x}(t), \\ \mathcal{L} &= \frac{1}{T-T_s}\int_{T_s}^{T}(r(t)-\hat{r}(t))^2dt,\end{aligned} \tag{3.5}$$

where T_s is the time of the stimulus offset and T is the time at the end of a trial. Note that, although we only demand that a weighted sum of the *activity* $\mathbf{x}(t)$ codes for the information, this also causes the *rates* $\tanh(\mathbf{x}(t))$ to phase-code for stimuli (Fig. 3.2a). In principle, one could also directly read-out the rates during training: $r(t) = \frac{1}{N}\mathbf{w}^{\mathsf{T}}\tanh(\mathbf{x}(t))$. Depending on the initialisation and training setup, this can either lead to a rate-coding solution (Fig. 3.2D-F), or a phase-coding solution (see Fig. S.3 for a more in-depth investigation).

We drew initial entries in the connectivity vectors from a zero-mean normal distribution, with a covariance matrix that is the identity matrix, except for an initial

covariance between $\mathbf{m}^{(1)}, \mathbf{n}^{(1)}$ of .6, an initial covariance between $\mathbf{m}^{(2)}, \mathbf{n}^{(2)}$ of 0.6 and a variance for $\mathbf{w}$ of 16 [56]. Note that the initial covariances also have an effect on the solution the model finds (Fig. S.3). We minimised Eq 3.5, by optimising with stochastic gradient descent all entries in $\mathbf{I}^{(osc)}, \mathbf{I}^{(s_a)}, \mathbf{I}^{(s_b)}, \mathbf{m}^{(1)}, \mathbf{m}^{(2)}, \mathbf{n}^{(1)}, \mathbf{n}^{(2)}$, as well as a scalar multiplying the readout weights $\mathbf{w}$. We used the Adam optimizer in Pytorch, with default decay rates (0.9, 0.999) and learning rate 0.01 [123, 170].

Each RNN was trained with LFP data from one rat. Shown in the main text is an RNN trained using LFPs from rat 2, whereas for the analysis of many trained models (Fig. S.3), also a third of the RNNs were trained using data from rat 1, and a third on data from rat 3. Before each training run, we split the LFP into short segments, of which 90% were used as reference signal for the RNN during training trials, and 10% as reference signal during validation trials. The validation and training trials only differ in the portion of the local field potential that was used as reference signal for the network, and in the seed used for generating the randomised stimulus onsets and offsets. For the models shown in the main text, we trained for 50 epochs, in batch sizes of 128, where one epoch denotes all training trials created from the LFP of one rat. Due to the simplicity of the task, all networks we used in the main Figures converged at this point (Fig. S.1), and we did not do any model selection.

Some of the models in the supplementary Figures (Fig. S.2 A,Fig. S.3, Fig. S.8 A), were guided to a phase-coding solution by adding a regularisation term Reg to the loss, which keeps the average firing rates close to 0:

$$\mathsf{Reg} = \frac{1}{N}\sum_{i}^{N}\left(\frac{1}{T-T_s}\int_{T_s}^{T}\mathbf{x}_i(t)dt\right)^2. \tag{3.6}$$

3.5.3 *Analysing dynamics of oscillating low-rank RNNs*

3.5.3.1 *Dimensionality and dynamics of low-rank RNNs with periodic input*

Here, we show that the dynamics of a rank-2 RNN with periodic input lie on a 3 dimensional manifold. To facilitate the analysis, we first consider the dynamics of the network in the absence of transient stimuli ($s_a(t) = s_b(t) = 0$), and no recurrent noise ($\sigma_{noise} = 0$). Furthermore we set $\mathbf{m}^{(1)} \perp \mathbf{m}^{(2)}$, which one can always obtain by singular value decomposition of $\mathbf{J}$, even if during training $\mathbf{m}^{(1)}$ and $\mathbf{m}^{(2)}$ became correlated. We split up $\mathbf{I}^{(osc)}$ in the parts parallel and orthogonal to the $\mathbf{m}$'s,

$$\mathbf{I}^{(osc)} = \mathbf{I}_\perp + \mathbf{m}^{(1)}\alpha_1 + \mathbf{m}^{(2)}\alpha_2.$$

We can then express $\mathbf{x}(t)$ in the orthogonal basis

$$\mathbf{x}(t) = \mathbf{m}^{(1)}\kappa_1(t) + \mathbf{m}^{(2)}\kappa_2(t) + \mathbf{I}_\perp v(t),$$

with $i \in \{1, 2\}$:

$$\kappa_i(t) = \frac{\mathbf{m}^{(i)\mathsf{T}}\mathbf{x}(t)}{\mathbf{m}^{(i)\mathsf{T}}\mathbf{m}^{(i)}},$$
$$v(t) = \frac{\mathbf{I}_\perp^\mathsf{T}\mathbf{x}(t)}{\mathbf{I}_\perp^\mathsf{T}\mathbf{I}_\perp}.$$

Here $v(t)$ is the reference oscillation filtered by the model's time constant τ. Using $*$ to denote convolution,

$$v(t) = \frac{1}{\tau}u(t) * e^{-\frac{t}{\tau}} + v(0)e^{-\frac{t}{\tau}}.$$

As both $u(t)$ and $v(t)$ explicitly depend on time, we first obtain the non-autonomous dynamical system $\mathbf{F}$,

$$\frac{d\kappa_1}{dt}, \frac{d\kappa_2}{dt} = \mathbf{F}(t, \kappa_1, \kappa_2), \tag{3.7}$$

with

$$\tau\frac{d\kappa_i}{dt} = -\kappa_i(t) + \frac{1}{N}\mathbf{n}^{(i)\mathsf{T}}\tanh(\mathbf{x}(t)) + \alpha_i u(t). \tag{3.8}$$

Next, we show that for periodic input this system can be considered as a three dimensional autonomous dynamical system. We introduce the variable $\theta = wt \mod 2\pi$ such that

$$\frac{d\theta}{dt} = \omega.$$

We take $u(t) = \sin(\theta)$ and we can now find the closed form solution for v,

$$v(t) = \frac{1}{\sqrt{(w\tau)^2 + 1}}\sin(\theta - \arctan(w\tau)) + ce^{-\frac{t}{\tau}}.$$

Here, the last term on the right hand side will decay to 0. Practically, we get rid of this dependence on t, by taking appropriate $v(0)$ such that $c = 0$ (or by assuming the simulation has run for a little amount of time).

Following this definition, both u and v are functions of θ, and we can consider the autonomous dynamical system with coordinates $\kappa_1, \kappa_2, \theta$,

$$\frac{d\kappa_1}{dt}, \frac{d\kappa_2}{dt}, \frac{d\theta}{dt} = \mathbf{G}(\kappa_1, \kappa_2, \theta).$$

3.5.3.2 *Coordinate system for phase space figures*

To create the phase space figures, we applied the following coordinate transform, where $\tilde{x}, \tilde{y}, \tilde{z}$ denote the coordinates in the figure,

$$\tilde{x} = \cos(\theta)(\tilde{r} - \kappa_1),$$
$$\tilde{y} = \sin(\theta)(\tilde{r} - \kappa_1),$$
$$\tilde{z} = \kappa_2.$$

Given an initial condition for κ_1, i.e. $\kappa_1(0)$, one can always pick an $\tilde{r}$ such that this is an injective map for all t. This requires $\tilde{r} < \kappa_1(t)$ for all t (otherwise trajectories would cross at the center). From Eq 3.8, we can see that $\kappa_1(t) > \frac{1}{N}|\mathbf{n}^{(1)}|_1 + |\alpha_1| \implies \frac{d\kappa_1}{dt} < 0$, thus if we pick $\tilde{r} > \kappa_1(0) \geq \frac{1}{N}|\mathbf{n}^{(1)}|_1 + |\alpha_1|$, the requirement is satisfied.

3.5.3.3 *Poincaré maps and linear stability analysis*

We used Poincaré maps to study the stability of limit cycles [34, 108]. We took a cross section $Q = \{(\kappa_1, \kappa_2, \theta) : \mod \theta = 2\pi\}$, and created the iterative map from Q to itself ($Q_0 \to Q_{2\pi}$),

$$\boldsymbol{\kappa}_{c+1} = \mathbf{P}(\boldsymbol{\kappa}_c),$$

where $\boldsymbol{\kappa_c} = \kappa_{1,c}, \kappa_{2,c}$ corresponds to the c'th intersection. Limit cycles correspond to fixed points $\boldsymbol{\kappa}^*$ for which

$$\boldsymbol{\kappa}^* = \mathbf{P}(\boldsymbol{\kappa}^*).$$

To study the stability of limit cycles we see what happens to a small perturbation η_0 to $\boldsymbol{\kappa}^*$. Applying a Taylor expansion around $\boldsymbol{\kappa}^*$, we have after one cycle,

$$\boldsymbol{\kappa}^* + \eta_1 = P(\boldsymbol{\kappa}^*) + D_{\boldsymbol{\kappa}}\mathbf{P}(\boldsymbol{\kappa}^*)\eta_0 + h.o.t.,$$

where $D_{\boldsymbol{\kappa}}$ denotes the gradient operator, the partial derivatives with respect to $\boldsymbol{\kappa}$. $D_{\boldsymbol{\kappa}}\mathbf{P}(\boldsymbol{\kappa}^*)$ is called the linearised Poincaré map at $\boldsymbol{k}^*$. To first order, perturbations scale proportional to the norm of its eigenvalues λ, called the Floquet (or characteristic) multipliers. To see this we can express η_0 as a linear combination of eigenvectors $\mathbf{e}$ of $D_{\boldsymbol{\kappa}}\mathbf{P}(\boldsymbol{\kappa}^*)$ (for some scalars v) and get the following expression for the perturbation after c cycles,

$$\eta_c = \sum_{i=1}^{2} v_i \mathbf{e}_i \lambda_i^c.$$

We now show how to obtain $D_{\boldsymbol{\kappa}}\mathbf{P}(\boldsymbol{\kappa}^*)$. Let $\psi(t, \boldsymbol{\kappa_0})$ denote the flow, a mapping from $(t, \boldsymbol{\kappa})$ to $\boldsymbol{\kappa}$, obtained by integrating $\mathbf{F}(t, \boldsymbol{\kappa})$ (Eq 3.7) for duration t, with initial conditions $\boldsymbol{\kappa_0}$. The Poincaré map then corresponds to [171]

$$\begin{aligned} \mathbf{P}(\boldsymbol{\kappa}_c) &= \psi(\mathcal{T}, \boldsymbol{\kappa_c}) \\ &= \int_0^{\mathcal{T}} \mathbf{F}(0, \psi(t, \boldsymbol{\kappa_c}))dt + \psi(0, \boldsymbol{\kappa_c}), \end{aligned} \tag{3.9}$$

where the second line is obtained by applying the fundamental theorem of calculus. $\mathcal{T}$ denotes one period of the reference oscillation. Defining $\mathbf{M}(t)$ as $D_{\boldsymbol{\kappa}}\psi(t, \boldsymbol{\kappa}_c)$, one can obtain a variational equation for the linearized Poincaré map,

$$\begin{aligned} D_{\boldsymbol{\kappa}}\mathbf{P}(\boldsymbol{\kappa}_c) &= \int_0^{\mathcal{T}} D_{\psi}\mathbf{F}(0, \psi(t, \boldsymbol{\kappa}_c)) D_{\boldsymbol{\kappa}_c}\psi(t, \boldsymbol{\kappa}_c)dt + \mathbf{I} \\ &= \mathbf{M}(\mathcal{T}) \end{aligned}$$

with

$$\frac{d\mathbf{M}(t)}{dt} = D_{\psi}\mathbf{F}(t, \psi(t, \boldsymbol{\kappa}_c))\mathbf{M}(t).$$

Here, $\mathbf{M}(0) = \mathbf{I}$. $\mathbf{M}(t)$ is called the circuit or monodromy matrix. Since we can not obtain the circuit matrix analytically we approximated $\mathbf{M}(\mathcal{T})$ using the Euler method with timestep h.

3.5.4 *Coupled oscillators and population structure*

3.5.4.1 *Extracting the coupling function*

We can rewrite the internal dynamics of the RNN in polar coordinates with

$$\kappa_1 = r\cos(\phi), \kappa_2 = r\sin(\phi).$$

We extracted the coupling function g,

$$\tau\frac{d\phi}{dt} = \frac{1}{r^2}\left(\frac{1}{N}(\kappa_1\mathbf{n}^{(2)} - \kappa_2\mathbf{n}^{(1)})^{\mathsf{T}}\tanh(\mathbf{x}) + (\kappa_1\alpha_2 - \kappa_2\alpha_1)u(\theta)\right),$$
$$g(\theta, \phi) = \frac{d\phi}{dt} - \omega.$$

Note that the radius of the two stable limit cycles might not be exactly constant, or equal to each other, so for Fig. 3.3B and Fig. 3.3C we took for r the mean radius over the two cycles.

3.5.4.2 *Reduced models*

In order to find a simplified description of the dynamics of our models we assumed entries in the weight vectors of our network are N samples, indexed by j, from a probability distribution $p(\mathbf{y})$ [49, 56, 59, 102]. To keep the equations brief, we assume one input v here, which can be straightforwardly extended to multiple inputs. Then we have

$$\mathbf{I}_{\perp j}, \mathbf{n}_j^{(1)}, \mathbf{n}_j^{(2)}, \mathbf{m}_j^{(1)}, \mathbf{m}_j^{(2)} \sim p(\mathbf{y}),$$
$$p(\mathbf{y}) = p(I, n^{(1)}, n^{(2)}, m^{(1)}, m^{(2)}).$$

We can then see Eq 3.8 as a sampling estimator for the recurrent input, which as $N \to \infty$ approaches the expectation,

$$\tau\frac{d\kappa_i}{dt} \overset{N\to\infty}{=} -\kappa_i + \mathbb{E}_{p(\mathbf{y})}[n^{(i)}\tanh(\kappa_1 m^{(1)} + \kappa_2 m^{(2)} + v_1 I)].$$

We assume $p(\mathbf{y})$ is a mixture of L zero-mean Gaussians [56],

$$p(\mathbf{y}) = \sum_l^L w_l \mathcal{N}(0, \Sigma_l).$$

We can then obtain a mean-field description for the dynamics (see previous studies [49, 56, 59, 102] for a derivation). Here the effective connectivity consists of the covariances of the mixture components, modulated by a nonlinear 'gain' function (which approaches 0 when the non-linearity saturates),

$$\tau \frac{d\kappa_i}{dt} = -\kappa_i + \sum_{l}^{L} w_l (\kappa_1 \sigma^{(l)}_{m^{(1)}n^{(i)}} + \kappa_2 \sigma^{(l)}_{m^{(2)}n^{(i)}} + v\sigma^{(l)}_{In^{(i)}}) \mathbb{E}_{p(z)}[\tanh'(\sqrt{\Delta^{(l)}}z)],$$

with $\Delta^{(l)} = (\sigma^{(l)}_{m^{(1)}}\kappa_1)^2 + (\sigma^{(l)}_{m^{(2)}}\kappa_2)^2 + (\sigma^{(l)}_{I}v)^2$ and $z = \mathcal{N}(0,1)$.

Instead of numerically approximating the expectation as in previous studies [49, 56], we substitute $\mathrm{erf}(\frac{\sqrt{\pi}}{2}x)$ for $\tanh(x)$ and analytically obtain a simpler approximation.

$$\begin{aligned}
\mathbb{E}_{p(z)}[\tanh'(\sqrt{\Delta}z)] &\approx \mathbb{E}_{p(z)}[\mathrm{erf}'(\frac{\sqrt{\pi\Delta}}{2}z)] \\
&\approx \mathbb{E}_{p(z)}[e^{-\frac{\pi\Delta}{4}z^2}] \\
&\approx \frac{1}{\sqrt{1+\frac{\pi}{2}\Delta}}.
\end{aligned}$$

This gives us the following mean-field description for the rank-2 RNNs,

$$\begin{aligned}
\tau \frac{d\kappa_i}{dt} &= -\kappa_i + \sum_{l}^{L} w_l (\kappa_1 \sigma^{(l)}_{m^{(1)}n^{(i)}} + \kappa_2 \sigma^{(l)}_{m^{(2)}n^{(i)}} + v\sigma^{(l)}_{In^{(i)}}) \frac{1}{\sqrt{1+\frac{\pi}{2}\Delta^{(l)}}}, \\
\Delta^{(l)} &= (\sigma^{(l)}_{m^{(1)}}\kappa_1)^2 + (\sigma^{(l)}_{m^{(2)}}\kappa_2)^2 + (\sigma^{(l)}_{I}v)^2.
\end{aligned} \tag{3.10}$$

3.5.4.3 *Connectivity for mean-field model*

By writing Eq 3.10 in polar coordinates we can see that the total change in phase of the model is the sum of the change in phase of its populations,

$$\tau \frac{d\phi(t)}{dt} = \sum_{l}^{L} w_l \frac{d\phi^{(l)}(t)}{dt}. \tag{3.11}$$

Based on reverse engineering the dynamics and connectivity of our trained networks, we can now choose covariances in order to design a model that approximates the coupled oscillator equations (Eq 3.3). We here assume for all populations l, $\sigma^{(l)}_{m^{(2)}} = \sigma^{(l)}_{m^{(1)}}$ (denoted as $\sigma^{(l)}_{m}$).

We first initialize a population that generates oscillations with angular velocity ω. We set this population unconnected to the input ($\sigma^{(i)}_{I} = 0$ for all i). When taking $\sigma_{m^{(2)}n^{(1)}} = -\sigma_{m^{(1)}n^{(2)}}$ and $\sigma_{m^{(1)}n^{(1)}} = \sigma_{m^{(2)}n^{(2)}}$ [49, 59, 164] this populations produces oscillations, with constant frequency,

$$\frac{d\phi^{(p1)}(t)}{dt} = \sigma^{(p_1)}_{n^{(1)}m^{(2)}} \frac{1}{\sqrt{1+\frac{\pi}{2}\Delta^{(p_1)}}},$$
$$\Delta^{(p1)} = \sigma^{2(p_1)}_{m} r^2.$$

Next, we create two populations that together implement the coupling function,

$$g(\phi,\theta) \propto \cos(2\phi)\frac{1}{\sqrt{1+\frac{\pi}{2}\Delta}}, \qquad \Delta = \sigma^2_m r^2 + \sigma^2_{I^{(osc)}} v(\theta)^2. \tag{3.12}$$

Based on Fig. 3.3B, it seems reasonable to assume that the bistable dynamics stem from $\sin(2\phi), \cos(2\phi)$ and $\sin(2\theta), \cos(2\theta)$ terms, which we obtained by setting $\sigma_{n^{(1)}m^{(2)}} = \sigma_{n^{(2)}m^{(1)}}$ (or $\sigma_{n^{(1)}m^{(1)}} = -\sigma_{n^{(2)}m^{(2)}}$), and $\sigma_{I^{(osc)}} \neq 0$.

In order to get the stimulus-induced bifurcation observed in Fig. 3.2C,Fig. 3.3C, we, additionally set the following connectivity for the input to the coupling populations, $\sigma^{(p_2)}_{I^{(s_a)}} \neq 0, \sigma^{(p_3)}_{I^{(s_b)}} \neq 0, \sigma^{(p_2)}_{I^{(osc)}n^{(1)}} = -\sigma^{(p_2)}_{I^{(osc)}n^{(2)}} = -\sigma^{(p_3)}_{I^{(osc)}n^{(1)}} = \sigma^{(p_3)}_{I^{(osc)}n^{(2)}}$. Then the equations for the coupling populations read,

$$\frac{d\phi^{(p2)}(t)}{dt} = \left(\frac{\sqrt{2}}{r}\sigma^{(p_2)}_{n^{(1)}I^{(osc)}} \sin(\phi+\frac{1}{4})v(\theta) + \sigma^{(p_2)}_{n^{(1)}m^{(2)}}\cos(2\phi)\right)\frac{1}{\sqrt{1+\frac{\pi}{2}\Delta^{(p_2)}}},$$
$$\Delta^{(p_2)} = \sigma^{2(p_2)}_{m} r^2 + \sigma^{2(p_2)}_{I^{(osc)}} v(\theta)^2 + \sigma^{2(p_2)}_{I^{(s_a)}} s_a(t)^2,$$
$$\frac{d\phi^{(p3)}(t)}{dt} = \left(\frac{\sqrt{2}}{r}\sigma^{(p_3)}_{n^{(2)}I^{(osc)}} \sin(\phi-\frac{3}{4})v(\theta) + \sigma^{(p_3)}_{n^{(1)}m^{(2)}}\cos(2\phi)\right)\frac{1}{\sqrt{1+\frac{\pi}{2}\Delta^{(p_3)}}},$$
$$\Delta^{(p_3)} = \sigma^{2(p_3)}_{m} r^2 + \sigma^{2(p_3)}_{I^{(osc)}} v(\theta)^2 + \sigma^{2(p_3)}_{I^{(s_b)}} s_b(t)^2.$$

When both s_a and s_b are zero, the $\sin(\phi)$ terms cancel out and we retrieve the desired coupling function (Eq 3.12; where the $\cos(2\phi)$ terms dominate). When either stimulus a or b is on, population 2 or 3 is inhibited, respectively (the non-linearities of the units in this population saturate), causing the term on the right to go to zero. Then the $\sin(\phi)$ term of the population that is unaffected by the stimulus takes over the coupling function, mimicking what we saw in Fig. 3.3B and Fig. 3.3C. For exact values of the parameters, see Fig. S.7.

3.6 ACKNOWLEDGMENTS

We thank Richard Gao, Kabir Dabholkar and Stefanie Liebe for insightful discussions, Pedro J. Gonçalves, Elia Turner and Zinovia Stefanidi for feedback on the manuscript, and the International Max Planck Research School for Intelligent Systems (IMPRS-IS) for supporting MP.

This work was supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) through SFB 1089 (Project-ID 227953431 to JHM) and Excellence Strategy (EXC number 2064/1 – 390727645 to JHM), the German Federal Ministry of Education and Research (BMBF) through the Tübingen AI Center (FKZ01IS18039A to JHM) and the ISRAEL SCIENCE FOUNDATION (grant No. 1442/21 to OB) and HFSP research grant (RGP0017/2021 to OB). The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

4

INFERRING STOCHASTIC LOW-RANK RECURRENT NEURAL NETWORKS FROM NEURAL DATA

A central aim in computational neuroscience is to relate the activity of large populations of neurons to an underlying dynamical system. Models of these neural dynamics should ideally be both interpretable and fit the observed data well. Low-rank recurrent neural networks (RNNs) exhibit such interpretability by having tractable dynamics. However, it is unclear how to best fit low-rank RNNs to data consisting of noisy observations of an underlying stochastic system. Here, we propose to fit stochastic low-rank RNNs with variational sequential Monte Carlo methods. We validate our method on several datasets consisting of both continuous and spiking neural data, where we obtain lower dimensional latent dynamics than current state of the art methods. Additionally, for low-rank models with piecewise-linear nonlinearities, we show how to efficiently identify all fixed points in polynomial rather than exponential cost in the number of units, making analysis of the inferred dynamics tractable for large RNNs. Our method both elucidates the dynamical systems underlying experimental recordings and provides a generative model whose trajectories match observed variability.

4.1 DECLARATION

This chapter is based on: Matthijs Pals, A Erdem Sağtekin, Felix Pei, Manuel Gloeckler, Jakob H Macke. Inferring stochastic low-rank recurrent neural networks from neural data. *The Thirty-eighth Annual Conference on Neural Information Processing Systems (NeurIPS)* (2024).

Author contributions

The method for fitting RNNs was developed by me with insights from Jakob H Macke and Manuel Gloeckler. I implemented the method and obtained the result on the number of fixed points in piecewise-linear RNNs. My initial proof was formalized and written up with assistance from Manuel Gloeckler, who also provided Fig S.11 and Fig S.12. Erdem Sağtekin performed initial tests on spiking data, applied the method to rat hippocampal recordings, and generated Fig. 4.5, 4.6, S.18, S.19 and S.20. Felix Pei applied the method to macaque motor cortex data and generated Fig. 4.7, S.17 and S.21. All other figures and experiments were done by me.

CRediT

Conceptualization: Matthijs Pals, Manuel Gloeckler, Jakob H Macke
Formal analysis: Matthijs Pals, Manuel Gloeckler
Funding acquisition: Jakob H Macke
Software: Matthijs Pals, A Erdem Sağtekin, Felix Pei
Supervision: Jakob H Macke
Writing – original draft: Matthijs Pals, Manuel Gloeckler, A Erdem Sağtekin, Felix Pei
Writing – review & editing: Jakob H Macke

4.2 INTRODUCTION

A common goal of many scientific fields is to extract the dynamical systems underlying noisy experimental observations. In particular, in neuroscience, much work is devoted to understanding the coordinated firing of neurons as being implemented through underlying dynamical systems [33, 35–38]. Recurrent neural networks (RNNs) constitute a common model class of neural dynamics [46, 62, 64–69] which can be reverse-engineered to form hypotheses about neural computations [47, 51]. As a result, several recent research directions have centered on interpretable or analytically tractable RNN architectures. In particular, RNNs with low-rank structure [49, 56, 59, 60, 101, 102, 172] admit a direct mapping between high-dimensional population activity and an underlying low-dimensional dynamical system. RNNs with piecewise-linear activations [64, 65, 70, 110–112] are tractable, as they have fixed points and cycles that can be accessed analytically.

To serve as useful models of brain activity, it is important that models also capture the observed brain activity, including trial-to-trial variability. Many methods that fit RNNs to data are restricted to RNNs with deterministic transitions [46, 62, 64, 66–68]. It is unlikely that, in general, all variability in the data can be explained by variability in the RNNs initial state. Thus, adopting stochastic transitions is imperative. While probabilistic sequence models are used effectively in neuroscience [31], they have so far largely consisted of state space models without an obvious mechanistic interpretation [40–43, 133].

Here, we demonstrate that we can fit large stochastic RNNs to noisy high-dimensional data. First, we show that, by combining variational sequential Monte Carlo methods [115–117] with low-rank RNNs, we can efficiently fit stochastic RNNs with many units by learning the underlying low-dimensional dynamical system. The resulting RNNs are generative models of neural data that can be used to sample trajectories of arbitrary length, and also allow for conditional generation with (both time-varying and stationary) inputs. Second, we show that, for low-rank networks with piecewise-linear activation functions, the resulting dynamics can be efficiently analyzed: In particular, we show how *all* fix points can be found with a polynomial cost in the number of units — dramatically more efficient than the exponential cost in the general case.

We first validate our method using several teacher-student setups and show that we recover both the ground truth dynamics and stochasticity. We then fit our model to several real-world datasets, spanning both spiking and continuous data, where we obtain a generative model which needs lower dimensional latent dynamics than current state of the art methods. We also demonstrate how in our low-rank RNNs fixed points can be efficiently inferred — potentially at a lower cost than approximate methods [70], while additionally coming with the guarantee that *all* fixed points are found.

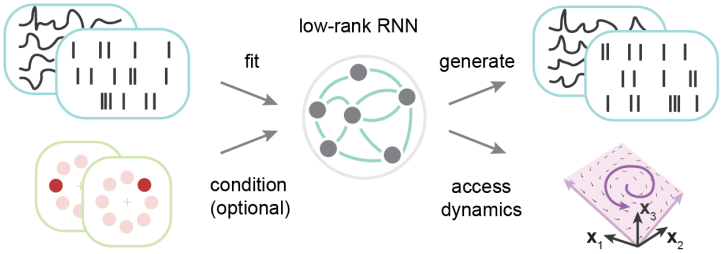

FIGURE 4.1: Our goal is to obtain generative models from which we can sample realistic neural data while having a tractable underlying dynamical system. We achieve this by fitting stochastic low-rank RNNs with variational sequential Monte Carlo.

4.3 THEORY AND METHODS

4.3.1 *Low-rank RNNs*

4.3.1.1 *Access to the low-dimensional dynamics underlying large networks*

Our goal is to infer recurrent neural network models of the form

$$\tau \frac{d\mathbf{x}}{dt} = -\mathbf{x}(t) + \mathbf{J}\phi(\mathbf{x}(t)) + \Gamma_{\mathbf{x}}\xi(t), \tag{4.1}$$

with neuron activity $\mathbf{x}(t) \in \mathbb{R}^N$, time-constant $\tau \in \mathbb{R}_{>0}$, recurrent weights $\mathbf{J} \in \mathbb{R}^{N\times N}$, element-wise nonlinearity ϕ, an R dimensional white noise process $\xi(t)$ and $\Gamma_{\mathbf{x}} \in \mathbb{R}^{N\times R}$. In particular, we are interested in the case where the weight matrix $\mathbf{J}$ has rank $R \leq N$, i.e., it can be written as $\mathbf{J} = \mathbf{M}\mathbf{N}^{\mathsf{T}}$, with $\mathbf{M}, \mathbf{N} \in \mathbb{R}^{N\times R}$ ([49, 56, 59, 102]). Assuming that $\mathbf{x}(0)$ lies in the subspace spanned by the columns of $\mathbf{M}$ and $\Gamma_{\mathbf{x}} = \mathbf{M}\Gamma_{\mathbf{z}}$, with $\Gamma_{\mathbf{z}} \in \mathbb{R}^{R\times R}$, we can rewrite Eq. 4.1 as an equivalent R dimensional system,

$$\tau \frac{d\mathbf{z}}{dt} = -\mathbf{z}(t) + \mathbf{N}^{\mathsf{T}}\phi(\mathbf{M}\mathbf{z}(t)) + \Gamma_{\mathbf{z}}\xi(t), \tag{4.2}$$

where we can switch between Eq. 4.1 and Eq. 4.2 by means of linear projection, $\mathbf{z}(t) = (\mathbf{M}^{\mathsf{T}}\mathbf{M})^{-1}\mathbf{M}^{\mathsf{T}}\mathbf{x}(t)$ and $\mathbf{x}(t) = \mathbf{M}\mathbf{z}(t)$. Note that we can directly extend these equations to include input, representing, e.g., experimental stimuli or context. Even if these stimuli are time-varying, $\mathbf{x}$ will be constrained to the span of the input weights and $\mathbf{M}$. By including input to the RNN, we can use the fit models for conditional generation (see Supplement .3.3.2).

4.3.1.2 *Low-rank RNNs as state space models*

We consider nonlinear latent dynamical systems with observations $\mathbf{y}_t$:

$$
\begin{aligned}
p(\mathbf{z}_{1:T}, \mathbf{y}_{1:T}) &= p(\mathbf{z}_1) \prod_{t=2}^{T} p(\mathbf{z}_t \mid \mathbf{z}_{t-1}) \prod_{t=1}^{T} p(\mathbf{y}_t \mid \mathbf{z}_t), \\
p(\mathbf{z}_t \mid \mathbf{z}_{t-1}) &= \mathcal{N}(F(\mathbf{z}_{t-1}), \Sigma_{\mathbf{z}}), \; p(\mathbf{z}_1) = \mathcal{N}(\mu_{\mathbf{z}_1}, \Sigma_{\mathbf{z}_1}), \\
p(\mathbf{y}_t \mid \mathbf{z}_t) &= G(\mathbf{z}_t),
\end{aligned}
$$

where the transition distribution is parameterised by discretising a low-rank RNN with timestep Δ_t, we have mean $F(\mathbf{z}_t) = a\mathbf{z}_t + \tilde{\mathbf{N}}^{\mathsf{T}} \phi(\mathbf{M}\mathbf{z}_t)$,

with $a = 1 - \frac{\Delta_t}{\tau}$ and $\tilde{\mathbf{N}} = \frac{\Delta_t}{\tau}\mathbf{N}$, and covariance $\Sigma_{\mathbf{z}}$ (see Supplement .3.3.1). The specific form of the observation function G, depends on the data-modality, e.g., here we use a Poisson distribution for count observations. This formulation allows one to keep the one-to-one correspondence between RNN units (or a subset of those) and recorded data neurons, (as was desired in previous work, e.g., [66–68, 173]). For example, assuming Gaussian observation noise, we can simply use that $\mathbf{x}_t = \mathbf{M}\mathbf{z}_t$ and define $G = \mathcal{N}(\mathbf{M}\mathbf{z}_t, \Sigma_{\mathbf{y}})$.

Once we learn $p(\mathbf{z}_{1:T}, \mathbf{y}_{1:T})$, we can use the obtained RNN as a generative model to sample trajectories, and reverse engineer the underlying dynamics to gain insight in the data generation process. Given the sequential structure of the RNN, we can do model learning by using variational sequential Monte Carlo (also called Particle Filtering) methods [115–117].

4.3.2 *Model learning with variational sequential Monte Carlo*

4.3.2.1 *Sequential Monte Carlo*

Sequential Monte Carlo (SMC) can be used to approximate sequences of distributions, such as those generated by our RNN, with a set of K trajectories of latents $\mathbf{z}_{1:T}$ (commonly called particles) [120]. As we do not have direct access to the posterior $\mathrm{p}(\mathbf{z}_t \mid \mathbf{y}_{1:t})$, we instead sample from a proposal distribution r, and adjust for the discrepancy between the proposal and target posterior distribution using importance weights. Thus, a crucial choice when doing SMC is picking the right proposal distribution r, from which we can sample latents conditioned on the previous latent $\mathbf{z}_{t-1}$ and observed data $\mathbf{y}_{1:T}$, or a subset of those. Given initial samples $\mathbf{z}_1^{1:K} \sim r$

and corresponding importance weights $\overline{w}_1^{1:K}$ (as defined below) SMC progresses by repeatedly executing the following steps:

$$\begin{aligned} &\textit{resample} && a_{t-1}^k \sim \mathsf{Discrete}(a_{t-1}^k \mid \overline{w}_{t-1}^k), \\ &\textit{propose} && \mathbf{z}_t^k \sim r(\mathbf{z}_t^k \mid \mathbf{y}_t, \mathbf{z}_{t-1}^{a_{t-1}^k}), \\ &\textit{reweight} && w_t^k = \frac{p(\mathbf{y}_t, \mathbf{z}_t^k \mid \mathbf{z}_{t-1}^{a_{t-1}^k})}{r(\mathbf{z}_t^k \mid \mathbf{y}_t, \mathbf{z}_{t-1}^{a_{t-1}^k})}, \end{aligned}$$

with $\overline{w}_t^k = \frac{w_t^k}{\sum_{j=1}^K w_t^j}$. Here, the resampling step avoids most of the weights from concentrating on very few particles. Using SMC, we obtain, at time t, a filtering approximation to the posterior,

$$q_{\mathsf{filt}}(\mathbf{z}_{1:t} \mid \mathbf{y}_{1:t}) = \sum_{k=1}^K \overline{w}_t^k \delta(\mathbf{z}_{1:t}^k). \tag{4.3}$$

The unnormalised weights give an unbiased estimate to the marginal likelihood,

$$\hat{p}(\mathbf{y}_{1:T}) = \prod_{t=1}^T \frac{1}{K} \sum_{k=1}^K w_t^k. \tag{4.4}$$

We now detail how we pick the proposal distribution r. For linear Gaussian observations $G = \mathcal{N}(\mathbf{W}\mathbf{z}_t, \Sigma_\mathbf{y})$, we set $r(\mathbf{z}_t \mid \mathbf{y}_t, \mathbf{z}_{t-1}) = p(\mathbf{z}_t \mid \mathbf{y}_t, \mathbf{z}_{t-1})$, as this is available in closed form and is optimal (in the sense that it minimizes the variance of the importance weights [120])

$$r(\mathbf{z}_t \mid \mathbf{y}_t, \mathbf{z}_{t-1}) = \mathcal{N}((\mathbf{I} - \mathbf{K}\mathbf{W})F(\mathbf{z}_{t-1}) + \mathbf{K}\mathbf{y}_t, \Lambda_\mathbf{z}) \tag{4.5}$$

with $\mathbf{K}$ the Kalman Gain: $\mathbf{K} = \Lambda_\mathbf{z}\mathbf{W}^\mathsf{T}\Sigma_\mathbf{y}^{-1}$, and $\Lambda_\mathbf{z} = (\Sigma_\mathbf{z}^{-1} + \mathbf{W}^\mathsf{T}\Sigma_\mathbf{y}^{-1}\mathbf{W})^{-1}$ (or equivalently $\mathbf{K} = \Sigma_\mathbf{z}\mathbf{W}^\mathsf{T}(\mathbf{W}\Sigma_\mathbf{z}\mathbf{W}^\mathsf{T} + \Sigma_\mathbf{y})^{-1}$, and $\Lambda_\mathbf{z} = (\mathbf{I} - \mathbf{K}\mathbf{W})\Sigma_\mathbf{z}$). For non-linear observations, we can not invert the observation process in closed form, so we instead jointly optimize a parameterized 'encoding' distribution $e(\mathbf{z}_t \mid \mathbf{y}_{t-t':t})$ (as in a variational autoencoder [123]). In particular, we assume e to be a multivariate normal with diagonal covariance, which we parameterize by a causal convolutional neural network, such that each latent is conditioned on the t' latest observations (although sometimes non-causal encoders can be advantageous, see Supplement .3.2.5). We then use the following proposal:

$$r(\mathbf{z}_t \mid \mathbf{z}_{t-1}, \mathbf{y}_{t-t':t}) \propto e(\mathbf{z}_t \mid \mathbf{y}_{t-t':t})p(\mathbf{z}_t \mid \mathbf{z}_{t-1}), \tag{4.6}$$

where we now also assume $p(\mathbf{z}_t \mid \mathbf{z}_{t-1})$ has a diagonal covariance matrix.

4.3.2.2 *Relationship to Generalised Teacher Forcing*

In our approach, the mean of the proposal distribution at time t is a linear combination between the RNN predicted state $F(\mathbf{z}_{t-1})$ and a data-inferred state $\hat{\mathbf{z}}_t$. A recent study obtained state-of-the art results for reconstructing dynamical systems by fitting deterministic RNNs with a method called Generalised Teacher Forcing (GTF), which similarly linearly interpolates between a data-inferred and an RNN predicted state at every time-step [64]; the model propagates forward in time as $\mathbf{z}_t = (1-\alpha)F(\mathbf{z}_{t-1}) + \alpha\hat{\mathbf{z}}_t$. Hess *et al.* [64] showed that by choosing the appropriate α, one can completely avoid exploding gradients, while still allowing backpropagation through time, and thus obtaining long-term stable solutions [100]. The optimal α can be picked based on the maximum Lyaponuv exponent of the system (a measure of how fast trajectories diverge in a chaotic system).

By including the RNN in the proposal distribution, we similarly to GTF allow backpropagation through time through the sampled trajectories. The linear combination is given by $\alpha = (\Sigma_{\mathbf{z}}^{-1} + \mathbf{W}^{\mathsf{T}}\Sigma_{\mathbf{y}}^{-1}\mathbf{W})^{-1}\mathbf{W}^{\mathsf{T}}\Sigma_{\mathbf{y}}^{-1}\mathbf{W}$ in Eq. 4.5, and similarly in Eq. 4.6 by $\alpha = (\Sigma_{\mathbf{z}}^{-1} + \Sigma_{\hat{\mathbf{z}}_t}^{-1})^{-1}\Sigma_{\hat{\mathbf{z}}_t}^{-1}$, where $\Sigma_{\hat{\mathbf{z}}_t}$ is the predicted variance of the encoding network. Thus, instead of interpolating based on an estimate of how chaotic the system is, our approach combines RNN and data inferred states adaptively (every time step, if Eq. 4.6 is used) based on how relatively noisy the transition distribution is with respect to the data-inferred states at time t, analogous to, e.g., the gain of a Kalman filter. In the formulation of GTF of Hess *et al.* [64], an invertable observation model is required. By learning an encoder that predicts a distribution over latents, our method naturally extends to models with non-invertable (e.g., Poisson) observations.

4.3.2.3 *Variational objective*

We can fit our RNNs to data by using SMC to specify a variational objective [115–117]. In variational inference, we specify a family of parameterized distributions Q, and optimize those parameters such that a divergence (usually the KL divergence) between the variational distribution $q(\mathbf{z}_{1:T}) \in Q$ and the true posterior $p(\mathbf{z}_{1:T} \mid \mathbf{y}_{1:T})$ is minimized. We do this by maximising a lower bound (ELBO) to the log likelihood $p(\mathbf{y}_{1:T})$. In particular, we can use Eq. 4.4 to specify the ELBO objective [115–117]

$$\mathcal{L} = \mathbb{E}_{q_{\mathsf{smc}}(\mathbf{z}_{1:T}^{1:K}, a_{1:T-1}^{1:K} | \mathbf{y}_{1:T})}[\log \hat{p}(\mathbf{y}_{1:T})], \tag{4.7}$$

with q_{smc} the sampling distribution:

$q_{\mathsf{smc}}(\mathbf{z}_{1:T}^{1:K}, a_{1:T-1}^{1:K} \mid \mathbf{y}_{1:T}) = \prod_{k=1}^{K} r(\mathbf{z}_1^k \mid \mathbf{y}_1) \prod_{k=1}^{K} \prod_{t=2}^{T} r(\mathbf{z}_t^k \mid \mathbf{z}_{t-1}^{a_{t-1}^k} \mathbf{y}_t) \mathsf{Discrete}(a_{t-1}^k \mid \overline{w}_{t-1}^k)$.

During each training iteration, we run SMC, using the closed form optimal proposal (Eq. 4.5) if observations are linear Gaussian, otherwise the proposal includes a parameterised encoder (Eq. 4.6). We can then use the resulting unnormalised importance weights (Eq. 4.4) to estimate the ELBO, which we maximise with backpropagation (through time). As suggested in previous studies [115–117, 122], we

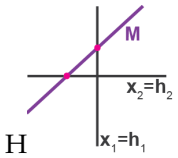

FIGURE 4.2: Proof sketch.

use biased gradients during optimization by dropping high-variance terms arising from the resampling.

4.3.3 *Finding fixed points in piecewise-linear low-rank RNNs*

After having learned our model, we can gain insight into the mechanisms underlying the data generation process by reverse engineering the learned dynamics [51], e.g., by calculating their fixed points. Here, we show that the fixed points can be found analytically and efficiently for low-rank networks with piecewise-linear activation functions $\phi(\mathbf{x}_i) = \sum_d^D \mathbf{b}_i^{(d)} \max(\mathbf{x}_i - \mathbf{h}_i^{(d)}, 0)$ includes, e.g., the standard ReLU ($\phi(\mathbf{x}_i) = \max(\mathbf{x}_i - \mathbf{h}_i, 0)$) or the 'clipped' variant ($\phi(\mathbf{x}_i) = \max(\mathbf{x}_i + \mathbf{h}_i, 0) - \max(\mathbf{x}_i, 0)$) [64] which we used in all experiments with real-world data here.

Naively, the cost of finding all fixed points piecewise-linear networks scales *exponentially* with the number of units in the networks: we would have to solve $(D+1)^N$ systems of N equations [65, 111]. If networks are low rank, it is straightforward to show that we can reduce this cost to solving $(D+1)^N$ systems of R equations (See Supplement .3.1). In addition, however, we show that the computational cost can be greatly reduced further: One can find *all* fixed points in a cost that is *polynomial* instead of *exponential* in the number of units:

Proposition 1. Assume Eq. 4.1, with $\mathbf{J}$ of rank R and piecewise-linear activations ϕ. For fixed rank R and fixed number of basis functions D, we can find all fixed points in the absence of noise, that is all $\mathbf{x}$ for which $\frac{d\mathbf{x}}{dt} = 0$, by solving at most $\mathcal{O}(N^R)$ linear systems of R equations.

Proof. See Supplement .3.1.

Sketch. Assuming $D = 1$, activations $\phi = \max(0, \mathbf{x}_i - \mathbf{h}_i)$; N units will partition the full phase space into 2^N regions in which the dynamics are linear (2 units, 4 regions in Fig. 4.2). We can thus, in principle, solve for all fixed points by solving all corresponding linear systems of equations [65, 111]. If dynamics are confined to the R-dimensional subspace spanned by the columns of $\mathbf{M}$, only a subset of the linear regions (3 in Fig. 4.2) can be reached. Each unit partitions the space spanned by the columns of $\mathbf{M}$ with a hyperplane (pink points in Fig. 4.2). The amount of linear regions in $\mathbf{M}$, becomes equivalent to 'how many regions can we create in R-dimensional space with N hyperplanes?' Using Zaslavsky's theorem [174], we can show that this at most $\sum_{r=0}^{R} \binom{N}{r} \in \mathcal{O}(N^R)$ (for fixed R).

4.4 EMPIRICAL RESULTS

4.4.1 *RNNs recover ground truth dynamics in student-teacher setups*

We validated our method using several student-teacher setups (Fig. 4.3; additional statistics in Fig. S.13). We first trained a 'teacher' RNN, with the weight matrix constrained to rank 2, to oscillate. We then simulated multiple trajectories with a high level of stochasticity in the latent dynamics (Fig. 4.3**a**, top left) and additional additive Gaussian observation noise (Fig. 4.3**a**, top right) on the observed neuron activity ($\mathbf{y}_i \sim \mathcal{N}(\mathbf{x}_i, \sigma_y^2)$, with $\mathbf{x} = \mathbf{M}\mathbf{z}$). A second 'student' RNN was then fit to the data drawn from the teacher, and both recovered the true latent dynamical system, as well as the right level of stochasticity (Fig. 4.3**a**, bottom; Fig. 4.3**d**).

We also verified that we can obtain covariance matrices $\Sigma_{\mathbf{z}}$ that are numerically close to the ground truth, for teacher networks with various levels of noise (Fig. S.14). When using the bootstrap proposal (i.e., sampling from the prior; $r = p(\mathbf{z}_t \mid \mathbf{z}_{t-1})$), or too few particles, the right level of stochasticity is not obtained, indicating that the use of multiple particles and a proposal that conditions on observed data is indeed beneficial.

Given that neurons emit action potentials, which are commonly approximated as discrete events, we repeated the initial teacher-student experiment with Poisson observations generated according to $\mathbf{y}_i \sim \mathsf{Pois}(\mathsf{softplus}(\mathbf{w}_i \mathbf{x}_i - \mathbf{b}_i))$. The student RNN again recovers the oscillatory latent dynamics. Note that because of the affine transformation in the observation model, the inferred dynamics can be scaled and translated with respect to the teacher model. To verify that samples from our inferred model follow the same distribution as samples from the teacher model, we computed several statistics, which all show a close match (Fig. 4.3**e**; Fig. S.13).

In our final teacher-student setups, we verified the ability to recover dynamics when there are known stimuli or contexts. In particular, we trained a rank-2 RNN on a task where, at each trial, it receives a transient pulse input corresponding to a particular angle θ (given as $\sin(\theta), \cos(\theta)$), and is asked to provide output matching the input after stimulus offset. The teacher RNN learns to perform the task by using an approximate ring attractor - which the student RNN accurately infers (Fig. 4.3**c**). Here, we inferred all fixed points by making use of Preposition 1. To demonstrate that our method also works when inputs are strongly time-varying, we included an additional setup where the teacher network was asked to report the sign of the mean of a noisy stimulus (Fig. S.15).

4.4.2 *Stochasticity allows recovering low-dimensional latents underlying EEG data*

After validating our model on a toy example, we went on to several challenging real-world datasets. We first used an EEG dataset [175, 176] with 64 channels containing one minute of continuous data sampled at 160 Hz (Fig. 4.4). This dataset was recently used in a study where generalized teacher forcing (GTF) was used to fit deterministic RNNs with low-rank structure [64]. The GTF method obtains state-of-

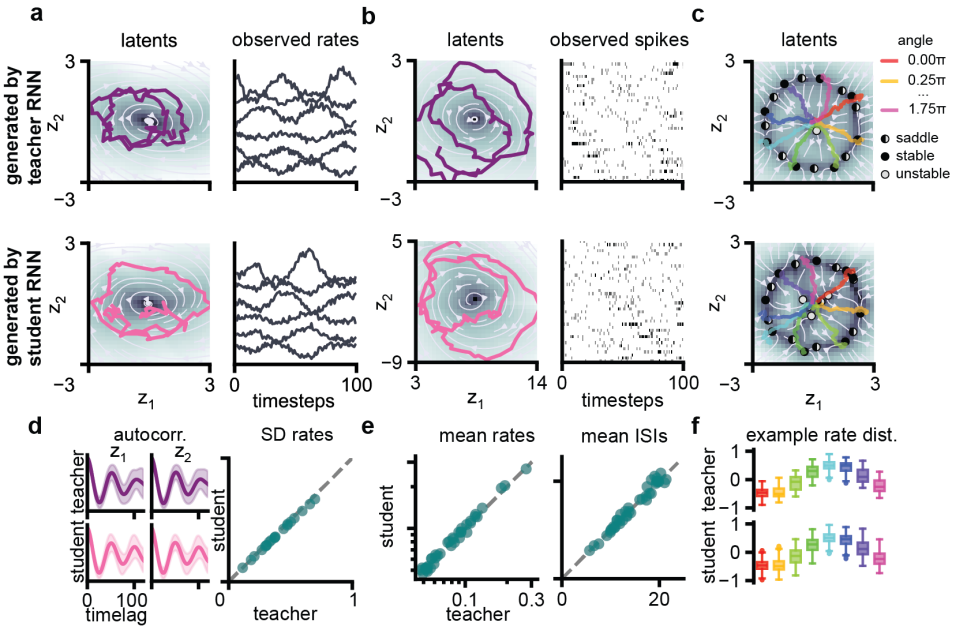

FIGURE 4.3: RNNs recover dynamics in teacher-student setups. **a)** Example ground truth latent trajectory and phase plane of low-rank RNN trained to oscillate (top left) and noisy observations of neuron activity (top right; 6/20 shown). A second low-rank RNN trained on the activity of the first recovers ground truth dynamics. **b)** Same set-up, but with Poisson observations. **c)** The teacher network was trained on a task where it has to provide an output corresponding to 8 different angles depending on an input cue. The student network, when given the same input during fitting, recovers the approximate ring attractor. **d)** Mean (±1SD) autocorrelation of the latents of the models from panel **a**, show the oscillation frequency is captured, as well as the decorrelation due to recurrent noise. The scale of the observed rates also agrees between student and teacher. **e)** Mean rates and ISI between student and teacher units of panel **b** match. **f)** Example rate distribution of one unit of the teacher and student RNN (of panel **c**), after onset of the 8 different stimuli.

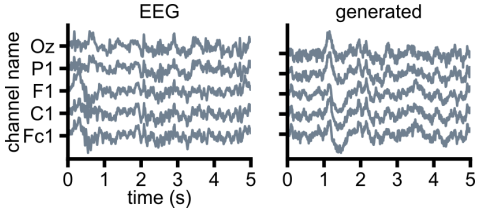

FIGURE 4.4: Example ground truth EEG [175, 176] and (unconditionally) generated traces by our model. Shown are 5/64 EEG channels.

Dataset	Method	$D_{\text{stsp}} \downarrow$	$D_H \downarrow$	dim	$\lvert\theta\rvert$
EEG (64d)	GTF [64]	2.1 ± 0.2	0.11 ± 0.01	16	17952
	adaptive GTF [64]	2.4 ± 0.2	0.13 ± 0.01	16	17952
	SMC (ours)	2.1 ± 0.1	0.11 ± 0.01	3	3920

TABLE 4.1: Lower dimensional latent dynamics than SOTA at same sample quality. We report median $\pm$ median absolute deviation over 20 independent training runs, 'dim' refers to the dimensionality of the model's underlying dynamics and $|\theta|$ denotes the total number of *trainable* parameters. Values for GTF taken from Hess *et al.* [64].

the-art results on several dynamical systems reconstruction tasks. It outperformed SINDy [177], neural differential equations [104], Long-Expressive-Memory [178], and other methods, while using a smaller latent dynamical system.

Here we show that using a stochastic RNN with SMC instead of a deterministic RNN with GTF, we can decrease the latent dimensionality even further, from 16 to just 3 latents, while matching the original reconstruction accuracy (Table 4.1). We hypothesize this is because the data can be well explained by stochastic transitions with simple underlying dynamics as opposed to complex deterministic chaos.

We evaluated samples from our RNN with two measures which were used in previous work [64], one KL divergence-based measure between the states (D_{stsp}), and one measure over time, based on the power spectra of generated and inferred dynamics (D_H; see Supplement .3.4.12). Unlike Hess *et al.* [64], who applied smoothing, we optimized our models directly on the raw EEG data. We also fit stochastic full-rank RNNs with variational SMC, however these models tend to have worse performance on this task, while also being less interpretable (Fig. S.16).

4.4.3 *Interpretable latent dynamics underlying spikes recorded from rat hippocampus*

We next investigated how well our model can capture the distribution of non-continuous time series. In particular, we used publicly available electrophysiological recordings from the hippocampus of rats running to drops of water or pieces of food [155, 156]. We binned the spiking data into 10ms bins and fit a rank-3 RNN to $\sim$850 s of data. Samples generated by running the fit RNN autonomously closely matched the statistics of the recordings (Fig. 4.5**a-c**). Previous investigations into this dataset have examined the relationship between spikes and theta (5-10 Hz) oscillations in the local field potential ([155]), and found that units were locked to the LFP rhythm, with the relative phase depending on the subregions from which the units were recorded. The latents generated by the RNN are visually similar to the average local field potential (Fig. 4.5**d**) and match its power spectrum (Fig. 4.5**e**). While the model was solely trained on the spikes, the posterior latents (Eq. 4.3) have a clear phase relationship with the LFP, as evidenced by a high coherence between the posterior latents and LFP. In contrast, and as expected, latents from running

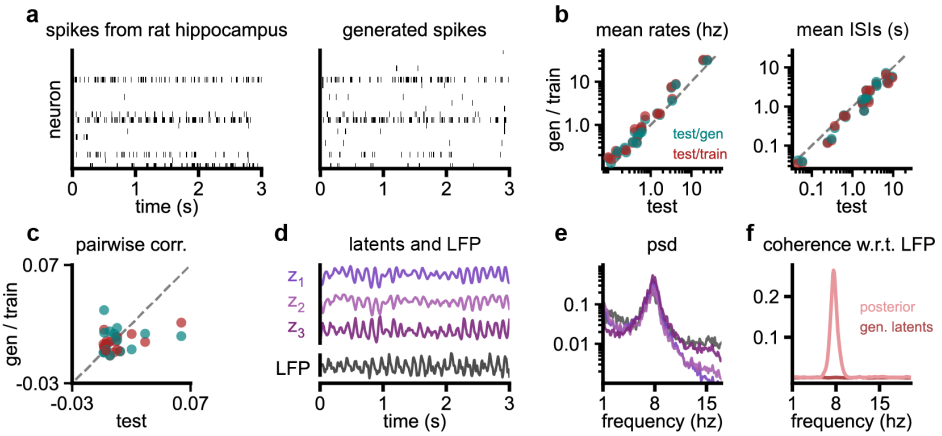

FIGURE 4.5: RNNs reproduce the stationary distribution of spiking data. **a)** We fit a rank-3 RNN to spike data recorded from rat hippocampus [155, 156] (left), and generate new samples from the RNN (right). **b)** Single neuron statistics. Mean rates and means of interspike interval (ISI) distributions of a long trajectory of data generated by the RNN (gen) match those of a held-out set of data (test). As a reference we additionally computed the same statistics between the train and test set. **c)** Population level statistics. We plot the pairwise correlations between all neurons for generated data against the pairwise correlations in the test data. **d)** The corresponding latents generated by running the RNN look visually similar to the local field potential (LFP). **e)** The peak in the power spectrum matches between latents and LFP. **f)** The posterior latents show coherence with the LFP. As a reference, we compute the coherence between the LFP and the latents generated by the RNN.

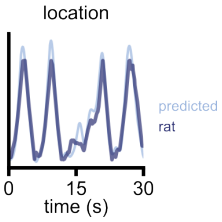

FIGURE 4.6: Posterior latents of our model (fit solely spikes) can be used to predict rat position.

the RNN are not correlated with the LFP (Fig. 4.5**f**). The correspondence between generated latents and LFP was absent when we use a related method for fitting RNNs (with deterministic transitions) to neural data (LFADS [46]; Fig. S.17). Using the bootstrap proposal also led to lower-quality samples (Fig. S.18).

Units in rat hippocampus have been shown to code for position, e.g., through place cells [179], which tend to fire if the animal is at a specific location. To further investigate how well we can model recordings from the hippocampus, we fit a rank-4 RNN to an additional set of recordings of rats running on a linear track [180–182] (Fig. S.19). As in Zhou & Wei [183], we first focus only on the spikes recorded while the rat is moving, which we bin into 25 ms bins. The RNN again accurately reconstructs the distribution of spikes and again has latent oscillations. Here the frequency at which power peaks is slightly higher than that of the LFP, potentially related to phase precession ([143]). While solely trained on spikes, the posterior latents also allowed us to predict the position of the rats with reasonable accuracy ($R^2 = 0.79 \pm 0.05$ mean $\pm$ SD, $N = 4$ RNNs; Fig. 4.6). We also fit rank-12 RNNs to around 15 minutes of recording (again with 25 ms bins), which includes long intermediate periods where the rat is stationary. Here our generative model learns to have higher theta power during running bouts, in line with the data (Fig. S.20).

4.4.4 *Extracting stimulus-conditioned dynamics in monkey reaching task*

We further investigated how well we can recover stimulus-conditioned dynamics. We applied our method to spiking activity recorded from the motor and premotor cortices of a macaque performing a delayed reaching task. This type of data has been popular for investigating neural dynamics underlying the control of movement [33, 36] and evaluating neuroscientific latent variable models [46, 184, 185]. We first validated the ability of our method to obtain a sensible posterior by evaluating it on the Neural Latents Benchmark [185] (Supplement .3.2.5, Table S.1).

We then went on to a set-up where we explicitly conditioned our model on external context. For simplicity, we constrained our experiment to trials with straight reach trajectories in the data. We fit a rank-5 model to these data while conditioning the RNN dynamics on the target position by providing the target position as input. Our model was able to infer single-trial latent dynamics and neuron firing rates

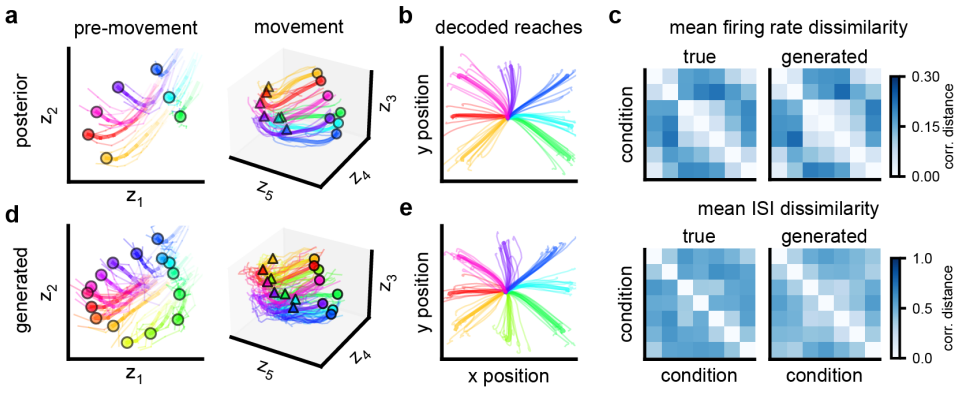

FIGURE 4.7: Inferred and generated dynamics from the model fit to macaque spiking activity during a reaching task. **a)** Latent states inferred from the macaque spiking data prior to movement initiation ('pre-movement') and during movement execution ('movement'), colored by the intended reach target. **b)** Reach trajectories decoded from model-inferred neural activity. **c)** Dissimilarity matrices computed across the seven conditions (i.e., the seven colors in **a**, **b)** for per-neuron mean firing rate and ISI. We generate neural activity from the model by providing the same conditioning stimuli as in the real data. Then, for each statistic, we compute and show the correlation distance between conditions in the real data (left) and model-generated data (right). **d**, **e)** Same as **a**, **b**, but with latent activity and behavioral predictions generated from the model with conditioning inputs including directions not seen in the real data (e.g., lime green). For clarity, we show only a subset of conditions in the decoded reaches.

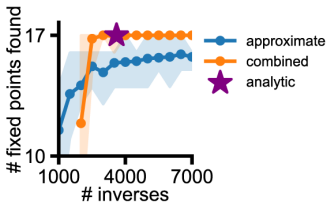

FIGURE 4.8: Comparison of our analytic method (star) and the approximate method proposed in Eisenmann *et al.* [70] (blue) for finding the fixed points of the teacher RNN in Fig. 4.3**c**. We can also use Proposition 1 to constrain the search space of the approximate method (orange). Error bars denote the minimum and maximum amount of fixed points found over 20 independent runs of the algorithm.

that predict reach velocity with high accuracy at lower latent dimensionalities than models without inputs (Fig. 4.7**b**, $R^2 = 0.90$ for this model, see Table S.2 for additional statistics).

We examined the posterior latents inferred by the model and found that our model recovers structured and interpretable latent dynamics. Before movement onset, latent states corresponded to the intended reach targets, which were near the edges of a rectangular screen (Fig. 4.7**a**, left), in line with [184]. During the movement period, the latents followed parallel curved trajectories that preserve target information (Fig. 4.7**a**, right) and can be decoded to predict monkey reach behavior (Fig. 4.7**b)**.

We then generated neural data from the RNN conditioned on stimulus input. Again, the distribution of spikes is well-captured (Fig. S.21). We additionally evaluated whether the model faithfully captures differences in spiking statistics across the seven reach directions, finding reasonable correspondence in dissimilarities between conditions in the generated and the real data (Fig. 4.7**c**). Finally, we simulated our trained RNN with conditioning inputs, including reach directions not present in the data, and found that the structured latent space recovered by the model enables realistic generalization to unseen reach conditions (Fig. 4.7**d**, **e**, lime green condition).

4.4.5 *Searching for fixed points*

In Proposition 1, we derived a bound on the number of systems of equations one has to solve in order to find *all* fixed points in piecewise-linear low-rank RNNs. Recently, an approximate algorithm for finding fixed points in piecewise-linear networks was proposed [70]. Here, we perform an exploration into how this compares to our analytic method by searching for fixed points of the RNN in Fig. 4.3**c** (top). For the same number of matrix inverses computed by our analytic method, the approximate method generally does not find all 17 fixed points (Fig. 4.8). We note, however, that (unlike ours) the convergence of the approximate method depends

on the dynamics of the RNN, and as a result, there are theoretical scenarios where the approximate method can be shown to be faster. Yet we empirically also found scenarios where the approximate methods failed to converge within the time-frame of our experiments (Fig. S.22).

Our analytic method relies on the insight that only a subset of all linear subregions formed by the piecewise-linear activations can be reached in low-rank networks. For networks with moderate rank, the cost of searching through all of the subregions might still be too high. We can, however, hugely reduce the search space of the approximate method [70] (from $(D+1)^N$ to $\sum_{r=0}^{R} D^r \binom{N}{r}$), at an upfront cost (Supplement .3.2.7; orange line in Fig. 4.8).

4.5 DISCUSSION

Here we proposed to fit low-rank RNNs to neural data using variational sequential Monte Carlo. The resulting RNNs are generative models with tractable underlying dynamics, from which we can sample long, stable trajectories of realistic data. We validated our method on several teacher-student setups and demonstrated the effectiveness of our method on multiple challenging real-world examples, where we generally needed a latent dynamical system with very few dimensions to accurately model the data. Besides our empirical results, we obtained a theoretical bound on the cost of finding fixed points for RNNs with piecewise-linear activation functions when they are also low-rank.

Adding stochastic transitions to low-rank RNNs can potentially hugely reduce the rank required to accurately model observed data, as demonstrated here with a network fit to EEG data where we could reduce the dimensionality from 16 to just 3. While many methods that fit RNNs to neural data (e.g., [46, 62, 64, 66–68]) assume deterministic transitions, there is a rich literature concentrating on probabilistic sequence models in neuroscience (e.g., [40–43, 133]). In particular, a recent work termed FINDR [43] uses variational inference (but not SMC), to similarly find very low-dimensional dynamical systems underlying neural data. These stochastic dynamical systems were parameterized using neural differential equations [104]. While Eq. 4.2 can be seen as a neural differential equation with one hidden layer, our particular formulation allows us to find its fixed-points effectively and map back to a regular, mechanistically interpretable RNN (Eq. 4.1) after fitting, which enables additional investigations into neural population dynamics [49, 56, 59, 172].

We here — similar to FINDR (and [186]) — did not use the adjoint method as is typical in the neural differential equation literature, but rather a simple Euler-Maruyama discretisation scheme and standard backpropagation through time. However, one could investigate how we can integrate our approach with variational approaches that use adjoint methods when fitting latent neural SDEs [187, 188] as well as with filtering approaches for continuous time systems [189]. This could be especially relevant for irregularly sampled time-series.

The reason we can do the mapping between a low-rank RNN (Eq. 4.1) and a latent dynamical system (Eq. 4.2) crucially relies on our assumption that samples from the recurrent noise process are correlated, such that they lie within the column-space of $\mathbf{M}$. Valente, Ostojic & Pillow [103] showed that for linear low-rank RNNs arbitrary covariances in the full N dimensional space can be used, when increasing the dimensionality of the latent dynamics to twice the rank R (to the column space of both $\mathbf{M}$ and $\mathbf{N}$), this however does not generalise to our non-linear setting. We do expect correlated recurrent noise to be appropriate for modeling stochasticity arising from unobserved inputs or from partial observations [103] —additionally, correlated noise constituted a pragmatic choice that allows building an *stochastic* model that can allow for trial-by-trial variability while maintaining the tractability of low-rank deterministic RNNs.

Still, future work can investigate training networks with more relaxed assumptions on the recurrent noise models, including extensions to non-Gaussian noise-processes. The latter could be of particular interest if more biologically plausible (i.e., spiking) neurons were used in the recurrence [87, 173].

Our results also open up further avenues to explore questions in neuroscience. The relation between LFP and spike (phase) in the hippocampus has been of great interest [136, 143, 155]. While we performe some preliminary investigation into the relation between the inferred latents and the local field potential, further studies could perform a systematic investigation into their relation, for instance, by using a multi-modal setup [69], or to investigate multi-region temporal relationships and interactions [66].

Taken together, by inferring low-rank RNNs with variational SMC, we obtained generative models of neural data whose trajectories match observed variability, and whose underlying latent dynamics are tractable.

CODE AVAILABILITY

Code to reproduce our results is available at:
`https://github.com/mackelab/smc_rnns`

ACKNOWLEDGMENTS

This work was supported by the German Research Foundation (DFG) through Germany's Excellence Strategy (EXC-Number 2064/1, PN 390727645), SFB 1089 (PN 227953431) and SPP2041 (PN 34721065), the German Federal Ministry of Education and Research (Tübingen AI Center, FKZ: 01IS18039; DeepHumanVision, FKZ: 031L0197B), and the European Union (ERC, DeepCoMechTome, 101089288), the 'Certification and Foundations of Safe Machine Learning Systems in Healthcare' project funded by the Carl Zeiss Foundation. MP and MG are members of the International Max Planck Research School for Intelligent Systems (IMPRS-IS). We thank Cornelius Schröder for feedback on the manuscript, and all members of Mackelab for discussions throughout the project.

5

ATTRACTOR DYNAMICS UNDERLIE SEQUENCE WORKING MEMORY IN DATA-CONSTRAINED RECURRENT NEURAL NETWORKS

Sequences of stimuli can be maintained in working memory by encoding each item by a distinct pattern of neural population activity. However competing hypotheses about the underlying dynamics have been proposed: the patterns of activity might be supported either by stable attractor states that persist over time, or by transient sequential dynamics. To investigate the dynamics underlying sequence memory, we fit low-rank recurrent neural networks (RNNs) to multi-session spiking recordings of macaques. Our data-constrained generative models accurately reproduced population spiking activity and recovered single-unit tuning properties across recording sessions. Consistent with prior analyses, the fitted networks represented items presented at different sequence positions in separate low-dimensional subspaces in the neural activity, alongside a dominant temporal component. The RNNs enabled direct investigation of the underlying dynamics. By isolating the temporal and stimulus subspaces, we found a manifold of fixed points emerged during the memory retention period. Targeted perturbation analyses further revealed that activity within this manifold is consistent with attractor-like dynamics. The models predict that brief optogenetic perturbations delivered along this attractive manifold can induce persistent shifts in neural activity and corresponding behavior, selectively for each sequence position. Our data-constrained low-rank RNNs suggest that sequence working memory is organized by coding in distinct subspaces, within which stable attractor-like dynamics support persistent representations.

5.1 DECLARATION

This chapter is based on: Matthijs Pals, Jakob H Macke. Attractor dynamics underlie sequence working memory in data-constrained recurrent neural networks. *manuscript in writing* (2026).

Author contributions

The dataset and research question were chosen by me, I performed the analysis, created the figures and wrote the initial draft of the paper. Jakob H Macke provided feedback on the analysis, writing and figures.

CRediT

Conceptualization: Matthijs Pals, Jakob H Macke
Formal analysis: Matthijs Pals
Funding acquisition: Jakob H Macke
Software: Matthijs Pals
Supervision: Jakob H Macke
Writing – original draft: Matthijs Pals
Writing – review & editing: Jakob H Macke

5.2 INTRODUCTION

Many behaviors require remembering ordered sequences, for instance holding a multi-digit authentication code in mind long enough to enter it correctly. Working memory is the capability that allows holding and manipulate information over short periods during such tasks [190]. Early theories proposed that working memory was instantiated by the persistent firing of neurons throughout the memory interval [5, 6, 191]. In addition to reports of persistent activity (e.g., [192, 193]), subsequent work has revealed that many individual neurons exhibit complex, time-varying responses [12–16]. One way to reconcile these observations is that the information can remain stable at the population level during memory maintenance, [17–19].

In sequence working-memory tasks, population activity exhibits additional structure: Items presented at different serial positions can occupy distinct low-dimensional subspaces within the population activity, as observed in experiments with both single unit resolution [74, 194, 195] and broadband MEG recordings [196]. This geometry resembles a representation in which items can be bound to sequence position along (approximately) orthogonal axes [197, 198]. Such a scheme enables ordered information to be maintained within a single population despite overlapping neural units.

However, representational geometry alone does not reveal the dynamical mechanisms that underlie these patterns. Multiple competing hypotheses remain under active investigation in the field [199]; the same stable population-level representation could arise either from persistent attractor states [7–11, 19, 50, 200] or from sequences of decaying transients [105, 201, 202]. We therefore ask: what dynamical regime underlies memory maintenance in the observed subspace during working memory for sequences?

To address this question, we fit recurrent neural networks (RNNs) to single-unit recordings from macaques performing a sequence working-memory task [74]. RNNs are widely used to model neural population activity and to generate hypotheses about the computations they implement [37, 47, 51, 63, 65, 67]. Here we focus on RNNs with low-rank connectivity [49, 56, 59, 60, 76, 102], which provides interpretability through a direct link between high-dimensional population activity and an underlying low-dimensional dynamics.

We first design a framework for fitting stochastic low-rank RNN such that we can fit one RNN to units recorded over multiple sessions at single trial resolution. The resulting RNN is a generative model of the data, that captures both spiking statistics and recovers experimentally observed population level representations. We perform stability analyses of the low-rank RNNs latent dynamics and in silico perturbation experiments, and show that the network maintains sequence information via stable attractor dynamics together with a time-varying component. Our results predict that transient perturbations delivered along the memory manifold during the delay period will induce persistent shifts in the neural state, resulting in behavioral changes.

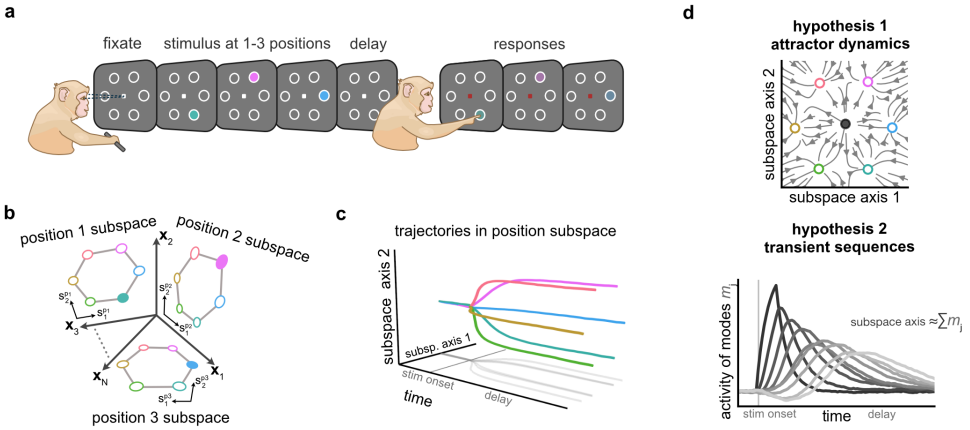

FIGURE 5.1: **Two hypotheses underlying population codes during working memory. a)** We use data from an experiment in which macaques performed a task requiring them to maintain sequences of 1–3 stimuli in working memory [74]. **b)** Previous studies found a separate (approximately orthogonal) subspace for each sequence position in the population activity of PFC neurons [74, 194, 195]. **c)** Subspaces are consistent throughout the delay. **d)** Stable decoding during the delay period could arise from either attractor dynamics (top; [10, 11, 19, 49]) or transient sequential activity (bottom; [105, 201, 202]).

5.3 RESULTS

5.3.1 *Dynamical mechanisms underlying stable working memory representations*

To study the dynamics underlying working memory, we use a dataset of single-unit recordings from the macaque prefrontal cortex (PFC) [74]. Macaques maintain sequences of 1–3 stimuli which, after a delay of 1.15–1.5s, are indicated by pressing on the screen (Fig. 5.1a). Chen *et al.* [74] found that each item in the sequence was represented in approximately orthogonal subspaces of the population activity (schematically in Fig. 5.1b), in line with other recent studies [194–196]. These subspaces can be consistently decoded throughout the memory period (schematically in Fig. 5.1c).

Stable decoding of remembered items during the delay could arise from distinct dynamical mechanisms (Fig. 5.1d): one possibility is that *attractor dynamics* underlie persistent activity patterns, for instance via a fixed point for each stimulus (Fig. 5.1d, top) [10, 11, 19, 49, 200]. Alternatively, the observed time-invariant subspace could emerge from *transient sequential activity*, in which a sequence of modes (patterns of activity) that trigger one another bridge the delay (Fig. 5.1d, bottom) [105, 201, 202]. Summing these transient modes can also result in a stable constant representations throughout a memory period [105]. To formalize these hypotheses, we constructed RNN models that maintain a stable memory representation while using either attractor or transient dynamics (Fig. S..23, S..24). While we observed a similar

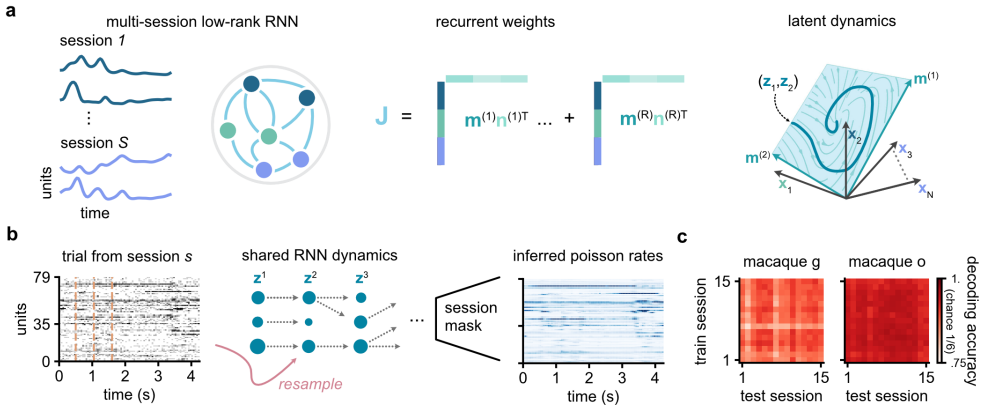

FIGURE 5.2: **Multi-session low-rank RNN with consistent latents between sessions. a)** Each unit in the RNN is identified with one recorded unit (left). We assume that the recurrent weights have a low-rank structure (middle), meaning that we can write the RNN's recurrent connectivity weights **J** as an outer product of left and right connectivity vectors **m** and **n**'s. The low-rank connectivity structure implies that the multi-session neural activity is constrained to a shared low-dimensional subspace (right; spanned by the **m** and input vectors). **b)** To fit the RNNs to neural data, we use sequential Monte Carlo methods where the latent dynamics are parameterized by a low-rank RNN that is shared between session. During training we alternate trials from different session. **c)** The latents between sessions are consistent with each other as shown by using the inferred latents during the late delay period to do cross-session decoding of the maintained items.

geometry of the latents between both models, we confirmed that our subsequent analyses can distinguish between the two models.

5.3.2 *Data simulated with multi-session RNNs match key properties of recordings*

To distinguish between the mechanisms proposed above, we train a generative RNN models of the spiking activity macaques. Because the recordings are collected over multiple days, we employ multi-session low-rank RNNs that capture shared population dynamics (Fig. 5.2a). Each recorded unit is mapped to a corresponding RNN unit (Fig. 5.2a, left), and the low-rank constraint on recurrent weights (Fig. 5.2a, middle) ensures that network activity evolves within a shared, low-dimensional subspace across sessions (Fig. 5.2a, right). We use RNNs with piecewise-linear (ReLU) activation functions, allowing fixed points and other dynamical features to be efficiently studied [64, 65, 70, 110, 112, 203].

To fit these RNN models at single-trial resolution, we develop a method for fitting low-rank RNNs toto multi-session data (Fig. 5.2b). This method builds on a previous framework Pals *et al.* [203] and leverages sequential Monte Carlo methods [115–117, 120]. During training we alternate data from different sessions. We use a so-called 'bootstrap proposal': At each time step, we propagate multiple

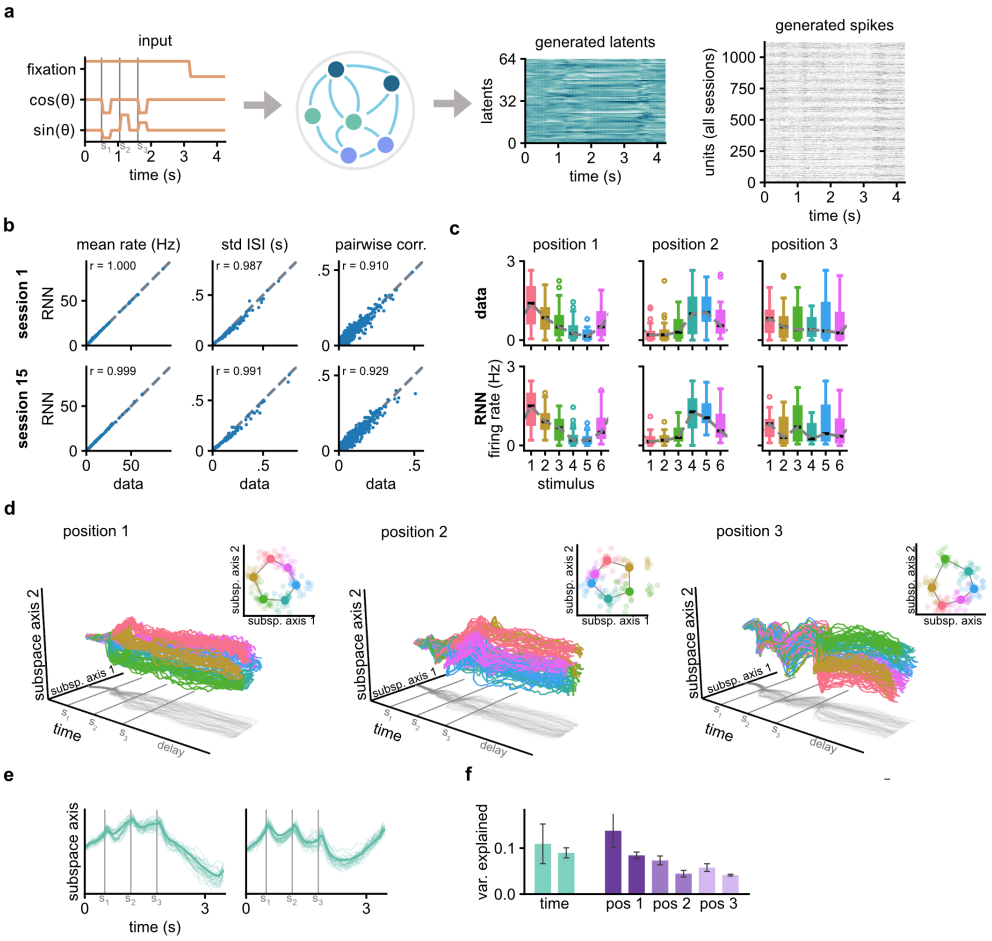

FIGURE 5.3: **Generative RNNs capture key properties of the data. a)** Our data-fitted RNNs act as generative models that produces stochastic spike trains for all recorded units, conditioned on task inputs: A constant fixation input until the end of the delay, and the sequence of task stimuli to be maintained (encoded as $(\sin(\theta, \cos(\theta)$, where θ is the angle of the stimulus). **b)** Key statistical properties of neural activity are accurately reproduced by the model. Comparisons between RNN-generated data and held-out recordings show agreement in mean firing rates, inter-spike-interval (ISI) variability, and pairwise correlations between neurons. Two example sessions are shown. **c)** Units with mixed-selectivity are captured by our RNN as shown by this example tuning curve of a neuron that jointly encodes stimulus identity and sequence position. **d)** Orthogonal transformations of the latent activity reveal three stimulus-encoding subspaces, each corresponding to one sequence position. Neural trajectories within each subspace separate according to the stimulus presented at that position (colors); each line represents the activity in one trial. **e)** Additional latent dimensions capture temporal dynamics that are largely independent of stimulus identity, including ramping activity during the delay and transient responses at stimulus onset. **f)** Identified latent dimensions consistently explain a large amount of delay activity across model seeds and macaques.

latent trajectories ('particles') forward according to the low-rank RNN dynamics. The sampled trajectories are then resampled based on how well they explain the observations in the currently presented session. We fit 5 RNN to 15 sessions of recordings of macaque G (1121 units, 10892 trials used for training) and 5 models to 15 sessions of macaque O (1367 units, 6852 trials). To verify that latent dynamics are consistent between sessions after fitting the model, we perform a cross-session decoding analysis. We first use our inference setup to obtain posterior latents for each session separately, then train session-wise decoders to decode presented stimuli from late-delay latents. By testing each decoder on the inferred latents of all other sessions, we confirmed our framework results in a low-rank RNN that successfully captures a shared latent structure underlying data collected over multiple sessions (cross-decoding accuracies: macaque G: 0.87 ± 0.009, 0.85 ± 0.01, 0.90 ± 0.01, macaque O: 0.97 ± 0.01, 0.94 ± 0.01, 0.92 ± 0.009, mean $\pm$ sd over 5 models, for sequence position 1 to 3, respectively; Fig. 5.2c).

After fitting the RNNs, we can generate new trajectories of latent dynamics and corresponding spiking activity for all units in the data, given stimulus inputs (Fig. 5.3a). The generated spiking activity shows a good match to the recorded activity across held-out spiking data from all sessions, both in terms of firing rates (macaque G: $r = 0.996 \pm 0.008$, macaque O: $r = 0.999 \pm 0.001$), inter-spike-interval (ISI) variability (G: $r = 0.980 \pm 0.01$, O: 0.984 ± 0.006), and pairwise correlations between units (G: $r = 0.888 \pm 0.04$, O: $r = 0.895 \pm 0.02$; all $\mu \pm$ sd over 15 sessions, after first averaging over 5 trained RNNs per macaque; Fig. 5.3b). Previous reports [194] have found units that jointly code for position and stimulus, i.e., fire in response to a stimulus only when it is shown at a particular sequence position, and might be responsive to different stimuli at different sequence positions. Such units are also found in our model (Fig. 5.3c). Moving on to population level analysis, we first find an orthonormal transformation of the RNN dynamics that allows us to find directions in RNN activity space that capture task-related variability [19]. In line with previous experiments [74, 194] and models [198, 204], we find separate two-dimensional subspaces for each sequence position. Additionally, we also find components in the latent dynamics that correlate with timing information [12, 205], such as stimulus onset and progression of the delay period (Fig. 5.3e). To quantify to what degree the neural activity during the delay can be explained by stimulus and timing information, we compute the variance explained by each of the found basis vectors (Fig. 5.3f). In total the 8 basis directions explain $64\% \pm 1.5\%$ ($\pm sd$ $n = 10$) of variance of the RNN's delay activity, indicating we have isolated relevant directions of neural activity during the memory period.

5.3.3 *Analysis of RNN indicates attractor-guided dynamics*

Having verified that our model generates activity that matches the recordings, we next analyse the RNNs dynamics. The time-varying activity observed during the delay period implies that the network state does not converge to a fixed-point attractor during memory maintenance. Yet time-varying activity does not

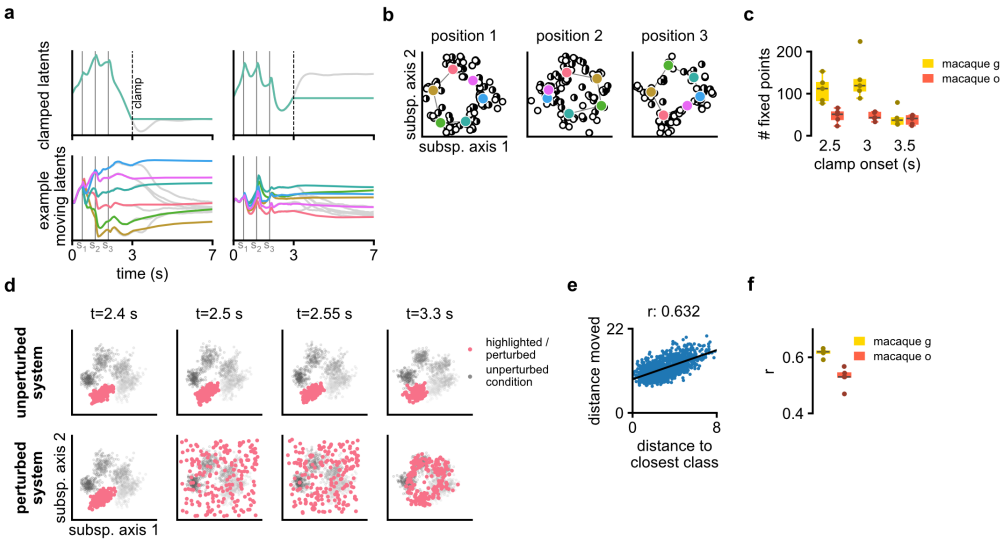

FIGURE 5.4: **Indications of attractor dynamics. a)** To isolate attractor-related dynamics in the presence of time-varying activity, we clamped the time latents to their mean value near the end of the delay period and simulated the remaining latents. Grey trajectories show how latents evolve when not clamped. **b)** Analysis of the reduced latent system reveals multiple stable fixed points (white) and saddle points (half-black/half-white) that cluster around the average neural activity observed during the delay periods (colored circles), suggesting that trajectories are guided toward a stable manifold in this region of state space during memory-maintenance. **c)** We find fixed points for all models, and when clamping the timelatents at different timepoints during the delay. **d)** Perturbations during the delay in all trials of the highlighted condition (pink) are most consistent with a stable manifold: perturbations that land near the activity observed in other trials (gray) move little, whereas trajectories that end outside this manifold quickly converge back to it. **e)** Quantification of perturbations (like in c, but for all conditions). In line with an attractive manifold, perturbations that land closer to one of the condition means (on-manifold) show less movement than those that land further away (off-manifold). **f)** We can consistently find indications of attractor dynamics across seeds for both macaques.

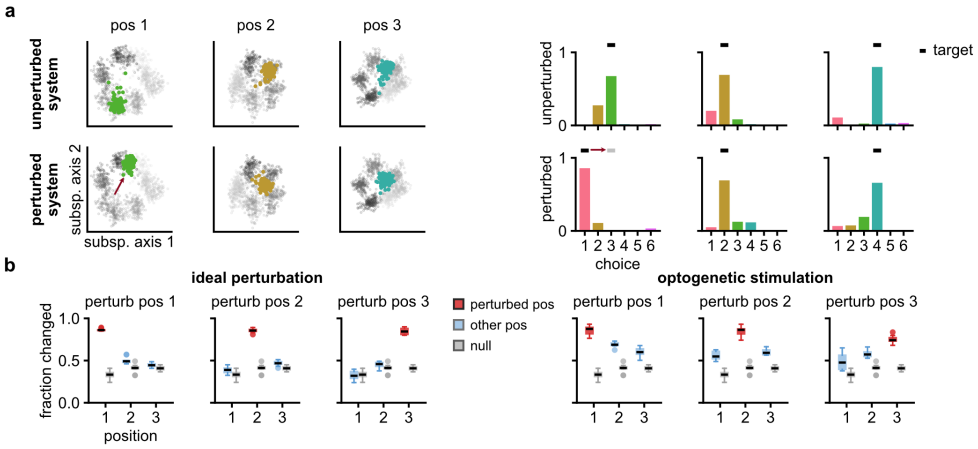

FIGURE 5.5: **Targeted perturbations selectively change behaviorial responses to one sequence position. a)** To test whether the identified position subspaces are relevant for behavior, we apply targeted perturbations that selectively change the models decoded responses for one particular sequence position. Here we add a perturbation to the position 1 subspace (left), on many repetitions of the same trial (with sequence 3,2,4). We can change the models response from choosing item 3 to item 1 for the first sequence position, while leaving the responses for the latter two positions mostly unaffected (right). **b)** Accross all possible permutations, and all trained models, we can consistently change the target position. We retain this effect if we use a perturbation where the same current is applied to a subset of units (mimicking optogenetic perturbations), although with a decreased position specificity.

rule out attractor dynamics [206], for example, chaotic or periodic attractors have been suggested to underlie neural representations during working memory [172, 207]. Alternatively, previous work suggests that activity may evolve in directions orthogonal to the subspace in which stimulus information is maintained by static attractors [18, 19].

To investigate the latter possibility, we simulate the RNN until it is near the end of the delay period. Then, we clamp the time-varying latent variables to their mean value, and continue the simulation for several additional seconds (Fig. 5.4a), while keeping the fixation input on. Under these conditions, the models tend to converge to a steady state. To identify the corresponding fixed points of the clamped system, we apply the algorithm of Eisenmann *et al.* [70]. We find both stable and saddle fixed points that cluster around the mean latent values during the delay period (Fig. 5.4b), for models trained on data of both macaques, irrespective of the exact clamping timepoint (Fig. 5.4c). This result suggests that the dynamics during the delay may be guided by an underlying attractor structure.

Different dynamical regimes can also be distinguished by their responses to perturbations ([200, 208, 209]; see Fig. S..23, S..24 for results with toy models). We therefore apply perturbations to the RNN activity within one of the stimulus subspaces, at the start of the delay period, and examine how the activity evolves thereafter. If the ring of activity reflects a manifold of stable states, perturbations that remain on the manifold should produce little subsequent movement, whereas perturbations that move the activity away from the manifold should decay back toward it, and therefore exhibit larger movement. Conversely, if the subspace were marginally stable during the delay, perturbations in any direction should produce little movement. Finally, if the subspace corresponded only to an output dimension rather than the internal subspace where stimuli are stored, all trajectories should decay back toward the original state. The behavior observed in (Fig. 5.4d, e, f) is most consistent with the presence of a stable manifold.

Finally, we use the model to generate experimental predictions: Can we use targeted perturbations during the memory period to change the macaques response for one sequence position, while keeping its response to the the other positions unaffected? To answer this, we first identify a neural subspace from which we could decode behavioral responses, using inferred latents matched to macaque behavioral outcomes. The found behavioral subspace is also relevant for activity generated by simulating the RNN; We can use it to decode the sequence of stimuli presented to the RNN with with an accuracy of 0.87 ± 0.029 , 0.77 ± 0.034, 0.72 ± 0.051 for sequence position 1, 2 and 3 respectively ($\mu \pm sd,\ n = 10$). Next, we apply targeted perturbations to the previously identified stimulus subspace, which produce persistent shifts in the network state that lead to corresponding changes in the decoded behavioral output (Fig. 5.5). To test whether these stimulus subspaces can be causally manipulated using experimentally relevant interventions, we apply optogenetic-style perturbations during the delay period. Specifically, we inject the same constant current into a subset of units associated with the target stimulus subspace. Because these perturbations cannot be perfectly aligned with the low-

dimensional position subspaces, their effects are somewhat less selective than idealized subspace-targeted perturbations, but nevertheless induce reliable shifts in the network state. These results suggests that a brief perturbation to a stimulus subspace during the delay can steer the macaque's response toward a desired target outcome, specifically for one stimulus position.

5.4 DISCUSSION

We fit low-rank RNNs to multiple sessions of neural recordings of macaques performing sequence working memory tasks [74]. The obtained RNNs are generative models that can generate realistic spiking data with underlying latent trajectories matching the previously reported geometry [74, 194, 195]. Our models allowed us to investigate the responsible dynamical system. Through both stability analysis and perturbation analysis, we find support for the RNNs using stable attractor dynamics while also displaying time-varying delay activity.

We obtained our results using a new approach for fitting multi-session low-rank RNNs. There are many other methods for fitting dynamical systems to multi-session data. One commonly used method is LFADS [46], with a potential additional session-alignment loss (NoMAD; [210]). In LFADS, variability is captured by the initial conditions, per-trial biases [211], or optionally inferred inputs. While well-suited for decoding applications, our approach of learning stochastic transitions might help with obtaining a generative model (see Pals *et al.* [203] for a direct comparison between LFADS and stochastic low-rank RNNs). XFADS [45] is similar to our method in that it learns stochastic transitions and uses a comparable formulation of the posterior latents, yet it imposes a Gaussian approximation rather than a more flexible particle-based representation. Other recent multi-session methods include learning session-specific low-rank perturbations [212], hypernetwork-generated connectivity [213], or sessions-specific reweighting of a shared connectivity basis [214]. These approaches all obtain session-specific models, while our framework learns a unified dynamical system that can jointly generate data across sessions, potentially providing a more holistic representation.

The underlying dynamics of our data-constrained models resemble ideas proposed in previous studies using task-trained or hand-designed models. Our decomposition of dynamics into stimulus-coding subspaces and orthogonal time-coding directions is closely related to Murray *et al.* [19] and Druckmann & Chklovskii [18], although we found a stable circular manifold storing the stimulus as opposed to the proposed marginally stable planes. Similarly, Chaisangmongkon *et al.* [215] found that activity in RNNs trained to perform working memory tasks displayed time-varying activity along one dimension and stimulus-dependent attraction in others (leading to trajectories converging to what they termed 'tunnels'), closely resembling what we find. Complementing these studies, we here obtained a model that is directly fit to neural data and additionally reproduces realistic population activity.

One might wonder whether the inferred attractor dynamics reflect the underlying system or biases of the model class. Recent work has highlighted potential identifiability issues when fitting RNNs to partially observed systems, showing that models can favor attractor-like solutions when approximating high-dimensional transient dynamics, particularly under limited capacity [216]. Here, we use comparatively large networks, which could potentially mitigate this effect. Additionally, in support of our results, we applied our fitting method to data generated from models with either transient or attractor dynamics. In these student-teacher setups, we can recover the correct underlying mechanism (Fig. S. .25, S. .26).

Yet transient and attractor dynamics make distinct predictions about how the system responds to perturbations or noise, as also noted by Qian *et al.* [216]. Perturbation approaches have recently been applied successfully in vivo to distinguish dynamical mechanisms [200, 209]. Our model makes clear predictions for how targeted perturbations should affect subsequent dynamics, and behavioral responses during sequence working memory tasks, providing a route for experimental validation. Analyzing trial-to-trial deviations from the mean dynamics [157] may offer an additional way to distinguish between these regimes.

In summary, our results indicate that sequence working memory representations are supported by attractor dynamics within position specific coding subspaces, while additional dimensions capture time-varying components. Future work combining data-constrained models with causal perturbations will be essential for verifying our proposed system.

5.5 ACKNOWLEDGMENTS

The authors are grateful to Prof. Liping Wang and dr. Jingwen Chen for providing the macaque single unit recordings. This work was supported by the German Research Foundation (DFG) through Germany's Excellence Strategy (EXC-Number 2064/1, PN 390727645) and SFB 1089 (PN 227953431), and the European Union (ERC, DeepCoMechTome, 101089288). MP is a members of the International Max Planck Research School for Intelligent Systems (IMPRS-IS).

5.6 METHODS

5.6.1 *Details of the data*

We used data from Chen *et al.* [74], see the corresponding paper for in-depth details. In brief, macaques were set in front of a screen and performed a sequential working memory task. Each trial started with the macaque pulling a level and fixating at a yellow fixation square. After 100 ms 6 empty circles situated on the vertices of a hexagon appear, and the fixation square turned white. After another 500 ms one to three stimuli were presented sequentially for 250 ms each with an inter-stimulus interval of 300-500 ms. After a delay of 1150 to 1500 ms the white square turned

blue, the macaques released the level and the touched locations on the screen. If the touched locations matched the stimuli, the macaques received a drop of water or juice. Note that the experiment contained both recording days were the monkeys had to report the stimuli in the presented (forward) order, as well as days were they had to report them in reverse order. For this study we only use forward recordings.

Recordings were obtained using a semichronic microdrive system (Gray Matter Research, USA [217]) implanted in the left hemisphere. A 157-channel tungsten electrode array (LS-157, AlphaOmega; $\sim$ 1 MΩ, 1.5 mm interelectrode spacing) was used and the recording chamber covered the prefrontal and premotor areas. Electrode depths were adjusted to maximize the number of recorded units. High-frequency signals (300–8000 Hz) were sorted offline using IronClust and manually curated (Plexon Inc.). Both single- and multi-unit activity were included. Units were discarded if they had firing rates <1.2 Hz or on average less than 5 spikes per trial. We discretized the recordings to 50ms bins. We used the first 15 recordings from two macaques, giving us 1121 (macaque G) and 1367 (Macaque O) units.

5.6.2 *Recurrent neural networks*

We use RNNs of the form (e.g., [54, 63, 77]):

$$\tau \frac{d\mathbf{x}(t)}{dt} = -\mathbf{x}(t) + \mathbf{J}\,\phi(\mathbf{x}(t)) + \mathbf{H}\mathbf{u}(t) + \Gamma_{\mathbf{x}}\boldsymbol{\xi}(t), \tag{5.1}$$

where $\mathbf{x}(t) \in \mathbb{R}^N$ is the vector of neural activity (currents), $\tau \in \mathbb{R}^+$ is the synaptic time constant, and $\mathbf{J} \in \mathbb{R}^{N\times N}$ is the recurrent connectivity matrix. The task stimuli are represented by input $\mathbf{u}(t) \in \mathbb{R}^{N_{inp}}$ and corresponding weights $\mathbf{H}^{N\times N_{inp}}$, and $\boldsymbol{\xi}(t) \in \mathbb{R}^R$ is white noise input with coupling matrix $\Gamma_{\mathbf{x}} \in \mathbb{R}^{N\times R}$. The nonlinearity is applied element-wise as:

$$\phi(x_i) = \max(x_i - b_i, 0).$$

We assume that each RNN unit maps to one recorded unit, and that the recorded spiking activity can be describe by an underlying continuous rate $r_i(t)$. Since the RNN activity $\phi(\mathbf{x}(t))$ is dimensionless, we introduce a per-neuron gain and bias.

$$r_i(t) = \text{softplus}\big(w_i\, x_i(t) + c_i\big),$$

where the $\text{softplus}(x) = \log(1 + e^x)$ ensures non-negative rates.

5.6.2.1 *Low-rank structure*

High-dimensional neural activity can be studied by looking at an underlying low-dimensional system [37]. This link can be made explicit in low-rank RNNs [49, 56, 76, 102]. Low-rank RNNs also facilitate our multi-session inference setup, as we model units recorded over different sessions, whose activity then lives in a shared

low-dimensional subspace. To construct these networks, we constrain the weight matrix $\mathbf{J}$ to have rank R:

$$\mathbf{J} = \mathbf{M}\mathbf{N}^\mathsf{T}, \quad \mathbf{M}, \mathbf{N} \in \mathbb{R}^{N \times R}. \tag{5.2}$$

This results in the dynamics being confined to the linear subspace spanned by $\mathbf{M}$ and $\mathbf{H}$ (Assuming $\mathbf{x}(0)$ in this space, or after initial transients have decayed, and $\Gamma_\mathbf{x} = \mathbf{M}\Gamma_\mathbf{z}$). This implies we can describe the RNN dynamics by an R-dimensional variable $\mathbf{z}(t)$ and an N_{inp}-dimensional variable $\mathbf{v}(t)$; we can write

$$\mathbf{x}(t) = \mathbf{M}\mathbf{z}(t) + \mathbf{H}\mathbf{v}(t).$$

We can now equivalently write the RNN dynamics (Eq. 5.1) by describing how the activity evolves in the $(R + N_{inp})$-dimensional subspace:

$$\begin{aligned}
\tau \frac{d\mathbf{z}(t)}{dt} &= -\mathbf{z}(t) + \mathbf{N}^\mathsf{T}\phi\big(\mathbf{M}\mathbf{z}(t) + \mathbf{H}\mathbf{v}(t)\big) + \Gamma_\mathbf{z}\boldsymbol{\xi}(t), \\
\tau \frac{d\mathbf{v}(t)}{dt} &= -\mathbf{v}(t) + \mathbf{u}(t).
\end{aligned}$$

5.6.2.2 *Discretization*

We discretized the latent dynamics using the Euler-Maruyama method with timestep Δ_t. Let $a = 1 - \frac{\Delta_t}{\tau}$, $\Sigma_\mathbf{z} = \frac{\Delta_t}{\tau^2}\Gamma_\mathbf{z}\Gamma_\mathbf{z}^\mathsf{T}$, $\tilde{\mathbf{N}} = \frac{\Delta_t}{\tau}\mathbf{N}$, we obtain

$$\mathbf{z}_{t+1} = a\,\mathbf{z}_t + \tilde{\mathbf{N}}^\mathsf{T}\phi\big(\mathbf{M}\mathbf{z}_t + \mathbf{H}\mathbf{v}_t\big) + \boldsymbol{\eta}_t, \tag{5.3}$$

$$\mathbf{v}_{t+1} = a\,\mathbf{v}_t + (1-a)\mathbf{u}_{t+1}, \tag{5.4}$$

with $\boldsymbol{\eta}_t \sim \mathcal{N}(0, \Sigma_\mathbf{z})$ and where, with a slight abuse of notation, we wrote $\mathbf{z}_{t+1}$, $\mathbf{v}_{t+1}$ for integrating the system one time-step ahead (i.e., for $\mathbf{z}_{t+\Delta t}$,$\mathbf{v}_{t+\Delta t}$). The currents of the RNN units, and corresponding predicted rates are given by:

$$\begin{aligned}
\mathbf{x}_t &= \mathbf{M}\mathbf{z}_t + \mathbf{H}\mathbf{v}_t, \\
\mathbf{r}_t &= \mathrm{softplus}\big(\mathrm{diag}(\mathbf{w})\mathbf{x}_t + \mathbf{c}\big).
\end{aligned} \tag{5.5}$$

5.6.2.3 *Probabilistic model*

Given observed data $\mathbf{y}_{1:T}$, the generative model corresponding to:

$$p(\mathbf{z}_{1:T}, \mathbf{y}_{1:T} \mid \mathbf{v}_{1:T}) = p(\mathbf{z}_1)\prod_{t=1}^{T-1} p(\mathbf{z}_{t+1} \mid \mathbf{z}_t, \mathbf{v}_t)\prod_{t=1}^{T} p(\mathbf{y}_t \mid \mathbf{z}_t, \mathbf{v}_t),$$

where $\mathbf{v}_{1:T}$ is deterministically defined from the input $\mathbf{u}_{1:T}$ by Eq. 5.4. The latent transition distribution is given by the low-rank RNN dynamic (Eq. 5.3):

$$p(\mathbf{z}_{t+1} \mid \mathbf{z}_t, \mathbf{v}_t) = \mathcal{N}\big(a\mathbf{z}_t + \tilde{\mathbf{N}}^\mathsf{T}\phi\big(\mathbf{M}\mathbf{z}_t + \mathbf{H}\mathbf{v}_t\big), \Sigma_\mathbf{z}\big),$$

and we also assume the initial condition to be Gaussian distributed: $p(\mathbf{z}_1) = \mathcal{N}(\boldsymbol{\mu}_1, \Sigma_1)$. Observations are assumed to be Poisson distributed:

$$\mathbf{y}_t \sim \text{Poisson}(\mathbf{r}_t),$$

with the rate $\mathbf{r}_t$ given by Eq. 5.5.

5.6.3 *Inference and training*

5.6.3.1 *Sequential Monte Carlo*

Because the exact posterior $p(\mathbf{z}_{1:T} \mid \mathbf{y}_{1:T})$ is intractable, we approximated it using Sequential Monte Carlo (SMC) with a set of K particles representing latent trajectories [120]. At each timestep, latents were proposed from the bootstrap proposal, i.e., from the RNN dynamics $p(\mathbf{z}_t \mid \mathbf{z}_{t-1})$ and assigned importance weights:

$$\begin{aligned} \mathbf{z}_t^k &\sim p(\mathbf{z}_t \mid \mathbf{z}_{t-1}^k), \\ w_t^k &= p(\mathbf{y}_t \mid \mathbf{z}_t^k). \end{aligned}$$

These were normalized as $\overline{w}_t^k = w_t^k / \sum_{j=1}^K w_t^j$ followed by a resampling step. This process results in a filtering approximation to the posterior at time t:

$$q_{\mathsf{filt}}(\mathbf{z}_{1:t} \mid \mathbf{y}_{1:t}) = \sum_{k=1}^K \overline{w}_t^k \delta(\mathbf{z}_{1:t}^k).$$

We optimized the RNN by maximizing an Evidence Lower Bound (ELBO) defined over the joint distribution of all particles and ancestor indices, q_{smc} [115–117]:

$$\mathcal{L} = \mathbb{E}_{q_{\mathsf{smc}}(\mathbf{z}_{1:T}^{1:K}, a_{1:T-1}^{1:K} \mid \mathbf{y}_{1:T})}[\log \hat{p}(\mathbf{y}_{1:T})], \tag{5.6}$$

where $\hat{p}(\mathbf{y}_{1:T}) = \prod_{t=1}^T \frac{1}{K} \sum_{k=1}^K w_t^k$ is the unbiased marginal likelihood estimate. During optimization we discarded high-variance gradient terms associated with the resampling step [115–117].

5.6.3.2 *Optimisation*

During training we alternated batches drawn from different recording sessions. For computing the loss, we only considered the rates given by RNN units matching the data units in the current batch. We minimised the ELBO (Eq.5.6) with stochastic gradient descent, using the Adam [218] optimiser in Pytorch [170].

We used batches of 128 trials, and trained for 750 epochs. Each training epoch consists of 15 sessions, with 344-571 (macaque O) and 566-850 (macaque G) trials per session. We used cosine decay to 10^{-6} for the learning rate, after an linear warm up of 50 epochs from 10^{-6} to 10^{-3}. Training took around 8 hours on one NVDIA 3080 for macaque O and 12 hours for macaque G. There was a training instability on some runs; In case NaNs or divergent losses appeared, a run was restarted. The models had a rank of $R = 64$, and we used $k = 32$ particles.

5.6.3.3 *Initialisation*

Models were initialized as follows:

$$\tilde{\mathbf{N}}_{ij} \sim \mathcal{U}_{[-\frac{1}{\sqrt{N}},\frac{1}{\sqrt{N}}]}, \qquad \mathbf{M}_{ij} \sim \mathcal{U}_{[-\frac{1}{\sqrt{R}},\frac{1}{\sqrt{R}}]},$$
$$\mathbf{H}_{ij} \sim \mathcal{U}_{[-\frac{1}{\sqrt{N_{inp}}},\frac{1}{\sqrt{N_{inp}}}]}, \qquad \mathbf{h}_i \sim \mathcal{U}_{[-\frac{1}{\sqrt{N}},\frac{1}{\sqrt{N}}]},$$
$$\mathbf{b} \leftarrow \mathbf{0}, \qquad \mathbf{c} \leftarrow \mathbf{0},$$
$$\mathbf{w} \leftarrow \mathbf{1}, \qquad a \leftarrow .9,$$
$$\Sigma_{\mathbf{z}} \leftarrow .01\mathbf{I}, \qquad \Sigma_{\mathbf{z}_1}.01 \leftarrow \mathbf{I},$$
$$\mu_{\mathbf{z}_1} \leftarrow \mathbf{0}.$$

We constrained a to be between 0 and 1 by instead optimizing $\tilde{a}$ with the following (sigmoidal) parameterization $a = \exp(-\exp(\tilde{a}))$ [219]. We assumed all covariances are diagonal and isotropic; parameterized by their standard deviations using $\Sigma = \mathbf{I}\,\mathrm{softplus}(\tilde{\sigma})^2$.

5.6.4 *Stimuli and time basis directions*

To obtain interpretable low-rank RNN dynamics, we expressed the latent activity in terms of orthogonal modes aligned with task variables. The connectivity vectors $\mathbf{M}$ were unconstrained during training and therefore do generally not define an orthogonal coordinate system. This means that individual latents $\mathbf{z}(t)$ can reflect arbitrary linear combinations of underlying modes. We obtained an orthogonal basis after training, by performing a singular value decomposition of the connectivity matrix,

$$\mathbf{J} = \mathbf{M}\mathbf{N}^{\mathsf{T}} = \mathbf{U}\mathbf{S}\mathbf{V}^{\mathsf{T}},$$

and projecting the latent state as $\mathbf{z} \leftarrow \mathbf{U}^{\top}\mathbf{M}\mathbf{z}$ (retaining only the first R singular vectors).

This basis is still agnostic to task structure. To better understand which latent dimensions encode variables such as stimulus identity we performed an additional orthonormal transformation that aligned components with task-relevant variables (inspired by Murray *et al.* [19]).

The latent activity was arranged as a tensor $\mathbf{Z} \in \mathbb{R}^{R \times S_1 \times S_2 \times S_3 \times T}$, where $R = 64$ latents, $S_i = 6$ stimulus identities at each of three sequence positions, and T denotes time. Because not all stimulus combinations were valid (each item could appear only at one position) and sequence elements had unequal durations, we defined a binary mask M indicating valid samples.

We first subtracted the global mean activity of each neuron across all valid samples:

$$\mu_r = \frac{\sum_m M_m \mathbf{Z}_{r,m}}{\sum_m M_m}, \quad \hat{\mathbf{Z}} = \mathbf{Z} - \boldsymbol{\mu},$$

where m runs along all task and time dimensions.

To isolate task-related variance, we computed marginalizations for time and each sequence position. For a given marginal (e.g., position 1: $i = S_1$), activity was averaged over all other dimensions using the mask:

$$\mathbf{Z}^{(i)} = \frac{\sum_{k \in \mathcal{A}_i} M_k \hat{\mathbf{Z}}_k}{\sum_{k \in \mathcal{A}_i} M_k},$$

where $\mathcal{A}_i$ denotes all axes except neurons and the marginal of interest. Only samples with nonzero mask support were retained, and the result was reshaped to $\mathbf{Z}^{(i)} \in \mathbb{R}^{N \times K_i}$.

We then performed PCA:

$$\mathbf{C}^{(i)} = \mathbf{Z}^{(i)} {\mathbf{Z}^{(i)}}^\top = \mathbf{U}^{(i)} \mathbf{\Lambda}^{(i)} {\mathbf{U}^{(i)}}^\top,$$

and the leading eigenvectors were retained to form $\mathbf{W}^{(i)}$. We kept 2 components for time and each of the three sequence positions.

Subspaces were extracted sequentially (time, position 1, position 2, position 3). After each step, the new basis $\mathbf{W}^{(i)}$ was orthogonalized with respect to previously extracted components W_{prev}:

$$\mathbf{W}^{(i)} \leftarrow \mathbf{W}^{(i)} - \mathbf{W}_{\text{prev}} \left(\mathbf{W}_{\text{prev}}^\mathsf{T} \mathbf{W}^{(i)} \right),$$

followed by normalization of the columns. To prevent variance leakage across marginals, we projected out the subspace at the current step:

$$\mathbf{Z} \leftarrow \mathbf{Z} - \mathbf{W}^{(i)} \mathbf{W}^{(i)\mathsf{T}} \mathbf{Z}.$$

All $\mathbf{W}^{(i)}$ were concatenated to form $\mathbf{W}_{\mathsf{enc}} \in \mathbb{R}^{R \times 8}$. This basis was extended to a full orthonormal basis $\mathbf{W}_{\mathsf{full}} \in \mathbb{R}^{R \times R}$ by appending random vectors and orthogonalizing the combined set via QR decomposition.

We computed the basis vectors $\mathbf{W}_{\mathsf{full}}$ using a train set of data generated by the RNN on 2400 trials (20 trials for each out of $6 \times 5 \times 4 = 120$ possible stimulus conditions). For all plots, we first generated new data, changed to an orthogonal basis using $\mathbf{U}^\top \mathbf{M}$ followed by subtracting with the mean $\boldsymbol{\mu}$ and an orthonormal rotation with $\mathbf{W}_{\mathsf{full}}$.

For analysis of the RNN dynamics (perturbations and fixed points), we transformed the RNN's latent dynamics such they are expressed in this basis. Take

$\mathbf{A} = \mathbf{W}_{\text{full}}^{\mathsf{T}}\mathbf{U}^{\mathsf{T}}\mathbf{M}$ and $\mathbf{c} = \mathbf{W}_{\text{full}}^{\mathsf{T}}\boldsymbol{\mu}$. Under the change of coordinates $\mathbf{z}_{new} = \mathbf{A}\mathbf{z} - \mathbf{c}$, the RNN parameters are updated as:

$$\begin{aligned} \mathbf{M} &\leftarrow \mathbf{M}\mathbf{A}^{-1}, & \tilde{\mathbf{N}} &\leftarrow \mathbf{A}\tilde{\mathbf{N}}, \\ \mathbf{h}_2 &\leftarrow \mathbf{M}\mathbf{A}^{-1}\mathbf{c} - \mathbf{b} + \mathbf{H}\mathbf{v}, & \mathbf{h}_1 &\leftarrow a\mathbf{c} - \mathbf{c}, \\ \mathbf{z}_0 &\leftarrow \mathbf{A}\mathbf{z}_0 - \mathbf{c}, \end{aligned}$$

with the RNN now described by:

$$\mathbf{z}_{t+1} = a\mathbf{z}_t + \tilde{\mathbf{N}}^{\mathsf{T}}\text{ReLU}\Big(\mathbf{M}\mathbf{z}_t + \mathbf{h}_2\Big) + \mathbf{h}_1.$$

In Fig 5.2, we plot the variance of the delay activity per component. Because the (fixation) input is constant across conditions during the delay, and because the latent trajectories were expressed in an orthonormal basis, the variance captured in the latent variables $\mathbf{z}$ corresponds directly to variance in RNN activity $\mathbf{x}$.

5.6.5 *Conditional fixed points*

To find fixed points conditioned on a subset of clamped latent coordinated, we partitioned the latent states into R_c *clamped* and R_m *moving* components:

$$\mathbf{z}_t = \begin{bmatrix} \mathbf{z}_t^{(c)} \\ \mathbf{z}_t^{(m)} \end{bmatrix}, \qquad R_c + R_m = R,$$

where $\mathbf{z}_t^{(c)}$ is clamped and $\mathbf{z}_t^{(m)}$ evolves according to the dynamics. We can treat the fixed coordinates as input or bias to the moving latents. Denoting, with $\ldots^{(m)}$ and $\ldots^{(c)}$ the columns or rows corresponding to moving and frozen coordinates, we obtained a set of R_m equations describing only the moving latents:

$$\begin{aligned} \mathbf{z}_{t+1}^{(m)} &= a\mathbf{z}_t^{(m)} + a\mathbf{z}_t^{(c)} + \tilde{\mathbf{N}}^{(m)\mathsf{T}}\text{ReLU}\Big(\mathbf{M}^{(m)}\mathbf{z}_t^{(m)} + \mathbf{M}^{(c)}\mathbf{z}_t^{(c)} + \mathbf{h}_2 + \mathbf{H}\mathbf{v}\Big) + \mathbf{h}_1^{(m)}, \\ &= a\mathbf{z}_t^{(m)} + \tilde{\mathbf{N}}^{(m)\mathsf{T}}\text{ReLU}\Big(\mathbf{M}^{(m)}\mathbf{z}_t^{(m)} + \mathbf{h}_2^{(m)}\Big) + \mathbf{h}_1^{(m)}, \end{aligned}$$

with effective offsets

$$\begin{aligned} \mathbf{h}_2^{(m)} &= \mathbf{M}^{(c)}\mathbf{z}_t^{(c)} + \mathbf{h}_2 + \mathbf{H}\mathbf{v}, \\ \mathbf{h}_1^{(m)} &= a\mathbf{z}_t^{(c)} + \mathbf{h}_1. \end{aligned}$$

Note we here assumed input $\mathbf{v}$ to be constant, allowing us to incorporate it as a bias term.

For fixed $\mathbf{z}_t^{(c)}$, the conditional fixed points are defined by solutions to

$$\mathbf{z}_*^{(m)} = a\mathbf{z}_*^{(m)} + \tilde{\mathbf{N}}^{(m)\mathsf{T}}\text{ReLU}\Big(\mathbf{M}^{(m)}\mathbf{z}_*^{(m)} + \mathbf{h}_2^{(m)}\Big) + \mathbf{h}_1^{(m)},$$

The full latent fixed point is then reconstructed as

$$\mathbf{z}_* = \begin{bmatrix} \mathbf{z}_t^{(c)} \\ \mathbf{z}_*^{(m)} \end{bmatrix}.$$

We identified fixed points by simulating the clamped RNN for 7 seconds (140 timesteps of 50 ms) while maintaining the fixation input. The latent state at the final timestep was extracted, perturbed with uniform noise, and used as the initial condition for the heuristic SCyFi algorithm [70].

The SCyFi algorithm is designed to compute fixed points in piecewise-linear systems, such as the ReLU-based RNNs used in this study. In piecewise-linear systems, the latent space is partitioned into regions, each governed by its own linear dynamical system. Given a latent state, SCyFi starts by obtaining the region in which that latent state resides, and computes the fixed point of the corresponding linear system. The resulting candidate fixed point may either lie within the same region, in which case it is a valid fixed point of the system, or fall into a different region. In the latter case, SCyFi reinitializes to the new region and computes the fixed point for the new corresponding linear system. This process is repeated until either a valid fixed point is found (within the current region) or a maximum number of iterations is reached.

5.6.6 *Perturbation analysis*

For the perturbation analysis, we simulated the RNNs and applied an impulse current to all RNN units 0.5s. after the last stimulus input. Let $\mathbf{z}_{\text{target}} \in \mathbb{R}^{R_p}$ be the target in latent space, where R_p denotes the amount of latents we want to perturb. We can define the perturbation $\mathbf{P}_\mathbf{z} \in \mathbb{R}^{R_p}$ to latents $\mathbf{z}_t^{(p)}$, or equivalently, the corresponding perturbation $\mathbf{P}_\mathbf{x} \in \mathbb{R}^N$ to units $\mathbf{x}_t$ as:

$$\begin{aligned} \mathbf{P}_\mathbf{z} &= \mathbf{z}_t^{(p)} - \mathbf{z}_{\text{target}}, \\ \mathbf{P}_\mathbf{x} &= \mathbf{M}^{(p)}(\mathbf{z}_{\text{target}} - \mathbf{M}^{(p)\mathsf{T}}\mathbf{x}), \end{aligned}$$

where $\mathbf{M}$ is assumed to be orthonormal (see above) and $\mathbf{M}^{(p)}$ picks out the columns corresponding to the latents we want to perturb. For Fig. 5.4d-f perturbation targets were sampled uniformly from a box spanning the delay-period range of each stimulus subspace coordinate, expanded by $\times 1.2$. Movement was integrated over the next 0.75s to compute the correlation coefficients in Fig. 5.4e and f.

To model optogenetic-style perturbations (Fig. 5.5b), we constrained perturbations to act with uniform intensity on a subset of recurrent units. We can think of the entries in the dense perturbation vector $\mathbf{P}_\mathbf{x}$ as computed above, as quantifying the contribution of each neuron to the desired latent-space displacement. We therefore selected the top 5% of units with the largest magnitudes in $\mathbf{P}_\mathbf{x}$, giving us a sparse mask $\mathbf{m} \in \{0,1\}^N$. Restricting the perturbation to a binary subset of neurons results in an unscaled latent-space perturbation $\tilde{\mathbf{P}}_\mathbf{z}^{\text{opto}} = \mathbf{M}^{(p)\mathsf{T}}\mathbf{m}$ that is generally not

perfectly aligned with the target displacement $\mathbf{P}_\mathbf{z}$. We determined the scaling c by minimizing the reconstruction error $\|\mathbf{P}_\mathbf{z} - c\tilde{\mathbf{P}}_\mathbf{z}^{\text{opto}}\|_2^2$ via orthogonal projection. The resulting scaling factor and the corresponding perturbation applied to the recurrent units are given by:

$$c = \frac{\mathbf{P}_\mathbf{z}^\top \tilde{\mathbf{P}}_\mathbf{z}^{\text{opto}}}{\|\tilde{\mathbf{P}}_\mathbf{z}^{\text{opto}}\|^2}$$
$$\mathbf{P}_\mathbf{x}^{\text{opto}} = c\mathbf{m}.$$

5.6.7 *Behavioral decoding*

We trained a shared 6-class logistic regression (with $l1$ regularization $C = 1$; [169]) to decode the macaques responses on (standardized) latents inferred from the macaques spikes using our sequential Monte Carlo framework. We averaged latents over a 0.25 s window ending .05s after the macaques responses, and fit the same decoder to the responses corresponding to each sequence position. The decoder trained on inferred latents was then applied to generated latent trajectories, with time windows defined based on the macaques average response times.

5.6.8 *Hand-constructed models*

To verify our methods, we implemented RNNs that maintain memory of one angular variable θ using either attractor or transient dynamics. We used methods from Beiran *et al.* [49] for constructing low-rank RNNs with Gaussian populations. The models were simulated using:

$$\tau\frac{d\mathbf{z}(t)}{dt} = -\mathbf{z}(t) + \frac{1}{N}\mathbf{N}^\top\phi\big(\mathbf{M}\mathbf{z}(t) + \mathbf{H}\mathbf{v}(t)\big) + \Gamma_\mathbf{z}\boldsymbol{\xi}(t),$$
$$\tau\frac{d\mathbf{v}(t)}{dt} = -\mathbf{v}(t) + \mathbf{u}(t).$$

5.6.8.1 *Transient dynamics*

We constructed a rank-20 recurrent neural network with $N = 500$ units, with $\phi(\mathbf{x}) = \text{ReLU}(\mathbf{x})$ and $\Gamma_\mathbf{x} = \mathbf{M}$. The dynamics consisted of two sets of 10 transient modes: one encoding $\sin(\theta)$ and one encoding $\cos(\theta)$.

The connectivity matrix $\mathbf{M}$ was initialized with i.i.d. Gaussian entries. To generate sequential dynamics, we imposed a feedforward chain within each set of modes. Let $\mathbf{D} \in \mathbb{R}^{N\times 20}$ be a random Gaussian matrix. The left connectivity vectors for one set of modes were defined as:

$$\mathbf{n}^{(i)} = 0.1\,\mathbf{d}^{(i)} + g_{\text{ff}}\,\mathbf{m}^{(i-1)}, \quad i = 2, \ldots, 10,$$
$$\mathbf{n}^{\text{in}} = 0.1\mathbf{d}^{(1)}.$$

The output mode accumulated contributions from all previous modes:

$$\mathbf{n}^{10} = 0.1\,\mathbf{d}^{10} + g_{\text{out}} \sum_{i=1}^{9} \mathbf{m}^{(i)}.$$

The input weights $\mathbf{I} \in \mathbb{R}^{N\times 2}$ were aligned with the first mode of each set:

$$\mathbf{I}^{(1)} = \sigma_I \left(\alpha\, \mathbf{m}^{(1)} + \sqrt{1-\alpha^2}\, \mathbf{d}_I^{(1)} \right).$$

Dynamics were simulated using Euler-Maruyama integration with time constant $\tau = 0.3$, time step $\Delta t = 0.05$, and Gaussian noise of amplitude $\sigma = 0.005$ injected in the low-rank subspace.

5.6.8.2 *Approximate ring attractor*

We constructed a rank-2 RNN with $N = 600$ units and activation $\phi(x) = \text{ReLU}(x + 2)$. Following Beiran *et al.* [49], the network was partitioned into $P = 6$ equally sized populations, each associated with a preferred angle

$$\theta_p = \frac{2\pi p}{P}, \quad p = 0, \ldots, P-1.$$

The connectivity vectors were defined using structured population means plus Gaussian noise. For each population p:

$$\mathbf{m}^{(m_1,p)} = \sqrt{2-\sigma_b^2}\,\cos(\theta_p) + \sigma_b \mathbf{d}_m^{(m_1,p)},$$
$$\mathbf{m}^{(m_2,p)} = \sqrt{2-\sigma_b^2}\,\sin(\theta_p) + \sigma_b \mathbf{d}_m^{(m_2,p)}.$$

The corresponding left connectivity vectors were

$$\mathbf{n}^{(m_1,p)} = \frac{g_{\text{rec}}}{\sqrt{2-\sigma_b^2}} \cos(\theta_p)\,\mathbf{1} + \sigma_b \left(\alpha_{mn}\, \mathbf{d}_m^{(m_1,p)} + \sqrt{1-\alpha_{mn}^2}\, \mathbf{d}_n^{(m_1,p)} \right),$$
$$\mathbf{n}^{(m_2,p)} = \frac{g_{\text{rec}}}{\sqrt{2-\sigma_b^2}} \sin(\theta_p)\,\mathbf{1} + \sigma_b \left(\alpha_{mn}\, \mathbf{d}_m^{(m_2,p)} + \sqrt{1-\alpha_{mn}^2}\, \mathbf{d}_n^{(m_2,p)} \right).$$

Parameters were set to $\sigma_b = 0.2$, $g_{\text{rec}} = 3$, and $\alpha_{mn} = -0.5$. The positive ReLU bias and recurrent gain push the dynamics away from the origin, while the negative alignment α_{mn} stabilizes activity along the ring. The input matrix $\mathbf{I} \in \mathbb{R}^{N\times 2}$ was aligned with the connectivity modes:

$$\mathbf{I} = \sigma_I \left(\alpha_I\, \mathbf{M} + \sqrt{1-\alpha_I^2}\, \mathbf{D}_I \right),$$

where $\alpha_I = 0.75$, $\sigma_I = 6$, The time constant was $\tau = 0.2$, and Gaussian noise with amplitude $\sigma = 0.1$ was used.

5.6.9 *Dataset for student-teacher setups*

To verify that we can in principle recover transient dynamics, we constructed a dataset using the hand-construct model with transient dynamics. Task input for a total of 6000 trials of length 5s steps was generated. Each trial had one stimulus presented: $\mathbf{u}_t = [\cos(\theta), \sin(\theta)]$, with θ drawn uniformly from 6 stimulus conditions. The stimuli were applied for 250 ms, following a random onset of $500 - 700$ ms. The RNN activity was mapped to spikes via

$$r_{i,t} = \text{softplus}(w_{\text{obs}} x_{i,t} + b_i)\,, \tag{5.7}$$

$$y_{i,t} \sim \text{Poisson}(r_{i,t}). \tag{5.8}$$

The gain w_{obs} and offsets b were set so that the resulting spike trains approximately matched the *marginal* statistics of the experimental data (mean firing rates, inter-spike-interval distributions, and pairwise spike-count correlations). In particular we chose, for the transient dynamics: $w_{\text{obs}} = 1.0$ and $b_i \sim \mathcal{N}(-0.25, 1.0)$, and for the attractor dynamics $w_{\text{obs}} = 0.1$ and $b_i \sim \mathcal{N}(-0.1, 0.5)$. Neurons and trials were partitioned into 8 disjoint sessions. Within each session, trials were divided into 75% training and 25% validation sets.

5.6.10 *Session-consistency loss for student-teacher setups*

In the student-teacher setups, we found that latent axes were not consistently aligned across sessions in the students with rank 2 or 3: the same stimulus could be represented in different coordinates (e.g. $\cos\theta$ in z_1 for one session and in z_2 for another). This was fixed by the consistency loss described here which encourages a shared latent geometry across sessions. Note that we did not apply this loss for any model fit to macaque data. We hypothesize that the alignment problem is more likely to appear in the student-teacher settings here, where the ground-truth dynamics are more low-dimensional and highly symmetric.

Concretely, during each training bach, we computed for each of the presented stimuli s, the mean latent vector $\bar{\mathbf{z}}_s$ over trials for a fixed delay window of .5s, starting .5s after stimulus offset. We maintained a session-aggregated reference centroid $\mathbf{c}_s$, initialized from the first batch in which s appeared, then updated with exponential momentum, $\mathbf{c}_s \leftarrow \beta\,\mathbf{c}_s + (1-\beta)\,\bar{\mathbf{z}}_s$, (with β set to .9). The auxiliary loss was

$$\mathcal{L}_{\text{centroid}} = \sum_s \|\mathbf{c}_s - \bar{\mathbf{z}}_s\|_2^2,$$

added to the variational objective with weight 50 for the rank-2 RNN fit to the attractor teacher model, and weight 1 for the rank-3 RNN fit to the transient teacher model. The reference centroid was updated after computation of the loss (after which gradients were also detached).

6

DISCUSSION AND FUTURE DIRECTIONS

6.1 SUMMARY

In this thesis we developed methods for fitting and interpreting recurrent neural networks (RNNs), and used these models to investigate neural dynamics. In Chapter 3, we used RNNs to investigate dynamics and connectivity during working memory maintenance in the presence of ongoing oscillations. We found that RNNs trained on a working memory task, while receiving oscillatory input can code information relative to the phase of this oscillation by using stable attractor dynamics. We showed that this temporal-coding mechanism was supported by distinct subpopulations: One population that is an intrinsic oscillator, and two additional populations that couple the intrinsic and input oscillation in a multi-stable fashion. In Chapter 4, we proposed a method for fitting stochastic low-rank RNNs to neural data, based on variational sequential Monte Carlo methods, and derived a bound on the number of fixed points in low-rank RNNs when they have piecewise-linear activation functions. In Chapter 5 we investigated neural dynamics during a sequence working memory task. We fit RNNs to data of macaques, using our method from Chapter 4, adapted such that we could fit one RNN to recordings collected over multiple sessions. We found that the RNNs used attractor dynamics to encode the stimuli and showed that targeted perturbations can selectively change the memory and behavioral response of one item in the sequence.

6.2 FUTURE DIRECTIONS

6.2.0.1 *Towards interpretable realistic models*

We motivated the use of RNNs by arguing that they strike a balance between interpretability and expressivity. Interpretability can be considered from two complementary perspectives. First, models may be mechanistic, with variables and parameters corresponding directly to biological properties. Second, models may provide an intermediate level of description that bridges mechanistic details with complex data.

From a mechanistic perspective, RNNs make substantial simplifications relative to biological neural networks. One might hope that properties not explicitly modeled are either irrelevant for the phenomena of interest, or average out at the population level. However, it is unlikely that intrinsic features of neural computation, such as spike timing and plasticity, can be entirely neglected [80, 220] (see also Section 2.1.1.1). Recent work has incorporated greater biological realism into trained RNNs, including short-term synaptic plasticity [221] and spiking dynamics [85, 88–

90, 222]. In our own work, we have contributed models with Hodgkin–Huxley-type dynamics trained on a simple working memory task [73].

If additional biological detail cannot be ignored, this means facing engineering challenges. Training large, biophysically detailed network models, and fitting them to neural data in a probabilistic framework, remains challenging. Our inference framework (Chapter 4) exploits low-rank structure to decouple inference complexity from the number of neurons in the model. However, introducing richer biological detail can break this decoupling; Increasing both the effective dimensionality and the number of parameters to be estimated. Combining low-rank connectivity with other structural constraints such as sparsity might be important for increasing expressivity while preserving tractability [223].

The second notion of interpretability we consider arises from the ability of models to map high-dimensional neural activity onto lower-dimensional, more tractable representations. From this perspective, ignoring biological detail may be justified if the underlying dynamics can still be recovered, i.e., if there exists a form of universality where diverse architectures, biological or artificial, converge to similar latent dynamical systems when solving the same task [52]. Whether such equivalences generally hold remains an open question, even across different training runs of the same model [57]. If dynamics are indeed universal, approaches that focus directly on inferring latent dynamical systems (e.g., [41–46]) would be well justified.

Emphasizing biological realism on the one hand and abstract dynamical equivalence on the other, are complementary. Biologically grounded dynamical systems models may better reveal the structure underlying neural computation [222], as well as the conditions under which low-dimensional abstractions remain valid. However, developing models that include more biophysical detail than standard rate-based RNNs, while still being expressible in terms of interpretable low-dimensional dynamics is an important open challenge.

6.2.0.2 *In need of validation*

There is an ever-growing cornucopia of methods for fitting dynamical systems models to neural data. We could organize them into various axis, such as whether or not they are probabilistic [40–45, 65] or use deterministic transitions [46, 62–64, 66–69, 210], or by whether they are general latent dynamical systems [40–46, 64, 69, 210]) or have some level of mechanistic interpretation [62, 63, 65–68]. Inspired by the availability of larger datasets, recent work outlines various strategies tailored to larger multi-session datasets, by using session-specific input biases [211], session-specific low-rank perturbations [212], hypernetwork-generated weights [213], or session-specific weighting of a shared pool of basis vectors [214].

It is however anything but clear which of these methods gives the best results. There are efforts for benchmarking methods in computational neuroscience, such as the Neural Latents Benchmark [185], which focus on inferring smoothed Poisson rates and behavioral predictions. Few-shot inference or prediction benchmarks might

be potentially better at distinguishing models [224, 225]. Yet, these benchmarks do neither directly test the quality of the generative model, nor how well an any underlying ground truth dynamic or mechanism is recovered. Validating our model-fitting methods seems paramount: A recent paper showed that in certain regimes with partial observations [216], all evaluated methods for fitting RNNs could spuriously infer attractor dynamics when the underlying system did not have them.

One possible path forward is to develop benchmarks that explicitly evaluate generative performance, particularly the ability to generate data conditioned on held-out perturbations. Such a framework could directly probe whether models capture the underlying dynamics rather than merely fitting observed activity. The usefulness of testing responses to perturbations is supported by recent work showing that more realistic biologically grounded models can better predict experimental perturbation responses than models without connectivity constraints, in some settings [222]. Moreover, perturbation-based analyses have proven effective at distinguishing between competing dynamical mechanisms in vivo [200, 209]. A complementary approach is to examine trial-to-trial variability around mean dynamics [157], which can be interpreted as probing the system with small perturbations.

6.2.0.3 *Closing remarks*

In this work, we have contributed to the broader effort of using machine learning to learn interpretable models from neural data, with a particular focus on recurrent neural networks exhibiting explicit low-dimensional dynamics. We introduced new methods for fitting these models to data and analyzed them to gain insight into the neural mechanisms underlying working memory.

Looking ahead, an important challenge will be to extend these approaches to more complex and biologically realistic models while maintaining interpretability and tractability. At the same time, modeling progress should be accompanied by the development of good evaluation strategies, as without such validation, it remains unclear under which conditions fitted models faithfully capture the underlying neural computations. Nevertheless, reverse-engineered machine-learning models serve as useful hypothesis generators for the dynamics underlying complex neural data. Ultimately, it will only be through the interplay between such models and new experiments that we can genuinely understand the computations performed by neural systems

REFERENCES

1. Herculano-Houzel, S. The human brain in numbers: a linearly scaled-up primate brain. *Frontiers in Human Neuroscience* **3** (2009).
2. Von Bartheld, C. S., Bahney, J. & Herculano-Houzel, S. The search for true numbers of neurons and glial cells in the human brain: A review of 150 years of cell counting. *J Comp Neurol* **524**, 3865 (2016).
3. Drachman, D. A. Do we have brain to spare? *Neurology* **64**, 2004 (2005).
4. Pakkenberg, B., Pelvig, D., Marner, L., Bundgaard, M. J., Gundersen, H. J. G., Nyengaard, J. R. & Regeur, L. Aging and the human neocortex. *Exp Gerontol* **38**, 95 (2003).
5. Fuster, J. M. & Alexander, G. E. Neuron Activity Related to Short-Term Memory. *Science* **173**, 652 (1971).
6. Kubota, K. & Niki, H. Prefrontal cortical unit activity and delayed alternation performance in monkeys. *Journal of Neurophysiology* **34**, 337 (1971).
7. Amit, D. J. & Brunel, N. Model of global spontaneous activity and local structured activity during delay periods in the cerebral cortex. *Cereb Cortex* **7**, 237 (1997).
8. Wang, X. J. Synaptic reverberation underlying mnemonic persistent activity. *Trends Neurosci* **24**, 455 (2001).
9. Durstewitz, D., Seamans, J. K. & Sejnowski, T. J. Neurocomputational models of working memory. *Nat Neurosci* **3**, 1184 (2000).
10. Hopfield, J. J. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences* **79**, 2554 (1982).
11. Amari, S.-i. Dynamics of pattern formation in lateral-inhibition type neural fields. *Biological Cybernetics* **27**, 77 (1977).
12. Miller, E. K., Erickson, C. A. & Desimone, R. Neural mechanisms of visual working memory in prefrontal cortex of the macaque. *Journal of Neuroscience* **16**, 5154 (1996).
13. Stokes, M. G., Kusunoki, M., Sigala, N., Nili, H., Gaffan, D. & Duncan, J. Dynamic coding for cognitive control in prefrontal cortex. *Neuron* **78**, 364 (2013).
14. Sreenivasan, K. K., Curtis, C. E. & D'Esposito, M. Revisiting the role of persistent neural activity during working memory. *Trends Cogn Sci* **18**, 82 (2014).

15. Stokes, M. G. 'Activity-silent' working memory in prefrontal cortex: a dynamic coding framework. *Trends Cogn Sci* **19**, 394 (2015).

16. Lundqvist, M., Herman, P. & Miller, E. K. Working Memory: Delay Activity, Yes! Persistent Activity? Maybe Not. *J Neurosci* **38**, 7013 (2018).

17. Singh, R. & Eliasmith, C. Higher-Dimensional Neurons Explain the Tuning and Dynamics of Working Memory Cells. *Journal of Neuroscience* **26**, 3667 (2006).

18. Druckmann, S. & Chklovskii, D. B. Neuronal Circuits Underlying Persistent Representations Despite Time Varying Activity. *Current Biology* **22**, 2095 (2012).

19. Murray, J. D., Bernacchia, A., Roy, N. A., Constantinidis, C., Romo, R. & Wang, X.-J. Stable population coding for working memory coexists with heterogeneous neural dynamics in prefrontal cortex. *Proceedings of the National Academy of Sciences* **114**, 394 (2017).

20. Stevenson, I. H. & Kording, K. P. How advances in neural recording affect data analysis. *Nat Neurosci* **14**, 139 (2011).

21. Urai, A. E., Doiron, B., Leifer, A. M. & Churchland, A. K. Large-scale neural recordings call for new insights to link brain and behavior. *Nature Neuroscience* **25**, 11 (2022).

22. Turing, A. M. Computing machinery and intelligence. *Mind* **59**, 433 (1950).

23. Goodfellow, I., Bengio, Y. & Courville, A. *Deep Learning* (MIT Press, 2016).

24. Ho, J., Jain, A. & Abbeel, P. *Denoising diffusion probabilistic models* in *Proceedings of the 34th International Conference on Neural Information Processing Systems* (2020).

25. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł. & Polosukhin, I. *Attention is all you need* in *Proceedings of the 31st International Conference on Neural Information Processing Systems* (2017).

26. Vetter, J., Macke, J. H. & Gao, R. Generating realistic neurophysiological time series with denoising diffusion probabilistic models. *Patterns* **5**, 101047 (2024).

27. Kapoor, J., Schulz, A., Vetter, J., Pei, F. C., Gao, R. & Macke, J. H. *Latent Diffusion for Neural Spiking Data* in *Advances in Neural Information Processing Systems (NeurIPS)* (2024).

28. Ye, J. & Pandarinath, C. Representation learning for neural population activity with Neural Data Transformers. *Neurons, Behavior, Data analysis, and Theory* **5** (2021).

29. Ye, J., Collinger, J. L., Wehbe, L. & Gaunt, R. *Neural Data Transformer 2: Multi-context Pretraining for Neural Spiking Activity* in *Thirty-seventh Conference on Neural Information Processing Systems* (2023).

30. Azabou, M., Arora, V., Ganesh, V., Mao, X., Nachimuthu, S. B., Mendelson, M. J., Richards, B. A., Perich, M. G., Lajoie, G. & Dyer, E. L. *A Unified, Scalable Framework for Neural Population Decoding* in *Thirty-seventh Conference on Neural Information Processing Systems* (2023).

31. Cunningham, J. P. & Yu, B. M. Dimensionality reduction for large-scale neural recordings. *Nature Neuroscience* **17**, 1500 (2014).

32. Chaudhuri, R., Gerçek, B., Pandey, B., Peyrache, A. & Fiete, I. The intrinsic attractor manifold and population dynamics of a canonical cognitive circuit across waking and sleep. *Nature Neuroscience* **22**, 1512 (2019).

33. Gallego, J. A., Perich, M. G., Miller, L. E. & Solla, S. A. Neural Manifolds for the Control of Movement. *Neuron* **94**, 978 (2017).

34. Strogatz, S. *Nonlinear dynamics and chaos: with applications to physics, biology, chemistry, and engineering* 2nd ed. (Westview Press, 2015).

35. Churchland, M. M., Yu, B. M., Sahani, M. & Shenoy, K. V. Techniques for extracting single-trial activity patterns from large-scale neural recordings. *Current Opinion in Neurobiology* **17**, 609 (2007).

36. Shenoy, K. V., Sahani, M. & Churchland, M. M. Cortical Control of Arm Movements: A Dynamical Systems Perspective. *Annual Review of Neuroscience* **36**, 337 (2013).

37. Vyas, S., Golub, M. D., Sussillo, D. & Shenoy, K. V. Computation Through Neural Population Dynamics. *Annual Review of Neuroscience* **43**, 249 (2020).

38. Barack, D. L. & Krakauer, J. W. Two views on the cognitive brain. *Nature Reviews Neuroscience* **22**, 359 (2021).

39. Durstewitz, D., Koppe, G. & Thurm, M. I. Reconstructing computational system dynamics from neural data with recurrent neural networks. *Nature Reviews Neuroscience* **24**, 693 (2023).

40. Petreska, B., Yu, B. M., Cunningham, J. P., Santhanam, G., Ryu, S., Shenoy, K. V. & Sahani, M. *Dynamical segmentation of single trials from population neural data* in *Advances in Neural Information Processing Systems* (2011).

41. Macke, J. H., Buesing, L., Cunningham, J. P., Yu, B. M., Shenoy, K. V. & Sahani, M. *Empirical models of spiking in neural populations* in *Advances in Neural Information Processing Systems* (2011).

42. Linderman, S., Johnson, M., Miller, A., Adams, R., Blei, D. & Paninski, L. *Bayesian Learning and Inference in Recurrent Switching Linear Dynamical Systems* in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics* (2017).

43. Kim, T. D., Luo, T. Z., Can, T., Krishnamurthy, K., Pillow, J. W. & Brody, C. D. *Flow-field inference from neural data using deep recurrent networks* bioRxiv:2023.11.14.567136. 2023.

44. Zhao, Y. & Linderman, S. *Revisiting Structured Variational Autoencoders* in *Proceedings of the 40th International Conference on Machine Learning* (2023).
45. Dowling, M., Zhao, Y. & Park, I. M. *eXponential FAmily Dynamical Systems (XFADS): Large-scale nonlinear Gaussian state-space modeling* in *Advances in Neural Information Processing Systems* (2024).
46. Pandarinath, C., O'Shea, D. J., Collins, J., Jozefowicz, R., Stavisky, S. D., Kao, J. C., Trautmann, E. M., Kaufman, M. T., Ryu, S. I., Hochberg, L. R., Henderson, J. M., Shenoy, K. V., Abbott, L. F. & Sussillo, D. Inferring single-trial neural population dynamics using sequential auto-encoders. *Nature Methods* **15**, 805 (2018).
47. Barak, O. Recurrent neural networks as versatile tools of neuroscience research. *Current Opinion in Neurobiology* **46**, 1 (2017).
48. Funahashi, K. & Nakamura, Y. Approximation of dynamical systems by continuous time recurrent neural networks. *Neural Networks* **6**, 801 (1993).
49. Beiran, M., Dubreuil, A., Valente, A., Mastrogiuseppe, F. & Ostojic, S. Shaping Dynamics With Multiple Populations in Low-Rank Recurrent Networks. *Neural Computation* **33**, 1572 (2021).
50. Zipser, D. Recurrent Network Model of the Neural Mechanism of Short-Term Active Memory. *Neural Comput* **3**, 179 (1991).
51. Sussillo, D. & Barak, O. Opening the Black Box: Low-Dimensional Dynamics in High-Dimensional Recurrent Neural Networks. *Neural Computation* **25**, 626 (2013).
52. Maheswaranathan, N., Williams, A., Golub, M., Ganguli, S. & Sussillo, D. *Universality and individuality in neural dynamics across large populations of recurrent networks* in *Advances in neural information processing systems* (2019).
53. Driscoll, L. N., Shenoy, K. & Sussillo, D. Flexible multitask computation in recurrent networks utilizes shared dynamical motifs. *Nature Neuroscience* **27**, 1349 (2024).
54. Song, H. F., Robert Yang, G. & Wang, X.-J. Training Excitatory-Inhibitory Recurrent Neural Networks for Cognitive Tasks: A Simple and Flexible Framework. *PLOS Computational Biology* **12**, 1 (2016).
55. Mante, V., Sussillo, D., Shenoy, K. V. & Newsome, W. T. Context-dependent computation by recurrent dynamics in prefrontal cortex. *Nature* **503**, 78 (2013).
56. Dubreuil, A., Valente, A., Beiran, M., Mastrogiuseppe, F. & Ostojic, S. The role of population structure in computations through neural dynamics. *Nature Neuroscience* **25**, 783 (2022).
57. Turner, E., Dabholkar, K. V. & Barak, O. *Charting and Navigating the Space of Solutions for Recurrent Neural Networks* in *Advances in Neural Information Processing Systems* (2021).

58. Barbosa, J., Proville, R., Rodgers, C. C., DeWeese, M. R., Ostojic, S. & Boubenec, Y. Early selection of task-relevant features through population gating. *Nature Communications* **14**, 6837 (2023).

59. Mastrogiuseppe, F. & Ostojic, S. Linking Connectivity, Dynamics, and Computations in Low-Rank Recurrent Neural Networks. *Neuron* **99**, 609 (2018).

60. Eliasmith, C. & Anderson, C. H. *Neural Engineering (Computational Neuroscience Series): Computational, Representation, and Dynamics in Neurobiological Systems* (MIT Press, Cambridge, MA, USA, 2002).

61. Claudi, F., Chandra, S. & Fiete, I. R. A theory and recipe to construct general and biologically plausible integrating continuous attractor neural networks (2025).

62. Sussillo, D. & Abbott, L. F. Generating Coherent Patterns of Activity from Chaotic Neural Networks. *Neuron* **63**, 544 (2009).

63. Sussillo, D., Churchland, M. M., Kaufman, M. T. & Shenoy, K. V. A neural network that finds a naturalistic solution for the production of muscle activity. *Nature neuroscience* **18**, 1025 (2015).

64. Hess, F., Monfared, Z., Brenner, M. & Durstewitz, D. *Generalized teacher forcing for learning chaotic dynamics* in *Proceedings of the 40th International Conference on Machine Learning* (2023).

65. Durstewitz, D. A state space approach for piecewise-linear recurrent neural networks for identifying computational dynamics from neural measurements. *PLOS Computational Biology* **13**, 1 (2017).

66. Perich, M. G., Arlt, C., Soares, S., Young, M. E., Mosher, C. P., Minxha, J., Carter, E., Rutishauser, U., Rudebeck, P. H., Harvey, C. D. & Rajan, K. *Inferring brain-wide interactions using data-constrained recurrent neural network models* bioRxiv:2020.12.18.423348. 2021.

67. Valente, A., Pillow, J. W. & Ostojic, S. *Extracting computational mechanisms from neural data using low-rank RNNs* in *Advances in Neural Information Processing Systems* **35** (2022).

68. Dinc, F., Shai, A., Schnitzer, M. & Tanaka, H. *CORNN: Convex optimization of recurrent neural networks for rapid inference of neural dynamics* in *Thirty-seventh Conference on Neural Information Processing Systems* (2023).

69. Brenner, M., Hess, F., Koppe, G. & Durstewitz, D. *Integrating Multimodal Data for Joint Generative Modeling of Complex Dynamics* in *Forty-first International Conference on Machine Learning* (2024).

70. Eisenmann, L., Monfared, Z., Göring, N. & Durstewitz, D. *Bifurcations and loss jumps in RNN training* in *Advances in Neural Information Processing Systems* (2023).

71. Dabholkar, K. V. & Barak, O. *Finding separatrices of dynamical flows with Deep Koopman Eigenfunctions* in *The Thirty-ninth Annual Conference on Neural Information Processing Systems* (2025).

72. Eisenmann, L., Brändle, A., Monfared, Z. & Durstewitz, D. *Detecting Invariant Manifolds in ReLU-Based RNNs* arXiv:2510.03814. 2025.

73. Deistler, M., Kadhim, K. L., Pals, M., Beck, J., Huang, Z., Gloeckler, M., Lappalainen, J. K., Schröder, C., Berens, P., Gonçalves, P. J. & Macke, J. H. Jaxley: differentiable simulation enables large-scale training of detailed biophysical models of neural dynamics. *Nature Methods* **22**, 2649 (2025).

74. Chen, J. *et al.* Flexible control of sequence working memory in the macaque frontal cortex. *Neuron* **112**, 3132 (2024).

75. Khajehabdollahi, S., Zeraati, R., Giannakakis, E., Schäfer, T. J., Martius, G. & Levina, A. *Emergent mechanisms for long timescales depend on training curriculum and affect performance in memory tasks* in *The Twelfth International Conference on Learning Representations* (2024).

76. Schmutz, V., Haydaroğlu, A., Wang, S., Feng, Y., Carandini, M. & Harris, K. D. *High-dimensional neuronal activity from low-dimensional latent dynamics: a solvable model* in *The Thirty-ninth Annual Conference on Neural Information Processing Systems* (2025).

77. Sompolinsky, H., Crisanti, A. & Sommers, H. J. Chaos in Random Neural Networks. *Phys. Rev. Lett.* **61**, 259 (3 1988).

78. Kadmon, J. & Sompolinsky, H. Transition to chaos in random neuronal networks. *Physical Review X* **5**, 041030 (2015).

79. Li, P., Cornford, J., Ghosh, A. & Richards, B. *Learning better with Dale's Law: A Spectral Perspective* in *Advances in Neural Information Processing Systems* (2023).

80. Brette, R. Philosophy of the Spike: Rate-Based vs. Spike-Based Theories of the Brain. *Frontiers in Systems Neuroscience* **9** (2015).

81. Dayan, P. & Abbott, L. *Theoretical Neuroscience* (MIT Press, Cambridge, MA, 2001).

82. Ostojic, S. Two types of asynchronous activity in networks of excitatory and inhibitory spiking neurons. *Nature Neuroscience* **17**, 594 (2014).

83. Engelken, R., Farkhooi, F., Hansel, D., van Vreeswijk, C. & Wolf, F. A reanalysis of "Two types of asynchronous activity in networks of excitatory and inhibitory spiking neurons". *F1000Research* **5**, 2043 (2016).

84. Schmutz, V., Brea, J. & Gerstner, W. Emergent Rate-Based Dynamics in Duplicate-Free Populations of Spiking Neurons. *Phys. Rev. Lett.* **134**, 018401 (1 2025).

85. Kim, R., Li, Y. & Sejnowski, T. J. Simple framework for constructing functional spiking recurrent neural networks. *Proceedings of the National Academy of Sciences* **116**, 22811 (2019).

86. DePasquale, B., Sussillo, D., Abbott, L. & Churchland, M. M. The centrality of population-level factors to network computation is demonstrated by a versatile approach for training spiking networks. *Neuron* **111**, 631 (2023).

87. Cimeša, L., Ciric, L. & Ostojic, S. Geometry of population activity in spiking networks with low-rank structure. *PLOS Computational Biology* **19**, 1 (2023).

88. Zenke, F. & Ganguli, S. SuperSpike: Supervised Learning in Multilayer Spiking Neural Networks. *Neural Comput* **30**, 1514 (2018).

89. Bellec, G., Scherr, F., Subramoney, A., Hajek, E., Salaj, D., Legenstein, R. & Maass, W. A solution to the learning dilemma for recurrent networks of spiking neurons. *Nature Communications* **11**, 3625 (2020).

90. Engelken, R. *SparseProp: Efficient Event-Based Simulation and Training of Sparse Recurrent Spiking Neural Networks* in *Advances in Neural Information Processing Systems* (2023).

91. Kim, R. & Sejnowski, T. J. Strong inhibitory signaling underlies stable temporal dynamics and working memory in spiking neural networks. *Nature Neuroscience* **24**, 129 (2021).

92. Werbos, P. Backpropagation Through Time: What It. *Proceedings of the IEEE* **78** (1990).

93. Pascanu, R., Mikolov, T. & Bengio, Y. *On the difficulty of training recurrent neural networks* in *Proceedings of the 30th International Conference on Machine Learning* (Atlanta, Georgia, USA, 2013).

94. Engelken, R. Gradient flossing: Improving gradient descent through dynamic control of jacobians. *Advances in Neural Information Processing Systems* **36**, 10412 (2023).

95. Langton, C. G. Computation at the edge of chaos: Phase transitions and emergent computation. *Physica D: nonlinear phenomena* **42**, 12 (1990).

96. Bertschinger, N. & Natschläger, T. Real-Time Computation at the Edge of Chaos in Recurrent Neural Networks. *Neural Computation* **16**, 1413 (2004).

97. Farrell, M., Recanatesi, S., Moore, T., Lajoie, G. & Shea-Brown, E. Gradient-based learning drives robust representations in recurrent neural networks by balancing compression and expansion. *Nature Machine Intelligence* **4**, 564 (2022).

98. Maass, W., Natschläger, T. & Markram, H. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural computation* **14**, 2531 (2002).

99. Jaeger, H. *Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the echo state network approach* **1** (GMD-Forschungszentrum Informationstechnik Bonn, 2002).

100. Doya, K. Bifurcations of Recurrent Neural Networks in Gradient Descent Learning. *IEEE Transactions on Neural Networks* (1993).

101. Seung, H. S. How the brain keeps the eyes still. *Proceedings of the National Academy of Sciences* **93**, 13339 (1996).

102. Schuessler, F., Dubreuil, A., Mastrogiuseppe, F., Ostojic, S. & Barak, O. Dynamics of random recurrent networks with correlated low-rank structure. *Physical Review Research* **2**, 013111 (2020).

103. Valente, A., Ostojic, S. & Pillow, J. W. Probing the Relationship Between Latent Linear Dynamical Systems and Low-Rank Recurrent Neural Network Models. *Neural Computation* **34**, 1871 (2022).

104. Chen, R. T. Q., Rubanova, Y., Bettencourt, J. & Duvenaud, D. K. *Neural Ordinary Differential Equations* in *Advances in Neural Information Processing Systems* (2018).

105. Goldman, M. S. Memory without feedback in a neural network. *Neuron* **61**, 621 (2009).

106. Christodoulou, G. & Vogels, T. The eigenvalue value (in neuroscience). *OSF Preprints* **10** (2022).

107. Katz, G. E. & Reggia, J. A. Using directional fibers to locate fixed points of recurrent neural networks. *IEEE transactions on neural networks and learning systems* **29**, 3636 (2017).

108. Sato, S. & Gohara, K. Poincaré Mapping of continuous Recurrent Neural Networks excited by Temporal External Input. *Int. J. Bifurc. Chaos* **10**, 1677 (2000).

109. Skarda, C. A. & Freeman, W. J. How brains make chaos in order to make sense of the world. *Behavioral and Brain Sciences* **10**, 161 (1987).

110. Curto, C., Geneson, J. & Morrison, K. Fixed Points of Competitive Threshold-Linear Networks. *Neural Computation* **31**, 94 (2019).

111. Brenner, M., Hess, F., Mikhaeil, J. M., Bereska, L. F., Monfared, Z., Kuo, P.-C. & Durstewitz, D. *Tractable Dendritic RNNs for Reconstructing Nonlinear Dynamical Systems* in *Proceedings of the 39th International Conference on Machine Learning* (2022).

112. Morrison, K., Degeratu, A., Itskov, V. & Curto, C. Diversity of Emergent Dynamics in Competitive Threshold-Linear Networks. *SIAM Journal on Applied Dynamical Systems* **23**, 855 (2024).

113. Brändle, A., Eisenmann, L., Götz, F. & Durstewitz, D. *Continuous-Time Piecewise-Linear Recurrent Neural Networks* arXiv:2602.15649. 2026.

114. Brennan, C., Aggarwal, A., Pei, R., Sussillo, D. & Proekt, A. One dimensional approximations of neuronal dynamics reveal computational strategy. *PLOS Computational Biology* **19**, 1 (2023).

115. Le, T. A., Igl, M., Rainforth, T., Jin, T. & Wood, F. *Auto-Encoding Sequential Monte Carlo* in *International Conference on Learning Representations* (2018).

116. Maddison, C. J., Lawson, J., Tucker, G., Heess, N., Norouzi, M., Mnih, A., Doucet, A. & Teh, Y. *Filtering Variational Objectives* in *Advances in Neural Information Processing Systems* (2017).

117. Naesseth, C., Linderman, S., Ranganath, R. & Blei, D. *Variational Sequential Monte Carlo* in *Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics* (2018).

118. Särkkä, S. & Svensson, L. *Bayesian Filtering and Smoothing* 2nd (Cambridge University Press, 2023).

119. Kalman, R. E. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering* **82**, 35 (1960).

120. Doucet, A. & Johansen, A. M. A Tutorial on Particle Filtering and Smoothing: Fifteen Years Later. *The Oxford Handbook of Nonlinear Filtering*, 656 (2011).

121. Liu, J. S. *Monte Carlo Strategies in Scientific Computing* (Springer, New York, 2001).

122. Zenn, J. & Bamler, R. *Resampling Gradients Vanish in Differentiable Sequential Monte Carlo Samplers* in *The First Tiny Papers Track at ICLR 2023, Tiny Papers @ ICLR 2023* (2023).

123. Kingma, D. P. & Ba, J. *Adam: A Method for Stochastic Optimization* in *International Conference on Learning Representations* (2015).

124. Kingma, D. P. & Welling, M. An Introduction to Variational Autoencoders. *Foundations and Trends® in Machine Learning* **12**, 307 (2019).

125. Burda, Y., Grosse, R. & Salakhutdinov, R. *Importance Weighted Autoencoders* in *International Conference on Learning Representations (ICLR)* (2016).

126. Cremer, C., Morris, Q. & Duvenaud, D. *Reinterpreting Importance-Weighted Autoencoders* in *International Conference on Learning Representations (ICLR) Workshop* (2017).

127. Tucker, G., Lawson, D., Gu, S. & Maddison, C. J. *Doubly Reparameterized Gradient Estimators for Monte Carlo Objectives* in *International Conference on Learning Representations (ICLR)* (2019).

128. Krishnan, R. G., Shalit, U. & Sontag, D. *Deep Kalman Filters* arXiv:1511.05121. 2015.

129. Mi, L., Xu, R., Prakhya, S., Lin, A., Shavit, N., Samuel, A. D. T. & C. Turaga, S. *Connectome-constrained Latent Variable Models of Whole-Brain Neural Activity* in *Proc. 10th International Conference on Learning Representations (ICLR 2022)* (2022).

130. Evensen, G. Sequential data assimilation with a nonlinear quasi-geostrophic model using Monte Carlo methods to forecast error statistics. *Journal of Geophysical Research: Oceans* **99**, 10143 (1994).

131. Wan, E. A. & van der Merwe, R. *The Unscented Kalman Filter for Nonlinear Estimation* in *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium* (2000), 153.

132. Mikhaeil, J. M., Monfared, Z. & Durstewitz, D. *On the difficulty of learning chaotic dynamics with RNNs* in *Proceedings of the 36th International Conference on Neural Information Processing Systems* (Red Hook, NY, USA, 2022).

133. Zhao, Y., Nassar, J., Jordan, I., Bugallo, M. & Park, I. Streaming Variational Monte Carlo. *IEEE Transactions on Pattern Analysis & Machine Intelligence* **45**, 1150 (2023).

134. Delcomyn, F. Neural Basis of Rhythmic Behavior in Animals. *Science* **210**, 492 (1980).

135. Marder, E. & Bucher, D. Understanding Circuit Dynamics Using the Stomatogastric Nervous System of Lobsters and Crabs. *Annual Review of Physiology* **69**, 291 (2007).

136. Buzsáki, G. *Rhythms of the Brain* 1st ed. (Oxford University Press, 2006).

137. Buzsáki, G. & Tingley, D. Space and Time: The Hippocampus as a Sequence Generator. *Trends in Cognitive Sciences* **22**, 853 (2018).

138. Lisman, J. E. & Jensen, O. The Theta-Gamma Neural Code. *Neuron* **77**, 1002 (2013).

139. Lisman, J. The theta/gamma discrete phase code occuring during the hippocampal phase precession may be a more general brain coding scheme. *Hippocampus* **15**, 913 (2005).

140. Lisman, J. E. & Idiart, M. A. P. Storage of 7 ± 2 Short-Term Memories in Oscillatory Subcycles. *Science* **267**, 1512 (1995).

141. Hopfield, J. J. Pattern recognition computation using action potential timing for stimulus representation. *Nature* **376**, 33 (1995).

142. Fell, J. & Axmacher, N. The role of phase synchronization in memory processes. *Nature Reviews Neuroscience* **12**, 105 (2011).

143. O'Keefe, J. & Recce, M. L. Phase relationship between hippocampal place units and the EEG theta rhythm. *Hippocampus* **3**, 317 (1993).

144. Kraskov, A., Quiroga, R. Q., Reddy, L., Fried, I. & Koch, C. Local Field Potentials and Spikes in the Human Medial Temporal Lobe are Selective to Image Category. *Journal of Cognitive Neuroscience* **19**, 479 (2007).

145. Turesson, H., Logothetis, N. & Hoffman, K. Category-selective phase coding in the superior temporal sulcus. *Proceedings of the National Academy of Sciences* **109**, 19438 (2012).

146. Watrous, A. J., Deuker, L., Fell, J., Axmacher, N. & Eichenbaum, H. Phase-amplitude coupling supports phase coding in human ECoG. *eLife* **4**, e07886 (2015).

147. Siegel, M., Warden, M. R. & Miller, E. K. Phase-dependent neuronal coding of objects in short-term memory. *Proceedings of the National Academy of Sciences* **106**, 21341 (2009).

148. Kayser, C., Ince, R. A. A. & Panzeri, S. Analysis of Slow (Theta) Oscillations as a Potential Temporal Reference Frame for Information Coding in Sensory Cortices. *PLOS Computational Biology* **8**, 1 (2012).

149. Liebe, S., Hoerzer, G. M., Logothetis, N. K. & Rainer, G. Theta coupling between V4 and prefrontal cortex predicts visual short-term memory performance. *Nature neuroscience* **15**, 456 (2012).

150. Kamiński, J., Brzezicka, A., Mamelak, A. N. & Rutishauser, U. Combined Phase-Rate Coding by Persistently Active Neurons as a Mechanism for Maintaining Multiple Items in Working Memory in Humans. *Neuron* **106**, 256 (2020).

151. Watrous, A. J., Miller, J., Qasim, S. E., Fried, I. & Jacobs, J. Phase-tuned neuronal firing encodes human contextual representations for navigational goals. *eLife* **7**, e32554 (2018).

152. Langdon, C. & Engel, T. A. Latent circuit inference from heterogeneous neural responses during cognitive tasks. *Nature Neuroscience* **28**, 665 (2025).

153. Finkelstein, A., Fontolan, L., Economo, M. N., Li, N., Romani, S. & Svoboda, K. Attractor dynamics gate cortical information flow during decision-making. *Nature Neuroscience* **24**, 843 (2021).

154. Duecker, K., Idiart, M., van Gerven, M. & Jensen, O. Oscillations in an artificial neural network convert competing inputs into a temporal code. *PLOS Computational Biology* **20**, 1 (2024).

155. Mizuseki, K., Sirota, A., Pastalkova, E. & Buzsáki, G. Theta Oscillations Provide Temporal Windows for Local Circuit Computation in the Entorhinal-Hippocampal Loop. *Neuron* **64**, 267 (2009).

156. Mizuseki, K., Sirota, A., Pastalkova, E. & Buzsáki, G. *Multi-unit recordings from the rat hippocampus made during open field foraging* Database: CRCNS [Internet]. 2009.

157. Galgali, A. R., Sahani, M. & Mante, V. Residual dynamics resolves recurrent contributions to neural computation. *Nature Neuroscience* **26**, 326 (2023).

158. Pahor, A. & Jaušovec, N. The Effects of Theta and Gamma tACS on Working Memory and Electrophysiology. *Frontiers in Human Neuroscience* **11** (2018).

159. Scheffer-Teixeira, R. & Tort, A. B. On cross-frequency phase-phase coupling between theta and gamma oscillations in the hippocampus. *eLife* **5**, e20515 (2016).

160. Stankovski, T., Pereira, T., McClintock, P. V. E. & Stefanovska, A. Coupling functions: Universal insights into dynamical interaction mechanisms. *Reviews of Modern Physics* **89**, 045001 (2017).

161. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M. & Duchesnay, E. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* **12**, 2825 (2011).

162. Stankovski, T., Duggento, A., McClintock, P. V. E. & Stefanovska, A. Inference of Time-Evolving Coupled Dynamical Systems in the Presence of Noise. *Physical Review Letters* **109**, 024101 (2012).

163. Khona, M. & Fiete, I. R. Attractor and integrator networks in the brain. *Nature Reviews Neuroscience* **23**, 744 (2022).

164. Susman, L., Brenner, N. & Barak, O. Stable memory with unstable synapses. *Nature Communications* **10**, 4441 (2019).

165. Rajan, K., Harvey, C. D. & Tank, D. W. Recurrent Network Models of Sequence Generation and Memory. *Neuron* **90**, 128 (2016).

166. Park, I. M., Ságodi, Á. & Sokół, P. A. *Persistent learning signals and working memory without continuous attractors* arXiv:2308.12585. 2023.

167. Hahn, L. A., Balakhonov, D., Lundqvist, M., Nieder, A. & Rose, J. Oscillations without cortex: Working memory modulates brainwaves in the endbrain of crows. *Progress in Neurobiology* **219**, 102372 (2022).

168. Garcia, S., Guarino, D., Jaillet, F., Jennings, T., Pröpper, R., Rautenberg, P., Rodgers, C., Sobolev, A., Wachtler, T., Yger, P. & Davison, A. Neo: an object model for handling electrophysiology data in multiple formats. *Frontiers in Neuroinformatics* **8** (2014).

169. Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C. J., Polat, İ., Feng, Y., Moore, E. W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P. & SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods* **17**, 261 (2020).

170. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J. & Chintala, S. *PyTorch: An Imperative Style, High-Performance Deep Learning Library* in *Advances in Neural Information Processing Systems* (2019).

171. Hirsch, M. W., Smale, S. & Devaney, R. L. *Differential Equations, Dynamical Systems, and an Introduction to Chaos* 3rd ed. (Academic Press, 2013).

172. Pals, M., Macke, J. H. & Barak, O. Trained recurrent neural networks develop phase-locked limit cycles in a working memory task. *PLOS Computational Biology* **20**, 1 (2024).

173. Sourmpis, C., Petersen, C., Gerstner, W. & Bellec, G. *Trial matching: capturing variability with data-constrained spiking neural networks* in *Advances in Neural Information Processing Systems* (2023).

174. Zaslavsky, T. Facing up to arrangements: face-count formulas for partitions of space by hyperplanes. *Memoirs of American Mathematical Society* **154**, 1 (1975).

175. Schalk, G., McFarland, D., Hinterberger, T., Birbaumer, N. & Wolpaw, J. BCI2000: a general-purpose brain-computer interface (BCI) system. *IEEE Transactions on Biomedical Engineering* **51**, 1034 (2004).

176. Moody, G., Mark, R. & Goldberger, A. PhysioNet: a research resource for studies of complex physiologic and biomedical signals. *Computers in cardiology* **27**, 179 (2000).

177. Brunton, S. L., L., P. J. & Kutz, J. N. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Science* **113**, 3932 (2016).

178. Rusch, T. K., Mishra, S., Erichson, N. B. & Mahoney, M. W. *Long Expressive Memory for Sequence Modeling* in *International Conference on Learning Representations* (2022).

179. O'Keefe, J. Place units in the hippocampus of the freely moving rat. *Experimental Neurology* **51**, 78 (1976).

180. Grosmark, A. D. & Buzsáki, G. Diversity in neural firing dynamics supports both rigid and learned hippocampal sequences. *Science* **351**, 1440 (2016).

181. Chen, Z., Grosmark, A. D., Penagos, H. & Wilson, M. A. Uncovering representations of sleep-associated hippocampal ensemble spike activity. *Scientific Reports* **6** (2016).

182. Grosmark A.D., L. J. & Buzsáki, G. *Recordings from hippocampal area CA1, PRE, during and POST novel spatial learning* Database: CRCNS. 2016.

183. Zhou, D. & Wei, X.-X. *Learning identifiable and interpretable latent models of high-dimensional neural activity using pi-VAE* in *Advances in Neural Information Processing Systems* (2020).

184. Santhanam, G., Yu, B. M., Gilja, V., Ryu, S. I., Afshar, A., Sahani, M. & Shenoy, K. V. Factor-Analysis Methods for Higher-Performance Neural Prostheses. *Journal of Neurophysiology* **102**, 1315 (2009).

185. Pei, F., Ye, J., Zoltowski, D. M., Wu, A., Chowdhury, R. H., Sohn, H., O'Doherty, J. E., Shenoy, K. V., Kaufman, M. T., Churchland, M., Jazayeri, M., Miller, L. E., Pillow, J., Park, I. M., Dyer, E. L. & Pandarinath, C. *Neural Latents Benchmark '21: Evaluating latent variable models of neural population activity* in *Advances in Neural Information Processing Systems (NeurIPS), Track on Datasets and Benchmarks* (2021).

186. Versteeg, C., Sedler, A. R., McCart, J. D. & Pandarinath, C. *Expressive dynamics models with nonlinear injective readouts enable reliable recovery of latent features from neural activity* in *Proceedings of the 2nd NeurIPS Workshop on Symmetry and Geometry in Neural Representations* (2024).

187. Li, X., Wong, T.-K. L., Chen, R. T. Q. & Duvenaud, D. *Scalable Gradients for Stochastic Differential Equations* in *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics* (2020).

188. Deng, R., Brubaker, M. A., Mori, G. & Lehrmann, A. *Continuous Latent Process Flows* in *Advances in Neural Information Processing Systems* (2021).

189. Sottinen, T. & Särkkä, S. Application of Girsanov theorem to particle filtering of discretely observed continuous-time non-linear systems. *Bayesian Analysis* **3**, 555 (2008).

190. Baddeley, A. & Hitch. in *Recent Advances in Learning and Motivation* (ed Bower, G.) 47 (Academic Press, United States, 1974).

191. Goldman-Rakic, P. S. Cellular basis of working memory. *Neuron* **14**, 477 (1995).

192. Kamiński, J., Sullivan, S., Chung, J. M., Ross, I. B., Mamelak, A. N. & Rutishauser, U. Persistently active neurons in human medial frontal and medial temporal lobe support working memory. *Nat Neurosci* **20**, 590 (2017).

193. Paluch, K., Magnuski, M., Średniawa, W., Ivanovski, D., Rysz, A., Służewska-Niedźwiedź, M., Pasterski, T., Fortuna, W., Smarzewska, K., Reinacher, P. C., Kaczor, S., Tabakow, P., Babu, H. & Kamiński, J. Unattended working memory items are coded by persistent activity in human medial temporal lobe neurons. *Nature Human Behaviour* **9**, 2099 (2025).

194. Xie, Y., Hu, P., Li, J., Chen, J., Song, W., Wang, X.-J., Yang, T., Dehaene, S., Tang, S., Min, B. & Wang, L. Geometry of sequence working memory in macaque prefrontal cortex. *Science* **375**, 632 (2022).

195. Tian, Z., Chen, J., Zhang, C., Min, B., Xu, B. & Wang, L. Mental programming of spatial sequences in working memory in the macaque frontal cortex. *Science* **385**, eadp6091 (2024).

196. Santo-Angles, A., Yang, J., Zhou, Y., Chu, W. K. H., Lindsay, G. W. & Sreenivasan, K. K. Neural Subspaces Encode Sequential Working Memory, but Neural Sequences Do Not (2025).

197. Smolensky, P. Tensor Product Variable Binding and the Representation of Symbolic Structures in Connectionist Systems. *Artificial Intelligence* **46**, 159 (1990).

198. Whittington, J. C., Dorrell, W., Behrens, T. E., Ganguli, S. & El-Gaby, M. A tale of two algorithms: Structured slots explain prefrontal sequence memory and are unified with hippocampal cognitive maps. *Neuron* **113**, 321 (2025).

199. Wang, X.-J. 50 years of mnemonic persistent activity: quo vadis? *Trends in Neurosciences* **44**, 888 (2021).

200. Inagaki, H. K., Fontolan, L., Romani, S. & Svoboda, K. Discrete attractor dynamics underlies persistent activity in the frontal cortex. *Nature* **566**, 212 (2019).

201. Savin, C. & Triesch, J. Emergence of task-dependent representations in working memory circuits. *Frontiers in Computational Neuroscience* **8**, 57 (2014).

202. Keller, T. A., Muller, L., Sejnowski, T. & Welling, M. *Traveling Waves Encode The Recent Past and Enhance Sequence Learning* in *The Twelfth International Conference on Learning Representations* (2024).

203. Pals, M., Sağtekin, A. E., Pei, F. C., Gloeckler, M. & Macke, J. H. *Inferring stochastic low-rank recurrent neural networks from neural data* in *The Thirty-eighth Annual Conference on Neural Information Processing Systems* (2024).

204. Wang, M., Fusi, S. & Stachenfeld, K. *Brain-like slot representation for sequence working memory in recurrent neural networks* in *Second Workshop on Representational Alignment at ICLR 2025* (2025).

205. Kaufman, M. T., Seely, J. S., Sussillo, D., Ryu, S. I., Shenoy, K. V. & Churchland, M. M. The Largest Response Component in the Motor Cortex Reflects Movement Timing but Not Movement Type. *eNeuro* **3** (2016).

206. Barbieri, F. & Brunel, N. Can attractor network models account for the statistics of firing during persistent activity in prefrontal cortex? *Frontiers in Neuroscience* **2**, 114 (2008).

207. Barak, O., Sussillo, D., Romo, R., Tsodyks, M. & Abbott, L. F. From fixed points to chaos: three models of delayed discrimination. *Prog Neurobiol* **103**, 214 (2013).

208. O'Shea, D. J., Duncker, L., Goo, W., Sun, X., Vyas, S., Trautmann, E. M., Diester, I., Ramakrishnan, C., Deisseroth, K., Sahani, M. & Shenoy, K. V. Direct neural perturbations reveal a dynamical mechanism for robust computation (2022).

209. Vinograd, A., Nair, A., Kim, J. H., Linderman, S. W. & Anderson, D. J. Causal evidence of a line attractor encoding an affective state. *Nature* **634**, 910 (2024).

210. Karpowicz, B. M., Ali, Y. H., Wimalasena, L. N., Sedler, A. R., Keshtkaran, M. R., Bodkin, K., Ma, X., Rubin, D. B., Williams, Z. M., Cash, S. S., Hochberg, L. R., Miller, L. E. & Pandarinath, C. Stabilizing brain-computer interfaces through alignment of latent dynamics. *Nature Communications* **16**, 4662 (2025).

211. Shah, N. P., Krasa, B. A., Kunz, E., Hahn, N., Kamdar, F., Avansino, D., Hochberg, L. R., Henderson, J. M. & Sussillo, D. Improved interpretability in LFADS models using a learned, context-dependent per-trial bias (2025).

212. Vermani, A., Nassar, J., Jeon, H., Dowling, M. & Park, I. M. *Meta-Dynamical State Space Models for Integrative Neural Data Analysis* in *The Thirteenth International Conference on Learning Representations* (2025).

213. Jamkhandi, A., Korojy, A., Codol, O., Lajoie, G. & Perich, M. G. *JEDI: Jointly Embedded Inference of Neural Dynamics* arXiv:2603.10489. 2026.

214. Brenner, M., Weber, E., Koppe, G. & Durstewitz, D. *Learning Interpretable Hierarchical Dynamical Systems Models from Time Series Data* in *The Thirteenth International Conference on Learning Representations* (2025).

215. Chaisangmongkon, W., Swaminathan, S. K., Freedman, D. J. & Wang, X.-J. Computing by Robust Transience: How the Fronto-Parietal Network Performs Sequential, Category-Based Decisions. *Neuron* **93**, 1504 (2017).

216. Qian, W., Zavatone-Veth, J. A., Ruben, B. S. & Pehlevan, C. *Partial observation can induce mechanistic mismatches in data-constrained models of neural dynamics* in *The Thirty-eighth Annual Conference on Neural Information Processing Systems* (2024).

217. Dotson, N. M., Hoffman, S. J., Goodell, B. & Gray, C. M. A Large-Scale Semi-Chronic Microdrive Recording System for Non-Human Primates. *Neuron* **96**, 769 (2017).

218. Liu, L., Jiang, H., He, P., Chen, W., Liu, X., Gao, J. & Han, J. *On the Variance of the Adaptive Learning Rate and Beyond* in *International Conference on Learning Representations* (2020).

219. Orvieto, A., Smith, S. L., Gu, A., Fernando, A., Gulcehre, C., Pascanu, R. & De, S. *Resurrecting recurrent neural networks for long sequences* in *Proceedings of the 40th International Conference on Machine Learning* (2023).

220. Mongillo, G., Barak, O. & Tsodyks, M. Synaptic theory of working memory. *Science* **319**, 1543 (2008).

221. Masse, N. Y., Robert Yang, G., Song, H. F., Wang, X.-J. & Freedman, D. J. Circuit mechanisms for the maintenance and manipulation of information in working memory. *Nature Neuroscience* **22**, 1159 (2019).

222. Sourmpis, C., Petersen, C. C., Gerstner, W. & Bellec, G. Biologically informed cortical models predict optogenetic perturbations. *eLife* **14** (eds Ostojic, S. & Poirazi, P.) RP106827 (2026).

223. Paninski, L. Fast Kalman filtering on quasilinear dendritic trees. *J Comput Neurosci* **28**, 211 (2010).

224. Dabholkar, K. V. & Barak, O. When predict can also explain: Few-shot prediction to select better neural latents. *PLOS Computational Biology* **21**, 1 (2026).

225. Karpowicz, B. M., Ye, J., Fan, C., Tostado-Marcos, P., Rizzoglio, F., Washington, C. B., Scodeler, T., de Lucena, D. S., Nason-Tomaszewski, S. R., Mender, M., Ma, X., Arneodo, E. M., Hochberg, L., Chestek, C., Henderson, J. M., Gentner, T. Q., Gilja, V., Miller, L. E., Rouse, A. G., Gaunt, R., Collinger, J. L. & Pandarinath, C. *Few-shot Algorithms for Consistent Neural Decoding (FALCON) Benchmark* in *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track* (2024).

226. Schläfli, L. *Theorie der vielfachen Kontinuität* (Birkhäuser Basel, Basel, 1901).

227. Buck, R. C. Partition of Space. *The American Mathematical monthly* **50**, 541 (1943).

228. Stanley, R. An introduction to hyperplane arrangements. *Geometric Combinatorics* **13**, 389 (2007).

229. Keshtkaran, M. R., Sedler, A. R., Chowdhury, R. H., Tandon, R., Basrai, D., Nguyen, S. L., Sohn, H., Jazayeri, M., Miller, L. E. & Pandarinath, C. A large-scale neural network training framework for generalized estimation of single-trial population dynamics. *Nature Methods* **19**, 1572 (2022).

230. Yu, B. M., Cunningham, J. P., Santhanam, G., Ryu, S., Shenoy, K. V. & Sahani, M. *Gaussian-process factor analysis for low-dimensional single-trial analysis of neural population activity* in *Advances in Neural Information Processing Systems* (2008).

231. Lawson, D., Raventós, A., Warrington, A. & Linderman, S. *SIXO: Smoothing Inference with Twisted Objectives* in *Advances in Neural Information Processing Systems* (2022).

SUPPLEMENT

.1 SUPPLEMENTARY INFORMATION BACKGROUND

.1.1 *Derivation of low-rank RNN mean-field equations*

This derivation is based on work by Beiran *et al.* [49], Dubreuil *et al.* [56], and Schuessler *et al.* [102], which build on Mastrogiuseppe & Ostojic [59].
We start with a rank-1 RNN:

$$\tau \frac{d\kappa(t)}{dt} = -\kappa(t) + \frac{1}{N}\sum_{i}^{N} \mathbf{n}_i^{\mathsf{T}} \phi\left(\kappa(t)\mathbf{m}_i + v(t)\mathbf{h}\right) \tag{.1}$$

Assuming the triplets $(\mathbf{n}_i, \mathbf{m}_i, \mathbf{h}_i)$ are all independently sampled per neuron from a joint probability distribution $p(\mathbf{y})$; $\mathbf{y} = (n, m, h)$, Eq. S.1 can be seen as a sampling estimator for the recurrent input, which, via the law of large numbers, approaches the expectation as $N \to \infty$:

$$\frac{d\kappa(t)}{dt} \overset{N\to\infty}{=} -\kappa(t) + \mathbb{E}_{p(\mathbf{y})}\left[n\phi\left(\kappa(t)m + v(t)h\right)\right]. \tag{.2}$$

We assume $p(\mathbf{y})$ is a mixture of L zero-mean Gaussians [49, 56]. This gives each component an intuitive interpretation as a separate subpopulation in the network.

$$p(n, m, h) = p(\mathbf{y}) = \sum_{l}^{L} w_l \mathcal{N}(0, \Sigma_l).$$

To obtain Eq. S.6, we start with rewriting Eq. S.2 in terms of standard Gaussian variables. We can use that any multivariate Gaussian (here any component $(n, m, h)^{(l)} \sim \mathcal{N}(0, \Sigma_l)$) can be written as a linear transformation $\mathbf{B}^{(l)}$ of independent standard normal variables:

$$(n, m, h)^{(l)} = \mathbf{B}^{(l)}\mathbf{q}, \quad \mathbf{q} \sim \mathcal{N}(0, \mathbf{I}_3),$$

where $\mathbf{B}^{(l)}$ could be obtained via a Cholesky decomposition of Σ_l. Equivalently, separating this expression per component, we can write, for e.g., $n^{(l)}$:

$$n^{(l)} = \sum_{d=1}^{3} b_{n,d}^{(l)} q^{(d)}, \quad q^{(d)} \sim \mathcal{N}(0, 1). \tag{.3}$$

Plugging these in in Eq. S.2 we obtain:

$$\begin{aligned}\tau \frac{d\kappa(t)}{dt} = &-\kappa(t) \\ &+ \sum_{l}^{L} w_l \mathbb{E}_{p(\mathbf{q})}\left[\sum_{d}^{3} b_{n,d}^{(l)} q^{(d)} \phi\left(\kappa(t)\sum_{d'}^{3} b_{m,d'}^{(l)} q^{(d')} + v(t)\sum_{d'}^{3} b_{h,d'}^{(l)} q^{(d')}\right)\right].\end{aligned} \tag{.4}$$

Next we apply integration by parts (Stein's lemma) $\mathbb{E}_{p(q)}[qf(q)] = \mathbb{E}_{p(q)}[\partial_q f(q)]$; $q \sim \mathcal{N}(0,1)$ for each $q^{(d)}$ of each of the L components:

$$\begin{aligned}
&\mathbb{E}_{p(\mathbf{q})}\left[\sum_{d}^{3} b_{n,d}^{(l)} q^{(d)} \phi\left(\kappa(t)\sum_{d'}^{3} b_{m,d'}^{(l)} q^{(d')} + v(t)\sum_{d'}^{3} b_{h,d'}^{(l)} q^{(d')}\right)\right], \\
&\quad = \mathbb{E}_{p(\mathbf{q})}\left[\sum_{d}^{3} b_{n,d}^{(l)} \partial_{q^{(d)}} \phi\left(\kappa(t)\sum_{d'} b_{m,d'}^{(l)} q^{(d')} + v(t)\sum_{d'} b_{h,d'}^{(l)} q^{(d')}\right)\right], \\
&\quad = \sum_{d}^{3} b_{n,d}^{(l)}(\kappa(t) b_{m,d}^{(l)} + v(t) b_{h,d}^{(l)})\, \mathbb{E}_{p(\mathbf{q})}\left[\phi'\left(\kappa(t)\sum_{d'} b_{m,d'}^{(l)} q^{(d')} + v(t)\sum_{d'} b_{h,d'}^{(l)} q^{(d')}\right)\right],
\end{aligned}$$

where we used the chain rule in the third line. Substituting this back into Eq. S.4, we arrive at:

$$\tau \frac{d\kappa(t)}{dt} = -\kappa(t) + \sum_{l}^{L} w_l \sum_{d}^{3} b_{n,d}^{(l)}(\kappa(t) b_{m,d}^{(l)} + v(t) b_{h,d}^{(l)}) \mathbb{E}_{p(\mathbf{q})}[\phi'(\dots)]. \tag{.5}$$

Finally we simplify again be rewriting the $b^{(l)}$ back in terms of covariances of $\mathbf{y}^{(l)}$. Starting with the terms in front of the expectation (e.g., $\sum_{d=1}^{3} b_{n,d}^{(l)} b_{m,d}^{(l)}$) we have:

$$\begin{aligned}
\sigma_{mn}^{(l)} &= \mathbb{E}_{p(\mathbf{y})}[n^{(l)} m^{(l)}] \\
&= \mathbb{E}_{p(\mathbf{q})}\left[\sum_{d} b_{n,d}^{(l)} q^{(d)} \sum_{d'} b_{m,d'}^{(l)} q^{(d')}\right] \\
&= \sum_{d,d'} b_{n,d}^{(l)} b_{m,d'}^{(l)} \mathbb{E}_{p(\mathbf{q})}[q^{(d)} q^{(d')}] \\
&= \sum_{d=1}^{3} b_{n,d}^{(l)} b_{m,d}^{(l)},
\end{aligned}$$

where we used the definition of covariances in the first line, Eq. S.3 in the second line and $\mathbb{E}_{p(\mathbf{q})}[q^{(d)} q^{(d')}] = \delta_{dd'}$ for independent d and d' in the last line. A similar argument applies to the terms inside the expectation:

$$
\begin{aligned}
&\mathbb{E}_{p(\mathbf{q})}\left[\phi'\left(\sum_{d'}(\kappa(t)b^{(l)}_{m,d'}+v(t)b^{(l)}_{h,d'})q^{(d')}\right)\right] \\
&\quad=\mathbb{E}_{p(z)}\left[\phi'\left(\sqrt{\sum_{d'}(\kappa(t)b^{(l)}_{m,d'}+v(t)b^{(l)}_{h,d'})^2}z\right)\right];\ z\sim\mathcal{N}(0,1) \\
&\quad=\mathbb{E}_{p(z)}\left[\phi'\left(\sqrt{\kappa(t)^2\sum_d(b^{(l)}_{m,d})^2+v(t)^2\sum_d(b^{(l)}_{h,d})^2+2\kappa v\sum_d b^{(l)}_{m,d}b^{(l)}_{h,d}}z\right)\right], \\
&\quad=\mathbb{E}_{p(z)}\left[\phi'\left(\sqrt{\kappa(t)^2\sigma^{2(l)}_m+v(t)^2\sigma^{2(l)}_h}z\right)\right], \\
&\quad=\mathbb{E}_{p(z)}\left[\phi'\left(\sqrt{\Delta^{(l)}(t)}z\right)\right],
\end{aligned}
$$

where we used in the second line that for independent zero-mean unit Gaussians $q^{(d)}$, the linear combination $\sum_d a^{(d)}q^{(d)}$ is equivalent to $\sqrt{\sum_d a^{(d)2}}z$ (where $z\sim\mathcal{N}(0,1)$), and in the fourth line that the cross-term $2\kappa(t)v(t)\sum_d b^{(l)}_{m,d}b^{(l)}_{h,d}$ vanishes because of the assumption that $m^{(l)}$ and $h^{(l)}$ are uncorrelated.

Substituting these into Eq. S.5 results in the desired result (Eq. S.6) which shows that the mean-field dynamics depend only on second-order statistics of the weight distribution, with nonlinear effects entering through a scalar gain term:

$$
\tau\frac{d\kappa(t)}{dt}=-\kappa(t)+\sum_l^L w_l\left(\kappa(t)\sigma^{(l)}_{mn}+v(t)\sigma^{(l)}_{hn}\right)\mathbb{E}_{p(z)}\left[\phi'\left(\sqrt{\Delta^{(l)}(t)}z\right)\right], \tag{.6}
$$

with $\Delta^{(l)}(t)=(\sigma^{(l)}_m\kappa(t))^2+(\sigma^{(l)}_h v(t))^2$ and $z=\mathcal{N}(0,1)$.

.2 SUPPLEMENTARY INFORMATION CHAPTER 3

.2.1 *Loss curves*

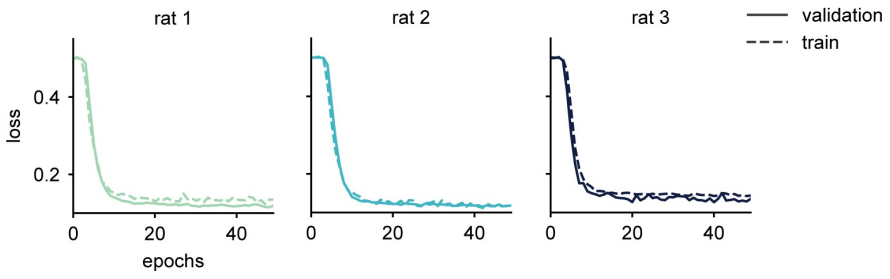

FIGURE .1: Loss over epochs for three models, each trained with LFP data from a separate rat. An epoch denotes one pass through all trials in the training or validation set. The validation trials are defined before training and are not used for calculating gradients. However, the training and validation error are almost identical throughout training. This is expected, as the validation and training trials only differ in the portion of the local field potential that was used as reference signal for the network, and in the seed used for generating the randomised stimulus onsets and offsets.

.2.2 *Dynamics of unconstrained networks are similar to low-rank RNNs*

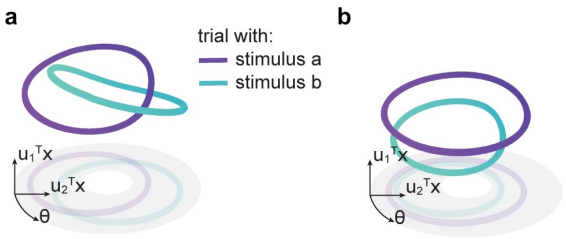

FIGURE .2: **a)** We also trained RNNs without rank constraint. For these networks initial entries in the recurrent weight matrix J were drawn from a zero-mean Gaussian with variance $\frac{g^2}{N}$. Here, we add a regularisation term to the loss, which keeps the average firing rates close to 0, to avoid a rate-coding solution (Eq 3.6). We set g to 0.6 and took a learning rate of 0.001, with the training setup otherwise as for the low-rank networks (Section 3.5.2.4). In order to find a basis similar to the one used for plotting the dynamics of the low-rank RNN we took the following approach. First we calculated basis vectors for the activity due to recurrent dynamics: $\mathbf{J}\tanh(\mathbf{x}(t))$, by performing a Principal Component Analysis (singular vector decomposition): $\mathbf{U}\mathbf{\Sigma}\mathbf{V}^\mathsf{T} = \mathbf{J}\tanh(\mathbf{X})$, where $\mathbf{X}$ is an $N \times 2\mathcal{T}$ matrix containing the activity of all units for one period of oscillation of each type of trial (stimulus a and b). We took the first two columns (principal components), $\mathbf{u}_1$ and $\mathbf{u}_2$, of $\mathbf{U}$, as well as the input vector $\mathbf{I}^{(osc)}$ orthogonalised with respect to these two principal components as basis for $\mathbf{X}$. This basis retains 77% of the variance of $\mathbf{X}$ (measured by r^2). Since now, similar to the low-rank case, we can write the projection of $\mathbf{x}(t)$ on $\mathbf{I}_\perp^{(osc)}$ as a function of θ, we can plot trajectories with coordinates $(\theta, \mathbf{u}_1^\mathsf{T}\mathbf{x}, \mathbf{u}_2^\mathsf{T}\mathbf{x})$, and we obtain two stable cycles, linked in phase-space, as in Fig 3.2. **b)** Here we train full-rank networks without regularisation, and find dynamics lying on two non-linked cycles, similar to the rate-coding model in Fig 3.2. The basis is constructed similar to a, here explaining 87% of the variance.

.2.3 *Influence of the training setup on the network solution*

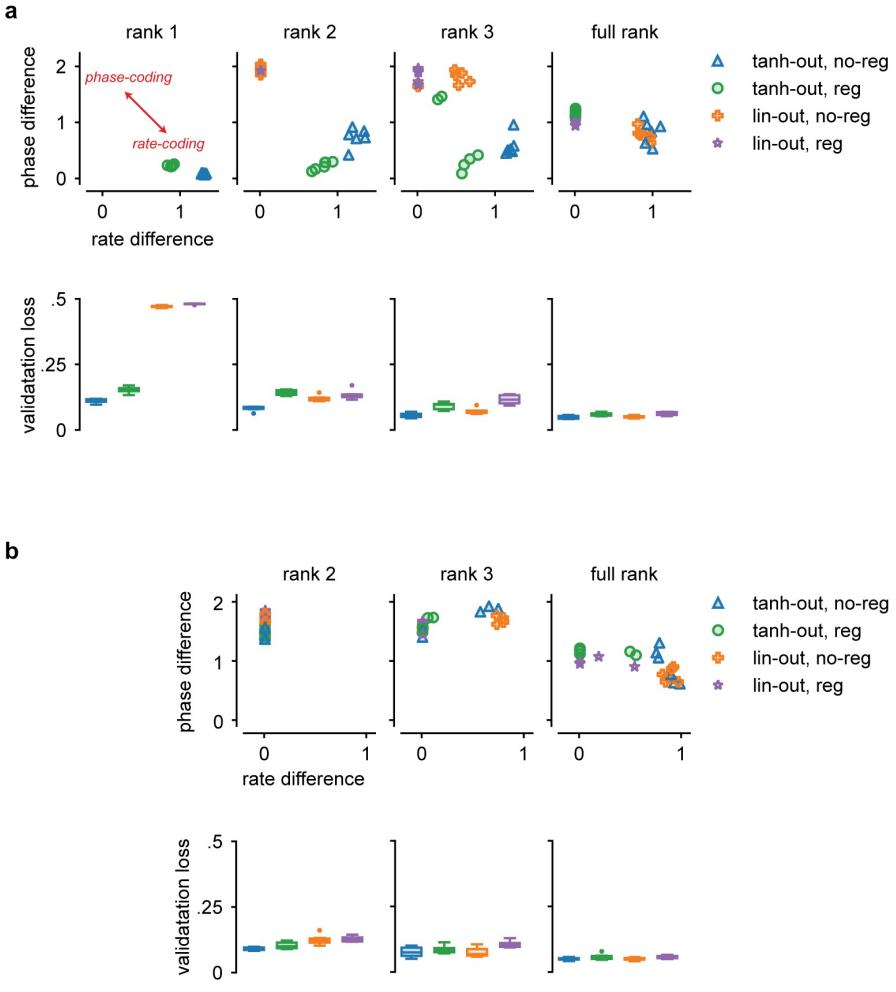

FIGURE .3: (*Caption on next page.*)

FIGURE .3: (*Figure on previous page.*) **a)** Here we analysed to what degree a model will learn a phase- versus a rate-coding solution, as a function of the training setup and initialisation. To quantify to what degree either solution is learned, we first computed for all units in a model the absolute difference between their (normalized) rate after stimulus *a* and their rate after stimuli *b*, as well as the absolute phase difference between trials with either stimulus. To get one measure of rate coding per model, we then calculated the mean over the absolute rate differences of all units, and similarly took the mean absolute phase differences as a measure of phase coding. We plot here in the first row these two measures for 6 models for each combination of the following conditions: ranks 1,2,3, full rank (columns); output during training is $\mathbf{W}\mathbf{x}$ (*lin-out*) or $\mathbf{W}\tanh(\mathbf{x})$ (*tanh-out*); regularisation that penalises deviations from the mean (Eq 3.6) is used (*reg*) or not (*no-reg*). All of these models use the default initialisation used in the manuscript and were trained for 100 epochs. Additionally we plot the validation loss after training in the second row. We observe that for rank 1 only models with *tanh-out* learn the task, and that these models learn a purely rate-coding solution. This is in line with our theory, as we previously showed that the phase-coding models couples its autonomously generated oscillations. Autonomous oscillations can only be generated by models of rank 2 and higher. For rank 2 and higher we can see that regularised models with a linear readout during training will generally learn a phase-coding solution, whereas models with a non-linear readout and no regularisation will generally learn a rate-coding solution.
b) We repeated the experiment, but now initialised each network to start out with oscillatory dynamics. We can do this by initialising the weight matrix with a pair of complex conjugate eigenvalues with real part larger than 1 [49, 59] (this is only possible for rank 2 and higher). We can observe that models with oscillatory initialisation are generally more likely to learn a phase-coding solution; in particular all rank 2 models learn a purely phase-coding solution.

.2.4 *Rate-coding solution in detail*

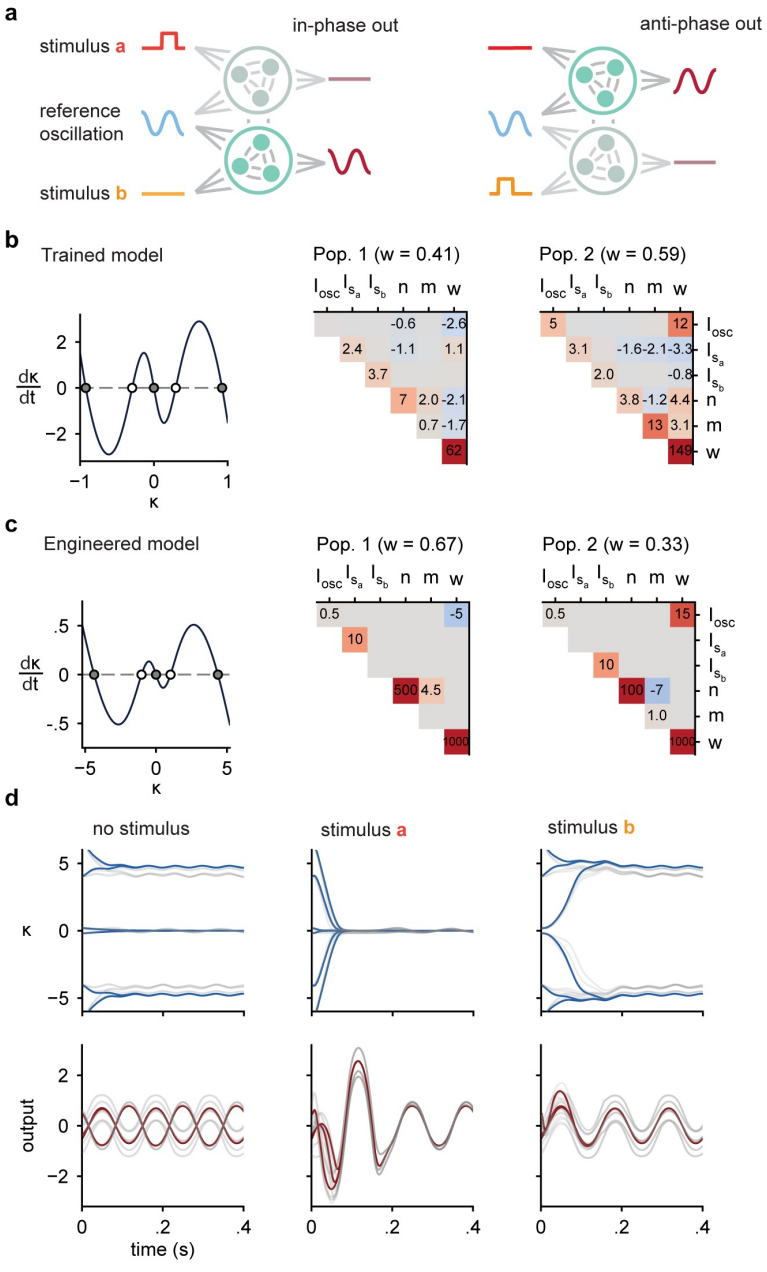

FIGURE .4: (*Caption on next page.*)

FIGURE .4: (*Figure on previous page.*) We here detail a simple rate-coding model that performs the working memory task. Such a model consists of two populations, one responsible for an oscillatory output in-phase with the reference oscillation, and one responsible for the anti-phase output oscillation. Depending on which stimulus was last seen, either population is dominating the readout.
b) We reverse engineered a trained RNN with rank-1 connectivity. Due to the rank constraint, in the absence of any input, the trained model implements a one-dimensional dynamical system (with variable κ). We can plot the change in κ as a function of κ and observe the model learned to have three stable fixed points (left). We can fit a mixture of two Gaussians to the connectivity ($\mathbf{I}$ = input, $\mathbf{n}, \mathbf{m}$ = left, right connectivity vector, $\mathbf{w}$ = output) which indicates that the model includes one population responsible for the fixed point at 0 (negative coupling between the connectivity vectors $\mathbf{m}$, and $\mathbf{n}$), while contributing little to the dynamics at the other fixed points (Due the large variance of $\mathbf{m}$, the units of this population will have saturated rates when κ is away from 0). This population produces an in-phase oscillation, due to the positive covariance between input ($\mathbf{I}^{(osc)}$) and output weights ($\mathbf{w}$). The other population is responsible for the two stable fixed points away from 0 (positive coupling between the connectivity vectors $\mathbf{m}$, and $\mathbf{n}$, small variance of $\mathbf{m}$), and produces the anti-phase oscillation. Note that this model is almost identical to the one described in [49] Figure 4 (but with oscillations added on top).
c) We can make the description from the previous panel more formal by engineering a rate-coding model. We initialised two covariance matrices (right), creating a two population model with similar dynamics as in the previous panel (left).
d) We plot the latent dynamics (top row) and output (bottom row) given by the mean-field equation describing the network activity in the limit of infinite units (blue, red), as well as finite size simulations (grey). For the finite size simulations, we used models (N=4096) with weights sampled from the mixture of Gaussians. We can see that in the absence of input, depending on the initial state the model will go to either the fixed point at 0, corresponding to an in-phase oscillation, or one of the two fixed points away from 0, corresponding to an anti-phase oscillation, respectively. For the second and third column, we provided the network with stimulus a, and b respectively, for the first .15s of the trial. Stimuli can successfully steer the network to the right fixed point and output oscillation phase, irrespective of initial condition.

.2.5 *Converged RNN dynamics match those of coupled oscillators*

FIGURE .5: We simulated two coupled oscillators with the coupling function extracted from a trained network (Fig 3.3b). The two oscillators represent the RNN's phase (ϕ) and the reference oscillation phase (θ). Starting simulations from various initial conditions demonstrates that the coupling function induces bistability, as all simulations converge to one of two stable cycles on the torus. Furthermore, the convergent trajectories of the coupled oscillators are a close match to those of the full RNN projected into the same space (ϕ, θ), indicating that the coupled oscillator description is appropriate for the RNN.

.2.6 *Connectivity of trained models*

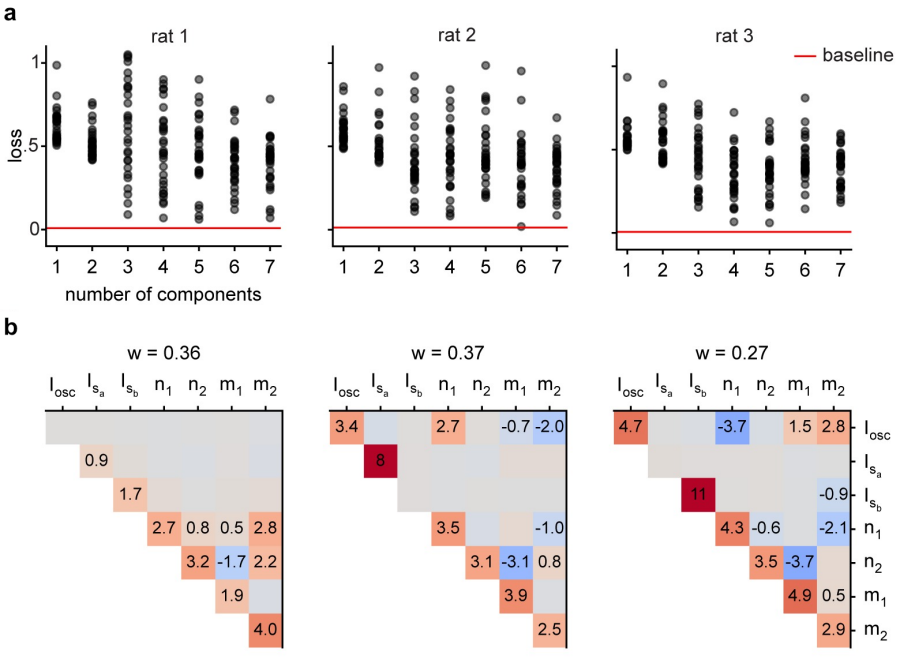

FIGURE .6: **a)** In order to study the connectivity of trained models, we fitted a mixture of Gaussians with 1 to 7 mixture components to the connectivity vectors of three different models [56], each trained with LFP data from a seperate rat. For this, we used variational inference with a Gaussian prior on the mean with precision 10^6 and mean 0. After each fit, we resampled the weights 30 times and computed the loss over a batch of 128 trials with a pure sine wave as reference oscillation. Altough for no amount of components we reliable were able to resample functioning models, from 3 components onwards a small fraction of sampled model have a comparable loss to the original trained model (red line). **b)** The covariance structure when fitting three components to weights of a trained network give us some hints as to what is needed to get a functioning model (top: covariances, bottom: pair plots). One component is unconnected (zero covariance) to the reference oscillation and has a skew-symmetric structure between the singular vectors. This structure generates oscillations [49, 59, 164]. The other two components, each connected to one stimulus and with opposite covariance between the input and singular vectors, together implement the coupling function.

.2.7 *Connectivity of mean-field model*

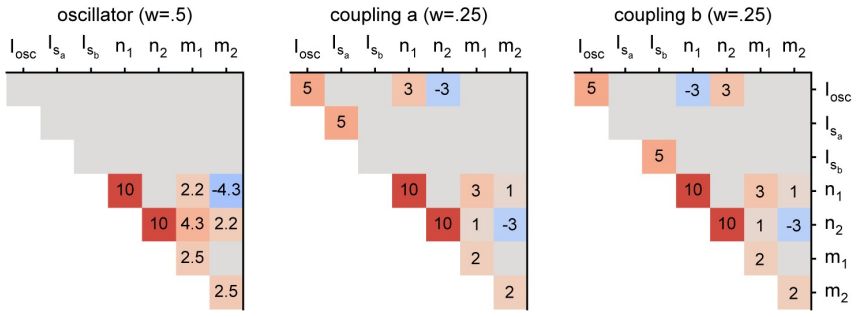

FIGURE .7: The designed covariance structure for the reduced model shown in Fig 3.4D,E, which leads to dynamics similar to the trained models. It also shares connectivity structure with trained models (Fig. S.6b), namely having one component unconnected to the reference oscillation that autonomously generates oscillations, and having the other two components, each connected to one stimulus, implement the coupling function.

.2.8 *Geometry is preserved in a model coding for 4 stimuli, but more coupling populations are required.*

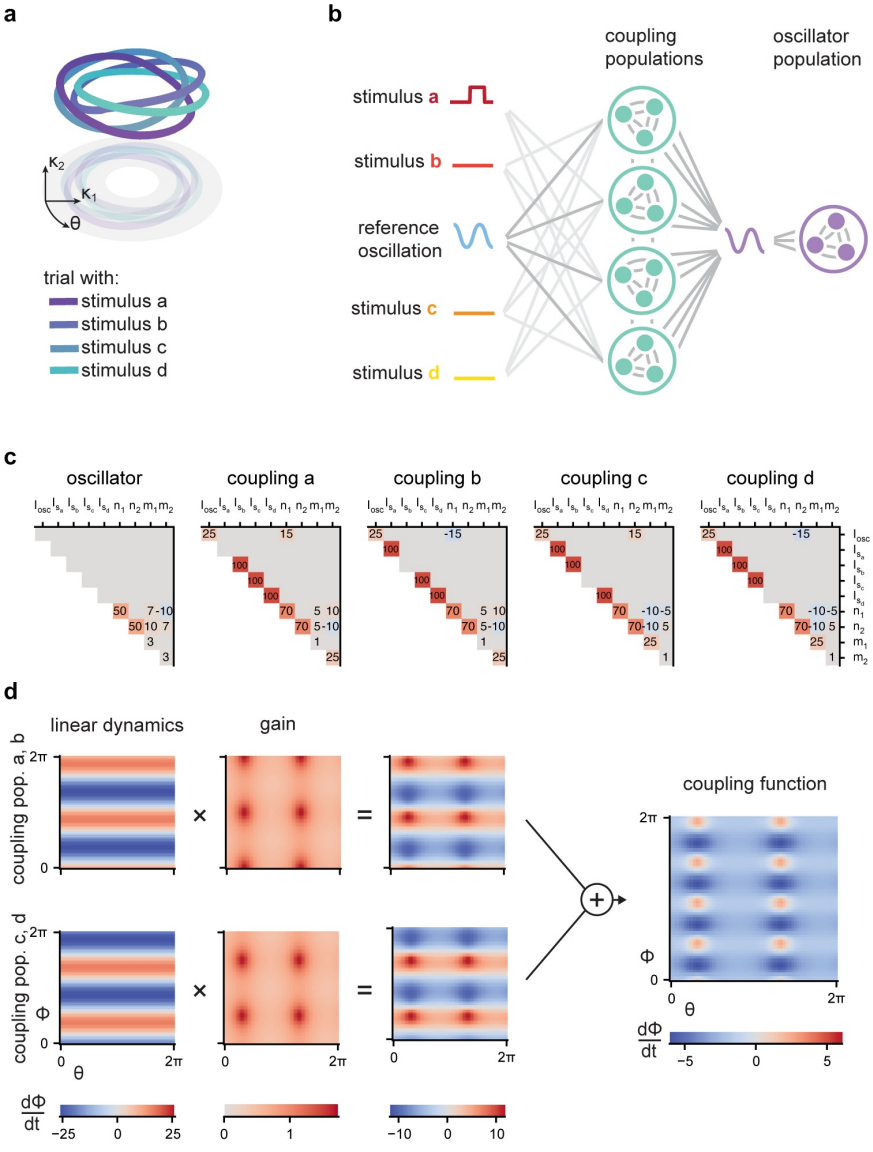

FIGURE .8: *(caption on next page.)*

FIGURE .8: (*Figure on previous page.*)**a)** Our model can be straightforwardly extended to code for more than 2 stimuli. Here, we trained a network to maintain 1 of 4 stimuli at a given trial, by producing an output oscillation at -0.2π, -0.7π, -1.4π or -1.7π radians offset with respect to the reference LFP, for stimulus *a*, *b*, *c* or *d*, respectively. Again we find a stable limit cycle for each stimuli, which form linked cycles in phase-space.
b) We now reverse engineered this model further, in particular we also found a reduced description of a model coding for 4 stimuli, in terms of 5 subpopulations. As before, 1 population generates oscillations, but we now have 4 coupling populations. Each stimulus inhibits all but one coupling population, so that the remaining coupling population guides the network dynamics to the limit cycle corresponding to the presented stimulus.
c) The reduced model is specified by a mixture of 5 Gaussians, here we show the covariances that can be used to generate the connectivity of a model coding for 4 stimuli.
d) Besides the connectivity with respect to the input, the recurrent dynamics (given by the overlaps, or covariances, between the connectivity vectors **m**'s and **n**'s) also had to be adjusted in order to allow memorising 4 stimuli instead of 2. We showed in our manuscript that rank 2 phase-coding models can described well by their coupling function, which gives the dynamics of the model as a function of the phase of the input oscillation (θ) and the phase of its internal oscillation (ϕ). The two limit cycles in the model described in the main text originated from the $\sin(2\phi)$ and $\cos(2\phi)$ terms in the coupling function. To code for 4 stimuli, we need higher frequency terms (e.g. $\cos(4\phi)$) in the coupling function. To understand how these originate from the recurrent connectivity, we decompose the coupling function. The coupling function generated by multiple populations is simply a summation over the coupling functions generated by the individual subpopulations. These in turn can be decomposed as the product of a linear part stemming from the covariances between the connectivity **m**'s and **n**'s of the subpopulations, and a non-linear part (the gain) that relates to how saturated neurons in this populations rates are (a neuron is saturated when its activity is at the flat part of the tanh non-linearity). Here we show that two populations that saturate at opposite phases of ϕ are enough to create a coupling function containing higher frequency terms, leading to dynamics with four limit cycles.

.2.9 *Phase precession through translation of the coupling function*

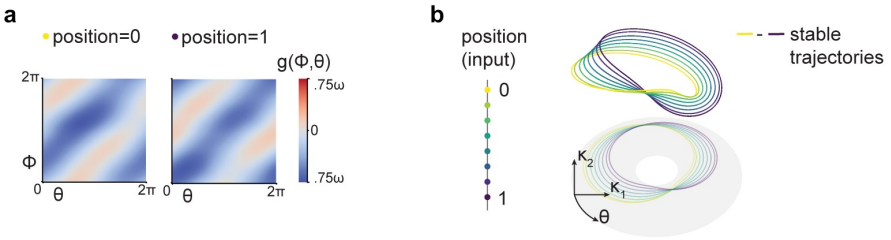

FIGURE .9: (*Caption on next page.*)

FIGURE .9: (*Figure on previous page.*) Phase precession in rat hippocampus entails a place cell changing its relative phase of firing as a result of rat moving through a place field [143]. We here show how to create a network that changes its relative phase of oscillation depending on a continuous valued stimulus input. We setup a network with connectivity drawn from a mixture of three Gaussians, again with two components implementing a coupling function and one component implementing an oscillator. The network receives a sinusoidal reference input with phase θ as $\sin(\theta), \cos(\theta)$ through input vectors $\mathbf{I}^{(osc_a)}, \mathbf{I}^{(osc_b)}$ respectively, as well as continuous valued stimulus input, representing place field position, $s(t) \in [0, 1)$ as $\sin(s(t)\frac{1}{2}\pi), \cos(s(t)\frac{1}{2}\pi)$ through input vectors $\mathbf{I}^{(s_a)}, \mathbf{I}^{(s_b)}$ respectively. The oscillator component has connectivity equal to Fig. S.7, whereas for the coupling components (p_2, p_3) we define the covariance matrices as follows: For the off-diagonal elements, $\sigma^{(p_2)}_{n^{(1)}I^{(osc_b)}} = -\sigma^{(p_2)}_{n^{(2)}I^{(osc_a)}} = \sigma^{(p_3)}_{n^{(2)}I^{(osc_a)}} = \sigma^{(p_3)}_{n^{(1)}I^{(osc_b)}}$, otherwise 0. For the diagonal elements, component two has zero variance for (is unconnected to) $\mathbf{I}^{(s_a)}$ and component three has zero variance for $\mathbf{I}^{(s_b)}$. This gives a coupling function of the form: $g(\theta, \phi) = \sin(\theta - \phi)\frac{a}{\sqrt{b+c\sin(\frac{\pi}{2}s(t))^2}} + \cos(\theta - \phi)\frac{a}{\sqrt{b+c\cos(\frac{\pi}{2}s(t))^2}}$, for constants a, b, c that depend on the exact values of the variances and covariances of the mixture components. For large c and small b, the coupling function will change from $a\sin(\theta - \phi)$ to $a\cos(\theta - \phi)$ as $s(t)$ changes from 0 to 1. We plotted here, the coupling function of a network (N=2056 units) with connectivity drawn from mixture of Gaussians as described above, for $s(t) = 0$ and $s(t) = 1$. The coupling function translates between these two states as a consequence of position input $s(t)$. **b)** Different trials on which $s(t)$ is tonically presented at different values lead the the network locking to a unique phase difference with respect to the reference oscillation. In phase-space, this appears as the stable cycle shifting along the surface of a torus.

.3 SUPPLEMENTARY INFORMATION CHAPTER 4

.3.1 *Proof of preposition 1*

.3.1.1 *Problem definition*

We are interested in finding all fixed points of the following equation:

$$\tau \frac{d\mathbf{x}}{dt} = -\mathbf{x}(t) + \mathbf{J}\phi(\mathbf{x}(t)), \tag{.7}$$

with $\mathbf{x}(t) \in \mathbb{R}^N$, element-wise nonlinearity $\phi(\mathbf{x}_i) = \sum_d^D \mathbf{b}_i^{(d)} \max(\mathbf{x}_i - \mathbf{h}_i^{(d)})$ and low-rank matrix $\mathbf{J} = \mathbf{M}\mathbf{N}^\mathsf{T}$, with $\mathbf{M}, \mathbf{N} \in \mathbb{R}^{N \times R}$ and $R \leq N$. Since τ only scales the speed of the dynamics, we will, for convenience and without loss of generality, assume $\tau = 1$.

.3.1.2 *Preliminaries: Fixed points in Piecewise-linear RNNs*

First, we briefly repeat results from Durstewitz [65]. Assume $D = 1$, $\phi(\mathbf{x}_i) = \max(\mathbf{x}_i - \mathbf{h}_i)$. To find all fixed points of Eq. .7, start by redefining ϕ by introducing a diagonal indicator matrix:

$$\mathbf{D}_\Omega = \begin{bmatrix} d_1 & & & \\ & d_2 & & \\ & & \ddots & \\ & & & d_N \end{bmatrix}, \tag{.8}$$

with $d_i = \begin{cases} 1, & \text{if } \mathbf{x}_i > \mathbf{h}_i \\ 0, & \text{otherwise} \end{cases}$.

Then our RNN equation, for a given $\mathbf{x}$ and corresponding $\mathbf{D}_\Omega$ reads:

$$\frac{d\mathbf{x}}{dt} = -\mathbf{x}(t) + \mathbf{J}\mathbf{D}_\Omega\mathbf{x}(t) - \mathbf{J}\mathbf{D}_\Omega\mathbf{h}.$$

Each of the 2^N configurations of $\mathbf{D}_\Omega$ corresponds to a region in which the dynamics are linear. Thus, for each configuration, we can solve:

$$\begin{aligned} \mathbf{0} &= -\mathbf{x} + \mathbf{J}\mathbf{D}_\Omega\mathbf{x} - \mathbf{J}\mathbf{D}_\Omega\mathbf{h}, \\ \mathbf{x}^* &= (\mathbf{J}\mathbf{D}_\Omega - \mathbf{I})^{-1}\mathbf{J}\mathbf{D}_\Omega\mathbf{h}. \end{aligned}$$

Next, we check whether the obtained $\mathbf{x}^*$ is consistent with the assumed $\mathbf{D}_\Omega$ (Eq. .8). If so, we found a fixed point of the RNN. We have to check, as the solution to the system of linear equations can lie outside of the linear regions specified by $\mathbf{D}_\Omega$. Note that if for some $\mathbf{D}_\Omega$ the matrix $\mathbf{J}\mathbf{D}_\Omega - \mathbf{I}$ is not invertible, then there is no single fixed point, but we still can find a structure of interest (e.g., a direction with eigenvalue 0 corresponds to marginal stability, i.e., a line attractor).

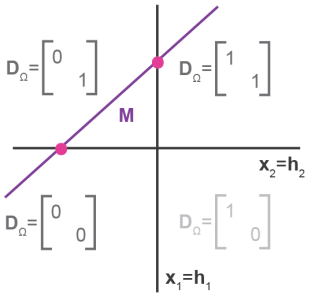

FIGURE .10: Proof sketch including $\mathbf{D}_\Omega$'s. The phase-space of an RNN with N (here 2) units with activation $\max(0, \mathbf{x}_i - \mathbf{h}_i)$ is partitioned into 2^N (here 4) regions in which the dynamics are linear, each corresponding to a configuration of $\mathbf{D}_\Omega$. If dynamics are confined to the R-dimensional subspace spanned by the columns of $\mathbf{M}$, only a subset (here 3) can be reached. Each unit intersects the space spanned by the columns of $\mathbf{M}$ with a hyperplane (the pink points in the Figure). The amount of linear regions in $\mathbf{M}$, thus becomes equivalent to "how many regions can we create in R-dimensional space with N hyperplanes?"

.3.1.3 *Preliminaries: Fixed points in Piecewise-linear low-rank RNNs*

First, assume $\mathbf{x}(0)$ is in the subspace spanned by the columns of $\mathbf{M}$. With the low-rank assumption, we can rewrite Eq. .7 for all $t \in [0, \infty)$, by projecting it on $\mathbf{M}$ [49, 56, 59]:

$$\frac{d\mathbf{z}}{dt} = -\mathbf{z}(t) + \mathbf{N}^\mathsf{T}\phi(\mathbf{M}\mathbf{z}(t) - \mathbf{h}) \tag{.9}$$

with $\mathbf{x}(t) = \mathbf{M}\mathbf{z}(t)$.

Now assume $\mathbf{x}(0)$ contains some part $\mathbf{x}^\perp(0)$ not in the subspace spanned by $\mathbf{M}$, i.e., we have $\mathbf{x}(0) = \mathbf{M}\mathbf{z}(0) + \mathbf{x}^\perp(0)$. The dynamics of $\mathbf{x}^\perp$(t) are simply given by $\frac{\mathbf{x}^\perp}{dt} = -\mathbf{x}^\perp(t)$ which will decay to its stable point at $\mathbf{0}$ irrespective of $\mathbf{z}(t)$, and can thus not contribute additional fixed points.

Naively, using the same strategy as before to obtain all fixed points $\mathbf{z}$, we would need to solve 2^N linear systems of R equations (again for all configurations of $\mathbf{D}_\Omega$):

$$\mathbf{z}^* = (\mathbf{N}^\mathsf{T}\mathbf{D}_\Omega\mathbf{M} - \mathbf{I})^{-1}\mathbf{N}^\mathsf{T}\mathbf{D}_\Omega\mathbf{h}, \tag{.10}$$

.3.1.4 *Preliminaries: Hyperplane arrangements*

In the subsequent section, we will turn to the question of how many equations we need to solve to find all possible fixed points. Recall that it is possible to calculate the fixed points analytically because piecewise-linear nonlinearities partition space into subregions in which dynamics are linear. Each of the linear regions corresponds to a configuration of $\mathbf{D}_\Omega$. For networks with low-rank connectivity, we have to

consider only a small subset of those, as only a small subset of all configurations of $\mathbf{D}_\Omega$ correspond to $\mathbf{x}$'s within the column space of $\mathbf{M}$ (See Fig. S.10). To find out exactly how many regions lie within the column space, we will need to answer the question: *in how many regions can we divide R-dimensional space, with N hyperplanes?*

To answer this question in general, we will need a theorem from the field of *hyperplane arrangements* [174, 226–228]. Here we give a brief introduction.

Introduction to hyperplane arrangements.
A finite arrangements of hyperplanes is a set of N affine subspaces $\mathcal{A} = \{a_1, \ldots, a_N\}$ in some vector space $V = \mathbb{R}^R$. Recall a hyperplane is an $R-1$ dimensional subspace defined by a linear equation $a_i := \{\mathbf{v} \in V | \mathbf{m}^\mathsf{T}\mathbf{v} = h\}$ for some $\mathbf{m} \in V, h \in \mathbb{R}$. Note that any linear system of equations $\mathbf{M}\mathbf{v} = \mathbf{h}$ with $\mathbf{M} \in \mathbb{R}^{N\times R}$ equivalents defines an arrangement of N hyperplanes in R dimensional space. In Fig. S.11**a**,**c**, we show arrangements of 3 hyperplanes in $\mathbb{R}^2$. In this case, a hyperplane is a line, but there are infinitely many possibilities on how we can arrange these lines in two-dimensional space. We are interested in

$$\mathcal{N}(\mathcal{A}) := \text{ number of regions } \mathcal{A} \text{ partitions } \mathbb{R}^R,$$

where regions correspond to the connected components of $\mathbb{R}^R \setminus \mathcal{A}$. In this simple case, we can visually verify that the arrangements in Fig. S.11**a** partitions the space into 7 regions, whereas the arrangement in Fig. S.11**c** partitions the space into only 6 regions. Clearly, the number of regions $\mathcal{A}$ partitions space in is strongly related to the number of unique intersections of lines. We have fewer regions in Fig. S.11**c**, simply because all lines intersect at the same point. If we can wiggle the hyperplanes a little, and not change the number of regions (as we can do in Fig. S.11**a**, but not Fig. S.11**c**), we call the hyperplanes in *general* position (see Theorem 1 for a formal definition).

To count the amount of regions for any arrangement of hyperplanes, we can leverage an algebraic construction called the *intersections poset* $\mathcal{L}(\mathcal{A})$. This is the set of all nonempty intersections of hyperplanes in $\mathcal{A}$ and includes V. Elements of this set are generally referred to as *flats*. The flats are ordered by reverse inclusion $x \leq y \iff x \supseteq y$ in the intersection poset. We visualized example intersection posets of the previous examples (Fig. S.11**b**, **d**). Here we organized the flats by dimensionality (such a visualization is called a Hesse Diagram). Importantly for any real arrangement $\mathcal{A}$, $\mathcal{N}(\mathcal{A})$ solely depends on $\mathcal{L}(\mathcal{A})$ (Corollary 2.1, [228]).

To calculate $\mathcal{N}(\mathcal{A})$ from $L(\mathcal{A})$, we need one last construction, namely the Möbius function, recursively defined by

$$\mu(\mathcal{X}, s) = \begin{cases} 1 & \text{if } s = \mathcal{X} \\ -\sum_{\mathcal{X} \supseteq s' \supset s} \mu(\mathcal{X}, s'), & \text{if } s \subset \mathcal{X}. \end{cases} \tag{.11}$$

The numerical values for the example are shown in Fig. S.11.

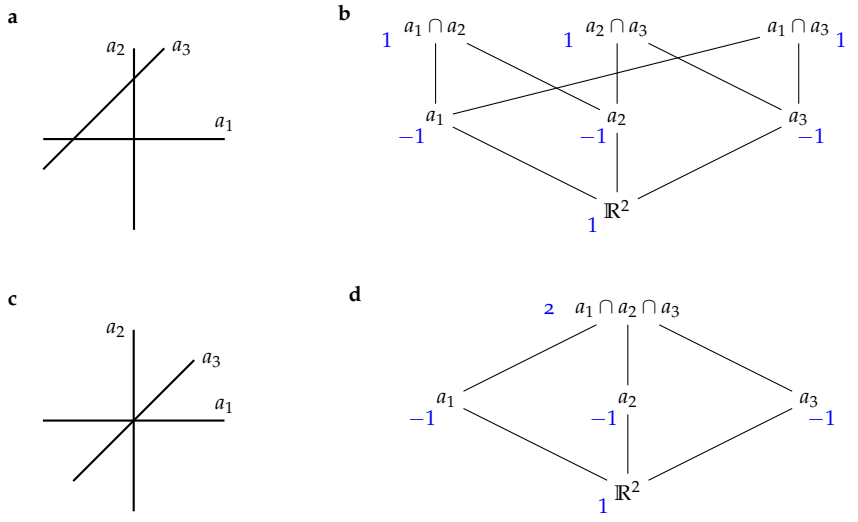

FIGURE .11: **a)** An arrangement of 3 hyperplanes a_1, a_2 and a_3 in *general position*. **b)** the associated intersection poset of the arrangement in **a**. **c)** An alternative arrangement with its associated intersection poset. **d)**. Blue numbers indicate the value of the Möbius function.

Theorem 1 (Zaslavsky's Theorem; [174, 228]). *Given a vector space $V = \mathbb{R}^R$ and an arrangement of N hyperplanes $\mathcal{A} = \{a_1, \ldots, a_N\}$ on V, then the number of regions $\mathcal{A}$ partitions V in (denoted $\mathcal{N}(\mathcal{A})$, can be expressed as follows*

$$\mathcal{N}(\mathcal{A}) = \sum_{s \in L(\mathcal{A})} \mu(\mathbb{R}^R, s)(-1)^{\dim(s)}$$

furthermore, it holds that

$$\mathcal{N}(\mathcal{A}) \leq \sum_{r=0}^{R} \binom{N}{r} \tag{.12}$$

with equality if and only if $\mathcal{A}$ is in general position i.e., $\mathcal{A}$ must satisfy

(i) $\{a_1, \ldots, a_p\} \subseteq \mathcal{A}$ *and* $p \leq R \Rightarrow \dim(\bigcap_{i=1}^{p} a_i) = N - p$

(ii) $\{a_1, \ldots, a_p\} \subseteq \mathcal{A}$ *and* $p > R \Rightarrow \bigcap_{i=1}^{p} a_i = \emptyset$

One can verify this fact for the given example shown in Fig. S.11. We refer to Stanley [228] for an in-depth formal introduction to this topic. Fundamentally, it is based on the following recursion that the number of regions for any arrangement satisfies

$$\mathcal{N}(\mathcal{A} \cup \{a_{N+1}\}) = \mathcal{N}(\mathcal{A}) + \mathcal{N}(\mathcal{A}^{a_{N+1}})$$

where $\mathcal{A}^{a_{N+1}} := \{a_{N+1} \cap a_i | a_i \in \mathcal{A}, a_{N+1} \cap a_i \neq \emptyset, a_{N+1} \nsubseteq a_i\}$ (Lemma 2.1, Stanley [228]). Note that a_{N+1} is itself an $R-1$ dimensional vector space, and each intersection $a_{N+1} \cap a_i$ is an $R-2$ dimensional hyperplane within a_{N+1} (e.g., the intersection

of two planes is a line within the planes). Hence $\mathcal{A}^{a_{N+1}}$ is itself an arrangement of N hyperplanes, but in an $R-1$ dimensional subspace. In fact, the intersection poset exhaustively enumerates the elements of all possible $\mathcal{A}^{a_i}$, and the Möbius function can be shown to satisfy the above recursion.

If we choose $\phi(\mathbf{x}_i) = \max(\mathbf{x}_i - \mathbf{h}_i, 0)$ i.e $D = 1$, each neuron would partition space by a single hyperplane $\mathbf{x}_i = \mathbf{h}_i$ or equivalently the R dimensional subspace by the hyperplane $\mathbf{M}_i\mathbf{z} = \mathbf{h}_i$. Hence, the hyperplane arrangement is determined by the matrix $\mathbf{M}$ and offset $\mathbf{h}$. As these quantities are learned during training of the RNN, this arrangement is often in a general position because it is simply numerically unlikely that two hyperplanes are exactly parallel or intersect in exactly the same "point".This does, however, change in the general case $D > 1$, for which we derive a tighter bound in the section below.

Arrangements of parallel families.

For the general case $\phi(\mathbf{x}_i) = \sum_{d=1}^{D} \mathbf{b}_i^{(d)} \max(\mathbf{x}_i - \mathbf{h}_i^{(d)}, 0)$ each neuron will partition space with D hyperplanes $\mathbf{b}_i^{(d)}\mathbf{x}_i = \mathbf{b}_i^{(d)}\mathbf{h}_i^{(d)} \iff \mathbf{x}_i = \mathbf{h}_i^{(d)}$ as before; equivalently each neuron partitions the R dimensional subspace with D hyperplanes $\mathbf{M}_i\mathbf{z} = \mathbf{h}_i^{(d)}$. Notably, all the D hyperplanes here will share the same row of $\mathbf{M}$, and thus they are *parallel*. Clearly, any such arrangement cannot be in general arrangement by definition.

The resulting arrangement will have a very specific structure. Let's define

$$A_i := \{a_{i1}, \ldots, a_{iD}\}$$

as a *family* of D parallel hyperplanes. Any pair of hyperplanes $a_{il}, a_{im} \in A_i$ is parallel. A low-rank RNN with N neurons and a general piecewise-linear activation function will thus lead to an arrangement consisting of N families of D parallel hyperplanes.

We can use this specific structure to obtain a tighter bound.

Lemma 1. *Let $\mathcal{A} = A_1 \cup \cdots \cup A_{N-1}$ be an arrangement of $N-1$ families of D parallel lines, then it satisfies the following recursion*

$$\mathcal{N}(\mathcal{A} \cup A_N) = \mathcal{N}(\mathcal{A}) + \sum_{d=1}^{D} \mathcal{N}\left(\mathcal{A}^{a_{Nd}}\right).$$

Furthermore, denote by $\mathcal{N}(N, R, D)$ the maximum number of regions attainable by any arrangement of N families of D parallel hyperplanes in R dimensional space then

$$\mathcal{N}(N, R, D) \leq \mathcal{N}(N-1, R, D) + D \cdot \mathcal{N}(N-1, R-1, D).$$

Proof. To add A_N to $\mathcal{A}$, we have to add D new parallel hyperplanes. We can do so by iteratively applying Lemma 2.1 [228]. We obtain

$$\begin{aligned}\mathcal{N}(\mathcal{A} \cup \{a_{N1}, \ldots, a_{ND}\}) &= \mathcal{N}(\mathcal{A} \cup \{a_{N1}, \ldots, a_{N(D-1)}\}) \\ &\quad + \mathcal{N}(\left(\mathcal{A} \cup \{a_{N1}, \ldots, a_{N(D-1)}\}\right)^{a_{ND}}) \\ &= \mathcal{N}(\mathcal{A}) + \sum_{d=1}^{D} \mathcal{N}\left(\left[\mathcal{A} \cup \bigcup_{i=1}^{d-1} \{a_{Ni}\}\right]^{a_{Nd}}\right).\end{aligned}$$

Now note that $\mathcal{A}^{a_{Nj}} := \{a_{Nj} \cap a_{lm} | a_{lm} \in \mathcal{A}, a_{Nj} \cap a_{lm} \neq \emptyset, a_{Nj} \nsubseteq a_{lm}\}$, hence by definition only hyperplanes that intersect with a_{Nj} are included in this set. As a_{Nj} is parallel to any other a_{Ni} for all $i \neq j$, all $a_{Nj} \cap a_{Ni}$ cannot be in the set. Hence for any d, we have that

$$\mathcal{N}\left(\left[\mathcal{A} \cup \bigcup_{i=1}^{d-1} \{a_{Ni}\}\right]^{a_{Nd}}\right) = \mathcal{N}(\mathcal{A}^{a_{Nd}})$$

which proves the first equation.

Recall that we define $\mathcal{N}(N, R, D)$ as the maximum number of regions attainable by any arrangement. Notice that $\mathcal{A}$ by construction is an arrangement of $N-1$ families of D parallel hyperplanes in R dimension. Thus by definition $\mathcal{N}(\mathcal{A}) \leq \mathcal{N}(N-1, R, D)$.

As noted before, the intersection set of two hyperplanes in dimension R is itself a hyperplane of dimension $R-1$. Furthermore the intersection sets of D parallel hyperplanes with a_{Nd}, remain parallel. Hence $\mathcal{A}^{a_{Nd}}$ is an arrangement of at most $N-1$ families of D parallel hyperplanes in $R-1$ dimensions. Thus $\mathcal{N}(\mathcal{A}^{a_{Nd}}) \leq \mathcal{N}(N-1, R-1, D)$ leaving us with

$$\mathcal{N}(\mathcal{A} \cup \{a_{N1}, \ldots, a_{ND}\}) \leq \mathcal{N}(N-1, R, D) + D \cdot \mathcal{N}(N-1, R-1, D).$$

As this holds for any arrangement, it also holds for the arrangement that has $\mathcal{N}(N, R, D)$ regions (i.e., which maximizes the number of regions) and, therefore, proves the second equation.

□

Lemma 2. *Let $\mathcal{A}$ be an arrangement of N families of D parallel hyperplanes. Then, it holds that*

$$N(\mathcal{A}) \leq \sum_{r=0}^{R} D^r \binom{N}{r}$$

with equality if each family is in a general position, i.e., that every subarrangement $\{a_{1j_1}, \ldots, a_{Nj_N}\}$ for all $1 \leq j_i \leq D$ is in general position.

Proof. We will first construct an intersection poset $L(\mathcal{A})$ on the level of families A_i in general position. After all, the *intersection properties* between these families is the

same as between their elements, e.g., if a_{i1} intersects a_{j1} then also all lines in A_i intersect all lines in A_j.

The resulting intersection poset $L(\mathcal{A})$ can be clustered into the corresponding families. We visualize the construction in Fig. S.12.

At each rank r (level from bottom to top), we can choose exactly $\binom{N}{r}$ families of hyperplanes that intersect (exactly the case if we just have N hyperplanes in general position). To obtain a flat of dimension $R - r$ we have to choose r out of the N hyperplane families without replacement.

If, e.g., two families of parallel hyperplanes A_i, A_j intersect, then any element a_{ik} will intersect with any element a_{jl} for all $1 \leq k, l \leq D$ leading to at most D^2 flats within each family (there can be less as other families might intersect in the same "point"). In general, each cluster of intersections of r families will contain at most D^r flats.

By construction of $L(\mathcal{A})$ and Theorem 1, the lemma follows directly.

To show that this construction indeed is an upper bound for all arrangements, we can use Lemma 1. There, we established a recursion, which any such upper bound must satisfy. Hence, assume $\mathcal{N}(N, R, D) = \sum_{r=0}^{R} D^r \binom{N}{r}$. Notice that using Pascal's identity, we can rewrite

$$
\begin{aligned}
\mathcal{N}(N, R, D) &= \sum_{r=0}^{R} D^r \binom{N}{r} \\
&= \sum_{r=0}^{R} D^r \left(\binom{N-1}{r} + \binom{N-1}{r-1} \right) \\
&= \sum_{r=0}^{R} D^r \binom{N-1}{r} + \sum_{r=0}^{R} D^r \binom{N-1}{r-1} \\
&= \sum_{r=0}^{R} D^r \binom{N-1}{r} + \underbrace{D^0 \binom{N-1}{-1}}_{:=0} + \sum_{r=1}^{R} D^r \binom{N-1}{r-1} \\
&= \mathcal{N}(N-1, R, D) + \sum_{r=0}^{R-1} D^{r+1} \binom{N-1}{r} \\
&= \mathcal{N}(N-1, R, D) + D \cdot \mathcal{N}(N-1, R-1, D)
\end{aligned}
$$

□

.3.1.5 *Proof of proposition*

Using the previously derived techniques, we will prove here the main proposition. Furthermore, in Algorithm 1, pseudo-code is given to compute all fixed points in practice.

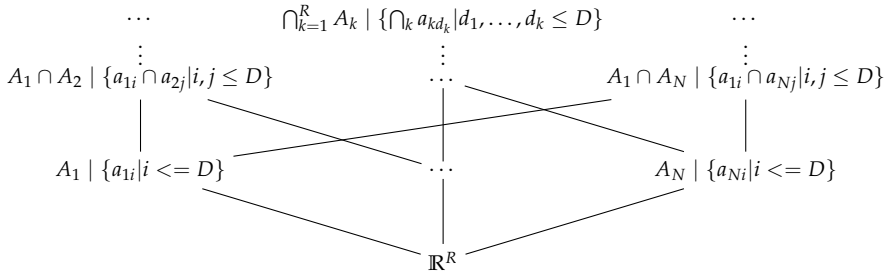

FIGURE .12: Construction of the intersection poset $L(\mathcal{A})$ for an arrangement of N families A_i of D parallel hyperplanes in "general position".

Proposition 1. *Assume the RNN of Eq. .7, with* $\mathbf{J}$ *of rank R and piecewise-linear activations:* $\phi(\mathbf{x}_i) = \sum_d^D \mathbf{b}_i^{(d)} \max(\mathbf{x}_i - \mathbf{h}_i^{(d)}, 0)$. *For fixed rank R and fixed number of basis functions D, we can find all fixed points in the absence of noise, that is all* $\mathbf{x}$ *for which* $\frac{d\mathbf{x}}{dt} = 0$, *by solving at most* $\mathcal{O}(N^R)$ *linear systems of R equations (for fixed R).*

Proof. By definition, each neuron partitions $\mathbb{R}^N$ in $D+1$ linear regions with D hyperplanes described by $\mathbf{x}_i^{(d)} = \mathbf{h}_i^{(d)}$, for the i'th neuron. Using that in the columnspace of $\mathbf{M}$, we have $\mathbf{x} = \mathbf{M}\mathbf{z}$, it follows that each neuron partitions the R dimensional subspace spanned by columns of $\mathbf{M}$, with D hyperplanes described by $\sum_r^R \mathbf{M}_{i,r}\mathbf{z}_r = \mathbf{h}_i^{(d)}$. Notice that these hyperplanes are parallel, as they all share the same coefficients $\mathbf{M}_i$ but have a different offset $\mathbf{h}_i^{(d)}$. Using Lemma 2 we know that there can only be $\sum_{r=0}^R D^r \binom{N}{r}$ such regions.

How do we find those regions? Let's first consider the case of $D = 1$, and assume that the hyperplanes are in general position. We can find the corresponding configurations of $\mathbf{D}_\Omega$ as follows. We first obtain the set of all intersections of R hyperplanes. For this we try to solve $\binom{N}{R}$ systems of R equations. Let $\mathbf{M}_R \in \mathbb{R}^{R\times R}$ be the matrix obtained by choosing R different rows $1, \ldots, R$ of $\mathbf{M} \in \mathbb{R}^{N\times R}$ (i.e., picking R neurons), then we may find the corresponding intersection of R hyperplanes by solving the following linear system of R equations

$$\mathbf{z}_\cap = \mathbf{M}_R^{-1}\mathbf{h}_R \qquad \text{and} \qquad \mathbf{x}_\cap = \mathbf{M}\mathbf{z}_\cap.$$

which will always have a unique solution if all hyperplanes are in general position, as then all $\mathbf{M}_R$ have rank R. Each $\mathbf{x}_\cap$ has 2^R possible bordering linear regions. We can find the corresponding $\mathbf{D}_\Omega = \mathrm{diag}([d_1, \ldots, d_N)$'s matrices of each of those subsections as follows. First $d_i = \mathbb{I}(\mathbf{x}_\cap < 0)$ for all $i <= N$. By construction $1, \ldots, R$ at $\mathbf{x}_\cap$ will be exactly at the threshold, by moving away from it d_R can become either zero or one, depending on in which region we and up. Hence, the 2^R regions correspond to one in which either combination of neurons $1, \ldots R$ is active (meaning that it is above the threshold). We thus just have to check all combinations $d_1, \ldots, d_R \in \{0, 1\}^R$. Using this, we will find at most $\sum_{r=0}^R \binom{N}{r}$ unique configurations (as this is the maximal number of regions possible for $D = 1$). To find all the

fixed points we hence have to solve Eq. .10 for each configuration. We thus end up with solving $\binom{N}{R}$ systems of R linear equations to find all regions, and another $\sum_{r=0}^{R}\binom{N}{r} \in \mathcal{O}(N^R)$ (for fixed R) systems of R linear equations to find all fixed points.

Let us now consider the case for $D > 1$. Note that an RNN with N units and D basis functions per unit, can be expanded to an RNN with ND units with activation $\phi(\mathbf{x}_i) = \max(\mathbf{x}_i - \mathbf{h}_i, 0)$ ([111], Theorem 1). Any fixed point can then still be analytically computed using Eq. .10. We expand the network but keep track of all $\sum_r^R D^r\binom{N}{R}$ possible intersections. It still holds that from each intersection, we can reach 2^R regions. In total, we will now find at most $\sum_{r=0}^{R} D^r\binom{N}{r}$ regions (Lemma 2). To find all the fixed points, we hence have to solve $\binom{N}{R}D^r + \sum_{r=0}^{R}\binom{N}{r}D^r$ systems of R linear equations, which for constant D and R has a cost of $\mathcal{O}(N^R)$

Finally, let's consider the case when hyperplanes are not in general position (which is unlikely to happen when doing numerical optimization). If there are intersections of more than R hyperplanes, we proceed as before, but in case the intersection of R hyperplanes we are currently considering intersects additional hyperplanes, set the diagonal elements of $\mathbf{D}_\Omega$ corresponding to these additional hyperplanes arbitrarily to 1 (as intersections including the additional hyperplanes are considered separately). On the other hand, in case some hyperplanes are only part of intersections of less than R hyperplanes (because they became parallel), we proceed as follows. Instead of considering only intersections of R hyperplanes, we now also consider all possible intersections of r hyperplanes, with $1 \leq r \leq R$. For this, we solve no more than $\sum_r^R \binom{N}{r}$ systems of r equations. Let $\mathbf{M}_r \in \mathbb{R}^{r\times R}$ be the matrix obtained by choosing r different linearly independent rows $1, \ldots, r$ of $\mathbf{M} \in \mathbb{R}^{N\times R}$; then we may find a point on the corresponding intersection of r hyperplanes (note that the intersection itself can now also be a hyperplane) by to solving the following linear system of r equations

$$\mathbf{z}_\cap = \mathbf{M}_r^\dagger \mathbf{h}_r \qquad \text{and} \qquad \mathbf{x}_\cap = \mathbf{M}\mathbf{z}_\cap.$$

with † being the pseudoinverse. We here now end up with solving no more than $\sum_r^R \binom{N}{r}$ systems of r linear equations to find all regions, which has an equal cost in N as the previous cases. □

We here provide pseudocode. For simplicity, we restrict ourselved to the case of $D = 1$ and assume that the arrangement specified by $\mathbf{M}$ and $\mathbf{h}$ is in general position. This can be generalized to the general setting as presented in the proof.

Algorithm 1: Improved exhaustive search for all fixed points

Data: $\mathbf{N} \in \mathbb{R}^{N \times R}, \mathbf{M} \in \mathbb{R}^{N \times R}, \mathbf{h} \in \mathbb{R}^{N}$
Result: *z_set* set of all fixpoints, *D_set* the set of all relevant $\mathbf{D}_\Omega$ configurations.

$D\_set := \{\}$;
$z\_set := \{\}$;
$idx = [1, \ldots, N]$;

// *Find feasible configurations*
idx_comb = all $\binom{N}{R}$ combinations of indices idx;
for $(i_1, \ldots, i_R)$ *in* idx_comb **do**
  $\mathbf{M}_R = \mathbf{M}[(i_1, \ldots, i_R), :]$;
  $\mathbf{h}_R = \mathbf{h}[(i_1, \ldots, i_R)]$;
  // $\mathbf{M}_R$ *is invertible as the arrangement is in general position*
  $\mathbf{z}_\cap = \text{solve}(\mathbf{M}_R, \mathbf{h}_R)$;
  $\mathbf{x}_\cap = \mathbf{M}\mathbf{z}_\cap$;
  $d\_init = \mathbf{x}_\cap > \mathbf{h}$;
  for $(v_1, \ldots, v_R)$ *in* $\{0, 1\}^R$ **do**
    $d = d\_init[(i_1, \ldots, i_R)].set(v_1, \ldots, v_R)$;
    $\mathbf{D}_\Omega = \text{diag}(d)$;
    $D\_set = D\_set \cup \{\mathbf{D}_\Omega\}$;

// *Find fixed points, for the at most* $\sum_{r=0}^{R} \binom{N}{r}$ *configurations*
for D_Ω *in* D_set **do**
  $\mathbf{z}^* = \text{solve}(\mathbf{N}^\mathsf{T}\mathbf{D}_\Omega\mathbf{M} - \mathbf{I}, \mathbf{N}^\mathsf{T}\mathbf{D}_\Omega\mathbf{h})$;
  $\mathbf{x}^* = \mathbf{M}\mathbf{z}^*$;
  // *check if fixed point is consistent with assumed* $\mathbf{D}_\Omega$
  if $\text{diag}(\mathbf{x}^* > \mathbf{h}) == \mathbf{D}_\Omega$ **then**
    $z\_set = z\_set \cup \{\mathbf{z}^*\}$;

.3.2 *Additional figures & tables*

.3.2.1 *Additional statistics for Teacher-Student setups*

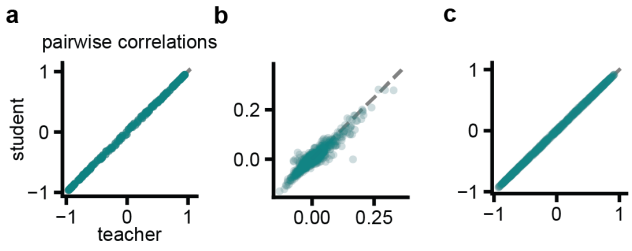

FIGURE .13: **a-c)** Pairwise correlations between units of the modes for panel **a-c**) of Fig. 4.3, respectively. Note that **c** is computed over all conditions.

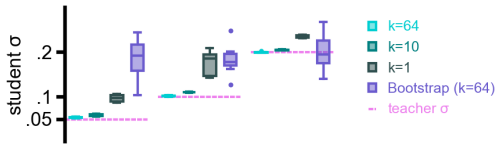

FIGURE .14: Our method allows recovering the true latent noise in student-teacher setups. We repeated the experiment of Fig. 4.3**a** for teacher networks with three levels of latent noise, with diagonal covariances matrices $\Sigma_{\mathbf{z}} = \sigma^2\mathbf{I}$. For each teacher we trained 5 student networks with varying number of particles (k), as well as using the bootstrap proposal (i.e., sampling from the prior). The standard deviations σ of the latent noise process only matches between the student and teacher, if we use enough particles during training. The bootstrap proposal (with k=64) is not as reliable as the optimal proposal.

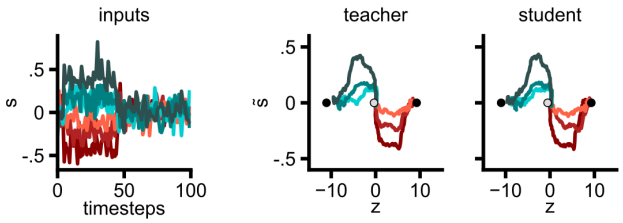

FIGURE .15: To demonstrate that our method works with time-varying input, a rank-1 teacher RNN with 60 units was trained to report the sign of a time-varying stimulus (left). We then generated 400 trials of data with observation noise covariance $\Sigma_{\mathbf{x}} = .01\mathbf{I}$ and latent noise covariance $\Sigma_{\mathbf{x}} = .0025\mathbf{I}$. We trained a student on the observed activity of the teacher for 400 epochs. The matching latent dynamics of the student and teacher lie in the column space of the recurrent and input weights (right; coordinates z and $\tilde{s}$, respectively; see Supplement .3.3.2).

.3.2.2 *EEG: Inferring full-rank RNNs*

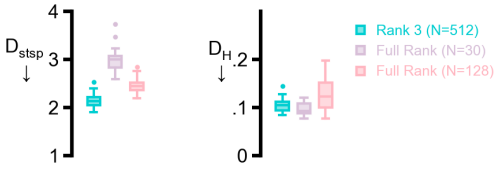

FIGURE .16: We fit full-rank RNNs to EEG data, by parameterising the mean of the transition distribution as $F(\mathbf{z}_t) = a\mathbf{z}_t + (1-a)\mathbf{J}\phi(\mathbf{z}_t)$. We trained full-rank RNNs with 30 units (a roughly similar amount of parameters as our rank-3 RNNs with 512 units), as well as full-rank RNNs with 128 units (over 10 times more parameters). The KL divergence-based measure (D_{stsp}), between generated samples and data is worse for the full-rank RNNs, while also being less interpretable.

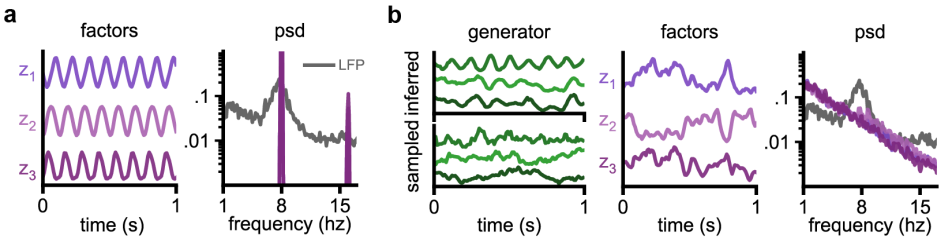

FIGURE .17: Latent Factor Analysis via Dynamical Systems (LFADS) [46] is a current method that can be used to fit RNNs to neural data, and while it is generally used for inference, can also be used for generation. We here explored sampling from LFADS, both using the autonomous version, and when using the controller (i.e., stochastic inputs). We fitted LFADS to the HPC-2 dataset (using AutoLFADS for model selection [229]). **a**) In the case of an autonomous LFADS model, one samples an initial condition from the prior and then simulates a deterministic RNN forward. As on long sequences not all variability can be explained by variability in the initial condition, the latents end up not representing any variability that resembles the underlying system (cf. Fig. 4.5). **b**) In the case of a full LFADS model with the controller, one can sample both an initial condition and time-varying inputs from the controller's auto-regressive prior. Here the full model seemed to rely overly on the controller's data-inferred inputs for inference, which deviated quite strongly from the samples from the controller's auto-regressive prior. As a consequence, the generated latents do not seem to represent variability that is meaningful.

.3.2.3 *HPC-2: Additional results*

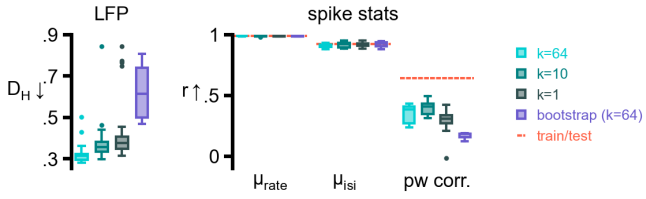

FIGURE .18: The Hellinger distance (D_H) between the power spectrum of latents and LFP of HPC-2 is lower when using the bootstrap proposal or too few particles (left), and simulated data is slightly worse when using the bootstrap proposal (right)

.3.2.4 *HPC-11: Additional results*

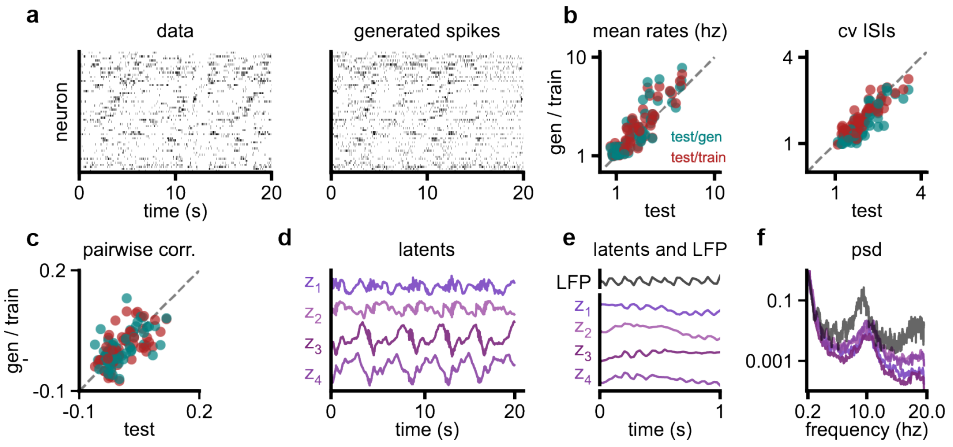

FIGURE .19: **a**) We fit a rank-4 RNN to spikes recorded from rat hippocampus [180–182], and generate new samples from the RNN (right), taking only the part of the recording where the rat is running. **b**) Single neuron statistics. The mean rates and coefficient of variations of interspike interval (ISI) distributions of a long trajectory of data generated by the RNN (gen) match those of a held-out set of data (test). As a reference we additionally computed the same statistics between the train and test set. **c)** Population level statistics. The pairwise correlations between neurons for generated data and the test data. **d**) The corresponding latents generated by the RNN consists of 10Hz (fast theta) oscillations on top of slower oscillations. **e**) Latents with further zooming in (on time), shown together with the LFP signal. **f**) The power spectrum of latents sampled from the RNN, which show power at a slightly higher frequency than that of the LFP [143].

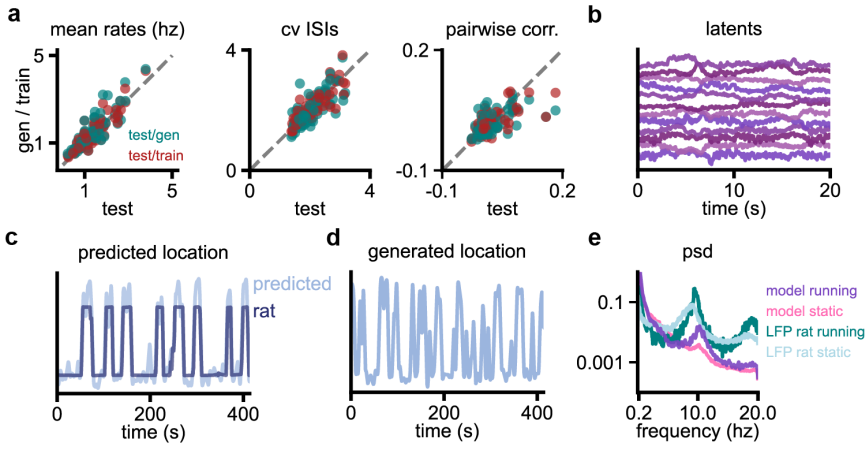

FIGURE .20: We additionally fit a rank-12 RNN to a whole recording (2067 seconds resampled to 40 Hz) which includes long bouts where the rat is stationary. **a**) We generate a new sample from the RNN and obtain matching spike statistics. **b**) Generated latents by our model. **c**) Inferred posterior latents can again be used to predict the location of the rat on a held-out set. **d**) Using the same decoder on generated latents, we obtain a model that also predicts alternating bouts of stationarity and running. **e**) The mean power of the latents at theta frequency during running bouts is higher then during stationarity bouts. The increased theta power during running is also there in the LFP data — again with latent dynamics obtained from spiking data oscillating at a slightly higher frequency than the LFP.

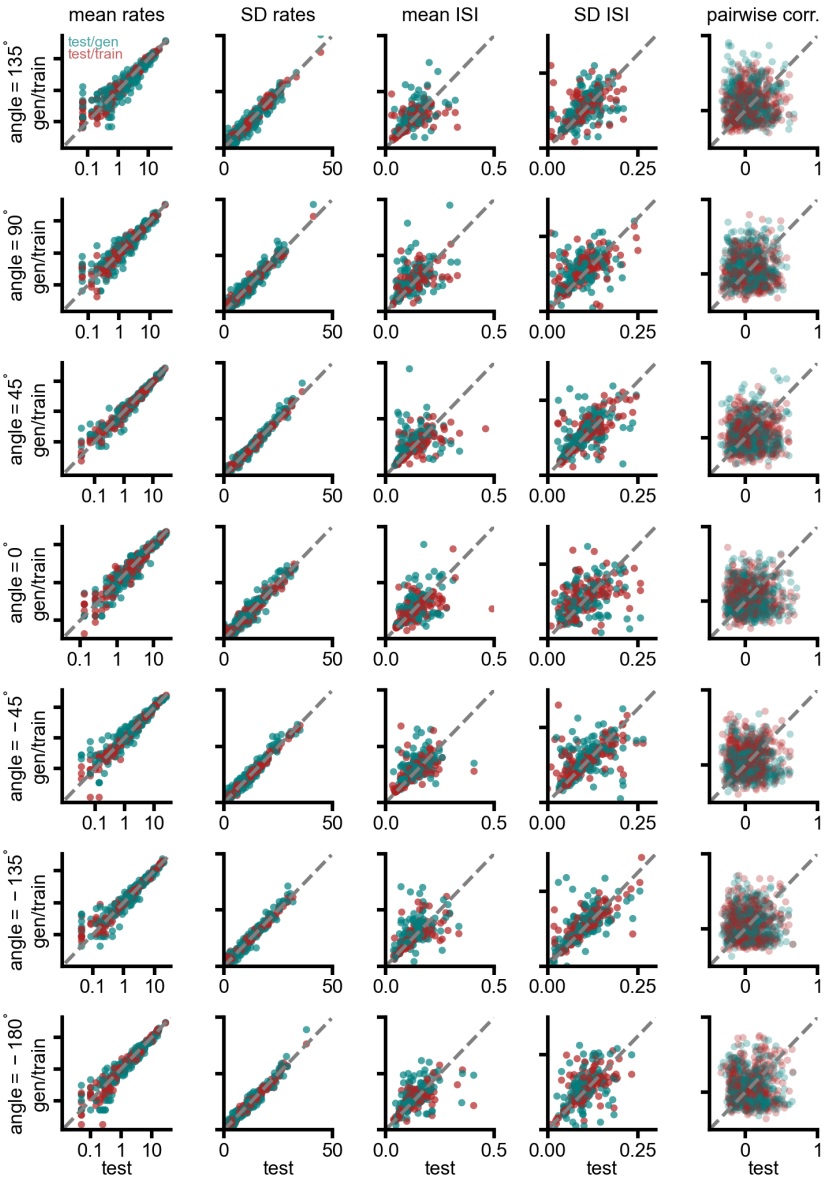

FIGURE .21: Spiking statistics of model-generated (teal) and train data (brick red) compared against test data.

.3.2.5 *Neural Latents Benchmark evaluation*

We applied our method to the `MC_Maze` dataset of the Neural Latents Benchmark (NLB) [185] at 20 millisecond bin size (Table .1), by using our method to obtain expected Poisson rates, given a filtering posterior over latents. The benchmark evaluates methods on a number of metrics: 'co-bps' (co-smoothing bits-per-spike) assesses the quality of firing rate predictions for a set of held-out neurons that are unobserved in the test data, evaluated with the Poisson likelihood of the true spiking activity given the rate predictions. 'vel R2' evaluates how well the model's inferred firing rates can predict the subject's hand velocity. 'PSTH R2' evaluates how well peri-stimulus time histograms (PSTHs) computed from model-inferred rates match empirical PSTHs from the data. 'fp-bps' evaluates predictions on heldout timesteps (which we predict by running the RNN forward from the last data-inferred step). We found that our method outperforms classical methods (GPFA [230] and SLDS [42]) while certain state-of-the-art deep learning (LFADS [46, 229], Neural Data Transformer [28]) are slightly better than our method on the 'co-bps' metric, but our method matches them in the 'vel R2' metric (in case we include smoothing information in the proposal) . We do note that NLB metrics center around evaluating the quality of smooth rates *inferred* from spikes, which is not the central focus of our method, which is *generation*, i.e., sampling noisy trajectories that reproduce variability in the data. We here found that the quality of inference increased when using an non-causal CNN encoder as part of the proposal distribution, and additional gains might be obtained by also changing the target distribution to a smoothing (instead of a filtering) one [231].

While our method also has comparatively lower dimensionality than the other deep learning approaches, a latent dimensionality of 36 is still considerably higher than all networks considered in the Main text. We reason that we need a high number of latents, because the full `MC_Maze` dataset has a large number of conditions (108), spanning multiple maze-configurations, which may be difficult to fully model with autonomous low-dimensional latent dynamics.

.3.2.6 *Stimulus-conditioning in monkey reaching task*

For the experiment with stimulus-conditioned dynamics in the monkey reaching task, we tested the performance of models with and without the conditioning inputs. We found that the conditioning inputs allow the networks to perform better on velocity decoding at lower dimensionalities.

Following the analysis in Fig. 5.5, we also further visualized the model's match to the spiking statistics, including mean and standard deviation (SD) of spiking rate, and mean, SD, and coefficient of variation (CV) of inter-spike intervals. We observed a good match to the mean and SD of the spiking rate across all conditions. Match to ISI statistics is also quite reasonable given the noise observed between estimates of the statistics from train and test.

method	dim	co-bps ↑	vel R2 ↑	PSTH R2 ↑	fp-bps ↑
Spike smoothing	137	0.2076	0.6111	−0.0005	—
GPFA	52	0.2463	0.6613	0.5574	—
SLDS	38	0.2117	0.7944	0.4709	−0.1513
LFADS	100	0.3554	0.8906	0.6002	0.2454
NDT	274	0.3597	0.8897	0.6172	0.2442
Ours	36	0.3225	0.8479	0.5927	0.2184
Ours (non-causal)	36	0.3407	0.8902	0.5963	0.2417

TABLE .1: Performance of our method on the `MC_Maze` dataset of the Neural Latents Benchmark, 'dim' refers to the dimensionality of the model's underlying dynamics (where possible).

conditioning	dim	vel R2 ↑
w/o conditioning	5	0.7897 ± 0.0687
	6	0.8944 ± 0.0039
	8	0.9085 ± 0.0048
	16	0.9196 ± 0.0041
with conditioning	5	0.8589 ± 0.0493
	6	0.9018 ± 0.0114

TABLE .2: Performance benefits of conditioning for monkey reaching task.

.3.2.7 *Comparison to an approximate method for finding fixed points*

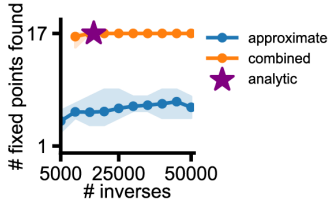

FIGURE .22: Repetition of the experiment of Fig. 4.8, but now with a rank-2 RNN with 128 units. Again, we show the number of fixed points found as a function of the number of matrix inverses computed, with errorbars denoting the minimum and maximum amount of fixed points found over 20 independent runs of the algorithm.

Recently an approximate method for finding fixed points in piecewise-linear RNNs was proposed [70]. The method proceeds by randomly selecting a linear region (a configuration of $\mathbf{D}_\Omega$, see Supplement .3.1.2) and calculating the corresponding fixed-point. If it is indeed a 'true' fixed point of the RNN (it is consistent with the assumed $\mathbf{D}_\Omega$), we store it. If the fixed point was inconsistent with the assumed $\mathbf{D}_\Omega$, we iteratively initialize $\mathbf{D}_\Omega$ according to the 'virtual' fixed point found and calculate the new fixed point corresponding to this $\mathbf{D}_\Omega$, until we either reach a 'true' fixed point or reach a certain amount of iterations. Then, we reinitialize at a randomly selected new configuration of $\mathbf{D}_\Omega$ and repeat the procedure.

Under some conditions, the approximate method can be shown to converge in linear time ($\|\mathbf{M}\tilde{\mathbf{N}}^\mathsf{T}\| + \|a\mathbf{I}\| \leq 1$) [64], where it will be faster than our exact method — however in general the convergence of the approximate method strongly depends on the dynamics of the networks. In particular, there are reasonable settings where the approximate method fails to find all fixed points, such as of a rank-2, 128 unit RNN with 17 fixed points (trained similarly to the teacher RNN of Fig. 4.3**c**; Fig. S.22). While an in-depth study of the approximate method is out of scope, we hypothesize that the failure to converge is because when initializing with randomly selected $\mathbf{D}_\Omega$s out of $(D+1)^N$ possible configurations, the approximate method tends to converges to the same set of $\mathbf{D}_\Omega$s.

Our method is completely independent of the dynamics of the system and has a fixed cost, after which one is guaranteed that all fixed points are found. However, we do note that there can be scenarios where our exact method is still too costly. In this scenario, we propose to use the approximate method, with one adjustment - we first pre-compute the subset of $\sum_r^R D^r\binom{N}{r}$ configurations that can contain fixed points, and then initialize the approximate method using randomly selected $\mathbf{D}_\Omega$ from this subset. Empirically, this leads to better convergence in at least some scenarios (Fig. 4.8, Fig. S.22)

For the approximate method, we used code from:
https://github.com/DurstewitzLab/CNS-2023, which was released with the GNU General Public License.

.3.3 *Additional details of low-rank RNNs*

.3.3.1 *Discretisation*

Given

$$\tau\frac{d\mathbf{z}}{dt} = -\mathbf{z}(t) + \mathbf{N}^\mathsf{T}\phi(\mathbf{M}\mathbf{z}(t)) + \Gamma_\mathbf{z}\xi(t),$$

Using the Euler–Maruyama method with timestep Δ_t:

$$\mathbf{z}_{t+1} = (1-\frac{\Delta_t}{\tau})\mathbf{z}_t + \frac{\Delta_t}{\tau}\mathbf{N}^\mathsf{T}\phi(\mathbf{M}\mathbf{z}_t) + \frac{\sqrt{\Delta_t}}{\tau}\Gamma_\mathbf{z}\epsilon_t,$$

and with $\epsilon_t \sim \mathcal{N}(0,\mathbf{I})$, define $a = 1-\frac{\Delta_t}{\tau}$, $\tilde{\mathbf{N}} = \frac{\Delta_t}{\tau}\mathbf{N}$, and $\Sigma_\mathbf{z} = \frac{\Delta_t}{\tau^2}\Gamma_\mathbf{z}\Gamma_\mathbf{z}^\mathsf{T}$, we obtain the transition distribution used in our experiments. Note the slight 'overloading' of

t here, as the discrete time indice t of e.g., $\mathbf{z}_t$ corresponds to the continuous time $\mathbf{z}((t-1)\Delta_t)$.

.3.3.2 *Conditional generation*

Given input weights $\mathbf{H} \in \mathbb{R}^{N \times N_s}$ and stimulus $\mathbf{s} \in \mathbb{R}^{N_s}$, we define our model as

$$\tau \frac{d\mathbf{x}}{dt} = -\mathbf{x}(t) + \mathbf{J}\phi(\mathbf{x}(t)) + \mathbf{H}\mathbf{s}(t) + \xi_{\mathbf{x}}.$$

Using the same assumptions as before, $\mathbf{x}$ can be described by $R + N_s$ variables

$$\begin{aligned}\tau \frac{d\mathbf{z}}{dt} &= -\mathbf{z}(t) + \mathbf{N}^\mathsf{T}\phi(\mathbf{M}\mathbf{z}(t) + \mathbf{H}\tilde{\mathbf{s}}(t)) + \xi_{\mathbf{z}},\\ \tau \frac{d\tilde{\mathbf{s}}}{dt} &= -\tilde{\mathbf{s}}(t) + \mathbf{s}(t),\end{aligned}$$

with $\mathbf{x} = \mathbf{M}\mathbf{z} + \mathbf{H}\tilde{\mathbf{s}}$, and $\begin{bmatrix}\mathbf{z}\\ \tilde{\mathbf{s}}\end{bmatrix} = ([\mathbf{M}, \mathbf{H}]^\mathsf{T}[\mathbf{M}, \mathbf{H}])^{-1}[\mathbf{M}, \mathbf{H}]^\mathsf{T}\mathbf{x}$.

We can write the distribution generated after discretization as:

$$\begin{aligned}p(\mathbf{z}_{1:T}, \mathbf{y}_{1:T}, \tilde{\mathbf{s}}_{1:T} | \mathbf{s}_{1:T-1}) &= p(\mathbf{z}_1)p(\tilde{\mathbf{s}}_1)\prod_{t=2}^{T} p(\tilde{\mathbf{s}}_t|\mathbf{s}_{t-1})p(\mathbf{z}_t|\tilde{\mathbf{s}}_{t-1}, \mathbf{z}_{t-1})\prod_{t=1}^{T} p(\mathbf{y}_t|\tilde{\mathbf{s}}_t, \mathbf{z}_t),\\ p(\mathbf{z}_t|\tilde{\mathbf{s}}_{t-1}, \mathbf{z}_{t-1}) &= \mathcal{N}(F(\tilde{\mathbf{s}}_{t-1}, \mathbf{z}_{t-1}), \Sigma_{\mathbf{z}}), \quad p(\mathbf{z}_1) = \mathcal{N}(\mu_{\mathbf{z}_1}, \Sigma_{\mathbf{z}_1}),\\ p(\tilde{\mathbf{s}}_t|\tilde{\mathbf{s}}_{t-1}, \mathbf{s}_{t-1}) &= \delta(a\tilde{\mathbf{s}}_{t-1} + (1-a)\mathbf{s}_{t-1}), \; p(\tilde{\mathbf{s}}_1) = \delta(\mathbf{0}),\end{aligned}$$

where the mean of the transition distribution is $F(\tilde{\mathbf{s}}_t, \mathbf{z}_t) = a\mathbf{z}_t + \tilde{\mathbf{N}}^\mathsf{T}\phi(\mathbf{M}\mathbf{z}_t + \mathbf{H}\tilde{\mathbf{s}}_t)$.

For constant input $\mathbf{s}$, $\tilde{\mathbf{s}}$ will converge to $\mathbf{s}$, and we can ignore the additional N_s variables, assuming $\mathbf{x}(0) = \mathbf{M}\mathbf{z}(0) + \mathbf{s}$. Similarly if $\mathbf{s}$ varies on a time scale slower than τ, $\mathbf{s} \approx \tilde{\mathbf{s}}$ is a good approximation [56]. Here, for experiments where the input is a constant context signal (Fig. 5.5), we substitute $\mathbf{s}$ for $\tilde{\mathbf{s}}$ and consider the R dimensional system described by $\mathbf{z}$ (which now has additional conditioning on $\mathbf{s}$):

$$\begin{aligned}p(\mathbf{z}_{1:T}, \mathbf{y}_{1:T}|\mathbf{s}_{1:T}) &= p(\mathbf{z}_1)\prod_{t=2}^{T} p(\mathbf{z}_t|\mathbf{s}_{t-1}, \mathbf{z}_{t-1})\prod_{t=1}^{T} p(\mathbf{y}_t|\mathbf{s}_t, \mathbf{z}_t),\\ p(\mathbf{z}_t|\mathbf{s}_{t-1}, \mathbf{z}_{t-1}) &= \mathcal{N}(F(\mathbf{s}_{t-1}, \mathbf{z}_{t-1}), \Sigma_{\mathbf{z}}), \; p(\mathbf{z}_1) = \mathcal{N}(\mu_{\mathbf{z}_1}, \Sigma_{\mathbf{z}_1}),\end{aligned}$$

where $F(\mathbf{s}_t, \mathbf{z}_t) = a\mathbf{z}_t + \tilde{\mathbf{N}}^\mathsf{T}\phi(\mathbf{M}\mathbf{z}_t + \mathbf{H}\mathbf{s}_t)$.

.3.3.3 *Linear transformations of the latent space and orthogonalisation*

Given

$$\begin{aligned}\mathbf{x}_{t+1} &= a\mathbf{x}_t + \mathbf{M}\tilde{\mathbf{N}}^\mathsf{T}\phi(\mathbf{x}_t) + \epsilon_{\mathbf{x}},\\ \mathbf{z}_{t+1} &= a\mathbf{z}_t + \tilde{\mathbf{N}}^\mathsf{T}\phi(\mathbf{M}\mathbf{z}_t) + \epsilon_{\mathbf{z}},\end{aligned}$$

with $\epsilon_{\mathbf{z}} \sim \mathcal{N}(0, \Sigma_{\mathbf{z}})$, $\epsilon_{\mathbf{x}} \sim \mathcal{N}(0, \mathbf{M}\Sigma_{\mathbf{z}}\mathbf{M}^{\mathsf{T}})$. We can do any linear transformation of the latent dynamics $\mathbf{z}$: $\hat{\mathbf{z}} = \mathbf{A}\mathbf{z}$, as long as $\mathbf{A}$ has rank R, without changing the neuron activity $\mathbf{x}$. To see this, define $\hat{\mathbf{M}} = \mathbf{M}\mathbf{A}^{-1}$, $\hat{\mathbf{N}} = \mathbf{A}\tilde{\mathbf{N}}$, and $\epsilon_{\hat{\mathbf{z}}} \sim \mathcal{N}(0, \mathbf{A}\Sigma_{\mathbf{z}}\mathbf{A}^{T})$, giving us:

$$\mathbf{x}_{t+1} = a\mathbf{x}_t + \hat{\mathbf{M}}\hat{\mathbf{N}}^{\mathsf{T}}\phi(\mathbf{x}_t) + \epsilon_{\mathbf{x}},$$
$$\hat{\mathbf{z}}_{t+1} = a\hat{\mathbf{z}}_t + \hat{\mathbf{N}}^{\mathsf{T}}\phi(\hat{\mathbf{M}}\hat{\mathbf{z}}_t) + \epsilon_{\hat{\mathbf{z}}},$$

which will leave $\mathbf{x}$ unchanged, while our latents $\mathbf{z}$ are expressed in a new basis. We typically got a more interpretable visualization of the latents by orthonormalising the columns of $\mathbf{M}$. Thus we applied for all visualisations after training $\mathbf{A} = \mathbf{U}^{\mathsf{T}}\mathbf{M}$, with $\hat{\mathbf{M}} = \mathbf{U}$, where $\mathbf{U}$ are the first R left singular vectors of $\mathbf{J} = \mathbf{M}\mathbf{N}^{\mathsf{T}}$.

.3.4 *Details of empirical experiments*

.3.4.1 *Training details*

.3.4.2 *Initialisation*

Our models are (unless noted otherwise) initialized as follows:

$$\begin{aligned}
\tilde{\mathbf{N}}_{ij} &\sim \mathcal{U}_{[-\frac{1}{\sqrt{N}}, \frac{1}{\sqrt{N}}]}, \\
\mathbf{M}_{ij} &\sim \mathcal{U}_{[-\frac{1}{\sqrt{R}}, \frac{1}{\sqrt{R}}]}, \\
\mathbf{W}_{ij} &\sim \mathcal{N}(0, \frac{2}{R}), \\
\mathbf{H}_{ij} &\sim \mathcal{U}_{[-\frac{1}{\sqrt{N_{inp}}}, \frac{1}{\sqrt{N_{inp}}}]}, \\
\mathbf{h}_i &\sim \mathcal{U}_{[-\frac{1}{\sqrt{N}}, \frac{1}{\sqrt{N}}]}, \\
\mathbf{b} &\leftarrow \mathbf{0}, \\
a &\leftarrow .9, \\
\Sigma_{\mathbf{z}} &\leftarrow .01\mathbf{I}, \\
\Sigma_{\mathbf{z}_1} &\leftarrow \mathbf{I}, \\
\mu_{\mathbf{z}_1} &\leftarrow \mathbf{0},
\end{aligned}$$

where $\mathbf{W}$ and $\mathbf{b}$ are the output weights and biases respectively. For Gaussian observations we initialise $\Sigma_{\mathbf{y}} \leftarrow .01\mathbf{I}$.

For experiments with Poisson observations, we jointly optimized a (usually causal) CNN encoder as part of the proposal distribution. The CNN was conditioned on observations and predicted the mean and log variance of a normal distribution. It consisted of common initial layers consisting of 1D convolutions, with a GeLU

activation function, and a separate output convolution for the predicted mean and (log) variance. The CNN was initialized to the Pytorch [170] defaults, except for the bias of the log variance output layer, to which we added a $\log(.01)$ term, such that the output matches the initially predicted variance of the RNN. The exact number of layers and channels are reported in the sections for each experiment.

For the teacher-student setups, we used as non-linearity $\phi(\mathbf{x}_i) = \max(\mathbf{x}_i - \mathbf{h}_i, 0)$ for both the students and the teachers, and for all experiments with real-world data, we used the 'clipped' $\phi(\mathbf{x}_i) = \max(\mathbf{x}_i + \mathbf{h}_i, 0) - \max(\mathbf{x}_i, 0)$ [64].

.3.4.3 *Parameterisation*

We constrain a to be between 0 and 1 by instead optimising $\tilde{a}$ with the following (sigmoidal) parameterisation $a = \exp(-\exp(\tilde{a}))$ [219]. In experiments with the optimal proposal, we estimate the full $\Sigma_{\mathbf{z}}$, which we constrain to be symmetric positive definite, by optimizing a lower triangular matrix $\mathbf{C}$ such that $\Sigma_{\mathbf{z}} = \mathbf{C}\mathbf{C}^T$, where we additionally constrain the diagonal of $\mathbf{C}$ to be positive using $\mathbf{C}_{ii} = \exp(\tilde{\mathbf{C}}_{ii}/2)$. For all diagonal covariances, we parameterise the diagonal elements using $\Sigma_{ii} = \exp(\tilde{\Sigma}_{ii})$. For Poisson observations, we apply a $\mathsf{Softplus}$ function to rectify the predicted rate.

.3.4.4 *Optimisation*

During training we minimise the variational SMC ELBO [115–117] (Eq.4.7) with stochastic gradient descent, using the RAdam [218] optimiser in Pytorch [170]. We generally use an exponentially decaying learning rate (details under each experiment).

.3.4.5 *Teacher student experiments*

.3.4.6 *Dataset description*

We created datasets by first training 'teacher' RNNs to perform a task and then generating observations by simulating the trained teacher RNNs.

For Fig. 4.3**a**, **b** we used code from [172] (https://github.com/mackelab/phase-limit-cycle-RNNs, Apache licence) to train rank-2 RNNs to produce oscillations, using a sine-wave with a periodicity of 50 time-steps as a target and an additional L2 regularisation on the rates. After training, we extracted the recurrent weights $\mathbf{M}, \mathbf{N}$ and biases $\mathbf{h}$, orthonormalized the columns of $\mathbf{M}$, and created a dataset by simulating the model for 75 timesteps, with $\Sigma_{\mathbf{z}} = .04\mathbf{I}$. For Fig. 4.3**a** we used $N = 20$ units and generated observations according to $G = \mathcal{N}(\mathbf{M}\mathbf{z}_t, \Sigma_{\mathbf{y}})$, with $\Sigma_{\mathbf{y}} = .01\mathbf{I}$. Fig. 4.3**b** we used $N = 40$ units and generated observations according to $G = \mathsf{Pois}(\mathsf{Softplus}(w\mathbf{M}\mathbf{z}_t - b)$, with $w = 4$ and $b = 3$.

For Fig. 4.3**c**, we followed a similar procedure but now trained the teacher RNN on a task where it has to use input. After an initial period of 25 time steps, a stimulus was presented for 25 timesteps consisting of $[\sin(\theta), \cos(\theta)]^{\mathsf{T}}$, where θ was randomly

selected every trial out of 8 fixed angles. The RNN was tasked to produce output that equals the transient stimulus for the next 100 time-steps. Here we used $N = 60$ units, $\Sigma_{\mathbf{z}} = .0025\mathbf{I}$ and generated observations according to $G = \mathcal{N}(\mathbf{M}\mathbf{z}_t, \Sigma_{\mathbf{y}})$, with $\Sigma_{\mathbf{y}} = .0025\mathbf{I}$. The training data for the student RNN was included for each trial the corresponding stimulus.

.3.4.7 *Training details*

The 'student' RNNs had $20, 40, 60$ units, respectively and rank $R = 2$, matching that of the teacher RNNs. For Fig. 4.3**a**, **c**. The observation model was a linear Gaussian according to $G = \mathcal{N}(\mathbf{M}\mathbf{z}_t, \Sigma_{\mathbf{y}})$, and we used the optimal proposal distribution. For Fig. 4.3**b** we used $G = \mathsf{Pois}(\mathsf{Softplus}(\mathbf{W}\mathbf{M}\mathbf{z}_t - \mathbf{b}))$, with $\mathbf{W}$ a diagonal matrix (scaling the output of each unit individually). For Fig. 4.3**b**, we used a causal CNN encoder as part of the proposal distribution. It consisted of 3 layers, with kernel sizes $(21, 11, 1)$, and channels $(64, 64, 2)$. We used (causal) circular padding.

For all three experiments, we used $k = 64$ particles, batch-sizes of 10, and decreased the learning rate exponentially from 10^{-3} to 10^{-5}. For Fig. 4.3**a** we trained for 1000 epochs of 200 trials, for Fig. 4.3**b** for 1500 epochs of 400 trials and for Fig. 4.3**c** for 200 epochs of 800 trials. We used a workstation with a NVIDIA GeForce RTX 3090 GPU for these runs. One model took about 3 to 4 hours to finish training.

.3.4.8 *Evaluation setup*

For Fig. 4.3 we generated long trajectories of $T = 10000$ time-steps of data for both the student and teacher RNNs. To facilitate visual comparisons between student and teacher dynamics, we also orthonormalized the columns of the students weights $\mathbf{M}$ after training, and for Fig. 4.3**a,c** picked signs of the columns of $\mathbf{M}$ such that the student and teacher match (note that after orthormalizing, the columns of $\mathbf{M}$ are equal to the non-zero singular vectors of the full weight matrix $\mathbf{J}$, which are only unique up to a sign flip). As noted before, this leaves the output of the model unchanged. The autocorrelation in Fig. 4.3**a** was computed by convolving a sequence of lag$=$ 120 steps of data with itself (with duration $2\times$lag), and normalising such that lag=0 corresponds to a correlation of 1. We repeated this for 80 sequences starting at different time-points of the whole trajectory.

.3.4.9 *EEG data*

.3.4.10 *Dataset description*

We used openly accessible electroencephalogram (EEG) data from [175, 176] (https://www.physionet.org/content/eegmmidb/1.0.0/, ODC-BY licence). The data was recorded from a human subject sitting still with eyes open (session S001R01), and was sampled at 160 Hz. Like [64], we used the full 1 minute of recording, but unlike [64], we did not smooth the data (but just standardized the data). Thus, to compare our performance to [64], who ran their evaluation using the smoothed

data, we smoothed our generated samples equivalently, using a Hann filter with a window length of 15-time bins, so that we can also compare our samples to the smoothed data.

.3.4.11 *Training details*

We used $N = 512$ units, and rank $R = 3$. The observation model was a linear Gaussian conditioned on the hidden state and we used the optimal proposal distribution. We trained for 1000 epochs consisting of 50 batches of size of 10, and $k = 10$ particles. The learning rate was decreased exponentially from 10^{-3} to 10^{-6}. Models were trained using NVIDIA RTX 2080 TI GPUs on a compute cluster. A single model took between 4 and 5 hours to finish training.

.3.4.12 *Evaluation setup*

We used our RNN to generate one long trajectory of $T = 9760$ steps of data, $\mathbf{y}_t$ (after discarding the first 2440 steps), which we compare to the EEG data, $\hat{\mathbf{y}}_t$, using two evaluation measures from [64, 111] (using code from https://github.com/DurstewitzLab/GTF-shPLRNN, GNU General Public License):

$\mathbf{D}_{\mathsf{stsp}}$: This is an estimate of the KL divergence between the ground truth and generated states. To compute this, we obtained kernel density estimates of the probability density functions (over states, not time), using a Gaussian kernel with standard deviation $\sigma = 1$. We get for the EEG data: $\hat{p}(\mathbf{y}) = \frac{1}{T}\sum_{t=1}^{T}\mathcal{N}(\hat{\mathbf{y}}_t, \mathbf{I})$, and for the generated data $\hat{q}(\mathbf{y}) = \frac{1}{T}\sum_{t=1}^{T}\mathcal{N}(\mathbf{y}_t, \mathbf{I})$. We then used the following Monte Carlo estimate of the KL divergence: $D_{\mathsf{stsp}} \approx \frac{1}{n}\sum \log \frac{\hat{p}(\hat{\mathbf{y}}^i)}{\hat{q}(\hat{\mathbf{y}}^i)}$, using $n = 1000$ samples $\hat{\mathbf{y}}^i$ drawn randomly from the EEG data.

$\mathbf{D}_{\mathbf{H}}$: This is an estimate of the difference in power spectra between the ground truth and generated states. We first computed for each data dimension the spectra $\hat{\mathbf{y}}_\omega^i$, $\mathbf{y}_\omega^i$ for the EEG and generated data, respectively. We used a Fast Fourier Transform, smoothed the estimates with a Gaussian kernel with standard deviation $\sigma = 20$, and normalized the spectra so they sum to 1. We computed the mean of the Hellinger distances between the spectra: $D_H = \frac{1}{64}\sum_i^{64}\frac{1}{\sqrt{2}}\|\sqrt{\hat{\mathbf{y}}_\omega^i} - \sqrt{\mathbf{y}_\omega^i}\|$.

.3.4.13 *Hippocampus HC-2*

.3.4.14 *Dataset description*

We used openly accessible neurophysiological data recorded from layer CA1 of the right dorsal hippocampus [155, 156] (https://crcns.org/data-sets/hc/hc-2/about-hc-2. Signals were recorded as the rats engaged in an open field task, chasing drops of water or pieces of food that were randomly placed. We used the session ec013.527 from rat ID ec13, which is approximately 1062 seconds long. From 37 units (neurons) we used 21 neurons that have maximal spike counts, discarding the rest of the comparatively silent neurons. We binned the spike data to 10ms. We

used the first 80 percent of the data for training, and the rest was saved for testing purposes.

.3.4.15 *Training details*

We used $N = 512$ units, and rank $R = 3$ for the run that was used in our Fig. 4.5. We used a causal CNN encoder as part of the proposal distribution, which consisted of 3 layers with kernel sizes (150, 11, 1), with (64, 64, 3) channels. During our study, we swept over multiple ranks and found that theta oscillations consistently emerged from rank 3 onwards, after which reconstruction accuracy was relatively stable. For each rank, we used three different seeds and two different first layer sizes for the encoder, 25 or 150. The duration of a randomly sampled trial (sequence length) from the whole data was 94 time steps when the first layer size was 25, and 219 when the first layer size was 150. We, however, also found that the choice of the duration did not affect the results much. We trained the model using 3000 epochs, each epoch consisting of 3000 trials with 64 batches and $k = 64$ particles. The learning rate was decreased exponentially from 10^{-3} to 10^{-6}. A single model took approximately 21 hours to finish training on a NVIDIA RTX 2080 TI GPU on a compute cluster.

.3.4.16 *Evaluation setup*

We used our RNN to generate data that matches the duration of the test data, which is 20810 time steps ($\sim$208 s) (after discarding the first 1000 steps). We compare different spike statistics of generated data with test data, and for comparison purposes, we also compared the same statistics measurements between train and test data as well. We calculated the mean firing rate of each neuron, mean of ISI distributions, and pairwise correlations. We used a band-pass filter 1-40 Hz for the latents and the LFP signal before calculating the powerspectrogram (Fig.4.5**e**).

.3.4.17 *Hippocampus HC-11*

.3.4.18 *Dataset description*

We used openly accessible neurophysiological data recorded from hippocampal CA1 region [180–182] (https://crcns.org/data-sets/hc/hc-11/about-hc-11). We used the subset of the dataset called the *maze* epoch, where a rat was running on a 1.6-meter linear track, with rewards located at each end (left and right). Throughout this task, neural activity was recorded from 120 identified pyramidal neurons. As in [69], we only used 60 neurons that had sufficient activity and discarded rest of the units. We used code from [183] (https://github.com/zhd96/pi-vae) to preprocess the spike data, and only use data corresponding to the rat running and the location data being available for the results shown in Fig. 5.4 and Fig. S.19. For the model shown in Fig. S.20 we used the last 1350s of data of the Maze epoch, which also includes bouts where the rat is stationary. We used 25ms bins.

.3.4.19 *Training details*

We used $N = 512$ units, and rank $R = 4$ for Fig. 5.4 and Fig. S.19 and rank $R = 12$ for Fig. S.20. We used a causal CNN encoder with zero padding with 3 layers (24, 11, 1), (64, 64, 4) channels, and 3 layers (150, 11, 1), (128, 64, 12) channels, respectively. The models were trained for 3000 epochs, each epoch having 3000 trials with a sequence length of 94 time bins (2.35 s) and 219 bins (5.48 s), respectively. We used batch sizes of 64 and $k = 64$ particles. The learning rate was decreased exponentially from 10^{-3} to 10^{-6}. A single model took approximately 21 hours to finish training on a NVIDIA RTX 2080 TI GPU on a compute cluster.

.3.4.20 *Evaluation setup*

We used our RNN to generate data that matches the duration of the test data, which is 4289 time steps ($\sim$107 s) (after discarding the first 1000 steps) for Fig. 5.4 and Fig. S.19 and 16539 time steps ($\sim$413 s) for Fig. S.20. We calculated the mean firing rate of each neuron, coefficient of variations of ISI distributions, and pairwise correlations. For the R^2 reported in the Main text, we fit a ridge regression model to the posterior latent variables on the training data, after smoothing with a Hann window of size 100, and apply the regression model to latents inferred from the test data.

.3.4.21 *Monkey Reach*

.3.4.22 *Dataset description*

We used the publicly available `MC_Maze` dataset from the Neural Latents Benchmark (NLB) [185] (https://dandiarchive.org/dandiset/000128, CC-BY-4.0 licence). The data were recorded from a macaque performing a delayed center-out reaching task with barriers, resulting in a variety of straight and curved reaches. For simplicity, we took only the trials with no barriers and thus straight reach trajectories, resulting in 592 training trials and 197 test trials. We binned the data at 20 ms and aligned each trial from 250 ms before to 450 ms after movement onset.

To create conditioning inputs for the model, we took the x and y coordinates of the target position for each trial and scaled them to be between -1 and 1. We then provide this scaled target position as constant context input to the RNN for the duration of the trial.

.3.4.23 *Training details*

We ran a random search of 30 different models with rank $r \in 3, 4, 5, 6$ and particle number $k \in 16, 32, 64$. All models had 512 units and used a causal CNN encoder with kernel sizes $(14, 4, 2)$ and channels $(128, 64, r)$. We used (causal) reflect padding. We trained each model for up to 2000 epochs, terminating training early if no improvement was seen for 50 epochs. Each model took around 3 to 4 hours to train on an NVIDIA RTX 2080 TI GPU on a compute cluster. Seeing that a rank of 5 was

sufficient for velocity decoding $R^2 \approx 0.9$, we took the best-performing rank-5 model for subsequent analyses.

.3.4.24 *Evaluation setup*

For qualitative evaluation of replication of cross-condition differences, we grouped the reach targets in the data into 7 conditions, one at each corner and the midpoint of each edge of the rectangular reach plane, excluding the midpoint directly at the bottom. We then generated data from the model RNN using conditioning inputs from the test trials of the real data. Then, for the test data and the model-generated data, we computed mean firing rate and inter-spike interval for each neuron for each condition. We then computed correlation distance ($1 - r$, where r is the Pearson correlation coefficient) on the neuron statistics between conditions in the test data and model-generated data.

For generation of data for Fig. 5.5**d**, **e**, we selected target locations by choosing angles from 0 to 360°, evenly spaced by 22.5°, and determined the corresponding reach endpoint on a square spanning from $(-1, -1)$ to $(1, 1)$. We then constructed conditioning inputs similar to the real data using these target locations and simulated the RNN with them. To decode the reaches, we used a linear decoder trained from inferred firing rates to reach velocity from the real data.

.3.4.25 *Neural Latents Benchmark*

.3.4.26 *Dataset description*

We again used the publicly available `MC_Maze` dataset from NLB (see Supplement .3.4.22). We resampled the data to 20 ms bin size and followed the standard data preprocessing procedures for the benchmark, as described in [185].

.3.4.27 *Training details*

We ran a random search of 30 different models with varying rank from 12 to 40 and particle number $k \in 16, 32, 64$. All models had 512 units and used a used a CNN encoder with kernel sizes $(14, 4, 2)$ and channels $(128, 64, 36)$, either with causal reflect padding or acausal zero padding. We trained each model for up to 2000 epochs, terminating training early if no improvement was seen for 50 epochs. Each model took around 10 to 12 hours to train on an NVIDIA RTX 2080 TI GPU on a compute cluster.

Because the primary task of the benchmark is co-smoothing, i.e., prediction of held-out neuron firing rates from held-in neurons, we provide the encoder with only the activity of held-in neurons. However, the observation likelihood component of the ELBO is computed on all neurons, held-in and held-out.

After training, we selected the model with the best co-smoothing score on the validation split and submitted its predictions to the benchmark for the final evaluation.

.3.4.28 *Evaluation setup*

Automated evaluation was performed on the benchmark platform, as described in [185].

We used for the prediction at timestep t, the expected Poisson rate of held-out neurons, conditioned on the activity of held-in neurons at the current and previous timesteps, by making use of the filtering Posterior (Eq. 4.3). We averaged over 32 sets of trajectories with 192 particles each.

.4 SUPPLEMENTARY INFORMATION CHAPTER 5

.4.1 *Ring attractor model*

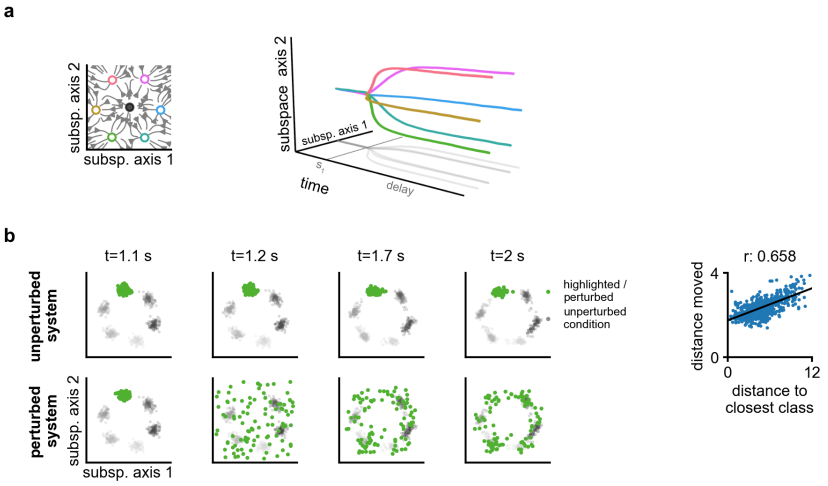

FIGURE .23: **a)** We constructed a rank-2 ring-attractor RNN, with 6 stable fixed points (left). Example trajectories for 6 different stimulus inputs are shown on the right. **b)** Perturbations reveal the underlying attractor structure. We perturb one of the stimulus conditions (green). After perturbation trajectories converge to one of the 6 stable fixed points.

.4.2 *Transient dynamics model*

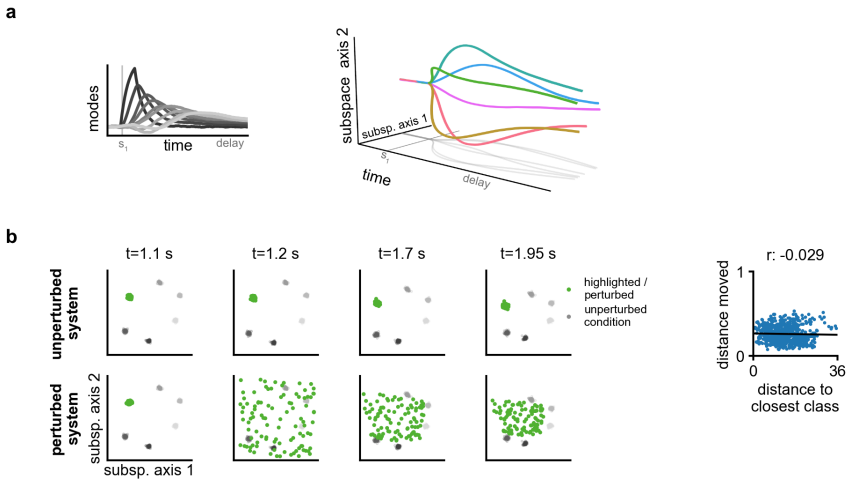

FIGURE .24: **a)** We constructed a rank-20 RNN, with 2 sets of 10 modes that sequentially activate (left). One set of modes maintains sin of the angular stimuli, and one set maintains cos. Example trajectories for 6 different stimulus inputs are shown on the right, with the subspace axis estimated similar to the main text. **b)** Perturbations again reveal the underlying attractor structure. We perturb one of the stimulus conditions (green). After perturbation trajectories do not move towards the different class means, indicating no evidence for a multi-stable attractor.

.4.3 *Student-teacher setup with attractor dynamics*

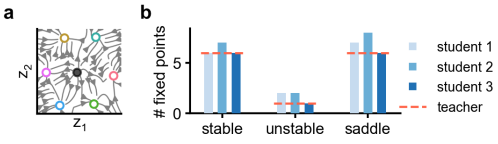

FIGURE .25: **a)** We fit student models with rank 2 to data generated from the teacher model in Fig. S. .23. Flow fields from the students matches that of the teachers quantitatively. **b)** Generally the students (3 seeds) are able to capture the multi-stable, approximate ring attractor of the teacher, however the students do sometimes find one additional spurious stable fixed point on the attractive manifold.

.4.4 *No spurious attractors in student-teacher setup with transient teacher*

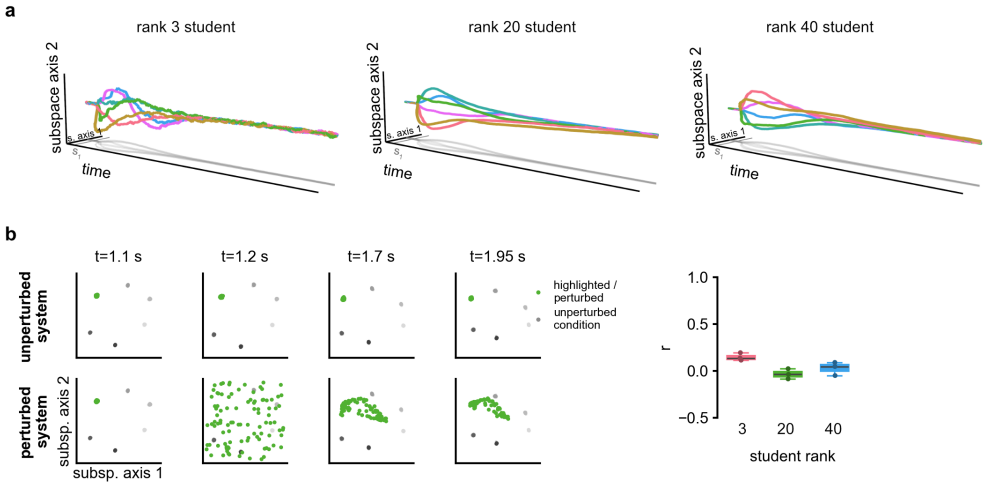

FIGURE .26: **a)** We fit student models with rank 3, 20 and 40, to data generated from the rank 20 teacher model in Fig. S. .24. In particular the rank 3 student case is interesting as based on Qian *et al.* [216], we might expect to find spurious attractors when the student does not have the capacity to match the teacher. In order to check whether or not the models learn transient dynamics, we first simulate the RNNs for an extended time after stimulus onset. For all models the latent dynamics collapse, indicating the stimulus maintenance was indeed transient. **b)** We also performed perturbation analysis. We plot one example perturbation on the left, and next, similar to Fig. 5.4, we compute the correlation coefficient over distance moved compared to how far the perturbations pushed the trajectories away from the class means. Across three seeds per student rank, we do not find any strong support for attractor dynamics.