Strong and weak principles of Bayesian machine learning for systems neuroscience

Kristopher Torp Jensen

Department of Engineering
University of Cambridge

This dissertation is submitted for the degree of
Doctor of Philosophy

Jesus College

April 2023

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements. This dissertation contains fewer than 65,000 words including appendices, footnotes, tables and equations and has fewer than 150 figures.

Kristopher Torp Jensen
April 2023

Strong and weak principles of Bayesian machine learning for systems neuroscience

Kristopher Torp Jensen

Neuroscientists are recording neural activity and behaviour at a rapidly increasing scale. This provides an unprecedented window into the neural underpinnings of behaviour, while also pushing the need for new techniques to analyse and model these large-scale datasets. Inspiration for such tools can be found in the Bayesian machine learning literature, which provides a set of principled techniques that allow us to perform inference in complex problem settings with large parameter spaces. When applied to neural population recordings, we propose that these approaches can be divided into 'weak' and 'strong' models of neural data. The weak models consist of tools for analysing experimental data, which build our own prior knowledge of neural circuits directly into the analysis pipeline. In contrast, strong Bayesian models of neural dynamics posit that the brain itself performs something akin to Bayesian inference. In this view, we can interpret our Bayesian machine learning models as algorithmic or mechanistic models of the learning processes and computations taking place in the biological brain. In this work, we first provide an overview of Bayesian machine learning and its applications to neuroscience, highlighting how both the strong and weak approaches have improved our understanding of neural computations in recent years. We then develop several new models in this field, which provide insights into neural computations ranging from motor control to navigation and decision making. These models can be grouped into three broad categories. First, we construct a series of new 'weak' latent variable models that allow us to infer the dimensionality and topology of neural data in an unsupervised manner. We highlight the utility of such approaches on synthetic data and across several biological circuits involved in motor control and navigation. Second, we propose a new method for Bayesian continual learning and relate it to longitudinal recordings of neural activity as a 'strong' model of biological learning and memory. Finally, we develop a new 'strong' model of planning and decision making through the lens of reinforcement learning formulated as Bayesian inference. In contrast to previous network models, we explicitly build in the capacity for planning-by-simulation and show that this explains many features of both human behaviour and rodent hippocampal replays. This results in a new theory of the role of hippocampus in flexible planning. The new methods developed in this work both expand the Bayesian toolbox available to systems neuroscientists and provide new insights into the neural computations driving natural behaviours.

Acknowledgements

Science is collaborative. Much of this work has therefore been done in collaboration with my excellent colleagues as specified in detail here. I have worked with Ta-Chu Kao and Jasmine Stone on large parts of Section 3.1, some of which was included in the MPhil thesis of Jasmine Stone and which has also been previously published (Jensen et al., 2021). Ta-Chu and I were both involved in the conceptualization of the work, while I performed most of the analyses, generated most of the figures, and wrote the bulk of the text. Jasmine and Ta-Chu were both heavily involved in writing the software used for this work and in later stages of writing and editing. I also collaborated with Ta-Chu Kao on much of the work in Section 3.2, part of which was included in their PhD thesis and which has also been previously published (Jensen et al., 2020, 2022b). For this work, my primary contribution was the conceptualization of the project and carrying out most of the analyses, while Ta-Chu helped substantially with software development and later stages of writing and editing. Additionally, I worked with Ta-Chu Kao on large parts of Section 4.1, which has been published previously (Kao et al., 2021a). Ta-Chu was responsible for the initial conceptualization of this work, while we collaborated on the experiments and analyses, and I wrote much of the initial text. In Section 4.2, I also compare our computational models to data from Jensen et al. (2022a). Some of this data was used in my previously submitted MPhil thesis, and it only forms a small part of the present PhD thesis, which is more focused on the computational modelling work. The code used to run human behavioural experiments in Chapter 5 was written by Guillaume Hennequin, while I colleceted the data, performed all analyses, trained the computational models, and wrote the majority of the text – with input from Guillaume Hennequin and Marcelo Mattar in all phases of the project. This work is currently under review and is available as a preprint (Jensen et al., 2023). All of the research contained in this thesis was carried out under the supervision of Guillaume Hennequin, and I am grateful for his advice and support throughout this journey.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

1.1 Bayesian machine learning in systems neuroscience

Machine learning has had a profound influence on computational neuroscience in recent years, as is evident from the ubiquity of these approaches in both papers and conferences for systems neuroscience (Figure 1.1; Saxe et al., 2021). This has followed substantial improvements in experimental methods for recording simultaneously from large populations of neurons (Pachitariu et al., 2017; Steinmetz et al., 2021), which has ushered in a new era of population-level analyses as opposed to classical single-neuron studies (Figure 1.1; Saxena and Cunningham, 2019). Within the large toolbox of machine learning, Bayesian methods have been particularly prominent in the field of computational neuroscience (Doya et al., 2007; Yu et al., 2009). This is in contrast to other areas of machine learning, many of which have been dominated by large overparameterized models – commonly known as 'deep learning' models (LeCun et al., 2015).

The primary difference between these two approaches is that in Bayesian machine learning, the goal is to infer a *posterior distribution* over models, and to use this posterior to make appropriate decisions given the residual uncertainty (Ghahramani, 2015). In machine learning, this is often achieved by explicitly modelling the uncertainty over (some subset of) model parameters – which in this case should actually be considered *random variables* instead (MacKay, 1992). The difficulty with this approach is that it requires us to perform inference in the space of models, which is often intractable or impractical. Additionally, it forces us to explicitly impose a prior over models, which can be difficult and unintuitive (Fortuin et al., 2021). For most machine learning applications, it has therefore been more fruitful to simply increase the number of parameters rather than spending additional computational resources trying to capture uncertainty (Kaplan et al., 2020; Wenzel et al., 2020). In the limit of large datasets, this is a logical approach to take as the posterior over models will be dominated by the likelihood term in the limit of infinite data, and we can approximate the posterior as a delta function at the maximum likelihood parameter setting.

If Bayesian machine learning has thus remained relatively niche when it comes to industrial applications (Jumper et al., 2021; Ouyang et al., 2022; Wenzel et al., 2020), why is it so prominent in the neuroscience community (Doya et al., 2007)? One plausible explanation is that most neuroscience applications have *not* been in the large data regime. Indeed, large neural recordings might cover a few thousand neurons for a couple of hours, yielding $\approx 10^9$ data

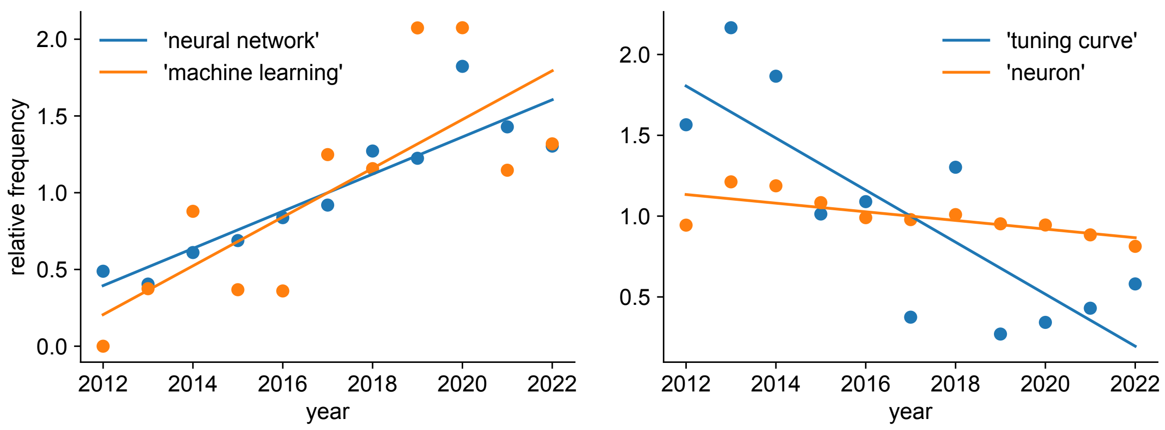

Figure 1.1 **Frequency of keywords in cosyne abstract books.** We computed the relative frequency of certain keywords in the Cosyne abstract book for all years from 2012 through 2022. The frequency was computed as the number of occurrences of each keyword, divided by the total number of words in the abstract book and normalized by the mean across all years. Left panel indicates canonical 'machine learning'-related keywords, which exhibit significantly positive trends with Pearson correlations of 0.90 for 'neural network' (p = 1e-4) and 0.77 for 'machine learning' (p = 6e-3). Right panel indicates canonical 'single neuron'-related keywords, which exhibit significantly negative trends with Pearson correlations of -0.81 for 'tuning curve' (p = 3e-3) and -0.74 for 'neuron' (p = 1e-2). We suggest that this is symptomatic of a general trend of systems neuroscience moving from single-neuron analyses towards population-based analyses over the past decade (Saxena and Cunningham, 2019).

points at a resolution of 100 Hz (Pachitariu et al., 2017; Steinmetz et al., 2021). By comparison, the ImageNet dataset (Deng et al., 2009) consists of 10^{12} pixels, which are commonly combined with additional data from applying various augmentation algorithms. This is several orders of magnitude larger than the largest neuroscience dataset but still dwarfed by the large datasets used by private companies, including the already out-of-date 10^{14}-pixel JFT-300M image dataset used by Google (Sun et al., 2017) or the 10^{11}-word MassiveText dataset used by DeepMind (Rae et al., 2021). In addition to these order-of-magnitude differences in the number of data points, image pixels and whole words contain many more bits of information than a single near-binary bin of neural activity.

Another reason for the ubiquity of Bayesian models in neuroscience might be that we have *good priors* compared to many other fields (Doya et al., 2007). As an example, we know that neural representations are likely to change continuously over time due to (i) biophysical contraints on the neurons themselves, and (ii) the smoothness of most physical processes which they might be representing (Yu et al., 2009). We also have fairly good descriptions of neural firing statistics, which we can build into our Bayesian likelihood functions (Duncker and Sahani, 2018; Keeley et al., 2020a; Liu and Lengyel, 2021; Zhao and Park, 2017). Additionally, we generally assume that neural representations are smooth functions of internal or external variables rather than being discontinuous – a necessary requirement for e.g. a 'tuning curve' to be well defined (Jensen et al., 2020). These ideas all provide soft constraints that can naturally be built into Bayesian models of neural data.

A separate reason why Bayesian ML is prominent in neuroscience is that the brain itself is often assumed to be 'Bayesian' (Doya et al., 2007), making Bayesian machine learning a natural language for describing and understanding neural computations. In this view, the brain is trying to infer some distribution over possible realities ('models') that is consistent with a set of observations, which allows us to make appropriate decisions in the face of the inevitable noise and uncertainty of the world around us (Doya et al., 2007). This also highlights an important distinction between the types of probabilistic models we commonly use in systems neuroscience: namely Bayesian machine learning as an *analysis tool* for making sense of noisy data, and Bayesian machine learning as a *mechanistic or algorithmic model* of computations performed by the brain.

We will term these two classes of models the 'weak' and 'strong' principle of Bayesian machine learning for systems neuroscience respectively – mirroring a similar recent distinction between different types of dimensionality reduction for neural data by Humphries (2020). The remainder of this section will provide a brief review of these two classes of approaches in systems neuroscience as well as a discussion of their respective use cases. This will be followed by the bulk of the thesis, which develops a series of new Bayesian approaches to systems neuroscience, starting from weak methods for data analysis and leading to stronger models of neural computations and behaviour. Finally, we will provide a summary and outlook of future directions in the field of Bayesian machine learning for neuroscience.

1.2 Weak principle of Bayesian machine learning

In systems neuroscience, it is common to record either behaviour or neural activity over extended periods of time (Dunn et al., 2021; Mathis et al., 2018; Pachitariu et al., 2017; Steinmetz et al., 2021). We often assume that these recordings constitute a noisy readout of either another observed quantity or some underlying 'latent' process (Ashwood et al., 2022; Bolkan et al., 2022; Pillow et al., 2008; Yu et al., 2009). A frequent goal is therefore to build an explicit model of this relationship in order to understand e.g. which covariates modulate the activity of a given set of neurons (Glaser et al., 2020). These models can be considered extensions of linear regression, and they are often formulated in an explicitly Bayesian framework, which allows us to build in prior knowledge such as temporal smoothness and non-negativity of firing rates (Cunningham and Byron, 2014; Pandarinath et al., 2018; Pillow et al., 2008; Schimel et al., 2021; Wu et al., 2017a; Yu et al., 2009). While these approaches can help us gain insights into the types of variables represented by neural populations, they also have practical use cases, e.g. in the context of brain-computer interfaces (Hochberg et al., 2012; Santhanam et al., 2006; Willett et al., 2021). Some of these approaches revolve around an idea that the brain is intrinsically 'low-dimensional', which motivates the use of latent variable models as a tool for extracting the underlying latent representations (Humphries, 2020). However, even in this setting they are generally not viewed as mechanistic models of neural computation, placing them in the class of 'weak' Bayesian machine learning models for systems neuroscience.

1.3 Strong principle of Bayesian machine learning

A wealth of evidence suggests that the brain takes into account uncertainty in processes ranging from perception (Ernst and Banks, 2002; Knill and Richards, 1996; Körding et al., 2007; Orbán et al., 2008) to decision making (Drugowitsch et al., 2014; Gold and Shadlen, 2001) and motor control (Heald et al., 2021; Körding and Wolpert, 2004, 2006). It has therefore been proposed that the brain solves many computational problems by performing 'inference' in a probabilistic graphical model. A prominent example comes from the sensory domain, where the visual system has been proposed to implement a process of inferring the latent factors giving rise to our sensory input (Dayan et al., 1995; Yuille and Kersten, 2006). However, similar theories have also been proposed of motor control as inverting a forward model of the motor system conditioned on a desired effect (Jordan and Rumelhart, 1992), and multi-step planning has been formulated as policy inference in a Markov decision process (Botvinick and Toussaint, 2012; Solway and Botvinick, 2012) – reminiscent of recent accounts of reinforcement learning as policy inference (Levine, 2018).

These findings and theories have also spurred a wide range of research into how the brain could carry out such 'Bayesian' computations (Rao, 2004). This includes methods centered around so-called probabilistic population codes (Beck et al., 2008; Ma et al., 2006), which resemble machine learning approaches based on variational inference (Kingma and Welling, 2013; MacKay, 2003; Rezende et al., 2014; Wainwright and Jordan, 2008). A separate strand of work suggests that the brain could perform sampling-based inference (Berkes et al., 2011; Fiser et al., 2010; Orbán et al., 2016), which resembles Bayesian machine learning methods that use Monte Carlo sampling (Hastings, 1970; Metropolis et al., 1953). Additionally, recent work has shown that circuits explicitly trained to perform such sampling-based inference exhibit cortical-like dynamics (Echeveste et al., 2020), and that learning itself can be modelled as a process of Bayesian inference (Aitchison et al., 2021). Finally, a Bayesian model known as a 'variational autoencoder', trained to predict future observations, learns similar representations to those found in the hippocampus and entorhinal cortex (Whittington et al., 2020). This has been taken as evidence that the hippocampal-entorhinal circuit could implement a similar computation. Together, these lines of work suggest that approaches and ideas from Bayesian machine learning can provide 'strong' mechanistic and algorithmic models of neural dynamics and behaviour, which can in turn inspire and constrain experiments to improve our understanding of the neural computations underlying natural behaviours.

1.4 Structure and outline

This thesis is structured as follows. First, we provide an introduction to Gaussian process regression (Section 2.1) and latent variables models (Section 2.2), which form the basis of many Bayesian approaches for neural data analysis. This is followed by an overview of approximate Bayesian inference with a focus on variational inference, since variational inference is used for computational tractability in much of the thesis (Section 2.3). We then present background sections on Bayesian continual learning (Section 2.4) and reinforcement learning (Section 2.5), both formulated as forms of probabilistic inference. Together, these five sections cover the relevant background and literature necessary to understand the original work in later chapters. We then proceed through the three main chapters containing original research. First, we present two new Bayesian latent variable models in Chapter 3, which further expand the toolbox of latent variable models for systems neuroscience. In particular, these new models facilitate inference of the latent dimensionality and topology of neural data – properties that are often of interest for neural data analysis (Humphries, 2020). These can be considered weak models of neural data, which allow us to better understand neural recordings without providing a mechanistic explanation of the observed phenomena (although they can be considered 'strong' models of dimensionality reduction in the terminology of Humphries, 2020). We then present a new Bayesian algorithm for continual learning in Chapter 4 and investigate how the neural dynamics arising from this learning paradigm compare to those associated with repeated engagement in a learned motor task in rodents. This provides a strong model of how the biological brain can retain past memories over long time periods; a topic of much debate in the recent neuroscience literature (Chestek et al., 2007; Clopath et al., 2017; Gallego et al., 2020; Jensen et al., 2022a; Rokni et al., 2007). In Chapter 5, we present a new model of planning and decision making in a reinforcement learning setting. We show how this strong model captures several features of behaviour and neural dynamics in humans and rodents and how using an appropriate prior over actions allows us to better model human planning. Finally, we discuss the utility of such Bayesian approaches for systems neuroscience in Chapter 6, where we also consider how they are likely to be used in the future given the current rapid increases in data sizes and computational capacity.

Chapter 2

Background

Notation

Bold lower-case letters are used to denote column vectors $\boldsymbol{x} \in \mathbb{R}^N$ with elements x_n, and 'hats' indicate normalized vectors $\hat{\boldsymbol{x}} = \boldsymbol{x}/\|\boldsymbol{x}\|_2$. Bold upper-case letters denote matrices $\boldsymbol{X} \in \mathbb{R}^{N \times M}$ with elements X_{nm}. $\boldsymbol{x}_n$ refers to the n$^{\text{th}}$ row of $\boldsymbol{X} \in \mathbb{R}^{N \times M}$ and $\boldsymbol{x}_m$ its m$^{\text{th}}$ column, both represented as column vectors. We will generally refer to model parameters as θ and variational parameters as ϕ. In a slight abuse of notation, θ is used to denote both a set of parameters and the corresponding parameter set stacked to form a vector. We use $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ to indicate that $\boldsymbol{x}$ is normally distributed with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$, occasionally using the notation $\mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$ to clarify that we are referring to a distribution over the random variable $\boldsymbol{x}$.

2.1 Gaussian process regression

Many popular approaches to Bayesian machine learning in systems neuroscience rely on *Gaussian processes* (GPs), and these will form the basis of much of the work in Chapter 3. We therefore provide a short practical introduction here and refer to Rasmussen and Williams (2006) for a more thorough overview. The problem we seek to solve in GP regression is to identify the functional mapping f between some set of regressors x (e.g. time or a stimulus presented to an experimental subject) and corresponding noisy observations y (e.g. position or neural firing rates).

In GP regression, we model our observations as $y = f(x) + \epsilon$ with Gaussian noise $\epsilon \sim \mathcal{N}(\epsilon; 0, \sigma_\epsilon^2)$. We additionally assume that the function f itself is sampled from a 'Gaussian process', which means that the result $\boldsymbol{f} \in \mathbb{R}^N$ of evaluting f at any finite number of inputs $\boldsymbol{x} \in \mathbb{R}^N$ will be jointly Gaussian. In other words, for finite N, we have

$$\boldsymbol{f} \sim \mathcal{N}(\boldsymbol{f}; \boldsymbol{\mu}, \boldsymbol{K}), \tag{2.1}$$

where $\boldsymbol{K} = k(\boldsymbol{x}, \boldsymbol{x}) \in \mathbb{R}^{N \times N}$ is the covariance matrix with elements $K_{ij} = k(x_i, x_j)$ for some kernel $k(\cdot, \cdot)$. $\boldsymbol{\mu}$ comes from a mean function, which is commonly assumed to be zero (Rasmussen and Williams, 2006). We can then write the distribution over the noisy observations $\boldsymbol{y}$ as

$$\boldsymbol{y} \sim \mathcal{N}(\boldsymbol{y}; 0, \boldsymbol{K} + \sigma_\epsilon^2 \boldsymbol{I}). \tag{2.2}$$

Our goal is now to *infer* the function f which gave rise to these noisy observations, evaluated at a finite set of test point $\boldsymbol{x}^*$. Additionally, we seek not just a single point estimate of f, but rather the full posterior distribution over functions that could have given rise to the data. To do this, we rely on Bayes' theorem, which gives us a recipe for evaluating such posteriors:

$$p(f|\boldsymbol{y}) \propto p(\boldsymbol{y}|f)p(f). \tag{2.3}$$

Here, $p(f)$ is a prior over functions induced by the kernel $k(\cdot,\cdot)$. By choosing an appropriate kernel, we can therefore build inductive biases into our model of the data, such as smoothness, linearity, or periodicity.

Since the generative model is jointly Gaussian, we can perform inference analytically. This gives rise to a posterior over $\boldsymbol{f}^*$ at $\boldsymbol{x}^*$ of the form

$$\boldsymbol{f}^* \sim \mathcal{N}(\boldsymbol{\mu}^*, \boldsymbol{\Sigma}^*), \tag{2.4}$$

where

$$\boldsymbol{\mu}^* = k(\boldsymbol{x}^*, \boldsymbol{x}) \left[k(\boldsymbol{x}, \boldsymbol{x}) + \sigma_\epsilon^2 \boldsymbol{I}\right]^{-1} \boldsymbol{y} \tag{2.5}$$

and

$$\boldsymbol{\Sigma}^* = k(\boldsymbol{x}^*, \boldsymbol{x}^*) - k(\boldsymbol{x}^*, \boldsymbol{x}) \left[k(\boldsymbol{x}, \boldsymbol{x}) + \sigma_\epsilon^2 \boldsymbol{I}\right]^{-1} k(\boldsymbol{x}, \boldsymbol{x}^*). \tag{2.6}$$

Here, $k(\boldsymbol{x}_1, \boldsymbol{x}_2) \in \mathbb{R}^{N \times M}$ indicates the matrix resulting from evaluating the kernel $k(\cdot,\cdot)$ for every pair of elements in $\boldsymbol{x}_1 \in \mathbb{R}^N$ and $\boldsymbol{x}_2 \in \mathbb{R}^M$.

Our model generally contains additional hyperparameters that we want to optimize. This could for example be the length scale, kernel scale, and noise variance in the commonly used squared exponential kernel (Rasmussen and Williams, 2006). It is common to optimize these parameters with respect to the *log marginal likelihood* of the model evaluated at the training data $\boldsymbol{x}$, $\log p(\boldsymbol{y}) = \log \int p(\boldsymbol{y}|\boldsymbol{f})p(\boldsymbol{f}|\boldsymbol{x})d\boldsymbol{f}$. This is analytically tractable since the observations are jointly Gaussian (c.f. Equation 2.2):

$$\log p(\boldsymbol{y}) = -\frac{1}{2}\boldsymbol{y}^T[\boldsymbol{K} + \sigma_\epsilon^2 \boldsymbol{I}]^{-1}\boldsymbol{y} - \frac{1}{2}\log|\boldsymbol{K} + \sigma_\epsilon^2 \boldsymbol{I}| - \frac{N}{2}\log 2\pi, \tag{2.7}$$

where N is the number of training data points.

A challenge with Gaussian process-based approaches is that computing the likelihood has a computational complexity of $\mathcal{O}(N^3)$, which makes optimization and inference expensive. For this reason, a rich literature has focused on developing computationally cheaper approximations to Gaussian processes (Bui et al., 2017; Quinonero-Candela and Rasmussen, 2005). A popular approach in this context is the variational sparse Gaussian process framework (Hensman et al., 2013, 2015a; Titsias, 2009), which forms the basis of some of the practical algorithms developed in Chapter 3 and will be covered in more depth there and in Section 2.3.

2.2 Latent variable models for high-dimensional data

2.2.1 Motivation and Bayesian formulation

In neuroscience, we often record high-dimensional neural activity $\boldsymbol{Y} \in \mathbb{R}^{N \times T}$, where N is the number of neurons and T the number of recorded timepoints. This consists of a series of noisy observations, which are correlated across both space and time (Yu et al., 2008). To relate these observations to neural computations or external variables, such as sensory stimuli or motor output, it is therefore common to perform some form of dimensionality reduction, which exploits these correlations to provide a more succinct and intuitive summary of the data (Cunningham and Byron, 2014; Humphries, 2020). Many of these approaches can be viewed as probabilistic generative models, which are fitted to the data under the assumption that the observations arise from a 'latent' process with states $\boldsymbol{X} \in \mathbb{R}^{D \times T}$ in a D-dimensional latent space with $D < N$ (Roweis and Ghahramani, 1999). We note that this is similar to the Gaussian process problem in Section 2.1, where the function f can be considered to describe the evolution of a 'latent variable'. In this section, we are interested in a low-dimensional representation of the data rather than such a 'denoised' but potentially still high-dimensional representation. The goal is then to (i) learn a functional mapping $f : \mathbb{R}^D \to \mathbb{R}^N$ between the low-dimensional latents and high-dimensional observations, $\boldsymbol{Y} \approx f(\boldsymbol{X})$, and (ii) infer the set of latent states $\boldsymbol{X}$.

Similar to the Gaussian process setting, these two challenges can be solved using Bayesian machine learning. In particular, we infer the latent variables using Bayes' rule:

$$p(\boldsymbol{X}|\boldsymbol{Y}) \propto p(\boldsymbol{Y}|\boldsymbol{X})p(\boldsymbol{X}). \tag{2.8}$$

To learn the functional mapping $f(\boldsymbol{x})$, one common approach is to use a parametric function $f_\theta(\boldsymbol{x})$ and learn the parameters θ by maximum likelihood estimation:

$$\theta = \arg\max_\theta p_\theta(\boldsymbol{Y}), \tag{2.9}$$

where

$$p_\theta(\boldsymbol{Y}) = \int_{\boldsymbol{X}} p(\boldsymbol{Y}|f_\theta(\boldsymbol{X}))p(\boldsymbol{X})d\boldsymbol{X}. \tag{2.10}$$

Here, $p(\boldsymbol{Y}|f_\theta(\boldsymbol{X}))$ incorporates the additional observation noise in our data, which is often assumed to be Gaussian or Poisson (Cunningham and Byron, 2014). An alternative to this parametric approach is to use a *non-parametric* model and perform inference over the functional mapping in addition to the latent variables (Titsias and Lawrence, 2010; Wu et al., 2017a):

$$p(f) \propto p(\boldsymbol{Y}|f)p(f). \tag{2.11}$$

We will see in Chapter 3 how this can be useful for several neuroscience applications.

2.2.2 Parametric models

A common family of parametric models is the set of *linear Gaussian models* (Roweis and Ghahramani, 1999). These assume that $f(\boldsymbol{X}) = \boldsymbol{C}\boldsymbol{X}$ is linear and that $p(\boldsymbol{Y}|f(\boldsymbol{X}))$ and $p(\boldsymbol{X})$ are both Gaussian. This family includes common methods such as PCA and factor analysis, which also assume that the prior over $\boldsymbol{X}$ factorizes across time,

$$p(\boldsymbol{X}) = \prod_t p(\boldsymbol{x}_t). \tag{2.12}$$

However, since neural activity and behaviour are often correlated in time, it can be useful to build this information into the prior – an example of how explicitly Bayesian approaches allow us to harness our prior knowledge to build more powerful inference algorithms. This has commonly been done using a dynamical-systems approach as is the case in e.g. the Kalman Filter (Kalman, 1960). More recent approaches, such as Gaussian process factor analysis (GPFA), have instead modelled the latent process as a Gaussian process (c.f. Section 2.1), such that $\boldsymbol{X} \sim \mathcal{GP}(0, k(\cdot,\cdot))$ (Rutten et al., 2020; Yu et al., 2008).

The linear Gaussian methods have the major advantage that they are analytically tractable – albeit with high computational complexity in the case of Gaussian process-based methods. However, another approach that has become increasingly popular with improvements in computing and approximate inference algorithms is to fit non-linear function approximators to the data, often inspired by variational autoencoders (Kingma and Welling, 2013; Rezende et al., 2014). Notable examples include 'Latent Factor Analysis with Dynamical Systems' (LFADS; Pandarinath et al., 2018) and the more recent 'iLQR-VAE' (Schimel et al., 2021). Both of these methods model $p(\boldsymbol{X})$ with a non-linear recurrent neural network, and they can use either linear or non-linear $f(\boldsymbol{X})$. These approaches are motivated by the observation that the brain itself can often be well-described as a non-linear dynamical system, suggesting that such a generative model could provide a good description of neural activity in an unsupervised setting as well. They generally also have many more free parameters compared to simple linear methods and tend to fit the data better with improved predictions of behavioural variables (Pandarinath et al., 2018; Schimel et al., 2021). However, such overparameterized non-linear methods come at the cost of increased model complexity, which both makes training of the model more expensive and brittle, while also reducing the interpretability of the model after training.

2.2.3 Non-parametric models

In contrast to the parametric methods, we have the set of non-parametric latent variable models. The most prominent examples of these assume that $f(\boldsymbol{X})$ is drawn from a Gaussian process (c.f. Section 2.1), such that

$$f \sim \mathcal{GP}(0, k(\cdot,\cdot)). \tag{2.13}$$

This gives rise to a so-called *Gaussian process latent variable model* (GPLVM), which was first developed by Lawrence (2005, 2004) and subsequently extended by Titsias and Lawrence (2010). In a neuroscience context, GPLVMs have been applied to both olfactory (Wu et al., 2018) and hippocampal (Wu et al., 2017a) data, where they provide a flexible yet interpretable model to describe non-linear relationships between latent variables and neural activity. For these purposes, the GPLVM framework was also extended to use a Poisson noise model and utilize a second GP prior over the latent variables themselves to build in the assumption of temporal continuity (Wu et al., 2017a). In the case of a GP prior over the latents, the model becomes an example of a (two-layer) 'deep Gaussian process' (Damianou and Lawrence, 2013), which is a class of models that uses a set of hierarchical Gaussian processes to map between two datasets (in this case 'time' and 'neural activity').

GPLVMs have commonly used the 'squared exponential' kernel to construct a prior over f, which builds in an assumption of smoothness in the mapping from latent variables to average neural firing rates. However, it is worth noting that this is not the only plausible choice of kernel, and we will explore alternatives in Chapter 3. Here, it is also worth mentioning that for a *linear* kernel, $k(\boldsymbol{x}_1, \boldsymbol{x}_2) = \boldsymbol{x}_1^T \boldsymbol{x}_2$, the GPLVM becomes equivalent to a linear Gaussian model, but where we compute the marginal likelihood and make predictions by *integrating out* the factor matrix $\boldsymbol{C}$ (with a unit Gaussian prior $p(C_{ij})$) instead of using a maximum-likelihood estimate. The simplest example of such an approach is Bayesian PCA (Bishop, 1999) – but even in this simple model, inference is intractable, and we have to resort to approximate inference procedures (c.f. Section 2.3). A corollary of this observation is that linear Gaussian models can be seen as a special case of the GPLVM, where the kernel is linear and the posterior is approximated with a delta function. This provides a unifying theoretical and computational framework for understanding and implementing many probabilistic latent variable models used in neuroscience – both linear and non-linear (Figure 2.1). Additionally, it makes explicit the different assumptions made by PCA-like methods compared to more canonical GPLVMs. In particular, the primary difference between these sets of approaches is that the canonical GPLVMs assume a *smooth* mapping from latents to neural activity, while PCA assumes a *linear* mapping from latents to neural activity. Which of these assumptions is most appropriate will of course depend on the data and application in question.

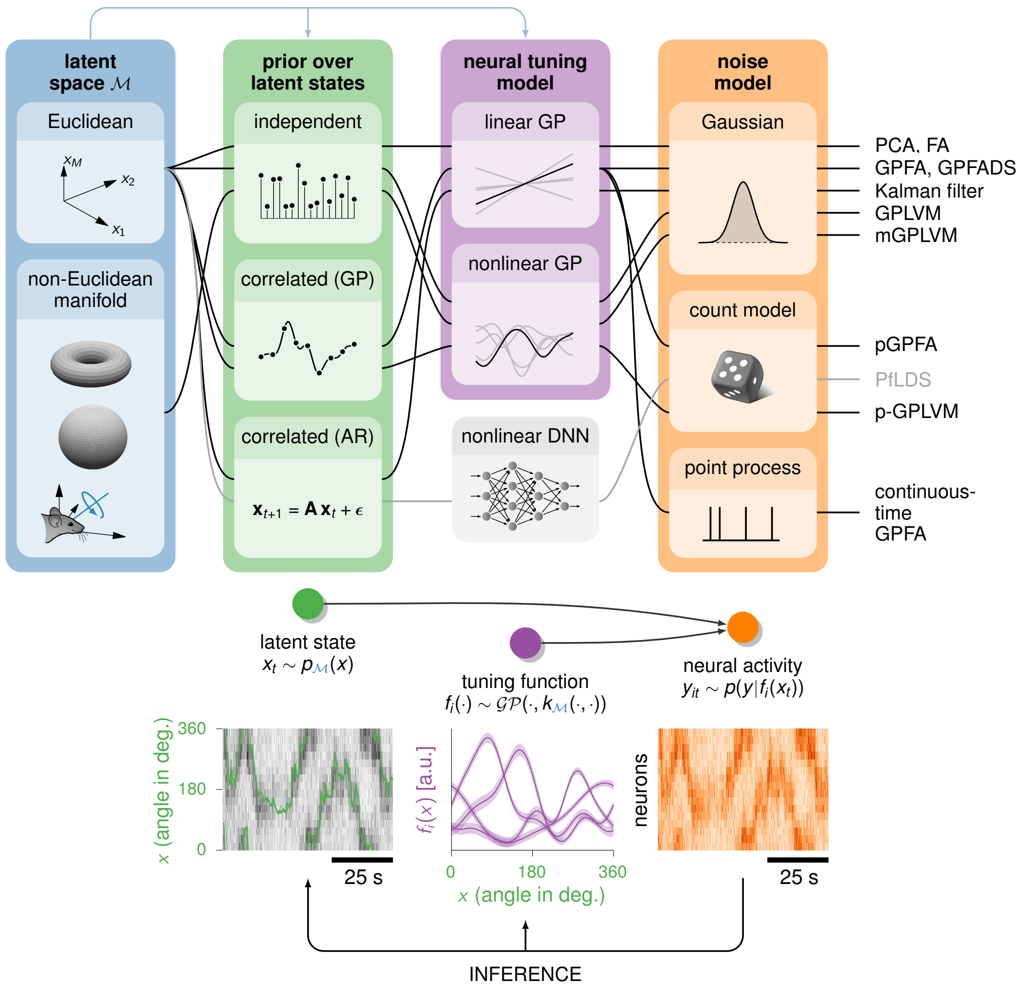

Figure 2.1 **Comparison of probabilistic latent variable models in neuroscience.** Many latent variable models in neuroscience are explicitly probabilistic and can be viewed as some combination of a set of common 'building blocks'. These building blocks include the nature of the latent space, which will be investigated further in Chapter 3, the prior over latents, the neural tuning model, and the noise model. Together, these model components specify a 'probabilistic generative model' (center), which defines a prior over models that the data could arise from. Notably, many of these models are contained within the family of 'Gaussian process latent variable models', which can incorporate both linear and non-linear tuning functions $f(\boldsymbol{x})$ as well as a variety of noise models and priors over latent states. Some of these extensions will be developed and considered further in Chapter 3. Given such a generative model and some dataset $\boldsymbol{Y}$, it is possible to perform *inference* over models to elucidate a distribution over latent variables $\boldsymbol{X}$ and functional mappings f conditioned on the data (bottom; example on circular latent space with non-linear f, c.f. Section 3.2).

2.3 Variational Bayesian inference for computational tractability

2.3.1 Inference can be formulated as optimization

As we have seen in the previous two sections, we generally want to perform *inference* on the basis of Bayes' theorem when applying methods from Bayesian machine learning to systems neuroscience (hence the name). Given a prior $p(\boldsymbol{X})$ over some set of unobserved, or latent, variables $\boldsymbol{X}$, and a likelihood $p(\boldsymbol{Y}|\boldsymbol{X})$, we thus want to compute the *posterior* $p(\boldsymbol{X}|\boldsymbol{Y})$ given our observations $\boldsymbol{Y}$:

$$p(\boldsymbol{X}|\boldsymbol{Y}) = \frac{p(\boldsymbol{Y}|\boldsymbol{X})p(\boldsymbol{X})}{p(\boldsymbol{Y})}. \tag{2.14}$$

Unfortunately this is often intractable, particularly if $p(\boldsymbol{Y}|\boldsymbol{X})$ hides a complicated generative model as is often the case in machine learning and computational neuroscience (e.g. Section 2.2). We are therefore frequently forced to approximate this posterior for practical applications. Two major approaches to this are Markov chain Monte Carlo methods (Hastings, 1970; Metropolis et al., 1953) and variational methods (Kingma and Welling, 2013; MacKay, 2003; Rezende et al., 2014; Wainwright and Jordan, 2008). Most of the work in this thesis relies on variational inference, and we therefore provide a brief introduction here.

When performing variational inference, we introduce a so-called *variational distribution* within a tractable family that will be used to approximate the posterior:

$$q_\phi(\boldsymbol{X}) \approx p(\boldsymbol{X}|\boldsymbol{Y}). \tag{2.15}$$

$q_\phi(\boldsymbol{X})$ contains a set of 'variational parameters' ϕ, which are tweaked to make $q(\boldsymbol{X})$ maximally similar to the true posterior ('ϕ' is suppressed in the following for notational simplicity). The degree of (dis)similarity to the posterior can be measured by the Kullback-Leibler (KL) divergence

$$\begin{aligned} KL[q(\boldsymbol{X})||p(\boldsymbol{X}|\boldsymbol{Y})] &= \int_X q(\boldsymbol{X}) \log \frac{q(\boldsymbol{X})}{p(\boldsymbol{X}|\boldsymbol{Y})} d\boldsymbol{X} \\ &= \mathbb{E}_q[\log q(\boldsymbol{X})] - \mathbb{E}_q[\log p(\boldsymbol{X}|\boldsymbol{Y})] \\ &= \mathbb{E}_q[\log q(\boldsymbol{X})] - \mathbb{E}_q[\log p(\boldsymbol{X},\boldsymbol{Y})] + \mathbb{E}_q[\log p(\boldsymbol{Y})], \end{aligned} \tag{2.16}$$

where $\mathbb{E}_q(\cdot)$ indicates expectations with respect to the variational distribution q.

Since $p(\boldsymbol{Y})$ does not depend on $\boldsymbol{X}$,

$$\mathbb{E}_q[\log p(\boldsymbol{Y})] = \log p(\boldsymbol{Y})\mathbb{E}_q[1] = \log p(\boldsymbol{Y}) = \text{Constant}. \tag{2.17}$$

We thus isolate this term and write

$$\log p(\boldsymbol{Y}) = KL[q(\boldsymbol{X})||p(\boldsymbol{X}|\boldsymbol{Y})] + \mathbb{E}_q[\log p(\boldsymbol{X},\boldsymbol{Y})] - \mathbb{E}_q[\log q(\boldsymbol{X})] \tag{2.18}$$

$$= KL[q(\boldsymbol{X})||p(\boldsymbol{X}|\boldsymbol{Y})] + ELBO, \tag{2.19}$$

where the *evidence lower bound* (ELBO) is defined as

$$\begin{aligned} ELBO &= \mathbb{E}_q[\log p(\boldsymbol{X}, \boldsymbol{Y})] - \mathbb{E}_q[\log q] && (2.20) \\ &= \log p(\boldsymbol{Y}) - KL[q || p(\boldsymbol{X}|\boldsymbol{Y})]. && (2.21) \end{aligned}$$

Since $\log p(\boldsymbol{Y})$ is constant with respect to ϕ, maximizing the ELBO is equivalent to minimizing the KL divergence between $q(\boldsymbol{X})$ and the true posterior $p(\boldsymbol{X}|\boldsymbol{Y})$. This maximization thus provides the best approximation to the posterior given our parameterization of q, as measured by the KL divergence. As a result, we have turned our *inference* problem into an *optimization* problem.

From Equation 2.21, we also see that the ELBO serves as a lower bound on the log marginal likelihood $\log p(\boldsymbol{Y})$, since $KL[q||p(\boldsymbol{X}|\boldsymbol{Y})] \geq 0$ (hence the name 'ELBO'). This is important since the likelihood $p(\boldsymbol{Y}|\boldsymbol{X})$ often contains additional parameters θ, which in a maximum likelihood setting we would like to optimize with respect to the marginal likelihood (c.f. Section 2.1 and Section 2.2). However, the marginal likelihood is generally intractable, and it is therefore common practice to optimize model parameters with respect to the ELBO as a proxy.

2.3.2 Optimization-based inference can be done by gradient descent

We now have a conceptual framework for training our models (optimizing θ) and performing inference (optimizing ϕ), and in this section we provide further details on how this is done in practice. We start by re-writing the ELBO slightly to arrive at

$$\begin{aligned} ELBO &= \mathbb{E}_q[\log p(\boldsymbol{Y}|\boldsymbol{X})] + \mathbb{E}_q[\log p(\boldsymbol{X})] - \mathbb{E}_q[\log q(\boldsymbol{X})] && (2.22) \\ &= \mathbb{E}_q[\log p(\boldsymbol{Y}|\boldsymbol{X})] - KL[q(\boldsymbol{X})||p(\boldsymbol{X})]. && (2.23) \end{aligned}$$

In many cases, $q(\boldsymbol{X})$ and $p(\boldsymbol{X})$ are chosen to be Gaussian (or otherwise tractable), such that the KL term can be computed analytically. This leaves the likelihood term, which is commonly approximated using Monte Carlo samples,

$$\mathbb{E}_q[\log p(\boldsymbol{Y}|\boldsymbol{X})] \approx \frac{1}{N} \sum_{\boldsymbol{X} \sim q(\boldsymbol{X})} \log p(\boldsymbol{Y}|\boldsymbol{X}), \tag{2.24}$$

where N indicates the number of Monte Carlo samples. This approach only requires us to evaluate the likelihood for individual samples rather than its expectation with respect to q.

When optimizing the parameters of the model, it is also necessary to differentiate through the computation of the ELBO, which is usually done using standard automatic differentiation software. In order to compute the ELBO in a differentiable manner, it is common to use the so-called 'reparameterization trick' (Kingma and Welling, 2013; Rezende et al., 2014). In order to sample from a parameterized distribution

$$\boldsymbol{X} \sim q_\phi(\boldsymbol{X}), \tag{2.25}$$

we rewrite the sampling process as a parameterized function of samples from a fixed reference distribution:

$$\boldsymbol{X} = g_\phi(\tilde{\boldsymbol{X}}) \tag{2.26}$$

$$\tilde{\boldsymbol{X}} \sim \tilde{q}(\tilde{\boldsymbol{X}}). \tag{2.27}$$

Here, $\tilde{\boldsymbol{X}}$ indicates samples from the reference distribution $\tilde{q}(\tilde{\boldsymbol{X}})$. The key idea is to move the parameters ϕ from the sampling process to the evaluation of a differentiable function, turning the gradient of our expectation into the expectation of a gradient over a fixed distribution $\tilde{q}$. This allows gradients to be approximated by Monte Carlo sampling, and we can use these to perform stochastic gradient descent on our objective.

2.4 Continual learning as Bayesian inference

2.4.1 Problem setting and notation

A notable difference between biological intelligence and modern machine learning systems is the ability of animals to learn continually without forgetting their past experiences and skills. This is in stark contrast to many machine learning systems, which often exhibit a phenomenon known as 'catastrophic forgetting' of previous abilities when learning a new task (Kirkpatrick et al., 2017). A natural question thus arises of *how* biological networks overcome this challenge, and whether we can take inspiration from biological systems to develop better ML systems. Much work has been done in this direction in recent years, with a wealth of new methods designed to overcome the challenge of catastrophic forgetting (Duncker et al., 2020; Huszár, 2017; Kirkpatrick et al., 2017; Loo et al., 2020; van de Ven et al., 2020; van de Ven and Tolias, 2019; Zeng et al., 2019). These approaches come in many different forms, each with a different algorithmic and mechanistic solution. A separate question is therefore whether these machine learning models can help us better understand how memories are stored and retained in *biological* systems by making explicit how network dynamics depend on such algorithmic choices. Many of these approaches to the continual learning problem can be formulated as a process of Bayesian inference, and we will provide an overview of the problem setting and existing methods in this section before discussing further extensions and comparisons to experimental data in Chapter 4.

We use $\boldsymbol{X} \otimes \boldsymbol{Y}$ to represent the Kronecker product between matrices $\boldsymbol{X} \in \mathbb{R}^{n\times n}$ and $\boldsymbol{Y} \in \mathbb{R}^{m\times m}$, such that $(\boldsymbol{X} \otimes \boldsymbol{Y})_{mi+k,mj+l} = \boldsymbol{X}_{ij}\boldsymbol{Y}_{kl}$. $\mathcal{D}_k$ refers to a 'dataset' corresponding to task k, which generally consists of a set of input-output pairs $\{\boldsymbol{x}_k^{(i)}, \boldsymbol{y}_k^{(i)}\}$ such that $\ell_k(\theta) := \log p(\mathcal{D}_k|\theta) = \sum_i \log p_\theta(\boldsymbol{y}_k^{(i)}|\boldsymbol{x}_k^{(i)})$ is the task-related performance on task k for a model with parameters θ. Finally, we use $\hat{\mathcal{D}}_k$ to refer to a dataset generated by inputs from the k^{th} task, where $\{\hat{\boldsymbol{y}}_k^{(i)} \sim p_\theta(\boldsymbol{y}|\boldsymbol{x}_k^{(i)})\}$ are drawn from the model distribution $\mathcal{M}$. When approaching a continual learning problem in machine learning, we train a model on a set of K tasks $\{\mathcal{D}_1, \ldots, \mathcal{D}_K\}$ that arrive sequentially, where the data distribution $\mathcal{D}_k$ for task k in general differs from $\mathcal{D}_{\neq k}$. The aim is to learn a probabilistic model $p(\mathcal{D}|\theta)$ that performs well on all tasks. The challenge in the continual learning setting stems from the sequential nature of learning, and in particular from the common assumption that the learner does not have access to "past" tasks (i.e., $\mathcal{D}_j$ for $j < k$) when learning task k.

2.4.2 Bayesian continual learning

The continual learning problem is naturally formalized in a Bayesian framework, whereby the posterior after $k-1$ tasks is used as a prior for task k. More specifically, we choose a prior $p(\theta)$ on the model parameters and compute the posterior after observing k tasks according to Bayes' rule:

$$\begin{aligned} p(\theta|\mathcal{D}_{1:k}) &\propto p(\theta) \prod_{k'=1}^{k} p(\mathcal{D}_{k'}|\theta) \\ &\propto p(\theta|\mathcal{D}_{1:k-1}) p(\mathcal{D}_k|\theta), \end{aligned} \tag{2.28}$$

where $\mathcal{D}_{1:k}$ is a concatenation of the first k tasks $(\mathcal{D}_1, \ldots, \mathcal{D}_k)$. In theory, it is thus possible to compute the exact posterior $p(\theta|\mathcal{D}_{1:k})$ after k tasks, while only observing $\mathcal{D}_k$, by using the posterior $p(\theta|\mathcal{D}_{1:k-1})$ after $k-1$ tasks as a prior. However, as is often the case in Bayesian inference, the difficulty here is that the posterior is typically intractable. To address this challenge, it is common to perform approximate online Bayesian inference. That is, the posterior $p(\theta|\mathcal{D}_{1:k-1})$ is approximated by a parametric distribution with parameters ϕ_{k-1}. The approximate posterior $q(\theta; \phi_{k-1})$ is then used as a prior for task k.

Online Laplace approximation A common approach is to use the Laplace approximation (MacKay, 2003), whereby the posterior $p(\theta|\mathcal{D}_{1:k-1})$ is approximated as a multivariate Gaussian q using local first and second derivatives of the log posterior (Huszár, 2017; Kirkpatrick et al., 2017; Ritter et al., 2018). This involves (i) finding a mode $\boldsymbol{\mu}_k$ of the posterior during task k, and (ii) performing a second-order Taylor expansion of the log posterior at $\boldsymbol{\mu}_k$ to construct an approximate Gaussian posterior $q(\theta; \phi_k) = \mathcal{N}(\theta; \boldsymbol{\mu}_k, \boldsymbol{\Lambda}_k^{-1})$, where $\boldsymbol{\Lambda}_k$ is the precision matrix and $\phi_k = (\boldsymbol{\mu}_k, \boldsymbol{\Lambda}_k)$. In this case, gradient-based optimization is used to find the posterior mode on task k (c.f. Equation 2.28):

$$\begin{aligned}\boldsymbol{\mu}_k &= \arg\max_\theta \ \log p(\theta|\mathcal{D}_k, \phi_{k-1}) && (2.29)\\ &= \arg\max_\theta \ \log p(\mathcal{D}_k|\theta) + \log q(\theta; \phi_{k-1}) && (2.30)\\ &= \arg\max_\theta \ \underbrace{\ell_k(\boldsymbol{\theta}) - \frac{1}{2}(\theta - \boldsymbol{\mu}_{k-1})^\top \boldsymbol{\Lambda}_{k-1}(\theta - \boldsymbol{\mu}_{k-1})}_{:= \mathcal{L}_k(\boldsymbol{\theta})} && (2.31)\end{aligned}$$

The precision matrix $\boldsymbol{\Lambda}_k$ is given by the Hessian of the negative log posterior at $\boldsymbol{\mu}_k$:

$$\boldsymbol{\Lambda}_k = -\nabla^2_\theta \log p(\theta|\mathcal{D}_k, \phi_{k-1})\Big|_{\theta=\boldsymbol{\mu}_k} = H(\mathcal{D}_k, \boldsymbol{\mu}_k) + \boldsymbol{\Lambda}_{k-1}, \quad (2.32)$$

where $H(\mathcal{D}_k, \boldsymbol{\mu}_k) = -\nabla^2_\theta \log p(\mathcal{D}_k|\theta)|_{\theta=\boldsymbol{\mu}_k}$ is the Hessian of the negative log likelihood of $\mathcal{D}_k$.

Continual learning with the online Laplace approximation thus involves two steps for each new task $\mathcal{D}_k$. First, given $\mathcal{D}_k$ and the previous posterior $q(\theta; \boldsymbol{\mu}_{k-1}, \boldsymbol{\Lambda}^{-1}_{k-1})$ (i.e. the new prior), $\boldsymbol{\mu}_k$ is found using gradient-based optimization (Equation 2.31). This step can be interpreted as optimizing the likelihood of $\mathcal{D}_k$ while penalizing changes in the parameters θ according to their importance for previous tasks, as determined by the prior precision matrix $\boldsymbol{\Lambda}_{k-1}$. Second, the new posterior precision matrix $\boldsymbol{\Lambda}_k$ is computed according to Equation 2.32.

Approximating the Hessian In practice, computing $\boldsymbol{\Lambda}_k$ presents two major difficulties. First, because $q(\theta; \phi_k)$ is a Gaussian distribution, $\boldsymbol{\Lambda}_k$ has to be positive semi-definite (PSD), which is not guaranteed for the Hessian $H(\mathcal{D}_k, \boldsymbol{\mu}_k)$. Second, if the number of model parameters n_θ is large, it may be prohibitive to compute a full $(n_\theta \times n_\theta)$ matrix. To address the first issue, it is common to approximate the Hessian with the Fisher information matrix (FIM; Huszár, 2017; Martens, 2014; Ritter et al., 2018):

$$\boldsymbol{F}_k = \mathbb{E}_{p(\hat{\mathcal{D}}_k|\theta)}\left[\nabla_\theta \log p(\hat{\mathcal{D}}_k|\theta) \nabla_\theta \log p(\hat{\mathcal{D}}_k|\theta)^\top\right]\Big|_{\theta=\boldsymbol{\mu}_k} \approx H(\mathcal{D}_k, \boldsymbol{\mu}_k) \quad (2.33)$$

The FIM is PSD, which ensures that $\boldsymbol{\Lambda}_k = \sum_{k'=1}^k \boldsymbol{F}_{k'}$ is also PSD. Computing $\boldsymbol{F}_k$ may still be impractical if there are many model parameters, and it is therefore common to further approximate the FIM using structured approximations with fewer parameters. In particular, a diagonal approximation to $\boldsymbol{F}_k$ recovers Elastic Weight Consolidation (EWC; Kirkpatrick et al., 2017), while a Kronecker-factored approximation (Martens and Grosse, 2015) recovers the method proposed by Ritter et al. (2018). We denote this method 'KFAC' and use it in Section 4.1.3 as a comparison for our own Kronecker-factored method.

2.5 Reinforcement learning as Bayesian inference

2.5.1 The reinforcement learning problem and policy gradients

Reinforcement learning is often used as a theory of how animals learn from experience (Daw et al., 2011; Geerts et al., 2020; Niv, 2009). Additionally, a wealth of neuroscience research has suggested that neural circuits directly represent quantities, such as reward prediction errors, that are central to theories of reinforcement learning in the machine learning literature (Dabney et al., 2020; Schultz et al., 1997; Wang et al., 2018). It has also recently been suggested that the process of reinforcement learning can itself be viewed as a process of Bayesian inference in the space of policies (Levine, 2018; Solway and Botvinick, 2012). This framework naturally allows for the integration of priors reminiscent of the biases commonly seen in natural behaviour (Lai and Gershman, 2021), and it could provide a fruitful avenue for improving our understanding of learning and generalization in biological organisms. In this section, we provide a brief overview of the reinforcement learning problem setting and canonical policy gradient algorithms for training RL agents. We then provide an overview of the theory of 'learning as inference', illustrating how the reinforcement learning problem is equivalent to approximate inference in the space of policies. These ideas will be used in Chapter 5, where we develop a new reinforcement learning model of planning and decision making.

Problem setting

Here we will provide a short introduction to the reinforcement learning problem in a discrete state and action space with a finite time horizon and no discounting. For a more general treatment, we refer to Sutton and Barto (2018). In the discrete problem setting, the environment consists of states $s \in \mathcal{S}$, and the agent can take actions $a \in \mathcal{A}$. The environment is characterized by transition and reward probabilities $p(s_{t+1}, r_t | s_t, a_t)$, where r_t is the reward at time t. We will further make the *Markov assumption* that the next state only depends on the current state and action, $p(s_{t+1}, r_t | s_t, a_t, s_{t-1}, a_{t-1}, ..., s_0, a_0) = p(s_{t+1}, r_t | s_t, a_t)$. We can then define a *trajectory* $\tau = \{s_t, a_t, r_t\}_{t=0}^{T}$, where

$$p(\tau) = p(s_0) \prod_{t=0}^{T} p(s_{t+1}, r_t | s_t, a_t) p(a_t | s_t). \tag{2.34}$$

$p(a_t|s_t)$ is the probability of taking action a_t in state s_t, which is usually controlled by the agent and denoted a *policy* $\pi(a_t|s_t)$. The objective of the agent is to maximize the expected total reward

$$J = \mathbb{E}_{\tau}\left[R_{\tau}\right] = \mathbb{E}_{\tau}\left[\sum_{t=0}^{T} r_t | \tau\right], \tag{2.35}$$

where $R_\tau := \sum_{t=0}^{T} r_t | \tau$. This is achieved by optimizing the policy $\pi_\theta(a|s)$, with parameters θ, to maximize the objective

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}\left[\sum_{t=0}^{T} r_t | \tau\right]. \tag{2.36}$$

Policy gradients

A conceptually simple way to maximize the reward in Equation 2.36 would be to use gradient descent with gradients given by

$$\begin{aligned} \nabla_\theta J(\theta) &= \nabla_\theta \mathbb{E}_{\tau \sim \pi_\theta}\left[R_\tau\right] && (2.37)\\ &= \sum_\tau R_\tau \nabla_\theta p_\theta(\tau). && (2.38) \end{aligned}$$

However, this requires differentiating through the environment, which we may not be able to do. Instead, we use the 'log-derivative trick', which takes advantage of the linearity of the expectation and the identity $\nabla_\theta \log f(\theta) = f(\theta)^{-1} \nabla_\theta f(\theta)$ to write

$$\begin{aligned} \nabla_\theta J(\theta) &= \sum_\tau R_\tau \nabla_\theta p_\theta(\tau) && (2.39)\\ &= \sum_\tau R_\tau p_\theta(\tau) \nabla_\theta \log p_\theta(\tau) && (2.40)\\ &= \mathbb{E}_{\tau \sim \pi_\theta}\left[R_\tau \nabla_\theta \log p_\theta(\tau)\right], && (2.41) \end{aligned}$$

Since the environment does not depend on θ, we can simplify the calculation of $\nabla_\theta \log p_\theta(\tau)$:

$$\begin{aligned} \nabla_\theta \log p_\theta(\tau) &= \nabla_\theta \left[\log p(s_0) + \sum_{t=0}^{T} \log p(s_{t+1}|s_t, a_t) + \log \pi_\theta(a_t|s_t)\right] && (2.42)\\ &= \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t). && (2.43) \end{aligned}$$

Inserting this in the expression for $\nabla_\theta J(\theta)$ and taking a Monte Carlo estimate of the expectation gives rise to the REINFORCE algorithm (Williams, 1992):

$$\begin{aligned} \nabla_\theta J(\theta) &= \mathbb{E}_{\tau \sim \pi_\theta}\left[R_\tau \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\right] && (2.44)\\ &\approx \frac{1}{N} \sum_{\tau \sim \pi_\theta} \left(\sum_{t=0}^{T} r_t\right)\left(\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\right). && (2.45) \end{aligned}$$

While the REINFORCE algorithm is unbiased, it also has high variance, which can make learning slow and unstable. It is therefore common to introduce modifications, which can help reduce the variance. The first of these comes from noting that an action taken at time t cannot affect the reward received at times $t' < t$. This allows us to define $R_t := \sum_{t'=t}^{T} r_{t'}$ and rewrite our REINFORCE update as

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{\tau \sim \pi_\theta} \sum_{t=0}^{T} R_t \nabla_\theta \log \pi_\theta(a_t|s_t). \tag{2.46}$$

It is also straightforward to show that for a baseline $B(s_t)$ that does not depend on a_t,

$$\mathbb{E}_{\tau \sim \pi_\theta} [B(s_t) \nabla_\theta \log \pi_\theta(a_t|s_t)] = \int_{s_t} B(s_t) p(s_t) \left[\nabla_\theta \int_{a_t} \pi_\theta(a_t|s_t) da_t \right] ds_t \tag{2.47}$$

$$= \int_{s_t} B(s_t) p(s_t) [\nabla_\theta 1] \, ds_t = 0. \tag{2.48}$$

A corollary of this result is that we can subtract such a baseline from our empirical reward and still have an unbiased estimator while reducing its variance. A common choice here is the expected future reward

$$V(s_t) = \mathbb{E}\left[\sum_{t'=t}^{T} r_{t'} | s_t \right] \tag{2.49}$$

This gives rise to the so-called 'actor-critic' algorithm

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{\tau \sim \pi_\theta} \sum_{t=0}^{T} (R_t - V(s_t)) \nabla_\theta \log \pi_\theta(a_t|s_t). \tag{2.50}$$

This algorithm and its probabilistic extension (Section 2.5.2) forms the basis of the work in Chapter 5. Additional variance-reduction methods include bootstrapping, but that is not used in the present work and we refer to Sutton and Barto (2018) for more details on this and other extensions.

2.5.2 Reformulating RL as Bayesian inference

While reinforcement learning is most commonly phrased as an optimization problem as in Section 2.5.1, we can also formulate it as a process of Bayesian inference in a probabilistic generative model following Levine (2018). This is in some sense the 'dual' of Section 2.3, where we reformulated inference as an optimization problem. As before, we consider an MDP of the form

$$p(\tau) = p(\{s_t, r_t, a_t\}_{t=0}^{T}) \tag{2.51}$$

$$= p(s_0) \prod_{t}^{T} p(s_{t+1}, r_t | s_t, a_t) \pi(a_t|s_t), \tag{2.52}$$

where $\pi(a|s)$ is a policy that we want to optimize. To perform inference in this setting, we now introduce an additional class of stochastic binary 'optimality' variables $O_t \in \{0,1\}$ (Levine, 2018; Solway and Botvinick, 2012). These are Bernoulli distributed with

$$p(O_t = 1|s_t, a_t) \propto \exp\left[r(s_t, a_t)\right], \tag{2.53}$$

where $r(s_t, a_t)$ is the reward associated with taking action a_t in state s_t. This allows us to transition from a scalar reward representation to a probabilistic formulation, where the probabilities now reflect reward *magnitude*.

In the following, we define $p(O_t) := p(O_t = 1)$ for notational simplicity. To proceed, we *condition* on $p(O_t = 1)$ and compute the corresponding posterior distribution over trajectories τ:

$$p(\tau|O_{1:T}) \propto p(O_{1:T}|\tau)p(\tau) \propto \exp\left[\sum_t r(s_t, a_t)\right] p(\tau). \tag{2.54}$$

This inference process yields the distribution over trajectories given that all of our optimality variables are '1' (as opposed to 0). This may seem somewhat arbitrary at first sight, but some intuition can be had from noting that the posterior probability of a trajectory increases exponentially with its associated total reward. It therefore allows us to move from a scalar to a probabilistic formulation while retaining the objective of maximizing reward, which is necessary for incorporating prior knowledge into our policy (see Solway and Botvinick, 2012 and Levine, 2018 for further motivation).

Approximate inference

The inference problem in Equation 2.54 is in general intractable, and to proceed, we will introduce a variational distribution $q_\phi(\tau)$ as discussed in Section 2.3. We choose q to factorize in the same way as the true $p(\tau)$, and to use the ground truth transition function:

$$q_\phi(\tau) = q_\phi(s_0) \prod_t^T q_\phi(s_{t+1}, r_t|s_t, a_t) q_\phi(a_t|s_t) \tag{2.55}$$

$$= p(s_0) \prod_t^T p(s_{t+1}, r_t|s_t, a_t) q_\phi(a_t|s_t). \tag{2.56}$$

We will optimize ϕ to minimize the KL divergence between $q_\phi(\tau)$ and $p(\tau|O_{1:T})$:

$$KL[q(\tau)||p(\tau|O_{1:T})] = \log p(O_{1:T}) - \mathcal{L}, \tag{2.57}$$

where

$$\mathcal{L} := \mathbb{E}_q\left[\log p(O_{1:T}|\tau)\right] - KL[q(\tau)||p(\tau)] \tag{2.58}$$

is the ELBO as defined in Section 2.3.

We now insert our expression for $\log p(O_t|a_t, s_t) = r_t$ to arrive at the 'learning as inference' objective

$$\mathcal{L} = \mathbb{E}_q\left[\sum_t r_t(a_t, s_t)\right] - KL[q(\tau)||p(\tau)]. \tag{2.59}$$

We see that this recovers our 'standard' RL objective of maximizing expected reward, but with an additional regularizer towards the prior $p(\tau)$. It is worth noting in passing that this KL-regularized RL objective is very similar to a range of recent models of biological reinforcement learning (de Saa and Renart, 2022; Lai and Gershman, 2021; Piray and Daw, 2021).

We can also introduce a temperature parameter β in the distribution over O,

$$p_\beta(O_t|s_t, a_t) \propto \exp\left[\beta r_t(s_t, a_t)\right]. \tag{2.60}$$

In this case, we instead need to maximize

$$\mathcal{L}_\beta = \beta \mathbb{E}_q\left[\log p_{\beta=1}(O_{1:T}|\tau)\right] - KL[q(\tau)||p(\tau)] \tag{2.61}$$

$$\propto \mathbb{E}_q\left[\log p_{\beta=1}(O_{1:T}|\tau)\right] - \beta^{-1} KL[q(\tau)||p(\tau)]. \tag{2.62}$$

β thus scales the relative importance of reward (through the likelihood) and the prior over actions. In the following, we will let $\beta = 1$ for notational simplicity and simply note that (i) this provides a way of scaling the prior while remaining faithful to the Bayesian formulation, and (ii) as $\beta \rightarrow \infty$, we recover canonical non-Bayesian RL since the KL regularization goes to zero – where our original policy π_θ has been replaced by the variational distribution $q_\phi(a_t|s_t)$ (Section 2.3). Finally we note that the transition terms cancel in the KL:

$$KL[q(\tau)||p(\tau)] = \mathbb{E}_q\left[\log q(\tau) - \log p(\tau)\right] \tag{2.63}$$

$$= \mathbb{E}_q[\log p(s_0) + \sum_t^T (\log p(r_t|s_t, a_t) + \log q_\phi(a_t|s_t)) \tag{2.64}$$

$$- \log p(s_0) - \sum_t^T (\log p(r_t|s_t, a_t) + \log p(a_t|s_t))]$$

$$= \mathbb{E}_q\left[\sum_t (\log q_\phi(a_t|s_t) - \log p(a_t|s_t))\right]. \tag{2.65}$$

This gives rise to our final 'learning as inference' objective:

$$\mathcal{L} = \mathbb{E}_{q(\tau)}\left[\sum_t r_t(a_t, s_t)\right] - \mathbb{E}_{q(\tau)}\left[\sum_t (\log q_\phi(a_t|s_t) - \log p(a_t|s_t))\right]. \tag{2.66}$$

To estimate the gradient of this objective, we use the 'log derivative trick' also used for the policy gradient algorithm in Section 2.5.1:

$$\nabla_\phi \mathbb{E}_{q_\phi}\left[R_\tau\right] = \mathbb{E}_{q_\phi}\left[\sum_t R_t \nabla_\phi \log q_\phi(a_t|s_t)\right]. \tag{2.67}$$

We can then estimate the expectation over q_ϕ by sampling rollouts of the agent given the policy and using Monte Carlo estimates of $R_t \nabla_\phi \log q_\phi(a_t|s_t)$. Since the KL term also involves an expectation over q, we have to use the same log derivative trick here:

$$\nabla_\phi \mathbb{E}_{q_\phi}\left[\sum_t \left[\log q_\phi(a_t|s_t) - \log p(a_t|s_t)\right]\right] = \mathbb{E}_{q_\phi}\left[\sum_t \nabla_\phi \log q_\phi(a_t|s_t) \sum_{t'=t}^{T} \left[\log q_\phi(a_{t'}|s_{t'}) - \log p(a_{t'}|s_{t'})\right]\right]. \tag{2.68}$$

It is worth considering the form of the objective under a uniform prior over actions $p(a_t|s_t) = 1/|\mathcal{A}|$. In this case, we can ignore the log prior term during optimization, and we arrive at the objective

$$\mathcal{L} = \mathbb{E}_{q(\tau)}\left[\sum_t r_t(a_t, s_t)\right] + \mathbb{E}_{q(\{s\})}\left[\sum_t H(q(a_t|s_t))\right], \tag{2.69}$$

where $H(q(a_t|s_t)) = -\sum_a [q(a_t|s_t)\log q(a_t|s_t)]$ is the *entropy* of the policy at time t and $q(\{s\})$ is the distribution of state sequences induced by the policy (marginalized over actions). Learning as inference thus naturally gives rise to 'maximum entropy' reinforcement learning (MERL). However, the true utility of this learning as inference approach arises when we move beyond such uniform priors. This has found applications in e.g. multi-task learning settings, where a prior over actions can arise from marginalizing the policy over tasks (Teh et al., 2017). Additionally, we will see in Chapter 5 how such a prior can be used to capture biases in human decision making when encountering a new task.

Chapter 3

Latent variable models

In this chapter, we develop two new 'weak' Bayesian machine learning models for neuroscience, which can be used to extract low-dimensional latent features from high-dimensional neural recordings. Much previous work has been done in this area, with a range of linear and non-linear latent variable models being used in the neuroscience community (Cunningham and Byron, 2014). The focus of the methods in this chapter is to infer the dimensionality (Section 3.1) and topology (Section 3.2) of the neural representations. This is important because neural activity is thought to be low-dimensional (Gallego et al., 2017) and often represents non-Euclidean features of the environment (Chaudhuri et al., 2019; Gardner et al., 2022). Capturing these properties in our statistical models is therefore essential for our understanding of how neural computations drive natural behaviours. A PyTorch implementation of the methods developed in this chapter can be found at https://github.com/tachukao/mgplvm-pytorch.

3.1 Bayesian Gaussian process factor analysis

This section has been peer reviewed and published as Jensen et al. (2021).

3.1.1 Introduction

The adult human brain contains upwards of 100 billion neurons (Azevedo et al., 2009). Yet many of our day-to-day behaviors such as navigation, motor control, and decision making can be described in much lower dimensional spaces. Accordingly, recent studies across a range of cognitive and motor tasks have shown that neural population activity can often be accurately summarised by the dynamics of a "latent state" evolving in a low-dimensional space (Chaudhuri et al., 2019; Churchland et al., 2012; Ecker et al., 2014; Minxha et al., 2020; Pandarinath et al., 2018). Inferring and investigating these latent processes can therefore help us understand the underlying representations and computations implemented by the brain (Humphries, 2020). To this end, numerous latent variable models have been developed and used to analyze the activity of populations of simultaneously recorded neurons. These models range from simple linear projections such as PCA to sophisticated non-linear and temporally correlated models (Cunningham and Byron, 2014; Gao et al., 2016; Jensen et al., 2020; Pandarinath et al., 2018; Schimel et al., 2021).

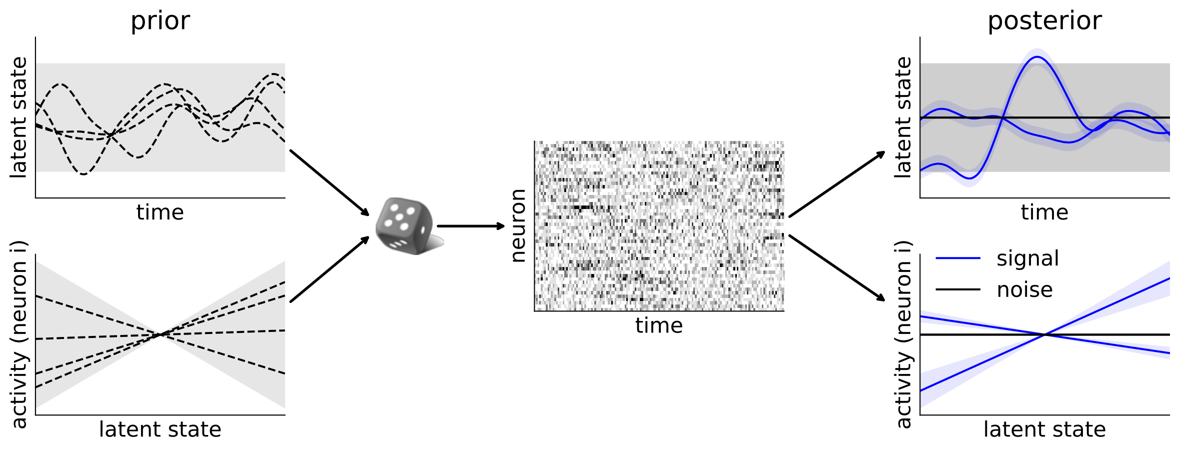

Figure 3.1 **Bayesian GPFA schematic.** Bayesian GPFA places a Gaussian process prior over the latent states in each dimension as a function of time $\boldsymbol{t}$ ($p(\boldsymbol{X}|\boldsymbol{t})$; top left) as well as a linear prior over neural activity as a function of each latent dimension ($p(\boldsymbol{F}|\boldsymbol{X})$; bottom left). Together with a stochastic noise process $p(\boldsymbol{Y}|\boldsymbol{F})$, which can be discrete for electrophysiological recordings, this forms a generative model that gives rise to observations $\boldsymbol{Y}$ (middle). From the data and priors, bGPFA infers posterior latent states for each latent dimension ($p(\boldsymbol{X}|\boldsymbol{Y})$; top right) as well as a posterior predictive observation model for each neuron ($p(\boldsymbol{Y}_{test}|\boldsymbol{X}_{test}, \boldsymbol{Y})$; bottom right). When combined with automatic relevance determination, the model learns to automatically discard superfluous latent dimensions by maximizing the log marginal likelihood of the data (right, black vs. blue).

A popular latent variable model for neural data analysis is Gaussian process factor analysis (GPFA), which has yielded insights into neural computations ranging from time tracking to movement preparation and execution (Afshar et al., 2011; Rutten et al., 2020; Sauerbrei et al., 2020; Sohn et al., 2019). However, fitting GPFA comes with a computational complexity of $\mathcal{O}(T^3)$ and a memory footprint of $\mathcal{O}(T^2)$ for T time bins. This prohibits the application of GPFA to time series longer than a few hundred time bins without artificially chunking such data into “pseudo-trials” and treating these as independent samples. Additionally, canonical GPFA assumes a Gaussian noise model while recent work has suggested that non-Gaussian models often perform better on neural data (Duncker and Sahani, 2018; Keeley et al., 2020a; Zhao and Park, 2017). Here, we address these challenges by formulating a scalable and doubly Bayesian version of GPFA (bGPFA; Figure 3.1) with a computational complexity of $\mathcal{O}(T \log T)$ and a memory cost of $\mathcal{O}(T)$. To do this, we introduce an efficiently parameterized variational inference strategy that ensures scalability to long recordings while also supporting non-Gaussian noise models. Additionally, the Bayesian formulation provides a framework for principled model selection based on approximate marginal likelihoods (Titsias and Lawrence, 2010). This allows us to perform automatic relevance determination and thus fit a single model without prior assumptions about the underlying dimensionality, which is instead inferred from the data itself (Bishop, 1999; Neal, 2012).

We validate our method on synthetic and biological data, where bGPFA exhibits superior performance to GPFA and Poisson GPFA with increased scalability and without requiring cross-validation to select the latent dimensionality. We then apply bGPFA to longitudinal, multi-area recordings from primary motor (M1) and sensory (S1) areas during a monkey self-paced reaching task spanning 30 minutes. bGPFA readily scales to such datasets, and the inferred latent trajectories improve decoding of kinematic variables compared to the raw data. This decoding improves further when taking into account the temporal offset between motor planning encoded by M1 and feedback encoded by S1. We also show that the latent trajectories for M1 converge to consistent regions of state space for a given reach direction at the onset of each individual reach. Importantly, the distance in latent space to this preparatory state from the state at target onset is predictive of reaction times across reaches, similar to previous results in a task that includes an explicit 'motor preparation epoch' where the subject is not allowed to move (Afshar et al., 2011). This illustrates the functional relevance of such preparatory activity and suggests that motor preparation takes place even when the task lacks well-defined trial structure and externally imposed delay periods, consistent with findings by Lara et al. (2018) and Zimnik and Churchland (2021). Finally, we analyze the task relevance of slow latent processes identified by bGPFA, which evolve on timescales of seconds; longer than the millisecond timescales that can be resolved by methods designed for trial-structured data. We find that some of these slow processes are also predictive of reaction time across reaches, and we hypothesize that they reflect task engagement, which varies over the course of several reaches.

3.1.2 Method

Generative model

Latent variable models for neural recordings typically model the neural activity $\boldsymbol{Y} \in \mathbb{R}^{N\times T}$ of N neurons at times $\boldsymbol{t} \in \mathbb{R}^T$ as arising from shared fluctuations in D latent variables $\boldsymbol{X} \in \mathbb{R}^{D\times T}$ (c.f. Section 2.2). Specifically, the probability of a given recording can be written as

$$p(\boldsymbol{Y}|\boldsymbol{t}) = \int p(\boldsymbol{Y}|\boldsymbol{F})\, p(\boldsymbol{F}|\boldsymbol{X})\, p(\boldsymbol{X}|\boldsymbol{t})\, d\boldsymbol{F}\, d\boldsymbol{X}, \tag{3.1}$$

where $\boldsymbol{F} \in \mathbb{R}^{N\times T}$ are intermediate, neuron-specific variables that can often be thought of as firing rates or a similar notion of noise-free activity. For example, GPFA (Yu et al., 2009) specifies

$$p(\boldsymbol{Y}|\boldsymbol{F}) = \prod_{n,t} \mathcal{N}(y_{nt}; f_{nt}, \sigma_n^2) \tag{3.2}$$

$$p(\boldsymbol{F}|\boldsymbol{X}) = \delta(\boldsymbol{F} - \boldsymbol{C}\boldsymbol{X}) \tag{3.3}$$

$$p(\boldsymbol{X}|\boldsymbol{t}) = \prod_{d} \mathcal{N}(\boldsymbol{x}_d; \boldsymbol{0}, \boldsymbol{K}_d) \qquad \text{with } \boldsymbol{K}_d = k_d(\boldsymbol{t}, \boldsymbol{t})) \tag{3.4}$$

That is, the prior over the d^{th} latent function $x_d(t)$ is a Gaussian process (Williams and Rasmussen, 1995; Section 2.1) with covariance function $k_d(\cdot,\cdot)$ (usually a radial basis function), and the observation model $p(\boldsymbol{Y}|\boldsymbol{X})$ is given by a parametric linear transformation with independent Gaussian noise.

In this work, we additionally introduce a prior distribution over the mixing matrix $\boldsymbol{C} \in \mathbb{R}^{N \times D}$ with scale parameters specific to each latent dimension. This allows us to *learn* an appropriate latent dimensionality for a given dataset using automatic relevance determination (ARD), similar to Bayesian PCA (Appendix A; Bishop, 1999), rather than relying on cross-validation or ad-hoc thresholds of variance explained. Unlike in standard GPFA, the log marginal likelihood (Equation 3.1) becomes intractable with this prior. We therefore develop a novel variational inference strategy (Wainwright and Jordan, 2008; Section 2.3), which also (i) provides a scalable implementation appropriate for long continuous neural recordings, and (ii) extends the model to general non-Gaussian likelihoods better suited for discrete spike counts.

In this new framework, which we call Bayesian GPFA (bGPFA), we use a Gaussian prior over $\boldsymbol{C}$ of the form $c_{nd} \sim \mathcal{N}(0, s_d^2)$, where s_d is a scale parameter associated with latent dimension d. Integrating $\boldsymbol{C}$ out in Equation 3.3 then yields the following observation model:

$$p(\boldsymbol{F}|\boldsymbol{X}) = \prod_{n} \mathcal{N}(\boldsymbol{f}_n; 0, \boldsymbol{X}^T \boldsymbol{S}^2 \boldsymbol{X}), \qquad \text{with } \boldsymbol{S} = \text{diag}(s_1, \ldots, s_D). \tag{3.5}$$

Moreover, we use a general noise model $p(\boldsymbol{Y}|\boldsymbol{F}) = \prod_{n,t} p(y_{nt}|f_{nt})$, where $p(y_{nt}|f_{nt})$ is any distribution for which we can evaluate its density or PMF in a differentiable manner.

Importantly, integrating over $\boldsymbol{C}$ means that the marginal likelihood in Equation 3.1 can *decrease* when adding superfluous dimensions that are not needed to explain the data. This is in contrast to canonical (GP)FA, where adding latent dimensions can only increase the marginal likelihood. This is countered by bGPFA having the capacity to learn that the scale factors (s_d) for any superfluous dimensions should be zero, in which case they do not contribute to the marginal likelihood or posterior predictive distribution. The posteriors over the corresponding latent dimensions collapse to the prior. This 'pruning' during training of dimensions that are not necessary to explain the data is referred to as 'automatic relevance determination' (ARD; Appendix A; Bishop, 1999). We can also fit bGPFA *without* ARD by tying all scale factors to a single shared value, $s_d = s \forall d$.

Variational inference and learning

To train the model and infer both $\boldsymbol{X}$ and $\boldsymbol{F}$ from the data $\boldsymbol{Y}$, we use a nested variational approach (Section 2.3). It is intractable to compute $\log p(\boldsymbol{Y}|\boldsymbol{t})$ (Equation 3.1) analytically for bGPFA, and we therefore introduce a lower bound on $\log p(\boldsymbol{Y}|\boldsymbol{t})$ at the outer level and another one on $\log p(\boldsymbol{Y}|\boldsymbol{X})$ at the inner level. These lower bounds are constructed from approximations to the posterior distributions over latents ($\boldsymbol{X}$) and noise-free activity ($\boldsymbol{F}$) respectively.

Distribution over latents At the outer level, we introduce a variational distribution $q(\boldsymbol{X})$ over latents and construct an evidence lower bound (ELBO; Wainwright and Jordan, 2008; Section 2.3) on the log marginal likelihood of Equation 3.1:

$$\log p(\boldsymbol{Y}|\boldsymbol{t}) \geq \mathcal{L} := \mathbb{E}_{q(\boldsymbol{X})}\left[\log p(\boldsymbol{Y}|\boldsymbol{X})\right] - \mathrm{KL}\left[q(\boldsymbol{X})||p(\boldsymbol{X}|\boldsymbol{t})\right]. \tag{3.6}$$

Conveniently, maximizing this lower bound is equivalent to minimizing $\mathrm{KL}\left[q(\boldsymbol{X})||p(\boldsymbol{X}|\boldsymbol{Y})\right]$ and thus also yields an approximation to the posterior over latents in the form of $q(\boldsymbol{X})$ (Section 2.3). We estimate the first term of the ELBO using Monte Carlo samples from $q(\boldsymbol{X})$ and compute the KL term analytically.

Here, we use a so-called whitened parameterization of $q(\boldsymbol{X})$ (Hensman et al., 2015b) that is both expressive and scalable to large datasets:

$$q(\boldsymbol{X}) = \prod_{d=1}^{D} \mathcal{N}(\boldsymbol{x}_d; \boldsymbol{\mu}_d, \boldsymbol{\Sigma}_d) \quad \text{with} \quad \boldsymbol{\mu}_d = \boldsymbol{K}_d^{\frac{1}{2}} \boldsymbol{\nu}_d \quad \text{and} \quad \boldsymbol{\Sigma}_d = \boldsymbol{K}_d^{\frac{1}{2}} \boldsymbol{\Lambda}_d \boldsymbol{\Lambda}_d^T \boldsymbol{K}_d^{\frac{1}{2}T}, \tag{3.7}$$

where $\boldsymbol{K}_d^{\frac{1}{2}}$ is any square root of the prior covariance matrix $\boldsymbol{K}_d$, and $\boldsymbol{\nu}_d \in \mathbb{R}^T$ is a vector of variational parameters to be optimized. $\boldsymbol{\Lambda}_d \in \mathbb{R}^{T \times T}$ is a positive semi-definite variational matrix whose structure is chosen carefully so that its squared Frobenius norm, log determinant, and matrix-vector products can all be computed efficiently, which facilitates the evaluation of Equations 3.8 and 3.9. This whitened parameterization has several advantages. First, it does not place probability mass where the prior itself does not. In addition to stabilizing learning (Murray and Adams, 2010), this also guarantees that the posterior is temporally smooth for a smooth prior. Second, the KL term in Equation 3.6 simplifies to

$$\mathrm{KL}[q(\boldsymbol{X})||p(\boldsymbol{X}|\boldsymbol{t})] = \frac{1}{2} \sum_d \left(||\boldsymbol{\Lambda}_d||_\mathrm{F}^2 - 2\log|\boldsymbol{\Lambda}_d| + ||\boldsymbol{\nu}_d||^2 - T \right). \tag{3.8}$$

Third, $q(\boldsymbol{X})$ can be sampled efficiently via a differentiable transform (i.e. the reparameterization trick) provided that fast differentiable $\boldsymbol{K}_d^{\frac{1}{2}} \boldsymbol{v}$ and $\boldsymbol{\Lambda}_d \boldsymbol{v}$ products are available for any vector $\boldsymbol{v}$:

$$\boldsymbol{x}_d^{(m)} = \boldsymbol{K}_d^{\frac{1}{2}} (\boldsymbol{\nu}_d + \boldsymbol{\Lambda}_d \boldsymbol{\eta}_d) \qquad \text{with} \qquad \boldsymbol{\eta}_d \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}), \tag{3.9}$$

where $\boldsymbol{x}_d^{(m)} \sim q(\boldsymbol{x}_d)$. This is important to form a Monte Carlo estimate of $\mathbb{E}_{q(\boldsymbol{X})}[\log p(\boldsymbol{Y}|\boldsymbol{X})]$.

To avoid the challenging computation of $\boldsymbol{K}_d^{\frac{1}{2}}\boldsymbol{v}$ for general $\boldsymbol{K}_d$ (Allen et al., 2000), we directly parameterize $\boldsymbol{K}_d^{\frac{1}{2}}$, the positive definite square root of $\boldsymbol{K}$, which implicitly defines the prior covariance function $k_d(\cdot,\cdot)$. In this work we use an RBF kernel for $\boldsymbol{K}_d$ and give the expression for $\boldsymbol{K}_d^{\frac{1}{2}}$ in Appendix A. Additionally, we use Toeplitz acceleration methods to compute $\boldsymbol{K}_d^{\frac{1}{2}}\boldsymbol{v}$ products in $\mathcal{O}(T\log T)$ time and with $\mathcal{O}(T)$ memory cost (Rutten et al., 2020; Wilson et al., 2015).

We implement and compare different choices of $\boldsymbol{\Lambda}_d$ in Appendix A. For the experiments in this work, we use the parameterization $\boldsymbol{\Lambda}_d = \boldsymbol{\Psi}_d\boldsymbol{C}_d$, where $\boldsymbol{\Psi}_d$ is diagonal with positive entries and $\boldsymbol{C}_d$ is circulant, symmetric, and positive definite. This parameterization enables cheap computation of KL divergences and matrix-vector products while maintaining sufficient expressiveness (Appendix A). All results are qualitatively similar when instead using a simple diagonal parameterization $\boldsymbol{\Lambda}_d = \boldsymbol{\Psi}_d$.

Distribution over neural activity Evaluating $\log p(\boldsymbol{Y}|\boldsymbol{X}) = \sum_n \log p(\boldsymbol{y}_n|\boldsymbol{X})$ for each sample drawn from $q(\boldsymbol{X})$ is intractable for general noise models. Thus, we further lower-bound the ELBO of Equation 3.6 by introducing an approximation $q(\boldsymbol{f}_n|\boldsymbol{X})$ to the posterior $p(\boldsymbol{f}_n|\boldsymbol{y}_n,\boldsymbol{X})$:

$$\log p(\boldsymbol{y}_n|\boldsymbol{X}) \geq \mathbb{E}_{q(\boldsymbol{f}_n|\boldsymbol{X})}[\log p(\boldsymbol{y}_n|\boldsymbol{f}_n)] - \mathrm{KL}[q(\boldsymbol{f}_n|\boldsymbol{X})||p(\boldsymbol{f}_n|\boldsymbol{X})]. \tag{3.10}$$

We repeat the whitened variational strategy described at the outer level by writing

$$q(\boldsymbol{f}_n|\boldsymbol{X}) = \mathcal{N}(\boldsymbol{f}_n;\hat{\boldsymbol{\mu}}_n,\hat{\boldsymbol{\Sigma}}_n) \quad \text{with} \quad \hat{\boldsymbol{\mu}}_n = \hat{\boldsymbol{K}}^{\frac{1}{2}}\hat{\boldsymbol{\nu}}_n \quad \text{and} \quad \hat{\boldsymbol{\Sigma}}_n = \hat{\boldsymbol{K}}^{\frac{1}{2}}\boldsymbol{L}_n\boldsymbol{L}_n^T(\hat{\boldsymbol{K}}^{\frac{1}{2}})^T, \tag{3.11}$$

where $\hat{\boldsymbol{\nu}}_n \in \mathbb{R}^D$ is a neuron-specific vector of variational parameters to be optimized along with a lower-triangular matrix $\boldsymbol{L}_n \in \mathbb{R}^{D\times D}$; and $\hat{\boldsymbol{K}}$ denotes the covariance matrix of $p(\boldsymbol{f}|\boldsymbol{X})$, whose square root $\hat{\boldsymbol{K}}^{\frac{1}{2}} = \boldsymbol{X}^T\boldsymbol{S}$ follows from Equation 3.5. The low-rank structure of $\hat{\boldsymbol{K}}$ enables cheap matrix-vector products and KL divergences:

$$\mathrm{KL}[q(\boldsymbol{f}_n|\boldsymbol{X})||p(\boldsymbol{f}_n|\boldsymbol{X})] = \frac{1}{2}\left(\|\boldsymbol{L}_n\|_\mathrm{F}^2 - 2\log|\boldsymbol{L}_n| + ||\hat{\boldsymbol{\nu}}_d||^2 - D\right). \tag{3.12}$$

Note that the KL divergence does not depend on $\boldsymbol{X}$ in this whitened parameterization (Appendix A). Moreover, $q(\boldsymbol{f}_n|\boldsymbol{X})$ in Equation 3.11 has the form of the exact posterior when the noise model is Gaussian (Appendix A), and it is equivalent to a stochastic variational inducing point approximation (Hensman et al., 2015a) for general noise models (Appendix A).

Finally, we need to compute the first term in Equation 3.10:

$$\mathbb{E}_{q(\boldsymbol{f}_n|\boldsymbol{X})}[\log p(\boldsymbol{y}_n|\boldsymbol{f}_n)] = \sum_t \mathbb{E}_{q(f_{nt}|\boldsymbol{X})}[\log p(y_{nt}|f_{nt})]. \tag{3.13}$$

Each term in this sum is simply a 1-dimensional Gaussian expectation which can be computed analytically in the case of Gaussian or Poisson noise (with an exponential link function), and otherwise approximated efficiently using Gauss-Hermite quadrature (Appendix A; Hensman et al., 2015a).

Summary of the algorithm

Putting all of our approximations together, optimization proceeds at each iteration by drawing M Monte Carlo samples $\{\boldsymbol{X}_m\}_1^M$ from $q(\boldsymbol{X})$ and estimating the overall ELBO as:

$$\begin{aligned}\mathcal{L} = \frac{1}{M} \sum_{\boldsymbol{X}_m \sim q(\boldsymbol{X})} &\left[\sum_{n,t} \mathbb{E}_{q(f_{nt}|\boldsymbol{X}_m)} \left[\log p(y_{nt}|f_{nt}) \right] \right] \\ &- \sum_n \mathrm{KL}\left[q(\boldsymbol{f}_n) || p(\boldsymbol{f}_n) \right] - \sum_d \mathrm{KL}\left[q(\boldsymbol{x}_d) || p(\boldsymbol{x}_d) \right], \end{aligned} \tag{3.14}$$

where the expectation over $q(f_{nt}|\boldsymbol{X})$ is evaluated analytically or using Gauss-Hermite quadrature depending on the noise model (Appendix A). We maximize $\mathcal{L}$ using stochastic gradient ascent with Adam (Kingma and Ba, 2014). This has a total computational time complexity of $\mathcal{O}(MNTD^2 + MDT \log T)$ and memory complexity of $\mathcal{O}(MNTD^2)$, where N is the number of neurons, T the number of time points, and D the latent dimensionality. For large datasets such as the monkey reaching data in Section 3.1.3, we compute gradients using mini-batches across time to mitigate the memory cost. That is, gradients for the sum over t in Equation 3.14 are computed in multiple passes. The algorithm is described in pseudocode with further implementation and computational details in Appendix A. The model learned by bGPFA can subsequently be used for predictions on held-out data by conditioning on partial observations as used for cross-validation in Section 3.1.3 and discussed in Appendix A. Latent dimensions that have been 'discarded' by automatic relevance determination will automatically have negligible contributions to the resulting posterior predictive distribution since the prior scale parameters s_d are approximately zero for these dimensions (see Appendix A for details).

3.1.3 Experiments and results

Synthetic data

We first generated an example dataset from the GPFA generative model (Equations 3.2-3.4) with a true latent dimensionality of 3. We proceeded to fit both factor analysis (FA), GPFA, and bGPFA with different latent dimensionalities $D \in [1, 10]$. Here, we fitted bGPFA without automatic relevance determination such that $s_d = s \forall d$. As expected, the marginal likelihoods increased monotonically with D for both FA and GPFA (Figure 3.2a; Appendix A). In contrast,

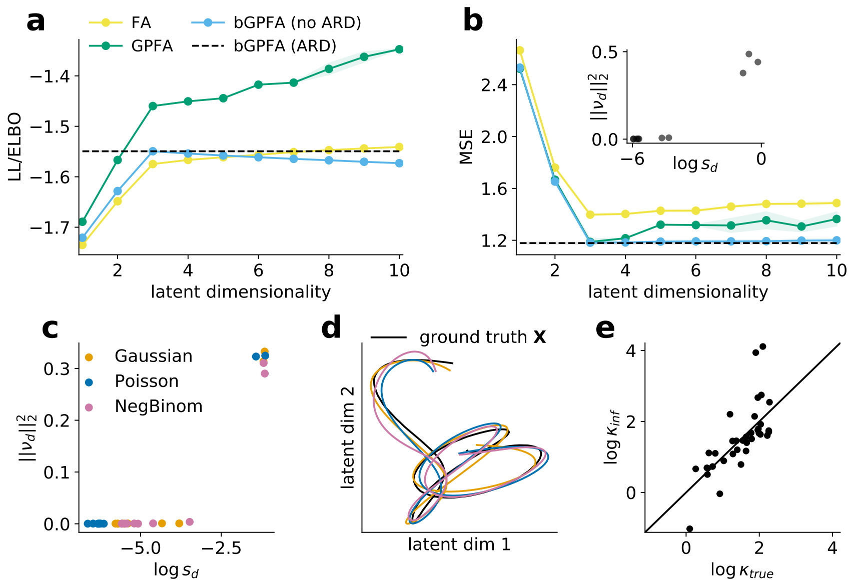

Figure 3.2 **Bayesian GPFA applied to synthetic data.** **(a)** Log likelihoods of factor analysis (yellow) & GPFA (green) and ELBO of Bayesian GPFA without ARD (blue) fitted to synthetic data with a ground truth dimensionality of three for different model dimensionalities. bGPFA with ARD recovered a three-dimensional latent space as well as the optimum ELBO of bGPFA without ARD (black dashed line). GPFA has a higher likelihood than FA because it includes a better prior over the latents, and GPFA has a higher likelihood than bGPFA because it uses a maximum likelihood estimate of $\boldsymbol{C}$ instead of integrating over $p(\boldsymbol{C})$. **(b)** Cross-validated prediction errors for the models in (a) (Appendix A). bGPFA with ARD recovered the performance of the optimal GPFA and bGPFA models without requiring a search over latent dimensionalities. Inspection of the learned prior scales $\{s_d\}$ and posterior mean parameters $||\boldsymbol{\nu}_d||_2^2$ (inset) indicates that ARD retained only $D^\star = 3$ informative dimensions (top right) and discarded the other 7 dimensions (bottom left). Shadings in (a) and (b) indicate ± 2 stdev. across 10 model fits. **(c)** Learned parameters of bGPFA with ARD and either Gaussian, Poisson or negative binomial noise models fitted to two-dimensional synthetic datasets with observations drawn from the corresponding noise models (Appendix A). The parameters clustered into two groups of informative (top right) and non-informative (bottom left) dimensions (Appendix A). **(d)** Latent trajectories in the space of the two most informative dimensions (c.f. (c)) for each model, with the ground truth shown in black. The inferred trajectories were aligned to the ground truth using linear transformations. **(e)** The overdispersion parameter κ_n for each neuron learned in the negative binomial model, plotted against the ground truth (Appendix A). Solid line indicates $y = x$; note that $\kappa_n \to \infty$ corresponds to a Poisson noise model.

the bGPFA ELBO reached its optimum value at the true latent dimensionality $D^\star = 3$. This is a manifestation of "Occam's razor", whereby fully Bayesian approaches favor the simplest model that adequately explains the data $\boldsymbol{Y}$ (MacKay, 2003). When instead considering the cross-validated predictive performance of each method, performance deteriorated rapidly for $D > 3$ for FA and GPFA, while Bayesian GPFA was more robust to overfitting (Figure 3.2b). Notably, the introduction of ARD parameters $\{s_d\}$ in bGPFA allowed us to fit a single model with large $D = 10$. This recovered the maximum ELBO of bGPFA without ARD and the minimum test error across GPFA and bGPFA without ARD (Figure 3.2a and b, black) without *a priori* assumptions about the latent dimensionality or the need to perform extensive cross-validation. Consistent with the ground truth generative process, only 3 of the scale parameters s_d remained well above zero after training (Figure 3.2b, inset). Similar to this illustrative example with Gaussian data, bGPFA with ARD and Poisson noise also exceeded the optimal performance of Poisson GPFA when applied to both synthetic and experimental spike count data (Appendix A).

We then proceeded to apply bGPFA ($D = 10$) to an example dataset drawn using Equations 3.4 and 3.5 with a ground truth dimensionality $D^{\star} = 2$, and either Gaussian, Poisson, or negative binomial noise. For all three datasets, the learned parameters clustered into a group of two latent dimensions with high information content (Appendix A) and a group of eight uninformative dimensions, consistent with the generative process (Figure 3.2c). In each case, we extracted the inferred latent trajectories corresponding to the informative dimensions and found that they recapitulated the ground truth up to a linear transformation (Figure 3.2d). Fitting flexible noise models such as the negative binomial model is important because neural firing patterns are known to be overdispersed in many contexts (Azouz and Gray, 1999; Fenton and Muller, 1998; Tomko and Crapper, 1974). However, it is often unclear how much of that overdispersion should be attributed to common fluctuations in hidden latent variables ($\boldsymbol{X}$ in our model) compared to private noise processes in single neurons (Low et al., 2018). In our synthetic data with negative binomial noise, we could accurately recover the single-neuron overdispersion parameters (Figure 3.2e; Appendix A), suggesting that such unsupervised models have the capacity to resolve overdispersion due to private and shared processes.

In summary, bGPFA provides a flexible method for inferring both latent dimensionalities, latent trajectories, and heterogeneous single-neuron parameters in an unsupervised manner. In the next section, we show that the scalability of the model and its interpretable parameters also facilitate the analysis of large neural population recordings.

Primate recordings

In this section, we apply bGPFA to biological data recorded from a rhesus macaque during a self-paced reaching task with continuous recordings spanning 30 minutes (Makin et al., 2018; O'Doherty et al., 2017; Figure 3.3a). The continuous nature of these recordings as one long trial makes it a challenging dataset for existing analysis methods that explicitly require the availability of many trials per experimental condition (Pandarinath et al., 2018), and poses computational challenges to Gaussian process-based methods that cannot handle long time series (Yu et al., 2009). While the ad-hoc division of continuous recordings into surrogate trials can still enable the use of these methods (Keshtkaran et al., 2021), here we show that our formulation of bGPFA readily applies to long continuous recordings. We fitted bGPFA with a negative binomial noise model to recordings from both primary motor cortex (M1) and primary somatosensory cortex (S1). For all analyses, we used a single recording session (`indy_20160426`, as in Keshtkaran et al., 2021), excluded neurons with overall firing rates below 2 Hz, and binned data at 25 ms resolution. This resulted in a data array $\boldsymbol{Y} \in \mathbb{R}^{200\times 70482}$ (130 M1 neurons and 70 S1 neurons).

We first fitted bGPFA independently to the M1 and S1 sub-populations with $D = 25$ latent dimensions. In this case, ARD retained 16 (M1) and 12 (S1) dimensions (Figure 3.3b). We then proceeded to train a linear decoder to predict hand kinematics in the form of x and y hand velocities from either the inferred firing rates or the raw data convolved with a 50 ms Gaussian kernel (Keshtkaran et al., 2021; Appendix A). We found that the model learned by bGPFA predicted kinematics better than the convolved spike trains, suggesting that (i) the latent space accurately captures kinematic representations, and (ii) the denoising and data-sharing across time in bGPFA aids decodability beyond simple smoothing of neural activity. Interestingly, by repeating this decoding analysis with an artificially imposed delay between neural activity and decoded behavior, we found that neurons in S1 predominantly encoded current behavior while neurons in M1 encoded a motor plan that predicted kinematics 100-150 ms into the future (Figure 3.3b). This is consistent with the motor neuroscience literature suggesting that M1 functions as a dynamical system driving behavior via downstream effectors (Churchland et al., 2012).

We then fitted bGPFA to the entire dataset including both M1 and S1 neurons. In this case, bGPFA retained 19 dimensions (Appendix A), and kinematic predictions improved over individual M1- and S1-based predictions (Figure 3.3b). In this analysis, the decoding performance as a function of delay between neural activity and behavior exhibited a broader peak than for the single-region decoding. We hypothesized that this broad peak reflects the fact that these neural populations encode both *current* behavior in S1 as well as *future* behavior in M1 (Figure 3.3c). Indeed, when we took this offset into account by shifting all M1 spike times by +100 ms and retraining the model, decoding performance increased from $68.56\% \pm 0.09$ to $69.81\% \pm 0.06$ (mean $\pm$ sem variance explained across ten model fits; Appendix A). Additionally, the shifted data exhibited a narrower decoding peak attained for near-zero delay between kinematics and latent trajectories (Figure 3.3d). Consistent with the improved kinematic decoding, we also found that shifting the M1 spikes by 100 ms increased the ELBO per neuron ($-34{,}637.0 \pm 0.7$ to $-34{,}631.1 \pm 0.6$) and decreased the dimensionality of the data (Appendix A; Recanatesi et al., 2019). These results suggest that M1 and S1 contain both overlapping but also non-redundant information, and that the most parsimonious description of the neural data is recovered by taking into account the different biological properties of M1 and S1.

We next wondered if bGPFA could be used to reveal putative motor preparation processes, which is non-trivial due to the lack of trial structure and well-defined preparatory epochs.

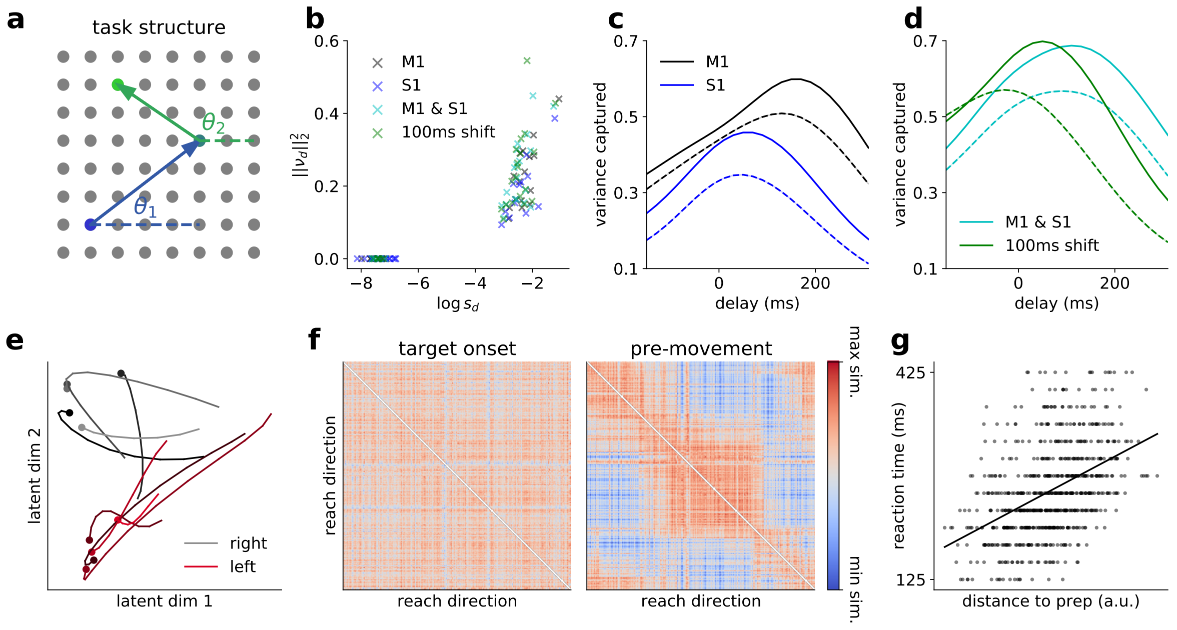

Figure 3.3 **Bayesian GPFA applied to primate data.** **(a)** Schematic illustration of the self-paced reaching task. When a target on a 17x8 grid is reached (arrows; 8x8 shown for clarity), a new target lights up on the screen (colours), selected at random from the remaining targets. In several analyses, we classify movements according to reach angle measured relative to horizontal (θ_1, θ_2). **(b)** Learned mean and scale parameters for the bGPFA models. Small prior scales s_d and posterior mean parameters ($||\boldsymbol{\nu}_d||_2^2$) indicate uninformative dimensions (Appendix A). **(c)** We applied bGPFA to monkey M1 and S1 data during the task and trained a linear model to decode kinematics from firing rates predicted from the inferred latent trajectories with different delays between latent states and kinematics. Neural activity was most predictive of future behavior in M1 (black) and current behavior in S1 (blue). Dashed lines indicate decoding from the raw data convolved with a Gaussian filter. **(d)** Decoding from bGPFA applied to the combined M1 and S1 data (cyan). Performance improved further when decoding from latent trajectories inferred from data where M1 activity was shifted by 100 ms relative to S1 activity (green). **(e)** Example trajectories in the two most informative latent dimensions for five rightward reaches (grey) and five leftward reaches (red). Trajectories are plotted from the appearance of the stimulus until movement onset (circles). During 'movement preparation', the latent trajectories move towards a consistent region of latent state space for each reach direction. **(f)** Similarity matrix of the latent state at stimulus onset showing no obvious structure (left) and 75 ms prior to movement onset showing modulation by reach direction (right). **(g)** Reaction time plotted against Euclidean distance between the latent state at target onset and the mean preparatory state for the corresponding reach direction ($\rho = 0.45$).

We partitioned the data post-hoc into individual 'reaches', each consisting of a period of time where the target location remained constant. For these analyses, we only considered 'successful' reaches, where the monkey eventually moved to the target location, and we defined movement onset as the first time during a reach where the cursor speed exceeded a low threshold (Appendix A). We began by visualizing the latent processes inferred by bGPFA as they unfolded prior to movement onset in each reach epoch. For visualization purposes, we ranked the latent dimensions based on their learned prior scales (a measure of variance explained; Appendix A) and selected the first two. Prior to movement onset, the latent trajectories tended to progress from their initial location at target onset towards reach-specific regions of state space (see example trials in Figure 3.3e for leftward and rightward reaches). To quantify this phenomenon, we computed pairwise similarities between latent states across all 762 reaches, during (i) stimulus onset and (ii) 75 ms before movement onset (chosen such that it is well before any detectable movement; Appendix A). We defined similarity as the negative Euclidean distance between latent states and restricted the analysis to 'fast' latent dimensions with timescales smaller than 200 ms to study this putatively fast process. When plotted as a function of reach direction, the latent similarities at target onset showed little discernable structure (Figure 3.3f, left). In contrast, the pairwise similarities became strongly structured 75 ms before movement onset where neighboring reach directions were associated with similar preparatory latent states (Figure 3.3f, right). Similar albeit noisier results were found when using factor analysis or GPFA instead of bGPFA (Appendix A). These findings are consistent with previous reports of monkey M1 partitioning preparatory and movement-related activity into distinct subspaces (Elsayed et al., 2016; Lara et al., 2018), as well as with the analogous finding that a 'relative target' subspace is active before a 'movement subspace' in previous analyses of this particular dataset (Keshtkaran et al., 2021).

Previous work on delayed reaches has shown that monkeys start reaching earlier when the neural state attained at the time of the go cue – which marks the end of a delay period with a known reach direction – is close to an "optimal subspace" (Afshar et al., 2011; Kao et al., 2021b). We wondered if a similar effect takes place during continuous, self-initiated reaching in the absence of explicit delay periods. Based on Figure 3.3e, we hypothesized that the monkey should start moving earlier if, at the time the next target is presented, its latent state is already close to the mean preparatory state for the required next movement direction. To test this, we extracted the mean preparatory state 75 ms prior to movement onset (as above) for each reach direction in the dataset. We found that the distance between the latent state at target onset and the corresponding mean preparatory state was strongly predictive of reaction time (Figure 3.3g, Pearson $\rho = 0.45$, $p = 4 \times 10^{-36}$). Such a correlation was also weakly present with factor analysis ($\rho = 0.21$, $p = 1.1 \times 10^{-8}$) but not detectable in the raw data ($\rho = 0.002$, $p = 0.95$). We also verified that the strong correlation found with bGPFA was not an artifact of the temporal correlations introduced by the prior (Appendix A). Taken together, our results suggest that motor preparation is an important part of reaching movements even in an unconstrained self-paced task. Additionally, we showed that bGPFA captures such behaviorally relevant latent dynamics better than simpler alternatives, and our scalable implementation enables its use on the large continuous reaching dataset analysed here.

Long-timescale latent processes

Some latent dimensions inferred by bGPFA also had long timescales on the order of 1.5 seconds, which is similar to the timescale of individual reaches (1-2 seconds; Appendix A). We hypothesized that these slow dynamics might reflect motivation or task engagement. Consistent with this hypothesis, we found that one of the slow latent processes ($\tau = 1.4$ s) was strongly correlated with reaction time during successful reaches (Pearson $\rho = 0.40$, $p = 3.4 \times 10^{-28}$). Interestingly, the information contained about reaction time in this long timescale latent dimension was largely complementary to that encoded by the distance to preparatory states in the 'fast' dimensions (Appendix A), suggesting that motor preparation and task engagement are orthogonal processes both contributing to task performance.

The experimental recordings were also characterized by a period of approximately five minutes towards the end of the recording session during which the monkey did not participate actively in the task and the cursor velocity was near-constant at zero (Figure 3.4a). When analysing neural activity across the periods with and without task participation, we found that neural dynamics moved to a different subspace as the monkey stopped engaging with the task (Figure 3.4b). Importantly, we were able to simultaneously capture these context-dependent changes as well as movement-specific and preparatory dynamics (Section 3.1.3) by fitting a single model to the full 30 minute dataset. This suggests that bGPFA can capture behaviorally relevant dynamics within individual contexts even when trained on richer datasets with changing contexts.

Finally, we wondered how the neural activity patterns during periods with and without task participation were related to the long-timescale latent dimensions predictive of task engagement. Here we found that the slow latent process considered above also exhibited a prominent change to a different state as the monkey stopped participating in the task (Figure 3.4c). This is consistent with our hypothesis that this latent process captures a feature related to task engagement, which slowly deteriorated during the first 24 minutes of the task followed by a discrete switch to a state with no engagement in the task. During the period of active task participation, this latent dimension was also correlated with time within the session. Indeed, reach number and latent state were both predictive of reaction time, but with the latent trajectory exhibiting a slightly stronger correlation (Pearson $\rho = 0.40$ vs. $\rho = 0.37$). It is not surprising that task engagement decreases with time, and it is in this case difficult to tease apart how motivation and time are differentially represented in such latent processes. However, based on the strong and abrupt modulation by task participation, this latent dimension appears to represent an aspect of engagement with the task beyond the passing of time.

Taken together, we thus find that bGPFA is capable of capturing not only single-reach dynamics and preparatory activity but also complementary processes evolving over longer timescales, which would be difficult to identify with methods designed for the analysis of many shorter trials.

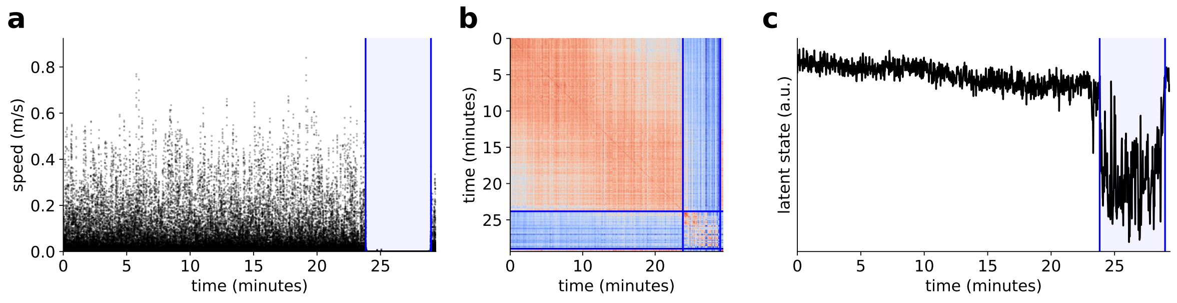

Figure 3.4 **Analysis of a period without task participation.** **(a)** Cursor speed over the course of the recording session. Blue horizontal lines indicate the last successful trial before and first successful trial after a period with no active task participation (blue shading). **(b)** Latent similarity matrix as a function of time during the task. The latent dynamics during task participation occur in a largely orthogonal subspace to the dynamics during the period with no active task participation. **(c)** Plot of latent state over time for a long-timescale latent dimension strongly correlated with reaction time ($\tau = 1.4$ s).

3.1.4 Discussion

Related work The generative model of bGPFA can be considered an extension of the canonical GPFA model proposed by Yu et al. (2009) to include a Gaussian prior over the loading matrix $\boldsymbol{C}$ (Section 3.1.2). In this view, bGPFA is to GPFA what Bayesian PCA is to PCA (Bishop, 1999); in particular, it facilitates automatic relevance determination to infer the dimensionality of the latent space from data (Bishop, 1999; Neal, 2012; Titsias and Lawrence, 2010). Similar to previous work in the field, we also use variational inference to facilitate arbitrary observation noise models, including non-Gaussian models more appropriate for electrophysiological recordings (Duncker and Sahani, 2018; Keeley et al., 2020a; Liu and Lengyel, 2021; Schimel et al., 2021; Zhao and Park, 2017; Zhao et al., 2020). While variational inference has proven a useful framework for such non-conjugate likelihood models, alternative approaches exist including the use of polynomial approximations to the non-linear terms in the likelihood (Keeley et al., 2020b). Another major challenge in the development of GP-based latent variable models such as bGPFA is to ensure scalability for longer time series. In this work, we utilize advances in variational inference (Kingma and Welling, 2013; Rezende et al., 2014) to facilitate scalability to the large datasets recorded in modern neuroscience. In particular, we contribute a new circulant variational GP posterior expressed partly in the Fourier domain that is both accurate and scalable. This is similar to Keeley et al. (2020a), who address the problem of scalability by assuming independence across Fourier features and formulating variational inference in the Fourier domain. However, we instead perform inference in the time domain and include additional factors in our variational posterior that ensure smoothness over time and allow for non-stationary posterior covariances. In contrast to these approaches, Zhao and Park (2017) rely on a low rank approximation to the prior covariance for inference and temporal subsamples for hyperparameter optimization to overcome the computational cost of model training. A conceptually similar approach employed by Duncker and Sahani (2018) is the use of inducing points, which has been studied extensively in the Gaussian process literature (Hensman et al., 2013, 2015a; Titsias, 2009). However, such low rank approximations can perform poorly on long time series where the number of inducing points needed is proportional to the recording duration (Chang et al., 2020).

bGPFA is also closely related to Gaussian process latent variable models (GPLVMs; Lawrence, 2005; Titsias and Lawrence, 2010; Section 2.2), which have recently found use in the neuroscience literature as a way of modelling flexible, non-linear tuning curves (Jensen et al., 2020; Liu and Lengyel, 2021; Wu et al., 2017a). This is because integrating out the loading matrix $\boldsymbol{C}$ in $p(\boldsymbol{Y}|\boldsymbol{X})$ with a Gaussian prior gives rise to a Gaussian process with a linear kernel. The low-rank structure of this linear kernel yields computationally cheap likelihoods, and our variational approach to estimating $\log p(\boldsymbol{Y}|\boldsymbol{X})$ is in fact equivalent to the sparse inducing point approximation used in the stochastic variational GP (SVGP) framework (Hensman et al., 2013, 2015a). In particular, our variational posterior is the same as that which would arise in SVGP with at least D inducing points irrespective of where those inducing points are placed (Appendix A). We also note that for a Gaussian noise model, the resulting low-rank Gaussian posterior is the form of the exact posterior distribution (Appendix A). Additionally, since the bGPFA observation model and prior over latents are both GPs, bGPFA is an example of a deep GP (Damianou and Lawrence, 2013) with two layers – the first with an RBF kernel and the second with a linear kernel. Finally, our parameterizations of the posteriors $q(\boldsymbol{x}_d)$ and $q(\boldsymbol{f}_n)$ can be viewed as variants of the 'whitening' approach introduced by Hensman et al. (2015b), which both facilitates efficient computation of the KL terms in the ELBOs and also stabilizes training (Section 3.1.2).

Conclusion In summary, bGPFA is an extension of the popular GPFA model in neuroscience that allows for regularized, scalable inference and automatic determination of the latent dimensionality as well as the use of non-Gaussian noise models appropriate for neural recordings. Importantly, the hyperparameters of bGPFA are efficiently optimized based on the ELBO on training data, which alleviates the need for cross-validation or complicated algorithms otherwise used for hyperparameter optimization in overparameterized models (Gao et al., 2016; Jensen et al., 2020; Keshtkaran and Pandarinath, 2019; Keshtkaran et al., 2021; Wu et al., 2017a; Yu et al., 2009). Our approach can also be extended to make it more useful to the neuroscience community. For example, replacing the spike count-based noise models with a point process model would provide higher temporal resolution (Duncker and Sahani, 2018), and facilitate inference of optimal temporal delays across neural populations (Lakshmanan et al., 2015). This will likely be useful as multi-region recordings become more prevalent in neuroscience (Keeley et al., 2020c).

3.2 Manifold Gaussian process latent variable models

This section has been peer reviewed and published as Jensen et al. (2020).

3.2.1 Introduction

As we have seen in several previous sections (Section 2.2, Section 3.1), it is common to use 'weak' Bayesian models to project high-dimensional neural data into lower-dimensional latent spaces as a first step towards linking neural activity to behaviour (Cunningham and Byron, 2014). As discussed in Section 2.2, this can be done using a variety of linear methods such as PCA or factor analysis (Cunningham and Ghahramani, 2015), or non-linear dimensionality reduction techniques such as tSNE (Maaten and Hinton, 2008). However, all these models project data into Euclidean latent spaces, thus failing to capture the inherent non-Euclidean nature of variables such as head direction, rotational motor plans, or grid cells (Bjerke et al., 2022; Chaudhuri et al., 2019; Finkelstein et al., 2015; Gardner et al., 2022; Seelig and Jayaraman, 2015; Wilson et al., 2018).

Most models in neuroscience assume that neurons are smoothly tuned to internal or external variables of interest (Stringer et al., 2019). As an example, a population of neurons representing an angular variable θ would respond similarly to some θ and to $\theta + \epsilon$ (for small ϵ). While it is straigthforward to model such smoothness by introducing smooth priors for response functions defined over $\mathbb{R}$, the activity of neurons modelled this way would exhibit a spurious discontinuity as the latent angle changes from 2π to $0 + \epsilon$. We see that appropriately modelling smooth neuronal representations requires keeping the latent variables of interest on their natural manifold (here, the circle), instead of an ad-hoc Euclidean space. While periodic kernels have commonly been used to address such problems in GP regression (MacKay, 1998), topological structure has not been incorporated into GP-based latent variable models due to the difficulty of doing inference in such spaces.

Here, we build on recent advances in non-Euclidean variational inference (Falorsi et al., 2019) to develop the manifold Gaussian process latent variable model (mGPLVM), an extension of the GPLVM framework (Lawrence, 2005; Titsias and Lawrence, 2010; Wu et al., 2018, 2017a) to non-Euclidean latent spaces including tori, spheres and $SO(3)$ (Figure 3.5). mGPLVM jointly learns the fluctuations of an underlying latent variable g *and* a probabilistic "tuning curve" $p(f_i|g)$ for each neuron i. The model therefore provides a fully unsupervised way of querying how the brain represents its surroundings and a readout of the relevant latent quantities. Importantly, the probabilistic nature of the model enables principled model selection between candidate manifolds, similar to how the ELBO was implicitly used for model selection across dimensionalities in Bayesian GPFA (Section 3.1). In this section, we provide a framework for scalable training and inference with mGPLVM, and we validate the model on both synthetic and experimental datasets.

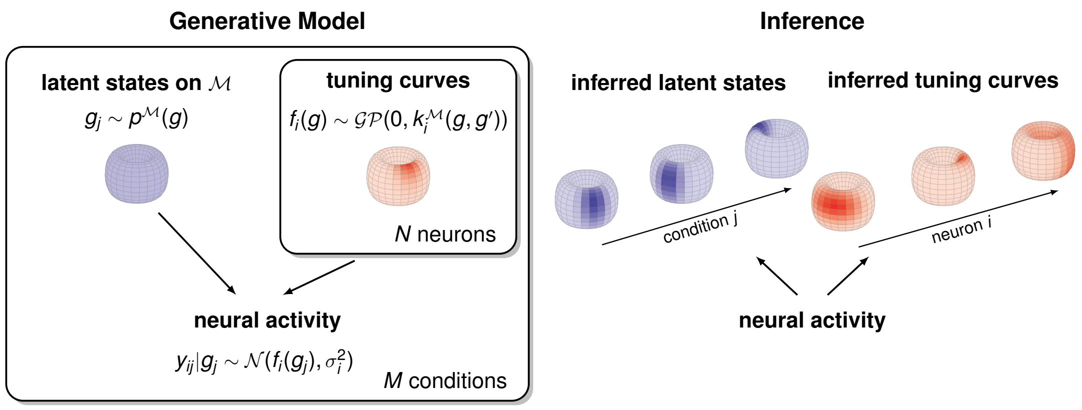
Generative Model
latent states on $\mathcal{M}$
$g_j \sim p^{\mathcal{M}}(g)$
tuning curves
$f_i(g) \sim \mathcal{GP}(0, k_i^{\mathcal{M}}(g, g'))$
N neurons
neural activity
$y_{ij}|g_j \sim \mathcal{N}(f_i(g_j), \sigma_i^2)$
M conditions
Inference
inferred latent states
inferred tuning curves
condition j
neuron i
neural activity

Figure 3.5 **Schematic illustration of the manifold Gaussian process latent variable model (mGPLVM).** In the generative model (left), neural activity arises from (i) M latent states $\{g_j\}$ on a manifold $\mathcal{M}$, each corresponding to a different condition j (e.g. time or stimulus), and (ii) the tuning curves of N neurons, modelled as Gaussian processes and sharing the same latent states $\{g_j\}$ as inputs. Using variational inference, mGPLVM jointly infers the global latent states and the tuning curve of each neuron on the manifold (right).

3.2.2 Method

The main contribution of this section is mGPLVM, a Gaussian process latent variable model (Titsias and Lawrence, 2010; Wu et al., 2018) defined for non-Euclidean latent spaces. We first present the generative model (Section 3.2.2), then explain how we perform approximate inference using reparameterizations on Lie groups (Falorsi et al., 2019; Section 3.2.2). Lie groups include Euclidean vector spaces $\mathbb{R}^n$ as well as other manifolds of interest to neuroscience such as tori T^n (Chaudhuri et al., 2019; Rubin et al., 2019) and the special orthogonal group $SO(3)$ (Finkelstein et al., 2015; Wilson et al., 2018; extensions to non-Lie groups are discussed in Appendix B). We then provide specific forms for variational densities and kernels on tori, spheres, and $SO(3)$ (Section 3.2.2). Finally we validate the method on both synthetic data (Section 3.2.3), calcium recordings from the fruit fly head direction system (Section 3.2.3), and extracellular recordings from the mouse anterodorsal thalamic nucleus (Appendix B).

Generative model

Let $\boldsymbol{Y} \in \mathbb{R}^{N \times M}$ be the activity of N neurons recorded in each of M conditions. Examples of "conditions" include time within a trial, stimulus identity, or motor output. We assume that all neuronal responses collectively encode a shared, condition-specific latent variable $g_j \in \mathcal{M}$, where $\mathcal{M}$ is some manifold. We further assume that each neuron i is tuned to the latent state g with a "tuning curve" $f_i(g)$, describing its average response conditioned on g. Rather than assuming a specific parametric form for these tuning curves, we place a Gaussian process prior on $f_i(\cdot)$ to capture the heterogeneity widely observed in biological systems (Churchland and

Shenoy, 2007; Hardcastle et al., 2017). The model is depicted in Figure 3.5 and can be formally described as:

$$g_j \sim p^{\mathcal{M}}(g) \qquad \text{(prior over latents)} \tag{3.15}$$
$$f_i \sim \mathcal{GP}(0, k_i^{\mathcal{M}}(\cdot,\cdot)) \qquad \text{(prior over tuning curves)} \tag{3.16}$$
$$y_{ij}|g_j \sim \mathcal{N}(f_i(g_j), \sigma_i^2) \qquad \text{(noise model)} \tag{3.17}$$

In Equation 3.15, we use a uniform prior $p^{\mathcal{M}}(g)$ inversely proportional to the volume of the manifold for bounded manifolds (Appendix B; see Jensen et al., 2022b for extensions to temporally correlated priors), and a Gaussian prior on Euclidean spaces to set a basic lengthscale. In Equation 3.16, $k_i^{\mathcal{M}}(\cdot,\cdot) : \mathcal{M} \times \mathcal{M} \to \mathbb{R}$ is a covariance function defined on manifold $\mathcal{M}$ – manifold-specific details are discussed in Section 3.2.2. In the special case where $\mathcal{M}$ is a Euclidean space, this model is equivalent to the standard Bayesian GPLVM (Titsias and Lawrence, 2010). While Equation 3.17 assumes independent noise across neurons, noise correlations can also be introduced as in (Wu et al., 2018) and Poisson noise as in (Wu et al., 2017a).

This probabilistic model can be fitted by maximizing the log marginal likelihood

$$\log p(\boldsymbol{Y}) = \log \int p(\boldsymbol{Y}|\{f_i\},\{g_j\})\, p(\{f_i\})\; p^{\mathcal{M}}(\{g_j\})\; d\{f_i\}d\{g_j\}. \tag{3.18}$$

Following optimization, we can query both the posterior over latent states $p(\{g_j\}|\boldsymbol{Y})$ and the posterior predictive distribution $p(\boldsymbol{Y}^\star|\mathcal{G}^\star, \boldsymbol{Y})$ at a set of query states $\mathcal{G}^\star$. While it is possible to marginalise out f_i when the states $\{g_j\}$ are known, further marginalising out $\{g_j\}$ is intractable, and maximizing Equation 3.18 requires approximate inference.

Learning and inference

To maximize $\log p(\boldsymbol{Y})$ in Equation 3.18, we use variational inference as previously proposed for GPLVMs (Titsias and Lawrence, 2010; Section 2.3). The true posterior over the latent states, $p(\{g_j\}|\mathbf{Y})$, is approximated by a variational distribution $q_\phi(\{g_j\})$ with parameters ϕ that are optimized to minimize the KL divergence between $q_\phi(\{g_j\})$ and $p(\{g_j\}|\boldsymbol{Y})$. This is equivalent to maximizing the evidence lower bound (ELBO) on the log marginal likelihood:

$$\mathcal{L}(\phi) = H(q_\phi) + \mathbb{E}_{q_\phi}[\log p^{\mathcal{M}}(\{g_j\})] + \mathbb{E}_{q_\phi}[\log p(\boldsymbol{Y}|\{g_j\})]. \tag{3.19}$$

Here, $\mathbb{E}_{q_\phi}[\cdot]$ indicates averaging over the variational distribution and $H(q_\phi)$ is its entropy. For simplicity, and because our model does not specify *a priori* statistical dependencies between the individual elements of $\{g_j\}$, we choose a variational distribution q_ϕ that factorizes over conditions:

$$q_\phi(\{g_j\}) = \prod_{j=1}^{M} q_{\phi_j}(g_j). \tag{3.20}$$

In the Euclidean case, the entropy and expectation terms in Equation 3.19 can be calculated analytically for some kernels (Titsias and Lawrence, 2010), and otherwise using the reparameterization trick (Kingma and Welling, 2013; Rezende et al., 2014). Briefly, the reparameterization trick involves first sampling from a fixed, easy-to-sample distribution (e.g. a normal distribution with zero mean and unit variance), and applying a series of differentiable transformations to obtain samples from q_ϕ (Section 2.3). We can then use these samples to estimate the entropy term and expectations in Equation 3.19.

For non-Euclidean manifolds, inference in mGPLVM poses two major problems. Firstly, we can no longer calculate the ELBO analytically nor evaluate it using the standard reparameterization trick. Secondly, evaluating the Gaussian process log marginal likelihood $\log p(\boldsymbol{Y}|\{g_j\})$ exactly becomes computationally too expensive for large datasets. We address these issues in the following.

Reparameterizing distributions on Lie groups To estimate and optimize the ELBO in Equation 3.19 when q_ϕ is defined on a non-Euclidean manifold, we use Falorsi et al.'s ReLie framework, an extension of the standard reparameterization trick to variational distributions defined on Lie groups.

Sampling from q_ϕ Since we assume that q_ϕ factorizes (Equation 3.20), sampling from q_ϕ is performed by independently sampling from each q_{ϕ_j}. We start from a differentiable base distribution $r_{\phi_j}(\boldsymbol{x})$ in $\mathbb{R}^n$. Note that $\mathbb{R}^n$ is isomorphic to the tangent space at the identity element of the group G, known as the Lie algebra. We can thus define a 'capitalized' exponential map $\mathrm{Exp}_G : \mathbb{R}^n \to G$, which maps elements of $\mathbb{R}^n$ to elements in G (Sola et al., 2018; Appendix B).

Importantly, Exp_G maps a distribution centered at zero in $\mathbb{R}^n$ to a distribution $\tilde{q}_{\phi_j}$ in the group centered at the identity element. To obtain samples from a distribution q_{ϕ_j} centered at an arbitrary g_j^μ in the group, we can simply apply the group multiplication with g_j^μ to samples from $\tilde{q}_{\phi_j}$. Therefore, obtaining a sample g_j from q_{ϕ_j} involves the following steps: (i) sample from $r_{\phi_j}(\boldsymbol{x})$, (ii) apply Exp_G to obtain a sample $\tilde{g}_j$ from $\tilde{q}_{\phi_j}$, and (iii) apply the group multiplication $g_j = g_j^\mu \tilde{g}_j$.

Estimating the entropy $H(q_\phi)$ Since $H(q_{\phi_j}) = H(\tilde{q}_{\phi_j})$ (Falorsi et al., 2019), we use K independent Monte Carlo samples from $\tilde{q}_\phi(\cdot) = \prod_{j=1}^{M} \tilde{q}_{\phi_j}(\cdot)$ to calculate

$$H(q_\phi) \approx -\frac{1}{K}\sum_{k=1}^{K}\sum_{j=1}^{M}\log \tilde{q}_{\phi_j}(\tilde{g}_{jk}), \tag{3.21}$$

where $\tilde{g}_{jk} = \text{Exp}_G \boldsymbol{x}_{jk}$ and $\{\boldsymbol{x}_{jk} \sim r_{\phi_j}(\boldsymbol{x})\}_{k=1}^{K}$.

Evaluating the density $\tilde{q}_\phi$ To evaluate $\log \tilde{q}_{\phi_j}(\text{Exp}_G \boldsymbol{x}_{jk})$, we use the result from Falorsi et al. (2019) that

$$\tilde{q}_\phi(\tilde{g}) = \sum_{\boldsymbol{x}\in\mathbb{R}^n \,:\, \text{Exp}_G(\boldsymbol{x})=\tilde{g}} r_\phi(\boldsymbol{x})|\boldsymbol{J}(\boldsymbol{x})|^{-1}, \tag{3.22}$$

where $\boldsymbol{J}(\boldsymbol{x})$ is the Jacobian of Exp_G at $\boldsymbol{x}$. Thus, $\tilde{q}_\phi(\tilde{g})$ is the sum of the Jacobian-weighted densities $r_\phi(\boldsymbol{x})$ in $\mathbb{R}^n$ at *all* those points that are mapped to $\tilde{g}$ through Exp_G. This is an infinite but converging sum, and following Falorsi et al. (2019) we approximate it by its first few dominant terms (Appendix B).

Note that $\text{Exp}_G(\cdot)$ and the group multiplication by g^μ are both differentiable operations. Therefore, as long as we choose a differentiable base distribution $r_\phi(\boldsymbol{x})$, we can perform end-to-end optimization of the ELBO. In this work we choose the reference distribution to be a multivariate normal $r_{\phi_j}(\boldsymbol{x}) = \mathcal{N}(\boldsymbol{x}; 0, \boldsymbol{\Sigma}_j)$ for each q_{ϕ_j}. We variationally optimize both $\{\boldsymbol{\Sigma}_j\}$ and the mean parameters $\{g_j^\mu\}$ for all j, and together these define the variational distribution.

Sparse GP approximation To efficiently evaluate the $\mathbb{E}_{q_\phi}[\log p(\boldsymbol{Y}|\{g_j\})]$ term in the ELBO for large datasets, we use the variational sparse GP approximation (Titsias, 2009), which has previously been applied to Euclidean GPLVMs (Titsias and Lawrence, 2010). Specifically, we introduce a set of m inducing points $\mathcal{Z}_i$ for each neuron i, and use a lower bound on the GP log marginal likelihood:

$$\log p(\boldsymbol{y}_i|\{g_j\}) \geq \underbrace{-\frac{1}{2}\boldsymbol{y}_i^T(\boldsymbol{Q}_i + \sigma_i^2\boldsymbol{I})^{-1}\boldsymbol{y}_i - \frac{1}{2}\log|\boldsymbol{Q}_i + \sigma_i^2\boldsymbol{I}| - \frac{1}{2\sigma^2}\text{Tr}(\boldsymbol{K}_i - \boldsymbol{Q}_i) + \text{const.}}_{\log \tilde{p}(\boldsymbol{y}_i|\{g_j\})} \tag{3.23}$$

$$\text{with } \boldsymbol{Q}_i = \boldsymbol{K}_{\{g_j\}\mathcal{Z}_i}\boldsymbol{K}_{\mathcal{Z}_i\mathcal{Z}_i}^{-1}\boldsymbol{K}_{\mathcal{Z}_i\{g_j\}}. \tag{3.24}$$

Here, $\boldsymbol{K}_{\mathcal{A}\mathcal{B}}$ denotes the Gram matrix associated with any two input sets $\mathcal{A}$ and $\mathcal{B}$. Note that the latents $\{g_j\}$ are shared across all neurons. In this work, we optimize the inducing points on G directly, but they could equivalently be optimized in $\mathbb{R}^n$ and projected onto G via Exp_G.

Using the sparse GP framework, the cost of computing the GP likelihood reduces to $\mathcal{O}(Mm^2)$ for each neuron and Monte Carlo sample. This leads to an overall complexity of $\mathcal{O}(KNMm^2)$ for approximating $\mathbb{E}_{q_\phi}[\log p(\boldsymbol{Y}|\{g_j\})]$ with K Monte Carlo samples, N neurons, M conditions and m inducing points (see Appendix B for further details on complexity and implementation).

Optimization We are now equipped to optimize the ELBO defined in Equation 3.19 using Monte Carlo samples drawn from a variational distribution q_ϕ defined on a Lie group G. To train the model, we use Adam (Kingma and Ba, 2014) to perform stochastic gradient descent on the following loss function:

$$\mathcal{L}(\phi) = \frac{1}{K}\sum_{k=1}^{K}\left[\sum_{j=1}^{M}\Big(\log p^{\mathcal{M}}(g_{jk}) - \log \tilde{q}_{\phi_j}(\tilde{g}_{jk})\Big) - \sum_{i}^{N}\log \tilde{p}(\boldsymbol{y}_i|\{g_{jk}\})\right], \tag{3.25}$$

where a set of K Monte-Carlo samples $\{\tilde{g}_{jk}\}_{k=1}^{K}$ is drawn at each iteration from $\{\tilde{q}_{\phi_j}\}$ as described in Section 3.2.2. In Equation 3.25, $g_{jk} = g_j^{\mu}\tilde{g}_{jk}$, where g_j^{μ} is a group element that is optimized together with all other model parameters. Finally, $\log \tilde{p}(\boldsymbol{y}_i|\{g_j\})$ is the lower bound defined in Equation 3.23, and $p^{\mathcal{M}}(g_{jk})$ is the prior described in Section 3.2.2. The inner sums run over conditions j and neurons i.

Posterior over tuning curves We approximate the posterior predictive distribution over tuning curves by sampling from the (approximate) posterior over latents. Specifically, for a given neuron i and a set of query states $\mathcal{G}^\star$, the posterior predictive over $\boldsymbol{f}_i^\star$ is approximated by:

$$p(\boldsymbol{f}_i^\star|\boldsymbol{Y},\mathcal{G}^\star) = \frac{1}{K}\sum_{k=1}^{K} p(\boldsymbol{f}_i^\star|\mathcal{G}^\star,\{\mathcal{G}_k,\boldsymbol{Y}\}), \tag{3.26}$$

where each $\mathcal{G}_k$ is a set of M latent states (one for each condition in $\boldsymbol{Y}$) independently drawn from the variational posterior $q_\phi(\cdot)$. In Equation 3.26, each term in the sum is a standard Gaussian process posterior (Rasmussen and Williams, 2006), which we approximate as described above (Section 3.2.2; Appendix B; Titsias, 2009).

Applying mGPLVM to tori, spheres and SO(3)

At this stage, we have yet to define the manifold-specific GP kernels $k^{\mathcal{M}}$ described in Section 3.2.2. These kernels ought to capture the topology of the latent space and express our prior assumptions that the neuronal tuning curves, defined on the manifold, have certain properties such as smoothness. Here we take inspiration from the common squared exponential covariance function defined over Euclidean spaces and introduce analogous kernels on tori, spheres, and $SO(3)$. This leads to the following general form:

$$k^{\mathcal{M}}(g,g') = \alpha^2 \exp\left(-\frac{d_{\mathcal{M}}(g,g')}{2\ell^2}\right) \qquad g,g' \in \mathcal{M}, \tag{3.27}$$

where α^2 is a variance parameter, ℓ is a characteristic lengthscale, and $d_{\mathcal{M}}(g, g')$ is a manifold-specific distance function. While squared geodesic distances might be intuitive choices for $d(\cdot,\cdot)$ in Equation 3.27, they result in positive semi-definite (PSD) kernels only for Euclidean latent spaces (Feragen et al., 2015; Jayasumana et al., 2015). Therefore, we build distance functions that automatically lead to valid covariance functions by observing that (i) dot product kernels are PSD, and (ii) the exponential of a PSD kernel is also PSD. Specifically, we use the following manifold-specific dot product-based distances:

$$d_{R^n}(g, g') = ||g - g'||_2^2 \qquad g \in \mathbb{R}^n \tag{3.28}$$

$$d_{S^n}(g, g') = 2(1 - g \cdot g') \qquad g \in \{\boldsymbol{x} \in \mathbb{R}^{n+1};\ \|\boldsymbol{x}\| = 1\} \tag{3.29}$$

$$d_{T^n}(g, g') = 2\textstyle\sum_k (1 - g_k \cdot g'_k) \qquad g \in \{(g_1, \cdots, g_n);\ \forall k:\ g_k \in \mathbb{R}^2, \|g_k\| = 1\} \tag{3.30}$$

$$d_{SO(3)}(g, g') = 4\left[1 - (g \cdot g')^2\right] \qquad g \in \{\boldsymbol{x} \in \mathbb{R}^4;\ \|\boldsymbol{x}\| = 1\} \tag{3.31}$$

Here, we have slightly abused notation by directly using "g" to denote a convenient parameterisation of the group elements which we define on the right of each equation. To build intuition, we note that the distance metric on the torus gives rise to a multivariate von Mises function; the distance metric on the sphere leads to an analogous von Mises Fisher function; and the distance metric on $SO(3)$ is $2(1 - \cos\varphi_{\text{rot}})$ where φ_{rot} is the angle of rotation required to transform g into g'. Notably, all these distance functions reduce to the Euclidean squared exponential kernel in the small angle limit. Laplacian (Feragen et al., 2015) and Matérn (Borovitskiy et al., 2020) kernels have previously been proposed for modelling data on Riemannian manifolds, and these can also be incorporated in mGPLVM.

Finally, we provide expressions for the variational densities (Equation 3.22) defined on tori, S^3 and $SO(3)$:

$$\tilde{q}_\phi(\text{Exp}_{T^n}\boldsymbol{x}) = \sum_{\boldsymbol{k} \in \mathbb{Z}^n} r_\phi(\boldsymbol{x} + 2\pi\boldsymbol{k}), \tag{3.32}$$

$$\tilde{q}_\phi(\text{Exp}_{SO(3)}\boldsymbol{x}) = \sum_{k \in \mathbb{Z}} \left[r_\phi(\boldsymbol{x} + \pi k\hat{\boldsymbol{x}}) \, \frac{2\|\boldsymbol{x} + \pi k\hat{\boldsymbol{x}}\|^2}{1 - \cos(2\|\boldsymbol{x} + \pi k\hat{\boldsymbol{x}}\|)} \right], \tag{3.33}$$

$$\tilde{q}_\phi(\text{Exp}_{S^3}\boldsymbol{x}) = \sum_{k \in \mathbb{Z}} \left[r_\phi(\boldsymbol{x} + 2\pi k\hat{\boldsymbol{x}}) \, \frac{2\|\boldsymbol{x} + 2\pi k\hat{\boldsymbol{x}}\|^2}{1 - \cos(2\|\boldsymbol{x} + 2\pi k\hat{\boldsymbol{x}}\|)} \right], \tag{3.34}$$

where $\hat{\boldsymbol{x}} = \boldsymbol{x}/\|\boldsymbol{x}\|$. Further details and the corresponding exponential maps are given in Appendix B. Since spheres that are not S^1 or S^3 are not Lie groups, ReLie does not provide a general framework for mGPLVM on these manifolds, which we therefore treat separately in Appendix B.

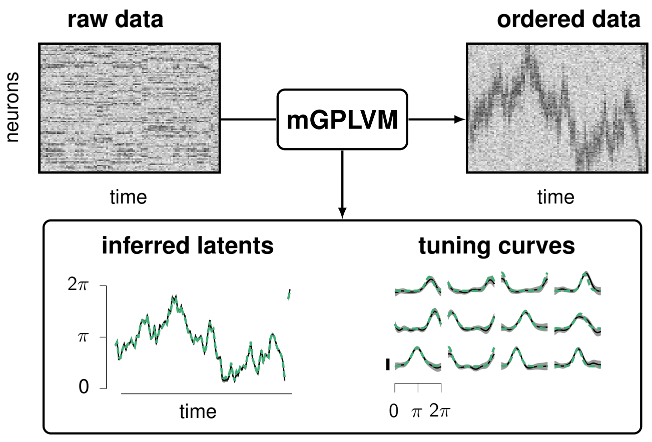

Figure 3.6 **Applying mGPLVM to synthetic data on the ring T^1. Top left**: neural activity of 100 neurons at 100 different conditions (here, time bins). **Bottom**: timecourse of the latent states (left) and tuning curves for 12 representative neurons (right). Green: ground truth; Black: posterior mean; Grey shaded regions: ± 2 posterior s.t.d. **Top right**: data replotted from the top left panel, with neurons reordered according to their preferred angles as determined by the inferred tuning curves.

3.2.3 Experiments and results

In this section, we start by demonstrating the ability of mGPLVM to correctly infer latent states and tuning curves in non-Euclidean spaces using synthetic data generated on T^1, T^2 and $SO(3)$. We also verify that cross-validated model comparison correctly recovers the topology of the underlying latent space, suggesting that mGPLVM can be used for model selection given a set of candidate manifolds. Finally, we apply mGPLVM to a biological dataset to show that it is robust to the noise and heterogeneity characteristic of experimental recordings.

Synthetic data

To generate synthetic data $\mathbf{Y}$, we specify a target manifold $\mathcal{M}$, draw a set of M latent states $\{g_j\}$ on $\mathcal{M}$, and assign a tuning curve to each neuron i of the form

$$f_i(g) = a_i^2 \exp\left(-\frac{d_{\text{geo}}^2(g, g_i^{\text{pref}})}{2b_i^2}\right) + c_i, \tag{3.35}$$

$$y_{ij}|g_j \sim \mathcal{N}(f_i(g_j), \sigma_i^2) \tag{3.36}$$

with random parameters a_i, b_i and c_i. Thus, the activity of each neuron is a noisy bell-shaped function of the geodesic distance on $\mathcal{M}$ between the momentary latent state g_j and the neuron's preferred state g_i^{pref} (sampled uniformly). While this choice of tuning curves is inspired by the common 'Gaussian bump' model of neural tuning, we emphasize that the non-parametric prior over f_i in mGPLVM can discover any smooth tuning curve on the manifold, not just Gaussian bumps. For computational simplicity, here we constrain the mGPLVM parameters α_i, ℓ_i and σ_i to be identical across neurons. Note that we can only recover the latent space up to symmetries which preserve pairwise distances. In all figures, we have therefore aligned model predictions and ground truth for ease of visualization (Appendix B).

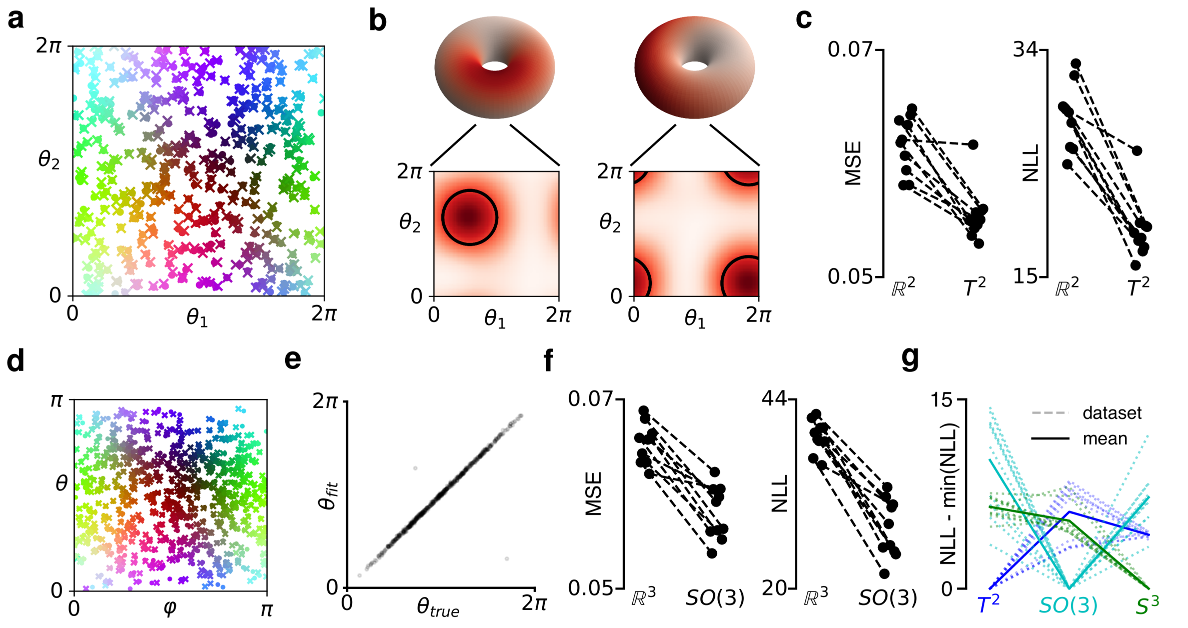

Figure 3.7 **Validating mGPLVM on synthetic data. (a-c)** Torus dataset. **(a)** True latent states $\{g_j \in T^2\}$ (dots) and posterior latent means $\{g_j^\mu\}$ (crosses). The color scheme is chosen to be smooth for the true latents. **(b)** Posterior tuning curves for two example neurons. Top: tuning curves on the tori. Bottom: projections onto the periodic $[0; 2\pi]$ plane. Black circles indicate locations and widths of the true tuning curves. **(c)** Mean squared cross-validated prediction error (left) and negative log likelihood (right) when fitting T^2 and $\mathbb{R}^2$ to data generated on T^2. Dashed lines connect datapoints for the same synthetic dataset. **(d-f)** $SO(3)$ dataset. **(d)** Axis of the rotation represented by the true latent states $\{g_j \in SO(3)\}$ (dots) and the posterior latent means $\{g_j^\mu\}$ (crosses) projected onto the (φ, θ)-plane. **(e)** Magnitude of the rotations represented by $\{g_j\}$ and $\{g_j^\mu\}$. **(f)** Same as (c), now comparing $SO(3)$ to $\mathbb{R}^3$. **(g)** Test log likelihood ratio for 10 synthetic datasets on T^2, $SO(3)$, & S^3, with mGPLVM fitted on each manifold (x-axis). Solid lines indicate mean across datasets.

We first generated data on the ring (T^1, Figure 3.6, top left), letting the true latent state be a continuous random walk across conditions for ease of visualization. We then fitted T^1-mGPLVM to the data and found that it correctly discovered the true latent states g as well as the ground truth tuning curves (Figure 3.6, bottom right). Reordering the neurons according to their preferred angles further exposed the population encoding of the angle (Figure 3.6, top right).

Next, we expanded the latent space to two dimensions with data now populating a 2-torus (T^2). Despite the non-trivial topology of this space, T^2-mGPLVM provided accurate inference of both latent states (Figure 3.7a) and tuning curves (Figure 3.7b). To show that mGPLVM can be used to distinguish between candidate topologies, we compared T^2-mGPLVM to a standard Euclidean GPLVM in $\mathbb{R}^2$ on the basis of both cross-validated prediction errors and importance-weighted marginal likelihood estimates (Burda et al., 2015). We simulated 10 different toroidal datasets; for each, we used half the conditions to fit the GP hyperparameters, and half the neurons to predict the latent states for the conditions not used to fit the GP parameters. Finally, we used the inferred GP parameters and latent states to predict the activity of the held-out neurons at the held-out conditions. As expected, the predictions of the toroidal model outperformed those of the standard Euclidean GPLVM which cannot capture the periodic boundary conditions of the torus (Figure 3.7c).

Beyond toroidal spaces, $SO(3)$ is of particular interest for the study of neural systems encoding 'yaw, pitch and roll' in a variety of 3D rotational contexts (Finkelstein et al., 2015; Shepard and Metzler, 1971; Wilson et al., 2018). We therefore fitted an $SO(3)$-mGPLVM to synthetic data generated on $SO(3)$ and found that it rendered a faithful representation of the latent space and outperformed a Euclidean GPLVM on predictions (Figure 3.7d-f). Finally we show that mGPLVM can also be used to select between multiple non-Euclidean topologies. We generated 10 datasets on each of T^2, $SO(3)$ and S^3 and compared cross-validated log likelihoods for T^2-, $SO(3)$- and S^3-mGPLVM, noting that $p(\mathcal{M}|\boldsymbol{Y}) \propto p(\boldsymbol{Y}|\mathcal{M})$ under a uniform prior over manifolds $\mathcal{M}$. Here we found that the correct latent manifold was consistently the most likely for all 30 datasets (Figure 3.7g). In summary, these results show robust performance of mGPLVM across various manifolds of interest in neuroscience and beyond, as well as a quantitative advantage over Euclidean GPLVMs which ignore the underlying topology of the latent space.

The *Drosophila* head direction circuit

Finally we applied mGPLVM to an experimental dataset to show that it is robust to biological and measurement noise. Here, we used calcium imaging data recorded from the ellipsoid body (EB) of *Drosophila melanogaster* (Turner-Evans, 2020; Turner-Evans et al., 2020), where the so-called E-PG neurons have recently been shown to encode head direction (Seelig and Jayaraman, 2015). The EB is divided into 16 'wedges', each containing 2-3 E-PG neurons that are not distinguishable on the basis of calcium imaging data, and we therefore treat each wedge as one 'neuron'. Due to the physical shape of the EB, neurons come 'pre-ordered' since their joint activity resembles a bump rotating on a ring (Figure 3.8a, analogous to Figure 3.6, "ordered data"). While the EB's apparent ring topology obviates the need for mGPLVM as an explorative tool for uncovering manifold representations, we emphasize that head direction circuits in higher organisms are not so obviously structured (Chaudhuri et al., 2019; Appendix B) – in fact, some brain areas such as the entorhinal cortex even embed concurrent representations of multiple spaces (Constantinescu et al., 2016; Hafting et al., 2005).

We fitted the full mGPLVM with separate GP hyperparameters for each neuron and found that T^1-mGPLVM performed better than $\mathbb{R}^1$-mGPLVM on both cross-validated prediction errors and log marginal likelihoods (Figure 3.8b). The model recovered latent angles that faithfully

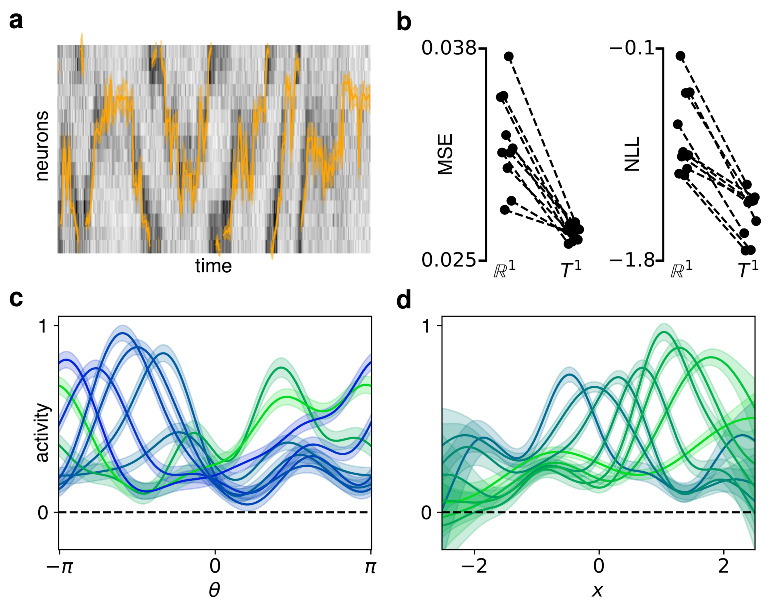

Figure 3.8 **The *Drosophila* head direction circuit.** **(a)** Input data overlaid with the posterior variational distribution over latent states of a T^1-mGPLVM. **(b)** Mean cross-validated prediction error (left) and negative log likelihood (right) for models fitted on T^1 and $\mathbb{R}^1$. Each datapoint corresponds to a different partition of the timepoints into a training set and a test set. **(c-d)** Posterior tuning curves for eight example neurons in T^1 (c) and $\mathbb{R}^1$ (d). Color encodes the position of the maximum of each tuning curve. Shadings in (a,c,d) indicate ± 2 s.t.d.

captured the visible rotation of the activity bump around the EB, with larger uncertainty during periods where the neurons were less active (Figure 3.8a, orange). When querying the posterior tuning curves from a fit in $\mathbb{R}^1$, these were found to suffer from spurious boundary conditions with inflated uncertainty at the edges of the latent representation – regions where $\mathbb{R}^1$-mGPLVM effectively has less data than T^1-mGPLVM since $\mathbb{R}^1$ does not wrap around. In comparison, the tuning curves were more uniform across angles in T^1, which correctly captures the continuity of the underlying manifold. In Appendix B, we describe similar results with mGPLVM applied to a dataset from the mouse head-direction circuit with more heterogeneous neuronal tuning and no obvious anatomical organization (Peyrache and Buzsáki, 2015).

3.2.4 Discussion

Conclusion We have presented an extension of the popular GPLVM model to incorporate non-Euclidean latent spaces. This is achieved by combining a Bayesian GPLVM with recently developed methods for approximate inference in non-Euclidean spaces and a new family of manifold-specific kernels. Inference is performed using variational sparse GPs for computational tractability with inducing points optimized directly on the manifold. We demonstrated that mGPLVM correctly infers the latent states and GP parameters for synthetic data of various dimensions and topologies, and that cross-validated model comparisons can recover the correct topology of the space. Finally, we showed how mGPLVM can be used to infer latent topologies and representations in biological circuits from calcium imaging data. We expect mGPLVM to be particularly valuable to the neuroscience community because many quantities encoded in the brain naturally live in non-Euclidean spaces (Chaudhuri et al., 2019; Finkelstein et al., 2015; Wilson et al., 2018).

Related work GP-based latent variable models with periodicity in the latent space have previously been used for motion capture, tracking and animation (Elgammal and Lee, 2008; Urtasun et al., 2008). However, these approaches are not easily generalized to other non-Euclidean topologies and do not provide a tractable marginal likelihood, which forms the basis of our Bayesian model comparisons. Additionally, methods have been developed for analysing the geometry of the latent space of GPLVMs (Tosi et al., 2014) and other latent variable models (Arvanitidis et al., 2017) after initially learning the models with a Euclidean latent. These approaches confer a degree of interpretability to the learned latent space but do not explicitly incorporate priors and topological constraints on the manifold during learning. Furthermore, GPs and GPLVMs with non-Euclidean outputs have been developed (Mallasto and Feragen, 2018; Mallasto et al., 2019; Navarro et al., 2017). These approaches are orthogonal to mGPLVM where the latent GP inputs, not outputs, live on a non-Euclidean manifold. mGPLVM can potentially be combined with these approaches to model non-Euclidean observations, and to incorporate more expressive GP priors over the latent states than the independent prior we have used here.

Finally, several methods for inference in non-Euclidean spaces have been developed in the machine learning literature. These have centered around methods based on VAEs (Davidson et al., 2018; Rey et al., 2019; Wang and Wang, 2019), normalizing flows (Rezende et al., 2020), and neural ODEs (Falorsi and Forré, 2020; Lou et al., 2020; Mathieu and Nickel, 2020). While non-Euclidean VAEs are useful for amortized inference, they constrain $f(g)$ more than a GP does and do not naturally allow expression of a prior over its smoothness. Normalizing flows and neural ODEs can potentially be combined with mGPLVM to increase the expressiveness of the variational distributions (Falorsi et al., 2019). This would allow us to model complex distributions over latents, such as the multimodal distributions that naturally arise in ambiguous environments with symmetries (Jacob et al., 2017).

mGPLVM extensions Here, we have assumed statistical independence across latent states, but prior dependencies could be introduced to incorporate e.g. temporal smoothness by placing a GP prior on the latents as in GPFA (Yu et al., 2009; see Jensen et al., 2022b for some extensions in this direction). To capture more statistical structure in the latents, richer variational approximations of the posterior could be learned by using normalizing flows on the base distribution (r_ϕ). It would also be interesting to exploit automatic relevance determination (ARD, Neal, 2012) in mGPLVM to automatically select the latent manifold dimension. We explored this approach by fitting a T^2-mGPLVM to the data from Figure 3.6 with separate lengthscales for the two dimensions, where we found that T^2 shrunk to T^1, the true underlying manifold (Appendix B).

Furthermore, the mGPLVM framework can be extended to direct products of manifolds, enabling the study of brain areas encoding non-Euclidean variables such as head direction jointly with global modulation parameters such as attention or velocity. As an example, fitting a $(T^1 \times \mathbb{R}^1)$-mGPLVM to the *Drosophila* data captures both the angular heading in the T^1 dimension as well as a variable correlated with global activity in the $\mathbb{R}^1$ dimension (Appendix B).

Future applications mGPLVM not only infers the most likely latent states but also estimates the associated uncertainty, which can be used as a proxy for the degree of momentary coherence expressed in neural representations. It would be interesting to compare such posterior uncertainties and tuning properties in animals across brain states. For example, uncertainty estimates could be compared across sleep and wakefulness or environments with reliable and noisy spatial cues.

In the motor domain, mGPLVM can help elucidate the neural encoding of motor plans for movements naturally specified in rotational spaces. Examples include 3-dimensional head rotations represented in the rodent superior colliculus (Masullo et al., 2019; Wilson et al., 2018) as well as analogous circuits in primates. Finally, it will be interesting to apply mGPLVM to artificial agents trained on tasks that require them to form internal representations of non-Euclidean environmental variables (Banino et al., 2018). Our framework could be used to dissect such representations, adding to a growing toolbox for the analysis of artificial neural networks (Sussillo and Barak, 2013).

Like Bayesian GPFA developed in Section 3.1, mGPLVM thus provides an additional tool in the Bayesian toolbox for neural data analysis. Together, these tools allow us to extract more interpretable features and lower-dimensional summaries from high-dimensional neural activity, and they demonstrate the importance of including appropriate inductive biases such as temporal continuity or non-Euclidean latent structure. In the remainder of this work, we will see how similar ideas can be used not just for data analysis but also to explicitly model the processes and computations taking place in biological neural circuits.

Chapter 4

Continual learning

In this chapter, we develop a new method for continual learning (Section 2.4). The method was originally developed with the aim of performing well in standard continual learning tasks from the machine learning literature (van de Ven and Tolias, 2019), and we compare it to several existing methods on these benchmarks (Section 4.1). We then go on to show how this and other algorithms for continual learning can also be considered strong models for computational neuroscience, where they lead to different predictions for the evolution of neural dynamics during sequential learning of multiple tasks. Finally, we compare these predictions to recent experimental data from rodent motor circuits after learning of a motor task, and we discuss the extent to which computational models can help us interpret experimental data on representational stability and drift (Section 4.2).

4.1 Natural continual learning

This section has been peer reviewed and published as Kao et al. (2021a).

4.1.1 Introduction

Catastrophic forgetting is a common feature of many machine learning algorithms, where training on a new task often leads to poor performance on previously learned tasks. This is in contrast to biological agents, which are capable of learning many different behaviors over the course of their lives with little to no interference across tasks. The study of continual learning in biological networks may therefore help inspire novel approaches in machine learning, while the development and study of continual learning algorithms in artificial agents can help us better understand how this challenge is overcome in the biological domain. This is particularly true for more challenging continual learning settings where task identity is not provided at test time, and for continual learning in recurrent neural networks (RNNs), which is important due to the practical and biological relevance of RNNs. However, continual learning in these settings has recently proven challenging for many existing algorithms, particularly those that rely on parameter regularization to mitigate forgetting (Duncker et al., 2020; Ehret et al., 2020; van de Ven and Tolias, 2019). In this work, we address these shortcomings by developing a continual learning algorithm that not only encourages good performance across tasks at convergence but also regularizes the optimization path itself using trust region optimization. This leads to improved performance compared to existing methods.

Previous work has addressed the challenge of continual learning in artificial agents using weight regularization, where parameters important for previous tasks are regularized to stay close to their previous values (Aljundi et al., 2018; Huszár, 2017; Kirkpatrick et al., 2017; Nguyen et al., 2017; Ritter et al., 2018; Zenke et al., 2017). This approach can be motivated by findings in the neuroscience literature of increased stability for a subset of synapses after learning (Xu et al., 2009; Yang et al., 2009). More recently, approaches based on projecting gradients into subspaces orthogonal to those that are important for previous tasks have been developed in both feedforward (Saha et al., 2021; Zeng et al., 2019) and recurrent (Duncker et al., 2020) neural networks. This is consistent with experimental findings that neural dynamics often occupy orthogonal subspaces across contexts in biological circuits (Ames and Churchland, 2019; Failor et al., 2021; Jensen et al., 2021; Kaufman et al., 2014). While these methods have been found to perform well in many continual learning settings, they also suffer from several shortcomings. In particular, while Bayesian weight regularization provides a natural way to weight previous and current task information, this approach can fail in practice due to its approximate nature and often requires additional tuning of the importance of the prior beyond what would be expected in a rigorous Bayesian treatment (van de Ven and Tolias, 2018). In contrast, while projection-based methods have been found empirically to mitigate catastrophic forgetting, it is unclear how the 'important subspaces' should be selected and how such methods behave when task demands begin to saturate the network capacity.

In this work, we develop natural continual learning (NCL), a new method that combines (i) Bayesian continual learning using weight regularization with (ii) an optimization procedure that relies on a trust region constructed from an approximate posterior distribution over the parameters given previous tasks. This encourages parameter updates predominantly in the null-space of previously acquired tasks while maintaining convergence to a maximum of the Bayesian approximate posterior. We show that NCL outperforms previous continual learning algorithms in both feedforward and recurrent networks. We also show that the projection-based methods introduced by Duncker et al. (2020) and Zeng et al. (2019) can be viewed as approximations to such trust region optimization using the posterior from previous tasks. Finally, we use tools from the neuroscience literature to investigate how the learned networks overcome the challenge of continual learning. Here, we find that the networks learn latent task representations that are stable over time after initial task learning, consistent with results from biological circuits. This provides a new strong Bayesian model of sequential task learning in biological circuits.

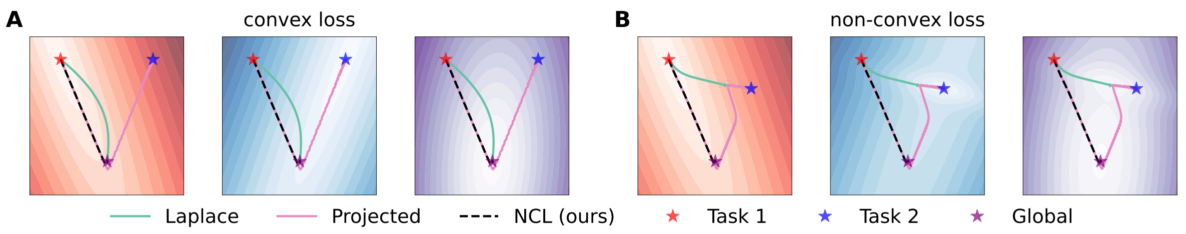

Figure 4.1 **Continual learning in a toy problem.** **(A)** Loss landscapes of task 1 (ℓ_1; left), task 2 (ℓ_2; middle) and the combined loss $\ell_{1+2} = \ell_1 + \ell_2$ (right). Stars indicate the global optima for ℓ_1 (red), ℓ_2 (blue), and ℓ_{1+2} (purple). We assume that θ has been optimized for ℓ_1 and consider how learning proceeds on task 2 using either the Laplace posterior ('Laplace', green), projected gradient descent on ℓ_2 with preconditioning according to task 1 ('Projected', pink), or NCL (black dashed). Laplace follows the steepest gradient of ℓ_{1+2} and transiently forgets task 1. NCL follows a flat direction of ℓ_1 and converges to the global optimum of ℓ_{1+2} with good performance on task 1 throughout. Projected gradient descent follows a similar optimization path to NCL but eventually diverges towards the optimum of ℓ_2. **(B)** As in (A), now with non-convex ℓ_2 (center), leading to a second local optimum of ℓ_{1+2} (right) while ℓ_1 is unchanged (left). In this case, Laplace can converge to a local optimum which has 'catastrophically' forgotten task 1. Projected gradient descent moves only slowly in 'steep' directions of ℓ_1 but eventually converges to a minimum of ℓ_2. Finally, NCL finds a local optimum of ℓ_{1+2} which retains good performance on task 1. See Appendix C for further mathematical details.

4.1.2 Method

Natural continual learning

While the online Laplace approximation discussed in Section 2.4 has been applied successfully in several continual learning settings (Kirkpatrick et al., 2017; Ritter et al., 2018), it has also been found to perform sub-optimally on a range of problems (Duncker et al., 2020; van de Ven and Tolias, 2018). Additionally, its Bayesian interpretation in theory prescribes a unique way of weighting the contributions of previous and current tasks to the loss. However, to perform well in practice, weight regularization approaches have been found to require ad-hoc re-weighting of the prior term by several orders of magnitude (Kirkpatrick et al., 2017; Ritter et al., 2018; van de Ven and Tolias, 2018). These shortcomings could be due to an inadequacy of the approximations used to construct the posterior (Section 2.4.1). However, we show in Figure 4.1 that standard gradient descent on the Laplace posterior has important drawbacks even in the exact case. First, we show that exact Bayesian inference on a simple continual regression problem can produce indirect optimization paths along which previous tasks are transiently forgotten as a new task is being learned (Figure 4.1A; green), unlike what is observed in biological agents. Second, when the loss is non-convex, we show that exact Bayesian inference can still lead to catastrophic forgetting (Figure 4.1B; green).

An alternative approach that has found recent success in a continual learning setting involves projection based methods which restrict parameter updates to a subspace that does not interfere with previous tasks (Duncker et al., 2020; Zeng et al., 2019). However, it is not immediately obvious how this projected subspace should be selected in a way that appropriately balances learning on previous and current tasks. Additionally, such projection-based algorithms have fixed points that are minima of the current task, but not necessarily minima of the (negative) Bayesian posterior. This can lead to catastrophic forgetting in the limit of long training times (Figure 4.1; pink), unless the learning rate is exactly zero in directions that interfere with previous tasks.

To combine the desirable features of both classes of methods, we introduce "Natural Continual Learning" (NCL) – an extension of the online Laplace approximation that also restricts parameter updates to directions which do not interfere strongly with previous tasks. In a Bayesian setting, we can conveniently express what is meant by such directions in terms of the prior precision matrix $\mathbf{\Lambda}$. In particular, 'flat' directions of the prior (low precision) correspond to directions that will not significantly affect the performance on previous tasks. Formally, we derive NCL as the solution of a trust region optimization problem. This involves minimizing the posterior loss $\mathcal{L}_k(\theta)$ within a region of radius r centered around θ with a distance metric of the form $d(\theta, \theta+\boldsymbol{\delta}) = \sqrt{\boldsymbol{\delta}^\top \mathbf{\Lambda}_{k-1}\boldsymbol{\delta}/2}$ that takes into account the curvature of the prior via its precision matrix $\mathbf{\Lambda}_{k-1}$:

$$\boldsymbol{\delta} = \arg\min_{\boldsymbol{\delta}} \mathcal{L}_k(\theta) + \nabla_\theta \mathcal{L}_k(\theta)^\top \boldsymbol{\delta} \quad \text{subject to } \frac{1}{2}\boldsymbol{\delta}^\top \mathbf{\Lambda}_{k-1}\boldsymbol{\delta} \leq r^2, \tag{4.1}$$

where $\mathcal{L}_k(\theta+\boldsymbol{\delta}) \approx \mathcal{L}_k(\theta) + \nabla_\theta \mathcal{L}_k(\theta)^\top \boldsymbol{\delta}$ is a first-order approximation to the updated Laplace objective. The solution to this subproblem is given by $\boldsymbol{\delta} \propto \mathbf{\Lambda}_{k-1}^{-1}\nabla_\theta \ell_k(\theta) - (\theta - \boldsymbol{\mu}_{k-1})$ (see Appendix C for a derivation), which gives rise to the NCL update rule

$$\theta \leftarrow \theta + \gamma\left[\mathbf{\Lambda}_{k-1}^{-1}\nabla_\theta \ell_k(\theta) - (\boldsymbol{\theta} - \boldsymbol{\mu}_{k-1})\right] \tag{4.2}$$

for a learning rate parameter γ (which is implicitly a function of r in Equation 4.1). To get some intuition for this learning rule, we note that $\mathbf{\Lambda}_{k-1}^{-1}$ acts as a preconditioner for the first (likelihood) term, which drives learning on the current task while encouraging parameter changes predominantly in directions that do not interfere with previous tasks. Meanwhile, the second term encourages θ to stay close to $\boldsymbol{\mu}_{k-1}$, the optimal parameters for the previous task. As we illustrate in Figure 4.1, this combines the desirable features of both Bayesian weight regularization and projection-based methods. In particular, NCL shares the fixed points of the Bayesian posterior while also mitigating intermediate or complete forgetting of previous tasks by preconditioning with the prior covariance. Notably, if the loss landscape is non-convex (as it generally will be), NCL can converge to a different local optimum from standard weight regularization despite having the same fixed points (Figure 4.1B).

Implementation The general NCL framework can be applied with different approximations to the Fisher matrix $\boldsymbol{F}_k$ in Equation 2.33 (see Section 2.4.1). In this work, we use a Kronecker-factored approximation (Martens and Grosse, 2015; Ritter et al., 2018). However, even after making a Kronecker-factored approximation to $\boldsymbol{F}_k$ for each task k, it remains difficult to compute the inverse of a sum of k Kronecker products (c.f. Equation 2.32). To address this challenge, we derived an efficient algorithm for making a Kronecker-factored approximation to $\boldsymbol{\Lambda}_k = \boldsymbol{F}_k + \boldsymbol{\Lambda}_{k-1} \approx \boldsymbol{A}_k \otimes \boldsymbol{G}_k$ when $\boldsymbol{\Lambda}_{k-1} = \boldsymbol{A}_{k-1} \otimes \boldsymbol{G}_{k-1}$ and $\boldsymbol{F}_k$ are also Kronecker products. This approximation minimizes the KL-divergence between $\mathcal{N}(\boldsymbol{\mu}_k, (\boldsymbol{A}_k \otimes \boldsymbol{G}_k)^{-1})$ and $\mathcal{N}(\boldsymbol{\mu}_k, (\boldsymbol{\Lambda}_{k-1} + \boldsymbol{F}_k)^{-1})$ (see Appendix C for details). Before training on the first task, we assume a spherical Gaussian prior $\theta \sim \mathcal{N}(\boldsymbol{0}, p_w^{-2}\boldsymbol{I})$. The scale parameter p_w can either be set to a fixed value (e.g. 1) or treated as a hyperparameter, and we optimize p_w explicitly for our experiments in feedforward networks. NCL also has a parameter α which is used to stabilize the matrix inversion $\boldsymbol{\Lambda}_{k-1}^{-1} \approx (\boldsymbol{A}_{k-1} \otimes \boldsymbol{G}_{k-1} + \alpha^2 \boldsymbol{I})^{-1}$ (Appendix C). This is equivalent to a hyperparameter used for such matrix inversions in OWM (Zeng et al., 2019) and DOWM (Duncker et al., 2020), and it is important for good performance with these methods. The p_w and α are largely redundant for NCL, and we generally prefer to fix α to a small value (10^{-10}) and optimize the p_w only. However, for our experiments in RNNs, we instead fix $p_w = 1$ and perform a hyperparameter optimization over α for a more direct comparison with OWM and DOWM. The NCL algorithm is described in pseudocode in Appendix C together with additional implementation and computational details.

Related work

As discussed in Section 2.4.1, our method is derived from prior work that relies on Bayesian inference to perform weight regularization for continual learning (Huszár, 2017; Kirkpatrick et al., 2017; Nguyen et al., 2017; Ritter et al., 2018). However, we also take inspiration from the literature on natural gradient descent (Amari, 1998; Kunstner et al., 2019) to introduce a preconditioner that encourages parameter updates primarily in flat directions of previously learned tasks.

Recent projection-based methods (Duncker et al., 2020; Saha et al., 2021; Zeng et al., 2019) have addressed the continual learning problem using an update rule of the form

$$\theta \leftarrow \theta + \gamma \boldsymbol{P}_L \nabla_\theta \ell_k(\theta) \boldsymbol{P}_R, \tag{4.3}$$

where $\boldsymbol{P}_L$ and $\boldsymbol{P}_R$ are projection matrices constructed from previous tasks which encourage parameter updates that do not interfere with performance on these tasks. Using Kronecker identities, we can rewrite Equation 4.3 as

$$\theta \leftarrow \theta + \gamma (\boldsymbol{P}_R \otimes \boldsymbol{P}_L) \nabla_\theta \ell_k(\theta). \tag{4.4}$$

This resembles the NCL update rule in Equation 4.2 where we identify $\boldsymbol{P}_R \otimes \boldsymbol{P}_L$ with the approximate inverse prior precision matrix used for gradient preconditioning in NCL, $\boldsymbol{\Lambda}_{k-1}^{-1} = \boldsymbol{A}_{k-1}^{-1} \otimes \boldsymbol{G}_{k-1}^{-1}$. Indeed, we note that for a Kronecker-structured approximation to $\boldsymbol{F}_k$, the matrix $\boldsymbol{A}_{k-1}$ approximates the empirical covariance matrix of the network activations experienced during all tasks up to $k-1$ (Bernacchia et al., 2018; Martens and Grosse, 2015, Appendix C), which is exactly the inverse of the projection matrix $\boldsymbol{P}_R$ used in previous work (Duncker et al., 2020; Zeng et al., 2019). We thus see that NCL takes the form of recent projection-based continual learning algorithms with two notable differences:

(i) NCL uses a *left* projection matrix $\boldsymbol{P}_L$ designed to approximate the posterior covariance of previous tasks $\boldsymbol{\Lambda}_{k-1}^{-1} \approx \boldsymbol{P}_R \otimes \boldsymbol{P}_L$ (i.e., the prior covariance on task k; Appendix C), while Zeng et al. (2019) use the identity matrix $\boldsymbol{I}$ and Duncker et al. (2020) use the covariance of recurrent inputs (Appendix C). Notably, both of these choices of $\boldsymbol{P}_L$ still provide reasonable approximations to $\boldsymbol{\Lambda}_{k-1}^{-1}$, and thus the parameter updates of OWM and DOWM can also be viewed as projecting out steep directions of the prior on task k.

(ii) NCL includes an additional regularization term $(\theta - \boldsymbol{\mu}_{k-1})$ derived from the Bayesian posterior objective, while Duncker et al. (2020) and Zeng et al. (2019) do not use such regularization. Importantly, this means that while NCL has a similar preconditioner and optimization path to these projection based methods, NCL has stationary points at the modes of the approximate Bayesian posterior while the stationary points of OWM and DOWM do not incorporate prior information from previous tasks (c.f. Figure 4.1).

It is also interesting to note that previous Bayesian continual learning algorithms include a hyperparameter λ that scales the prior compared to the likelihood term for the current task (Loo et al., 2020):

$$\mathcal{L}_k^{(\lambda)}(\theta) = \log p(\mathcal{D}_k|\theta) - \lambda(\theta - \boldsymbol{\mu}_{k-1})^\top \boldsymbol{\Lambda}_{k-1}(\theta - \boldsymbol{\mu}_{k-1}). \tag{4.5}$$

To minimize this loss and thus find a mode of the approximate posterior, it is common to employ pseudo-second-order stochastic gradient-based optimization algorithms such as Adam (Kingma and Ba, 2014) that use their own gradient preconditioner based on an approximation to the Hessian of Equation 4.5. Interestingly, this Hessian is given by $\boldsymbol{H}_k = -H(\mathcal{D}_k, \theta) - \lambda \boldsymbol{\Lambda}_{k-1}$, which in the limit of large λ becomes increasingly similar to preconditioning with the prior precision as in NCL. Consistent with this, previous work using the online Laplace approximation has found that large values of λ are generally required for good performance (Kirkpatrick et al., 2017; Ritter et al., 2018; van de Ven and Tolias, 2018). Recent work has also combined Bayesian continual learning with natural gradient descent (Osawa et al., 2019; Tseran et al., 2018), and in this case a relatively high value of $\lambda = 100$ was similarly found to maximize performance (Osawa et al., 2019).

4.1.3 Experiments and results

NCL in feedforward networks

To verify the utility of NCL for continual learning, we first compared our algorithm to standard methods in feedforward networks across two continual learning benchmarks: split MNIST and split CIFAR-100 (see Appendix C for task details). For each benchmark, we considered three continual learning settings (van de Ven and Tolias, 2019). In the 'task-incremental' setting, task identity is available to the network at test time, in our case via a multi-head output layer (Chaudhry et al., 2018). In the 'domain-incremental' setting, task identity is unavailable at test time, and the output layer is shared between all tasks. Finally, in the 'class-incremental' setting, the network has to both infer task identity and solve the task, in our case by performing classification over all possible classes irrespective of which task the input in question is drawn from.

van de Ven and Tolias previously showed that parameter regularization methods such as EWC perform poorly in the domain- and class-incremental settings (van de Ven and Tolias, 2019). We therefore applied NCL as well as synaptic intelligence [SI; Zenke et al., 2017], online EWC (Schwarz et al., 2018), Kronecker factored EWC [KFAC; Ritter et al., 2018], and orthogonal weight modification [OWM; Zeng et al., 2019] to split MNIST and split CIFAR-100 in the task-, domain- and class-incremental learning settings. For these continual learning problems, we found that NCL outperformed all the baseline methods in the task- and domain-incremental learning settings (Figure 4.2). In the class-incremental settings, we found that NCL performed comparably to but slightly worse than OWM. However, both OWM and NCL comfortably outperformed the other compared methods in this setting. These results suggest that the subpar performance of parameter regularization methods can be alleviated by regularizing their optimization paths, particularly in the domain- and class-incremental learning settings.

For the split MNIST and split CIFAR-100 experiments, each baseline method had a single hyperparameter (c for SI, λ for EWC and KFAC, α for OWM, and p_w for NCL; Appendix C) that was optimized on a held-out seed (see Appendix C). However, by setting the NCL prior to a unit Gaussian, we were also able to achieve good performance across task sets in a hyperparameter-free setting, further highlighting the robustness of the method (see "NCL (no opt)" in Figure 4.2).

NCL in recurrent neural networks

We then proceeded to consider how NCL compares to previous methods in recurrent neural networks (RNNs), a setting that has recently proven challenging for continual learning (Duncker et al., 2020; Ehret et al., 2020) and which is of interest to the study of continual learning in

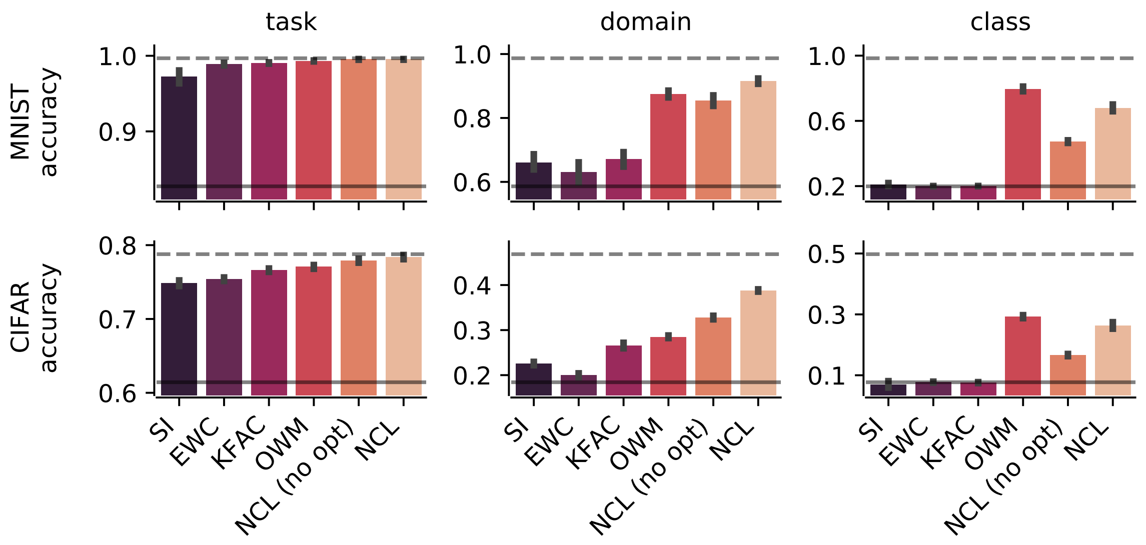

Figure 4.2 **NCL performance in feedforward networks.** Average test accuracy after learning all tasks on split MNIST (top row) and split CIFAR-100 (bottom-row) in the task-, domain- and class-incremental learning settings. Dashed horizontal lines denote average performance when networks are trained simultaneously on all tasks. Solid horizontal lines denote average performance when networks are trained sequentially on each task without applying any continual learning methods. Error bars denote standard error across 20 (MNIST) or 10 (CIFAR) random seeds. 'NCL' indicates natural continual learning where the initial prior has been optimized on a held-out random seed, and 'NCL (no opt)' indicates NCL with a simple unit Gaussian prior and no hyperparameter optimization. Numerical results for these experiments are provided in Table C.2 in Appendix C.

biological circuits (Duncker et al., 2020; Yang et al., 2019). In these experiments, the task identity is available to the RNN (i.e., we consider the task-incremental learning setting).

Stimulus-response tasks In this section, we consider a set of neuroscience inspired 'stimulus-response' (SR) tasks (Yang et al., 2019; details in Appendix C). We first compared the performance and behavior of NCL to OWM, the top performing method in the feedforward setting (Figure 4.2), and to the projection-based DOWM method designed explicitly for RNNs (Duncker et al., 2020). For a more direct comparison with OWM and DOWM, we fixed the NCL prior to a unit Gaussian for all RNN experiments and instead performed a hyperparameter optimization over 'α' used to regularize the matrix inversions for all three methods (Section 4.1.2, Appendix C,Duncker et al., 2020, Zeng et al., 2019). Following previous work, we trained RNNs with 256 recurrent units to sequentially solve six stimulus-response tasks (Duncker et al., 2020; Yang et al., 2009). While NCL, OWM and DOWM all managed to learn the six tasks without catastrophic forgetting, we found that NCL achieved superior average performance across tasks after training (Figure 4.3A).

We then compared NCL, OWM, and DOWM to KFAC, the top performing parameter regularization method in our feedforward experiments (Figure 4.2), which uses Adam (Kingma and Ba, 2014) to optimize the objective in Equation 2.31 with a Kronecker-factored approximation to the posterior precision matrix (Section 2.4.1; Ritter et al., 2018). Consistent with the results shown in Duncker et al. (2020), we found that NCL, OWM, and DOWM outperformed KFAC with $\lambda = 1$ (Figure 4.3A; see also Duncker et al., 2020 for a comparison of DOWM and EWC). We note that NCL and KFAC optimize the same objective function (Equation 2.31) and approximate the posterior precision matrix in the same way, but they differ in the way they precondition the gradient of the objective. These results thus demonstrate empirically that the choice of optimization algorithm is important to prevent forgetting, consistent with the intuition provided by Figure 4.1.

In feedforward networks, poor performance with weight regularization approaches such as EWC and KFAC has been mitigated by optimizing the hyperparameter λ, which increases the importance of the prior term compared to a standard Bayesian treatment (Equation 4.5; Section 4.1.3, Kirkpatrick et al., 2017; Loo et al., 2020; Ritter et al., 2018). We confirmed this here by performing a grid search over λ, which showed that KFAC with $\lambda \in [100, 1000]$ could perform comparably to the projection-based methods (Appendix C; Figure 4.3A). We hypothesize that the good performance provided by high λ is partly due to the approximate second order nature of Adam which, together with the relative increase in the prior term compared to the data term, leads to preconditioning with a matrix resembling the prior $\mathbf{\Lambda}_{k-1}$ (Section 4.1.2). In support of this hypothesis, we found that the KL divergence between the Adam preconditioner and the approximate prior precision $\mathbf{\Lambda}_{k-1}$ decreased with increasing λ, and that the performance of KFAC with Adam could also be rescued by increasing λ only when computing the preconditioner while retaining $\lambda = 1$ when computing the gradients (Appendix C).

Stroke MNIST One way to challenge the continual learning algorithms further is to increase the number of tasks. We thus considered an augmented version of the stroke MNIST dataset [SMNIST; de Jong, 2016]. The original dataset consists of the MNIST digits transformed into pen strokes with the direction of the stroke at each time point provided as an input to the network. Similar to Ehret et al. (2020), we constructed a continual learning problem by considering consecutive binary classification tasks inspired by the split MNIST task set. We further increased the number of tasks by including a set of extra digits where the x and y dimensions have been swapped in the input stroke data, and another set where both the x and y dimensions have changed sign. We also added high-variance noise to the inputs to increase the task difficulty. This gave rise to a total of 15 binary classification tasks, each with unique digits not used in other tasks, which we sought to learn in a continual fashion using an RNN with 30 recurrent units (see Appendix C for details).

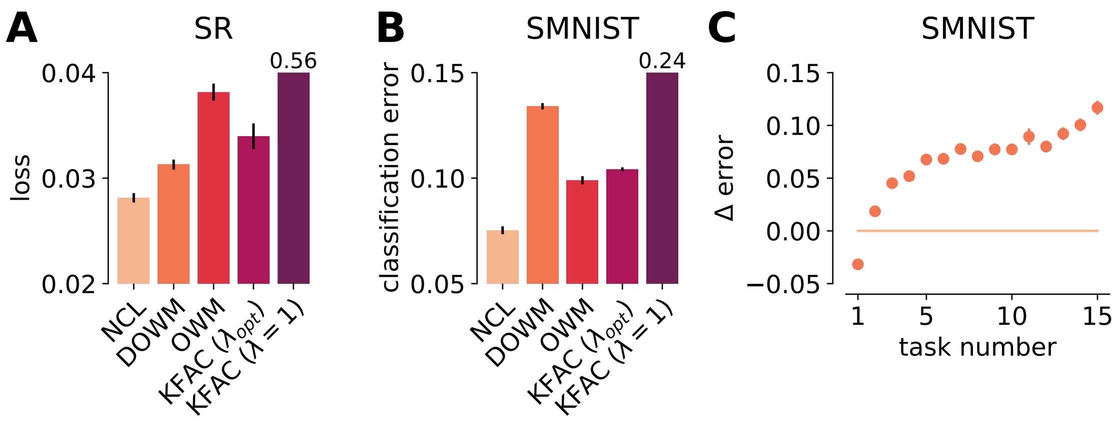

Figure 4.3 **Performance on SR and SMNIST tasks. (A)** Mean loss of NCL, DOWM, OWM, KFAC (optimal λ), and KFAC ($\lambda = 1$) across stimulus-response tasks after sequential training on all tasks. Error bars indicate standard error across 5 random seeds. Here and in (B), KFAC with $\lambda = 1$ failed catastrophically, and its performance is indicated in text as it does not fit on the axes. **(B)** Mean classification error across SMNIST tasks after sequential training. **(C)** Difference between the mean classification error of Laplace-DOWM and NCL as a function of task number. Error bars in (B) and (C) indicate standard error across 100 random task permutations.

As for the SR task set in Section 4.1.3, we found that NCL outperformed previous projection-based methods (Figure 4.3B). We again found that weight regularization with a KFAC approximation performed poorly with $\lambda = 1$, and that this poor performance could be partially rescued by optimizing over λ (Figure 4.3B). To investigate how the difference in performance between NCL and DOWM was affected by their different approximations to the Fisher matrix, we implemented NCL using the DOWM projection matrices as an alternative approximation to the inverse Fisher matrix. We refer to this method as Laplace-DOWM. We then considered how the performance on each task at the end of training depended on task number, averaged over different task permutations (Figure 4.3C). We found that while Laplace-DOWM outperformed NCL on the first task, this method generally performed worse on subsequent tasks. Notably, Laplace-DOWM exhibited a near-monotonic decrease in relative performance with task number, which is consistent with the intuition that DOWM overestimates the dimensionality of the parameter subspace that matters for previous tasks (Appendix C). In contrast, although neural circuits are known to use orthogonal subspaces in different contexts, there is no general sense that learning more tasks in the past should systematically hinder learning in future contexts for biological agents.

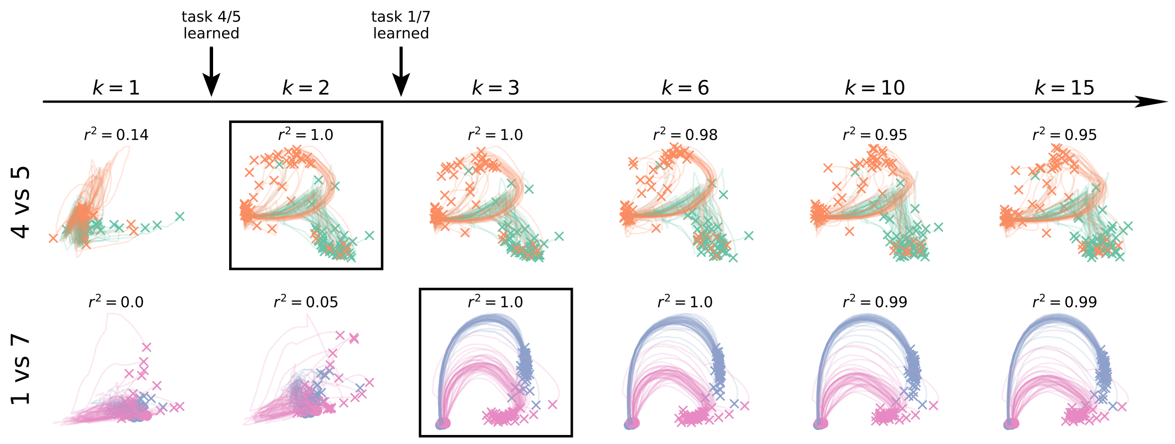

Figure 4.4 **Latent dynamics during SMNIST.** We considered two example tasks, 4 vs 5 (top) and 1 vs 7 (bottom). For each task, we simulated the response of a network trained by NCL to 100 digits drawn from that task distribution at different times during learning. We then fitted a factor analysis model for each example task to the response of the network right after the correponding task had been learned (squares; $k = 2$ and $k = 3$ respectively). We used this model to project the responses at different times during learning into a common latent space for each example task. For both example tasks, the network initially exhibited variable dynamics with no clear separation of inputs and subsequently acquired stable dynamics after learning to solve the task. The r^2 values above each plot indicate the similarity of neural population activity with that collected immediately after learning the corresponding task, quantified across all neurons (not just the 2D projection).

Dissecting the dynamics of networks trained on the SMNIST task set

To further investigate how the trained RNNs solve the continual learning problems and how this relates to the neuroscience literature, we dissected the dynamics of networks trained on the SMNIST task set using the NCL algorithm. To do this, we analyzed latent representations of the RNN activity trajectories, as is commonly done to study the collective dynamics of artificial and biological networks (Gallego et al., 2020; Jensen et al., 2021, 2020; Mante et al., 2013; Yu et al., 2009). We considered two consecutive classification tasks, namely classifying 4's vs 5's ($k = 2$) and classifying 1's vs 7's ($k = 3$). For each of these tasks, we trained a factor analysis model right after the task was learned, using network activity collected while presenting 50 examples of each of the two input digits associated with the task. We then tracked the network responses to the same set of stimuli at various stages of learning, both before and after the task in question was acquired, using the trained factor analysis model to visualize low-dimensional summaries of the dynamics (Figure 4.4).

Consistent with the network having successfully learned to solve these two tasks, we found that latent trajectories diverged over time for the two types of inputs in each task. Critically, these diverging dynamics only emerged after the task was learned, and remained highly stable thereafter (Figure 4.4). The stability of the task-associated representations is consistent with recent work in the neuroscience literature showing that, in a primate reaching task, latent neural trajectories remain stable after learning (Gallego et al., 2020). Since here we have access to the activity of all neurons throughout the task, we proceeded to quantify the source of this stability at the level of single units. The stability of such single-neuron dynamics after learning has recently been a topic of much interest in biological circuits (Clopath et al., 2017; Lütcke et al., 2013; Rule et al., 2019). In the RNNs, we found that the single-unit representations of a given digit changed during learning of the task involving that digit but stabilized after learning, consistent with work in several distinct biological circuits (Chestek et al., 2007; Dhawale et al., 2017; Ganguly and Carmena, 2009; Jensen et al., 2022a; Katlowitz et al., 2018; Peters et al., 2014). Similar results were found using the DOWM algorithm, which was explicitly designed to preserve network dynamics on previously learned tasks (Duncker et al., 2020).

4.1.4 Discussion

In summary, we have developed a new framework for continual learning based on approximate Bayesian inference combined with trust-region optimization. We showed that this framework encompasses recent projection-based methods and found that it performs better than naive weight regularization. This was particularly evident when task identity was not provided at test time and in recurrent neural networks, settings which have previously been challenging for many continual learning algorithms (Duncker et al., 2020; Ehret et al., 2020; van de Ven and Tolias, 2019). Furthermore, we showed that our principled probabilistic approach outperforms previous projection-based methods (Duncker et al., 2020; Zeng et al., 2019), in particular when the number of tasks and their complexity challenges the network's capacity. Finally, we analyzed the dynamics of the learned RNNs in a sequential binary classification problem, where we found that the latent dynamics adapt to each new task. We also found that the task-associated dynamics were subsequently conserved during further learning, consistent with experimental reports of stable neural representations (Dhawale et al., 2017; Gallego et al., 2020; Jensen et al., 2022a). Importantly, our results suggest that preconditioning with the prior covariance can lead to improved performance over existing continual learning algorithms. In future work, it will therefore be interesting to apply this idea to other weight regularization approaches such as EWC with a diagonal approximate posterior (Kirkpatrick et al., 2017). Finally, a separate branch of continual learning utilizes replay-like mechanisms to reduce catastrophic forgetting (Cong et al., 2020; Li and Hoiem, 2017; Pan et al., 2020; Shin et al., 2017; Titsias et al., 2020; van de Ven and Tolias, 2018). While our work has focused on weight regularization, such regularization and replay are not mutually exclusive. Instead, these two approaches have been found to further improve robustness to catastrophic forgetting when combined (Nguyen et al., 2017; van de Ven et al., 2020).

Impact and limitations While we have shown that NCL represents an important conceptual and methodological advance for continual learning, it also comes with several limitations. One such limitation arises from the relative difficulty of computing the prior Fisher matrix, which is needed for our projection step. Indeed the success of methods such as Adam (Kingma and Ba, 2014) and EWC (Kirkpatrick et al., 2017) is due in part to their ease of implementation, which facilitates broad applicability. It will therefore be interesting to investigate how approximations such as a running average of a diagonal approximation to the empirical Fisher matrix as used in Adam could facilitate the development of simple yet powerful variants of NCL.

Furthermore, while NCL mitigates the need to overcount the prior from previous tasks via λ as in KFAC, it does introduce two other (largely redundant) hyperparameters in the form of (i) the scale of the prior before the first task, and (ii) the parameter α used to regularize the inversion of the prior Fisher matrix, similar to OWM and DOWM (Duncker et al., 2020; Zeng et al., 2019). While α is an important hyperparameter for OWM and DOWM and we also optimize it in the RNN setting for a more direct comparison (Section 4.1.3), we find it more natural to set this parameter to a constant small value present only for numerical stability (Appendix C). This leaves the prior scale, which we optimize explicitly in the feedforward setting (Section 4.1.3). However, in future work it would be interesting to consider whether a good prior can be determined in a data free manner to make NCL a hyperparameter-free method. Finally, computing the Fisher matrix used for pre-conditioning requires explicit knowledge of task boundaries. In future work, it will therefore be interesting to develop an algorithm similar to NCL that also works for online learning problems with continually changing task distributions.

4.2 Representational stability in biological and artificial circuits

4.2.1 Introduction

Humans and many other animals have the ability to learn many new behaviours over the course of their lives. Despite this flexibility, most of the learned behaviours also remain highly stable after learning, even after long periods without continued practice (Krakauer and Shadmehr, 2006; Melnick, 1971). The stability of such remembered behaviours is seemingly at odds with the high degree of synaptic turnover observed in the brain (Holtmaat and Svoboda, 2009; Xu et al., 2009; Yang et al., 2009).

In agreement with a turnover of dendritic spines and other cellular components, many studies in neuroscience have found that task-associated neural representations drift over timescales of a few hours to days (Carmena et al., 2005; Driscoll et al., 2017; Rokni et al., 2007; Schoonover et al., 2021). Such unstable single-neuron representations have been hypothesized to still facilitate stable behaviour either by restricting drift to a functional 'null-space' (Gallego et al., 2020), or by limiting drift to directions in parameter space that only require minor changes to a downstream neural decoder for stable performance (Rule et al., 2020). However, under this hypothesis, it remains an open question how neural circuits would identify such null or error-limiting directions in which to drift.

On the other hand, a separate set of studies has suggested that neural representations of a given task tend to be stable after learning (Chestek et al., 2007; Dhawale et al., 2017; Flint et al., 2016; Jensen et al., 2022a). This is consistent with the observation of long-term stability in the zebra finch song circuit after initial learning in the juvenile bird (Katlowitz et al., 2018). However, in the context of zebra finch song learning, plasticity is limited to a single 'critical period' during development (Sizemore and Perkel, 2011), and the resulting circuit does not have to adapt to learning new behaviours. In contrast, the brain regions studied in mammals tend to undergo continual learning throughout life and therefore must maintain the ability to learn and adapt in the face of these seemingly stable representations. There are thus contrasting results on representational stability and drift in the neuroscience literature, with the additional complication that different studies have often been carried out with different tasks and methods in different circuits and even organisms. Reconciling these findings or pinpointing the reasons for the observed differences therefore remains an important topic in systems neuroscience.

While the topic of stable neural representations and the apparent paradox with lifelong learning has long been a topic of interest and study in the neuroscience community, it has also recently begun to be addressed in the machine learning literature (Section 4.1). In particular, while biological agents are capable of learning over a lifetime with little to no loss in performance on previously learned tasks, artificial agents often undergo 'catastrophic forgetting', whereby the performance on previous tasks deteriorates rapidly as new tasks are learned. This shortcoming of artificial agents has been addressed using methods ranging from 'replay' of examples from previous tasks (Li and Hoiem, 2017; Pan et al., 2020; Shin et al., 2017; van de Ven and Tolias, 2018) to regularizing parameters important for previous tasks (Kirkpatrick et al., 2017; Nguyen et al., 2017; Ritter et al., 2018), and projecting parameter updates into subspaces that do not interfere with previous tasks (Duncker et al., 2020; Zeng et al., 2019).

In this section, we attempt to relate experimental findings on neural stability to the machine learning literature and consider how qualitatively different approaches to addressing the continual learning problem can lead to different levels of representational stability at the single-neuron level. We also discuss these results in light of experimental findings from different regions of the brain in terms of both the stability of the neural representations and experimental evidence for different mechanisms that might help overcome catastrophic forgetting.

To illustrate the implications of different continual learning algorithms for the stability of neural representations, we use the stroke MNIST task also considered in Section 4.1 (de Jong, 2016). In contrast to most approaches to continual learning, we will work with a recurrent neural network model as in Section 4.1. Continual learning in such recurrent neural networks has recently become a topic of interest in the machine learning community (Duncker et al., 2020; Ehret et al., 2020) and is of significant interest when trying to understand how catastrophic forgetting is mitigated in the brain – a large, noisy network with a high degree of recurrence.

4.2.2 Two classes of continual learning algorithms

We will consider two broad classes of continual learning algorithms, namely those that regularize the *parameters* of the network and those that regularize the *input-output mapping* of the network. The first class of algorithms broadly considers loss functions of the form

$$\mathcal{L}_k(\theta_k) = \ell(\theta_k; \mathcal{D}_k) + d(\theta_k, \theta_{<k}), \tag{4.6}$$

where $\ell(\theta_k; \mathcal{D}_k)$ is the loss on the current task k, and $d(\theta_k, \theta_{<k})$ indicates some divergence between the current parameters (θ_k) and those optimized for previous tasks ($\theta_{<k}$). Notably, this divergence is measured directly in parameter space. In contrast, functional regularization algorithms consider loss functions of the form

$$\mathcal{L}_k(\theta_k) = \ell(\theta_k; \mathcal{D}_k) + d(f_{\theta_k}(\tilde{\mathcal{D}}), f_{\theta_{<k}}(\tilde{\mathcal{D}})). \tag{4.7}$$

Here, the divergence $d()$ is specified in the space of functional mappings *induced* by the parameters θ, as quantified over some dataset $\tilde{\mathcal{D}}$. This functional divergence metric is more directly related to our final objective of performance across all tasks. However, it can be much harder to specify and optimize – especially in a putative biological circuit, where information about the global functional mapping may not be available locally, while local parameter information does exist.

From the set of parameter regularization algorithms, we consider 'natural continual learning' (NCL; Section 4.1; Kao et al., 2021a), which regularizes the loss function on later tasks using the posterior over network parameters from previous tasks as a prior. This is combined with a form of gradient projection that encourages searching for local minima in the space of solutions to previous tasks, yielding the following learning rule on task k (c.f. Section 4.1):

$$\theta \leftarrow \theta - \gamma \left[\mathbf{\Lambda}_{k-1}^{-1} \nabla_\theta \ell_k(\theta) + (\theta - \boldsymbol{\mu}_{k-1}) \right]. \tag{4.8}$$

Here, γ is a learning rate, θ are the network parameters, and $q_\phi(\theta) = \mathcal{N}(\theta; \boldsymbol{\mu}_{k-1}, \mathbf{\Lambda}_{k-1}^{-1})$ is a Laplace approximation to the posterior over θ constructed from tasks 1 to k (see Kao et al., 2021a and Section 4.1 for details).

From the set of functional regularization algorithms, we consider a relatively naive implementation of continual learning using replay. In this method, the learner estimates the task-specific loss $\ell_k(\theta)$ using examples from the current task as above. In addition, the learner gets to 'replay' a set of examples $\{\boldsymbol{x}^{(k')}, \boldsymbol{y}^{(k')}\}$ from previous tasks at every iteration to estimate the expected loss on earlier tasks

$$\ell_{<k}(\theta) = \frac{1}{k-1} \sum_{k'=1}^{k-1} \mathbb{E}\left[\sum_t \log p_\theta(\{\boldsymbol{y}_t^{(k')}\}|\{\boldsymbol{x}_t^{(k')}\})\right]. \tag{4.9}$$

The parameters are then updated as

$$\theta \leftarrow \theta - \gamma \left[\frac{1}{k}\nabla_\theta \ell_k(\theta) + \frac{k-1}{k}\nabla_\theta \ell_{<k}(\theta)\right]. \tag{4.10}$$

Note that while we explicitly replay examples drawn from the true data distribution for previous tasks for simplicity, these examples could instead be drawn from a generative model that is learned in a continual fashion together with the discriminative model (van de Ven et al., 2020; van de Ven and Tolias, 2018).

Task representations

We trained an RNN with 30 recurrent units on 15 sequential binary classification tasks from the extended SMNIST dataset used in Section 4.1. We stored the parameters of the network after each task and simulated the responses of the corresponding networks to a set of 100 digits drawn from the task distribution of each task, similar to Figure 4.4. We then computed the similarity of the neural dynamics at different stages of the sequential learning process ($k \in [1, 15]$) to the dynamics right after task learning. For this analysis, similarity was quantified as the correlation between neural responses to the same set of stimuli at two different points in time, averaged over neurons (c.f. Jensen et al., 2022a).

For networks trained by NCL, we found that neural activity remained largely stable after learning (Figure 4.5). This is consistent with neuroscience studies suggesting a high degree of representational stability after learning a task (Chestek et al., 2007; Dhawale et al., 2017; Flint et al., 2016; Katlowitz et al., 2018). Furthermore, the learning rule employed by NCL regularizes changes in parameters from those used in previous tasks if they are 'important' for those tasks (c.f. Equation 4.8; see also Kirkpatrick et al., 2017). This is mechanistically similar to the observation of a stable subset of dendritic spines following task learning in previous experimental work (Fu et al., 2012; Yang et al., 2009), which lends some support to the implementation of mechanisms resembling parameter regularization for continual learning in biological systems.

When we trained the network using replay instead of NCL, the overall task performance was comparable. However, in contrast to NCL (and other weight regularization approaches such as EWC and KFAC), the networks trained with replay exhibited drifing representations of

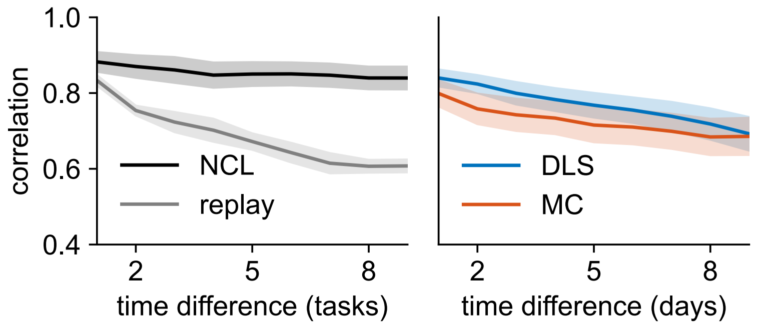

Figure 4.5 **Comparison of computational and biological dynamics. (left)** Similarity of neural dynamics as a function of time difference in artificial networks. We computed the response of the network to a set of input digits from each of the first five SMNIST tasks at different points during training. We then computed the correlation of neural responses as a function of time difference, measured in terms of the number of tasks learned. Lines and shadings indicate mean and standard error across tasks for networks trained with NCL (black) or replay (grey). **(right)** Representational similarity as a function of time difference in a biological circuit described by Jensen et al. (2022a). Some of the drift in neural activity could be attributed to a concomitant drift in behaviour (Jensen et al., 2022a). It is not clear how the passing of biological time in this data relates to task learning in the artificial circuits.

previously acquired tasks. This is consistent with reports of changing neural representations following learning in a different subset of the neuroscience literature (Carmena et al., 2005; Driscoll et al., 2017; Rokni et al., 2007; Schoonover et al., 2021). Additionally, we hypothesize that representational drift under functional regularization will be even more prominent for a noisy optimisation in larger circuits with more degeneracy.

We proceeded to compare these results from artificial networks explicitly to a biological dataset recently described by Jensen et al. (2022a), which quantified the similarity of neural activity over time after learning a motor task (Kawai et al., 2015). In this dataset, neural representations remained qualitatively stable over several weeks of recording (Figure 4.5). Additionally, analyses by Jensen et al. (2022a) suggested that this level of stability is itself likely to be an underestimate given potential confounds of drifting behaviour and other unmeasured processes that we do not account for. Unfortunately, it is difficult to compare the timescale of drift in artificial circuits and biological circuits directly, since artificial networks only see discrete tasks, while biological circuits have to cope with the passing of continuous time and the associated turnover of cellular parameters (Fu et al., 2012; Holtmaat and Svoboda, 2009). It is therefore not clear what amount of task-specific learning in an artificial circuit is the equivalent of a single day of experience in the biological circuit.

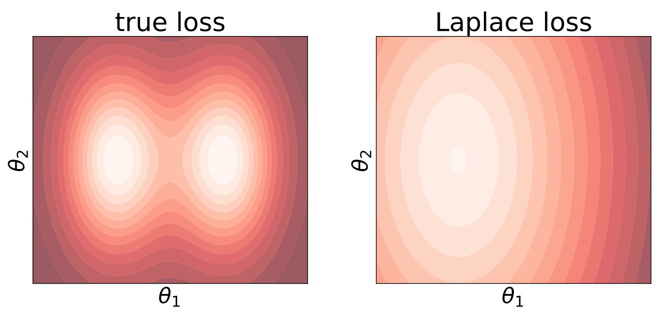

Figure 4.6 **Symmetry breaking in the Laplace approximation.** True loss for a hypothetical 'task 1' (left) and the approximate loss given by the Laplace approximation at one of two degenerate local minima (right).

Local and global loss functions

Returning to the artificial networks, we can understand the qualitative differences in task-associated neural dynamics between networks trained with NCL and replay by considering the locality of the loss function that is optimized. Consider an example loss function, ℓ_1, for a hypothetical 'task 1' with two nearby local minima separated by a small energy barrier (Figure 4.6; left). When proceeding to train on task 2 after learning task 1, the replay-based approach to continual learning (Equation 4.9) provides an unbiased estimate of $\ell_1(\theta)$, although it may be very noisy if the number of replay events is small. This allows θ to move between adjacent local minima with similar task 1 performance, and the rate at which these transitions occur will increase with increasing noise levels. If replay examples are instead drawn from a learned generative model, they are likely to provide a biased estimate of $\tilde{\ell}_1 \approx \ell_1$ but will still allow relatively uninhibited transitions between nearby minima of the approximate loss function. Any such transitions will lead to representational drift, and in the limit of noisy updates in large parameter spaces with many adjacent or even continuous minima, drift is likely to be substantial.

Consider instead the Laplace approximation to $\ell_1(\theta)$ (Figure 4.6; right). In this case, the approximation $\tilde{\ell}_1 \approx \ell_1$ is inherently local and will always favour parameters θ that resemble the prior mean $\boldsymbol{\mu}_1$. This will break degeneracies between otherwise equivalent parameter sets and encourage a constancy of parameters that drives a constancy of dynamics. A simple example of the degeneracy breaking introduced by the Laplace approximation is seen when permuting the identity of all recurrent units using some permutation operator $\hat{P}_u$. If the parameters θ are permuted accordingly, this will lead to a system with the exact same input-output mapping and task performance. However, under the Laplace approximation, this new parameter set $\tilde{\theta} = \hat{P}_u(\boldsymbol{\mu})$ will have a much higher loss than the original parameters $\boldsymbol{\mu}$ given the $||\theta - \boldsymbol{\mu}||_2^2$ term of the Laplace loss function (Section 2.4). While we have discussed such degeneracy breaking specifically for the Laplace approximation here, the same is also true for other local approximations that restrict changes from the parameters used in previous tasks or project gradients based on the parameters or dynamics observed in previous tasks.

4.2.3 Discussion

In this section, we have highlighted how different continual learning algorithms can lead to qualitatively different properties of neural dynamics in recurrent networks over the course of learning. In particular, we have argued that continual learning algorithms based on local parameter regularization or projection matrices constructed from previous tasks will tend to preserve neural dynamics and lead to stable representations even as further tasks are learned. In contrast, algorithms that regularize the input-output mapping, e.g. via exact or generative replay, are susceptible to representational drift over the course of further learning.

It is interesting to speculate whether such algorithmic differences may explain some of the discrepancies in experimental reports of neural stability after task learning. Notably, several reports suggesting stable neural representations have centered around motor circuits (Chestek et al., 2007; Dhawale et al., 2017; Flint et al., 2016; Gallego et al., 2020; Jensen et al., 2022a), where previous work has also suggested that representations stabilize over the course of learning (Ganguly and Carmena, 2009; Peters et al., 2014). This is consistent with our results for parameter regularized networks (Figure 4.5), which restrict how much particular connections in the network can change depending on their importance for prior tasks. Importantly, experimental evidence for such regularization of specific connections has previously been reported in rodent motor cortex (Fu et al., 2012; Yang et al., 2009), suggesting a potential mechanistic explanation for the observed representational stability.

Conversely, neural representations have been reported to drift more rapidly in several higher brain regions, including hippocampus (Ziv et al., 2013) and posterior parietal cortex (Driscoll et al., 2017; Rule et al., 2020). This is consistent with the important role of hippocampal replay in memory consolidation, which is well-established in the neuroscience literature (Carr et al., 2011; van de Ven et al., 2016; Wilson and McNaughton, 1994). Additionally, replay-like events have been observed in neocortical regions including posterior parietal cortex (Qin et al., 1997) and prefrontal cortex (Shin et al., 2019). It is possible that such neural replay of past experiences provides a substrate for continual learning in these brain regions, which obviates the need for stability at the level of single synapses and allows for rapidly drifting representations.

Finally, it is worth noting the difficulty of directly comparing rates of drift in biological and artificial circuits as highlighted in Figure 4.5. One reason for this is that the artificial setting is 'perfectly controlled' in the sense that there are no unaccounted behavioural or latent processes that might affect the apparent stability of neural dynamics (Jensen et al., 2022a). Additionally, it is difficult directly to compare the passing of time in these two cases. For example, it is unclear how many 'tasks' in a machine learning setting should be learned in the equivalent of a day or a week of biological time passing. Indeed in the biological setting, it is often artificial to only speak of specific task learning in the first place, since it can be argued that animals are always learning new associations and behaviours, and it will be difficult to quantify this ‘learning’ exactly. Biological time is thus not a discrete quantity of ‘learning a task’ or ‘not learning a task’, but rather a continuum of noise and plasticity in circuits with different degrees of relevance for different behaviours performed for different durations. Additionally, while we have considered a binary distinction between ‘parameter regularization’ and ‘functional regularization’, it is also plausible that some combination of mechanisms is used to preserve memories in biological circuits, similar to recent work in the machine learning literature (Nguyen et al., 2017; van de Ven et al., 2020).

For these reasons, we do not suggest a direct comparison between the timescale of drift in biological and artificial circuits, which is unlikely to be meaningful. Instead, we propose that thinking about the consequences of different algorithms for observed neural dynamics has the potential to shed light on differences between brain regions, where principled comparisons can be made using a fixed set of experimental paradigms and analysis methods. While it may still seem overly simplistic to relate such differences in stability to putative algorithmic differences, we believe that trying to relate our experimental data to tangible computational models is a fruitful approach for making sense of the growing datasets recorded in modern systems neuroscience. For this reason, we hope that strong models of continual learning may help shed light on the literature on representational drift.

Chapter 5

Reinforcement learning to plan

In the previous chapter, we saw how continual learning formulated as Bayesian inference can provide a strong model of representational stability and drift in biological circuits. In this chapter, we develop another strong model of planning and decision making, formulated in the language of reinforcement learning (Section 2.5). This model is inspired by recent work on meta-reinforcement learning by Wang et al. (2018) and captures the ability of humans to trade off actions for time spent thinking when encountering new tasks and task settings. The 'outer loop' of this reinforcement learning model can be formulated as inference in policy space, treating the model parameters as variational parameters (Section 2.5). However, an important new feature is that the model also learns to do inference in the space of hidden network states as a form of 'planning' driven by policy rollouts. This provides a strong model of planning and decision making in humans and other animals and sheds further light on a range of experimental findings in the hippocampal replay literature.

This chapter is currently under review and is available as a preprint (Jensen et al., 2023).

5.1 Introduction

Humans and other mammals have a unique ability to adapt rapidly to new information and changing environments. Such adaptation often involves spending extended and variable periods of time contemplating possible futures before taking an action (Callaway et al., 2022; van Opheusden et al., 2021). For example, we might take a moment to think about which route to take to work depending on traffic conditions. The next day, some roads might be blocked due to roadworks, requiring us to adapt and mentally review the available routes in a process of re-planning before leaving the house. Since thinking does not involve the acquisition of new information or interactions with the environment, it is perhaps surprising that it is so ubiquitous for human decision making. However, thinking allows us to perform more computations with the available information, which can lead to improved performance on downstream tasks (Bansal et al., 2022). Since physically interacting with the environment can consume time and other resources, or incur unnecessary risk, the benefits of planning often more than make up for the time that was lost to the planning process itself.

Despite a wealth of cognitive science research on the algorithmic underpinnings of planning (Callaway et al., 2022; Mattar and Daw, 2018; Mattar and Lengyel, 2022; Solway and Botvinick, 2012), little is known about the underlying neural mechanisms. This question has been difficult to address due to a scarcity of intracortical recordings during planning, and during contextual adaptation more generally. However, neuroscientists have begun to collect large-scale neural recordings during increasingly complex behaviors from the hippocampus and prefrontal cortex, brain regions known to be important for memory, decision making, and adaptation (Gillespie et al., 2021; Jadhav et al., 2016; Pfeiffer and Foster, 2013; Samborska et al., 2022; Wang et al., 2018; Widloski and Foster, 2022; Wu et al., 2017b). These studies have demonstrated the importance of prefrontal cortex for generalizing abstract task structure across contexts (Samborska et al., 2022; Wang et al., 2018). Additionally, it has been suggested that planning could be mediated by the process of hippocampal forward replays (Agrawal et al., 2022; Foster, 2017; Jiang et al., 2022; Johnson and Redish, 2007; Mattar and Daw, 2018; Pfeiffer and Foster, 2013; Widloski and Foster, 2022). Despite these preliminary theories, little is known about how hippocampal replays could be integrated within the dynamics of downstream circuits to implement planning-based decision making and facilitate adaptive behavior (Yu and Frank, 2015). While prevailing theories of learning from replays generally rely on dopamine-mediated synaptic plasticity (De Lavilléon et al., 2015; Gomperts et al., 2015; Mattar and Daw, 2018), it is currently unclear whether this process could operate sufficiently fast to also inform online decision making.

It has recently been suggested that some forms of fast adaptation could result from recurrent meta-reinforcement learning (meta-RL; Duan et al., 2016; Wang et al., 2018, 2016). Such meta-RL models posit that adaptation to new tasks can be directly implemented by the recurrent dynamics of the prefrontal network. The dynamics themselves are learned through gradual changes in synaptic weights, which are modified over many different environments and tasks in a slow process of reinforcement learning. Importantly, such recurrent neural network (RNN)-based agents are able to adapt rapidly to a new task or environment after training by integrating their experiences into the hidden state of the RNN, with no additional synaptic changes (Alver and Precup, 2021; Duan et al., 2016; Wang et al., 2018, 2016; Zintgraf et al., 2019). However, previous models are generally only capable of making *instantaneous* decisions and thus do not have the ability to improve their choices by 'thinking' prior to taking an action. Wang et al. (2018) explored the possibility of allowing multiple steps of network dynamics before making a decision, but this additional computation was also pre-determined by the experimenter and not adaptively modulated by the agent itself.

In this work, we propose a model that similarly combines slow synaptic learning with fast adaptation through recurrent dynamics in the prefrontal network. In contrast to previous work, however, this recurrent meta-learner can *choose* to momentarily forgo physical interactions with the environment and instead 'think' (Hamrick et al., 2017; Pascanu et al., 2017). This process of thinking is formalized as the simulation of sequences of imagined actions, sampled from the policy of the agent itself, which we refer to as 'rollouts' (Figure 5.1A). We introduce a flexible

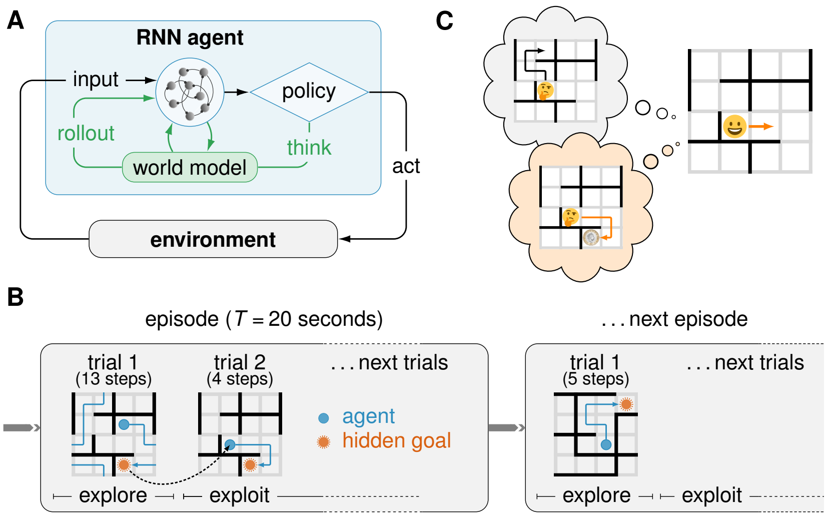

Figure 5.1 **Task and model schematics. (A)** The RL agent consisted of a recurrent neural network, which received information about the environment and executed actions in response. The primary output of the agent was a policy from which the next action was sampled. This action could either be to move in the environment in a given direction (up, down, left or right), or to 'plan' by using an internal world model to simulate a possible future trajectory (a 'rollout'). The agent was trained to maximize its average reward per episode and to predict (i) the upcoming state, (ii) the current goal location, and (iii) the value of the current state. When the agent decided to plan, the first two predictors were used in an open-loop planning process, where the agent iteratively sampled 'imagined' actions and predicted what the resulting state would be, and whether the goal had been (virtually) reached. The output of this planning process was appended to the agent's input on the subsequent time step (details in text). A physical action was assumed to take 400 ms and a rollout was assumed to take 120 ms (Kurth-Nelson et al., 2016). **(B)** Schematic illustrating the dynamic maze task. In each episode lasting $T = 20$ seconds, a maze and a goal location were randomly sampled. Each time the goal was reached, the subject received a reward and was subsequently "teleported" to a new random location, from which it could return to the goal to receive more reward. The maze had periodic boundaries, meaning that subjects could exit one side of the maze to appear at the opposite side. **(C)** Schematic illustrating how policy rollouts can improve performance by altering the momentary policy. An agent might perform a policy rollout leading to low value (top; black), which would *decrease* the probability of physically performing the corresponding sequence of actions. Conversely, a rollout leading to high value (bottom; orange) would *increase* the probability of the corresponding action sequence. Notably, these policy changes occur at the level of network dynamics rather than parameter updates.

maze navigation task to study the relationship between the behavior of such RL agents and that of humans (Figure 5.1B). In this task, both human participants and RL agents (collectively 'subjects') have to discover the spatial location of an unknown goal in a novel environment, and they subsequently have to return to this goal from multiple different starting locations (Banino et al., 2018; Morris, 1981). Intriguingly, RL agents trained on this task learn to use rollouts to improve their policy and better generalize to previously unseen environments, and they selectively trigger rollouts in situations where humans also spend more time deliberating.

Additionally, we draw explicit parallels between the model rollouts and hippocampal replays through novel analyses of recent hippocampal recordings from rats performing a similar maze task (Widloski and Foster, 2022). We find that the content and behavioral effects of hippocampal replays in this dataset have a striking resemblance to the content and effects of policy rollouts in our computational model. Our work thus addresses two key questions from previous studies on hippocampal replays and planning. First, we show that a recurrent network can meta-learn when to plan instead of having to precompute a 'plan' in order to decide whether to use it (Mattar and Daw, 2018; Russek et al., 2022). Second, we propose a new theory of replay-mediated planning, which utilizes fast network dynamics for real-time decision making that could operate in parallel to slower synaptic plasticity (Gomperts et al., 2015). To formalize this second point, we provide a normative mathematical theory of how replays can improve decision making via feedback to prefrontal cortex by approximating policy gradient optimization (Sutton and Barto, 2018). We show that such an optimization process naturally arises in our RL agent trained for rapid adaptation and suggest that biological replays could implement a similar process of rollout-driven decision making (Figure 5.1C).

Our work provides new insights into the neural underpinning of 'thinking' by bridging the gap between recurrent meta-RL (Wang et al., 2018), machine learning research on adaptive computation (Banino et al., 2021; Graves, 2016; Hamrick et al., 2017), and theories of meta-cognition (Botvinick et al., 2020; Botvinick and Cohen, 2014; Griffiths et al., 2015). We link these ideas to the phenomenon of hippocampal replays and provide a new theory of how forward replays can modulate behavior through recurrent interactions with prefrontal cortex.

5.2 Results

5.2.1 Humans think for different durations in different contexts

To characterize the behavioral signatures of planning, we recruited 94 human participants from Prolific to perform an online experiment. The experiment consisted of a maze navigation task in which the walls and goal location periodically changed, thus requiring rapid adaptation. The environment was a 4×4 grid with periodic boundaries, a set of impassable walls, and a single hidden reward location (Figure 5.1B; Appendix D.2). The task consisted of a succession of 'episodes', each lasting $T = 20$ seconds. At the beginning of each episode, both the wall configuration and the reward location were randomly initialized and remained fixed until the next episode. The initial position of the subject was also randomly sampled. Subjects first had to explore the maze by taking discrete steps in the cardinal directions until they found the hidden reward location. Upon finding this goal, subjects were immediately moved to a new random location, initiating a phase of exploitation during which they repeatedly had to return to the same goal (Figure 5.1B). We refer to a single instance of navigating from a random starting location to the goal as a 'trial'. To encourage good performance, human participants were paid a monetary bonus proportional to the average number of trials completed per episode (Appendix D.2; Figure D.1), and the behavior of all subjects was recorded over 40 episodes.

We first examined human performance as a function of trial number within each episode, comparing the first exploration trial with subsequent exploitation trials. We found that participants exhibited a rapid 'one-shot' transition to goal-directed navigation after the initial exploration phase (Figure 5.2A, black). This was true even though each new maze was not seen before, and it is consistent with previous work demonstrating the ability of humans and animals to adapt rapidly to new information in a 'meta-learning' setting (Wang et al., 2018). We next investigated the time participants spent thinking during the exploitation phase. We estimated the 'thinking time' for each action as the posterior mean under a probabilistic model that decomposes the total response time for each action (Figure 5.2B; top) into the sum of the thinking time (Figure 5.2B; bottom) and a perception-action delay. The prior distribution over perception-action delays was estimated for each individual using a separate set of trials, where participants were explicitly cued with the optimal path and thus did not have to plan a route themselves (Appendix D.2; Figure D.1). Since the first step within each trial required participants to parse their new position in the maze, a separate prior was fitted for the first action in a trial.

Participants exhibited a wide distribution of thinking times during the exploitation phase of the task (Figure 5.2B; bottom). To reveal any task-related structure in this variability, we partitioned thinking times by within-trial action number and by distance to goal (Figure 5.2C). We found that participants exhibited longer thinking times when further from the goal, consistent with planning of longer routes taking more time. Furthermore, subjects exhibited substantially longer thinking times for the first action of each trial (Figure D.2), consistent with them having to initially plan a new route to the goal. These patterns confirm that the broad marginal distribution of thinking times (Figure 5.2B) does not simply reflect a noisy decision making process or task-irrelevant distractions. On the contrary, variability in thinking time is an important feature of human behavior that reflects the variable moment-to-moment cognitive demands for decision making.

5.2.2 A recurrent network model of planning

To model the rapid adaptation and the detailed patterns of thinking times displayed by human subjects, we considered an RNN model trained in a meta-reinforcement learning setting (Figure 5.1A; Appendix D.2; Duan et al., 2016; Wang et al., 2018, 2016). The RL agent consisted of 100 gated recurrent units (GRUs; Cho et al., 2014; Figure D.3) and was characterized by a time-varying internal activation state $\boldsymbol{h}_k$, which evolved dynamically according to

$$\boldsymbol{h}_k = \phi_\theta(\boldsymbol{x}_k, \boldsymbol{h}_{k-1}) \tag{5.1}$$

$$\boldsymbol{y}_k = \zeta_\theta(\boldsymbol{h}_k). \tag{5.2}$$

Here, θ denotes the set of all model parameters, $\boldsymbol{x}_k$ are momentary inputs to the RNN, and $\boldsymbol{y}_k$ are momentary network outputs computed from the current state $\boldsymbol{h}_k$, which was reset at the beginning of each episode. k indexes the evolution of the network dynamics and can in general be *different* from the wallclock time t in agents that have the ability to 'think' for variable periods of time (see below). Inputs consisted of the current agent location $\boldsymbol{s}_k$, the previous action taken a_{k-1} and associated reward signal r_{k-1}, the elapsed time t since the beginning of the episode, and the locations of all walls (Appendix D.2). Thus, while the reward location was hidden and had to be both discovered and memorized, the rest of the environment was fully observed. Outputs consisted primarily of a policy $\pi_\theta(a_k|\boldsymbol{h}_k)$, i.e. a set of probabilities associated with each possible action, which depended on the current hidden state of the RNN. At each step, an action a_k was sampled from this distribution and triggered changes in the environment ψ according to:

$$\boldsymbol{x}_{k+1}, \boldsymbol{s}_{k+1} = \psi(a_k, \boldsymbol{s}_k). \tag{5.3}$$

This yielded both a new location $\boldsymbol{s}_{k+1}$ of the agent and the new inputs $\boldsymbol{x}_{k+1}$, which were fed back to the agent on the subsequent iteration (Figure 5.1A). In addition to the policy, the output of the agent included a value function and predictions of the new location and current goal location.

As in standard RL settings, we quantified the performance of the agent in a given environment as the expected total reward,

$$J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{k=1}^{K} r_k\right]. \tag{5.4}$$

Training proceeded by gradually adjusting the parameters θ to maximize the average $J(\theta)$ *across environments*, using a policy gradient algorithm (Appendix D.2; Sutton and Barto, 2018; Wang et al., 2018). In Equation 5.4, K refers to the total number of iterations in an episode, with each episode terminating once t exceeded the episode duration of $T = 20$ seconds as in the human data (Figure 5.1B). Since our agent had no intrinsic notion of wallclock time, we considered each discrete action to consume $\Delta t = 400$ ms, meaning that there was time for 50 actions in a single RL episode. This was calculated to approximately match the number of actions taken in a typical RL episode to the human data. In this canonical formulation, the RL agent always takes an instantaneous action in response to a given set of inputs. It therefore does not have any ability to perform temporally extended planning, implying constant (zero) 'thinking time' in all situations. As a consequence, such a canonical meta-RL agent cannot explain the salient patterns of thinking times observed in human participants (recall Figure 5.2C). At first glance, temporally extended planning might also appear *unnecessary* in the RL agent, since it already has access to the current state, wall configuration, and reward information needed for decision making. However, this was also the case for our human participants, who chose to spend time thinking nonetheless. We hypothesized that the RL agent could similarly benefit from the ability to trade off time for additional *processing* of the available information in difficult tasks, where the agent has not learned a perfect policy (Hamrick et al., 2017; Pascanu et al., 2017).

To investigate the effect of such thinking for recurrent meta learners and account for the observed variability in human thinking times, we augmented the RL agent with the ability to perform temporally extended planning in the form of imagined policy rollouts. Specifically, we expanded the action space of the agent to give it the option of sampling a hypothetical trajectory from its own policy at any moment in time (a 'rollout'; Figure 5.1A; Hamrick et al., 2017; Pascanu et al., 2017). In other words, the agent was allowed to either perform a physical action, or to perform a mental simulation of its policy. If the agent chose to perform a rollout, a flattened array of the imagined action sequence was fed back to the network as additional inputs on the subsequent time step, together with an indication of whether or not the simulated action sequence reached the goal. These inputs in turn affected the policy by modulating $\boldsymbol{h}_k$ through a set of learnable input weights (Figure 5.1A). This is reminiscent of canonical RL algorithms that change their *parameters* θ to yield a new and improved policy on the basis of trajectories sampled from the current policy. In our formulation of planning, the agent's policy is instead induced by the *hidden state* $\boldsymbol{h}_k$, which can similarly be modulated on the basis of the imagined policy rollouts to improve performance.

Each rollout was terminated either upon reaching the goal, or after a maximum duration of 8 simulated actions (see Figure D.3 for different network sizes and maximum planning horizons). Importantly, both the generation of a mental rollout and the corresponding success feedback relied on an *internal* model of the environment that was obtained from the agent itself. This internal model was trained alongside the RNN and the policy, by learning to predict the reward location and state transitions from the momentary hidden state of the RNN ($\boldsymbol{h}_k$) and the action taken (a_k; Appendix D.2; Figure D.4). Thus, rollouts did not provide the agent with any privileged information that it did not already possess. Instead, they allowed the agent to trade off time for additional computational capacity – similar to thinking in humans and other animals. Furthermore, to capture the fact that mental simulation is faster than physical actions (Kurth-Nelson et al., 2016; Liu et al., 2019), we assumed each full rollout to consume only 120 ms. In other words, a single iteration of the network dynamics ($k \rightarrow k+1$ in Equation 5.1) incremented time by 120 ms if the agent chose to perform a rollout and 400 ms if the agent chose a physical action. This allowed the agent to perform many simulated actions in the time it would take to physically move only a short distance (Agrawal et al., 2022). Importantly,

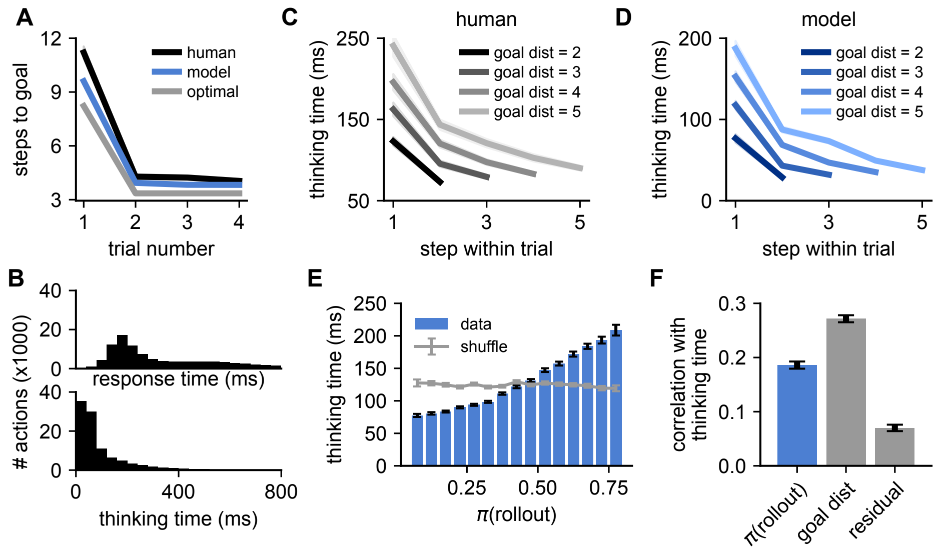

Figure 5.2 **Trained RL agents perform more rollouts in situations where humans spend longer thinking.** **(A)** Performance – quantified as the number of actions needed to reach the goal – as a function of trial number within each episode, computed for both human participants (black) and RL agents (blue). Shading indicates standard error of the mean across human participants ($n = 94$) or RL agents ($n = 5$) and mostly falls within the interval covered by the solid lines. Gray line indicates optimal performance, computed separately for exploration (trial 1) and exploitation (trials 2-4; Appendix D.2). **(B)** Distribution of human response times (top) and thinking times (bottom), spanning ranges on the order of a second (Appendix D.2). **(C)** Human thinking time as a function of the step-within-trial (x-axis) for different initial distances to the goal at the beginning of the trial (lines, legend). Shading indicates standard error of the mean across 94 participants. Participants spent more time thinking further from the goal and before the first action of each trial (Figure D.2). **(D)** Model 'thinking times' separated by time-within-trial and distance-to-goal, exhibiting a similar pattern to human participants. To compute thinking times for the model, each rollout was assumed to last 120 ms as described in the main text. Shading indicates standard error of the mean across 5 RL agents. **(E)** Binned human thinking time as a function of the probability that the agent chooses to perform a rollout, π(rollout). Error bars indicate standard error of the mean within each bin. Gray horizontal line indicates a shuffled control, where human thinking times were randomly permuted before the analysis. **(F)** Correlation between human thinking time and the regressors (i) π(rollout) under the model, (ii) distance-to-goal, and (iii) π(rollout) after conditioning on distance-to-goal ('residual'; Appendix D.2). Bars and error bars indicate mean and standard error across human participants ($n = 94$).

since an episode had a fixed duration of 20 seconds, choosing to perform more rollouts had a temporal opportunity cost by leaving less time for physical actions towards the goal.

Biologically, we interpret these mental simulations as prefrontal cortex (the RNN) interacting with the hippocampal formation (the world model), which allows the agent to simulate a sequence of state transitions from the current policy and evaluate their consequences. Importantly, while we endowed the agent with the ability to perform policy rollouts, we did not build in any prior knowledge about when, how, or how much they should be used. The agent instead had to learn this over the course of training on many different environments. Therefore, while the rollouts phenomenologically resembled hippocampal forward replays *by design*, our computational model allowed us to investigate (i) whether and how such rollouts can drive policy improvements, (ii) whether their temporal patterns can explain human response times, and (iii) whether biological replays appear to be implementing a similar computation.

The RL agent was trained by slowly adjusting its parameters θ over 8×10^6 episodes, sampled randomly from 2.7×10^8 possible environment configurations. This implied that the majority of environments seen at test time would be novel to the agent, requiring generalization across tasks. Parameter adjustments followed the gradient of a cost function that combined terms designed to (i) maximize the expected reward in Equation 5.4, (ii) learn the internal model by accurately predicting the reward location and state transitions, and (iii) minimize a standard entropy cost to encourage exploration (Appendix D.2; Wang et al., 2016). Importantly, parameters were frozen after training, and the agent adapted to the wall configuration and goal location of each new environment using only internal network dynamics (Duan et al., 2016; Wang et al., 2018).

5.2.3 Human thinking times correlate with agent rollouts

Having specified our computational model of planning, we analyzed its behavior and compared it to that exhibited by humans. We trained 5 copies of our RL agent to solve the same task as the human participants and found that the agents robustly learned to navigate the changing maze. Similar to humans, the trained agents exhibited a rapid transition from exploration to exploitation upon finding the reward, reaching near-optimal performance in both phases (Figure 5.2A, blue). This confirmed that these RNNs are capable of adapting to changing environments using only internal network dynamics with fixed parameters, corroborating previous work on recurrent meta-RL (Banino et al., 2018; Duan et al., 2016; Wang et al., 2018). The trained networks also used their capacity to perform rollouts, choosing to do so approximately 30% of the time. Importantly, there was temporal variability in the probability of performing a rollout, and the networks sometimes performed multiple successive rollouts between consecutive physical actions. When we queried the conditions under which the trained agents performed these rollouts, we found striking similarities with the pattern of human thinking times observed previously. In particular, the RL agent performed more rollouts earlier in a trial and further from the goal (Figure 5.2D) – situations where the human participants also spent more time thinking before taking an action (Figure 5.2C). On average, thinking times in the RL agent were approximately 50 ms lower than in humans. This difference could e.g. be due to (i) the choice of prior in the probabilistic model used to infer human thinking times, (ii) the agent having a better 'base policy' than humans, or (iii) the hyperparameters determining the temporal cost of planning.

To further study the relationship between rollouts and human 'thinking', we simulated the RL agent in the same environments as the human participants. We did this by clamping the physical actions of the agent to those taken by the participants, while still allowing it to sample on-policy rollouts (Appendix D.2). In this setting, the agent's probability of choosing to perform a rollout when encountering a new state, $\pi(\text{rollout})$, was a monotonically increasing function of human thinking time in the same situation (Figure 5.2E). The Pearson correlation between these two quantities was $r = 0.186 \pm 0.007$ (mean $\pm$ sem across participants), which was significantly higher than expected by chance (Figure 5.2F, '$\pi(\text{rollout})$'; chance level $r = 0 \pm 0.004$). An above-chance correlation between thinking times and $\pi(\text{rollout})$ of $r = 0.070 \pm 0.006$ persisted after conditioning on the distance-to-goal (Figure 5.2F, 'residual'), which was also correlated with thinking times ($r = 0.272 \pm 0.006$). The similarity between planning in humans and RL agents thus extends beyond this salient feature of the task, including an increased tendency to plan on the first step of a trial (Figure D.2).

In addition to the similarities during the *exploitation* phase, a significant correlation was also observed between human thinking time and $\pi(\text{rollout})$ during *exploration* ($r = 0.098 \pm 0.008$). In this phase, both humans and RL agents spent more time thinking during later stages of exploration (Figure D.5). Model rollouts during exploration corresponded to planning towards an imagined goal from the posterior over goal locations, which becomes narrower as more states are explored (Figure D.5). This finding suggests that humans may similarly engage in increasingly goal-directed behavior as the goal posterior becomes narrower over the course of exploration. Taken together, our results show that a meta-reinforcement learning agent, endowed with the ability to perform rollouts, learns to do so in situations similar to when humans appear to plan. This provides a putative normative explanation for the variability in human thinking times observed in the dynamic maze task.

Participants also exhibited a bias against moving through the periodic boundaries of the arena, with human participants at a boundary being 1.84 ± 0.03 times as likely to move into the arena as they were to move through the boundary (mean $\pm$ standard error across participants). To capture this bias in our computational model, we replaced the standard policy entropy regularization with a KL regularization term that discouraged moving through the boundaries, which can be interpreted as a form of 'prior over actions' (Section 2.5, Appendix D.2). Such a prior will lead to the presence of a similar bias against moving through boundaries in the model behavior *by design*. Interestingly, introducing this bias also resulted in a stronger residual correlation between human thinking times and model thinking times after regressing out the effect of distance-to-goal, and it led to a stronger correlation between model and human thinking times during exploration (Figure D.6). We interpret this as the introduction of an action prior leading to a different notion of what constitutes a 'difficult' or 'far away' state for a given arena, which is more consistent with the human notion of what constitutes a difficult state. This suggests that humans solving sequential decision making tasks may make use of similar priors learned from previous tasks, which can lead to particular biases and structured reaction times when encountering a new task.

5.2.4 Rollouts improve the policy of the RL agent

In the previous section, we saw that an RL agent can learn to use policy rollouts as part of its decision making process, and that the timing and number of rollouts correlates with variability in human thinking times. In this section, we aim to understand *why* the agent chooses to perform rollouts and *how* they guide its behavior. To do this, we considered the agent right after it first located the goal in each episode (i.e., at the first time step of trial 2; Figure 5.1B) and forced it to perform a pre-defined number of rollouts, which we varied. We then quantified the number of actions that the agent took to return to the goal while preventing any further rollouts during this return phase (Appendix D.2).

The average number of actions needed to reach the goal decreased monotonically as the number of forced rollouts increased up to at least 15 rollouts (Figure 5.3A). Interestingly, this was the case despite the unperturbed behavior of the agent rarely including more than a few consecutive rollouts (Figure 5.2D), suggesting that the agent learned a robust algorithm for policy optimization on the basis of such rollouts (Hamrick et al., 2017; Schrittwieser et al., 2020). The increase in performance with rollout number was also associated with a concomitant decrease in policy entropy (Figure 5.3B). Thus, performing more rollouts both improved performance and increased the agent's confidence in its actions (Appendix D.2). These findings confirm that the agent successfully learned to use policy rollouts to optimize its future behavior. However, the question remains of whether this policy improvement is appropriately balanced with the temporal opportunity cost of performing a rollout.

In general, performing a rollout is beneficial in situations where the policy improvement resulting from the rollout is greater than the temporal cost of 120 ms of performing the rollout. To investigate whether the agent learned to trade off the cost and benefit of rollouts (Agrawal et al., 2022; Hamrick et al., 2017; Pascanu et al., 2017), we computed the performance of the agent in a surrogate environment where rollouts were not allowed. In this setting, each action was instead sampled from the distribution over physical actions only (Appendix D.2). When preventing rollouts in this way, the agent only collected 6.54 ± 0.11 rewards per episode compared to 7.54 ± 0.03 in the presence of rollouts (mean $\pm$ standard error across agents), confirming that it used rollouts to increase expected reward (Figure 5.3C). To investigate whether the temporal structure of rollouts described in Figure 5.2 was important for this performance improvement, we performed an additional control, where the number of rollouts was kept fixed for each environment, but their occurrence was randomized in time. In this case,

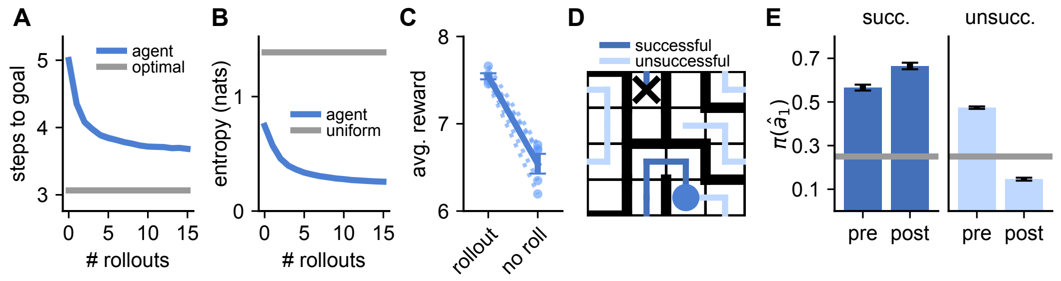

Figure 5.3 **Rollouts improve the network policy. (A)** Performance on trial 2 as a function of the number of rollouts enforced at the beginning of the trial. Performance was quantified as the average number of steps needed to reach the goal in the absence of further rollouts. Gray horizontal line indicates optimal performance. **(B)** Policy entropy as a function of the number of rollouts enforced at the beginning of trial 2. The entropy was computed after re-normalizing the policy over the four 'physical' actions, and the horizontal gray line indicates the entropy of a uniform policy. **(C)** Original performance of the RL agent (left) and its performance when re-normalizing the policy over physical actions to prevent any rollouts (right). Performance was quantified as the average number of rewards collected per episode, and dashed lines indicate individual RL agents, while the solid line indicates mean and standard error across agents. **(D)** Schematic showing an example of a 'successful' (dark blue) and an 'unsuccessful' (light blue) rollout from the same physical location (blue circle). Black cross indicates the goal location (not visible to the agent or human participants). **(E)** Probability of taking the first simulated action of the rollout, $\hat{a}_1$, before ($\pi^{\text{pre}}(\hat{a}_1)$) and after ($\pi^{\text{post}}(\hat{a}_1)$) the rollout. This was evaluated separately for successful (left) and unsuccessful (right) rollouts. $\pi^{\text{pre}}(\hat{a}_1)$ was above chance (gray line) in both cases and increased for successful rollouts, while it decreased for unsuccessful rollouts. Error bars represent standard error across five independently trained agents. The magnitude of the change in $\pi(\hat{a}_1)$ for successful and unsuccessful rollouts depended on the planning horizon (Figure D.3).

performance dropped to 6.75 ± 0.04 rewards per episode, confirming that the RL agent chose to use rollouts specifically when they improved performance.

To further dissect the effect of rollouts on agent behavior, we classified each rollout, $\hat{\tau}$ (a sequence $\{\hat{a}_1, \hat{a}_2, \ldots\}$ of rolled-out actions), as being either 'successful' if it reached the goal according to the agent's internal world model, or 'unsuccessful' if it did not (Figure 5.3D). We hypothesized that the policy improvement observed in Figure 5.3A could arise from *upregulating* the probability of following a successful rollout and *downregulating* the probability of following an unsuccessful rollout. To test this hypothesis, we enforced a single rollout after the agent first found the reward and analyzed the effect of this rollout on the policy, separating the analysis by successful and unsuccessful rollouts. Importantly, we could compare the causal effect of rollout success by matching the history of the agent and performing rejection sampling from the rollout process until either a successful or an unsuccessful rollout had occurred (Appendix D.2). Specifically, we asked how the rollout affected the probability of taking the first rolled-out action, $\hat{a}_1$, by comparing the value of this probability before ($\pi^{\text{pre}}(\hat{a}_1)$) and after ($\pi^{\text{post}}(\hat{a}_1)$) the rollout. $\pi^{\text{pre}}(\hat{a}_1)$ was slightly higher for successful rollouts than unsuccessful rollouts, with both types of rollouts exhibiting a substantially higher-than-chance probability – a consequence of the model rollouts being drawn 'on-policy' (Figure 5.3E). However, while successful rollouts *increased* $\pi(\hat{a}_1)$, unsuccessful rollouts *decreased* $\pi(\hat{a}_1)$ (Figure 5.3E). This finding demonstrates that the agent combines the spatial information of a rollout with knowledge about its consequences, based on its internal world model, to guide future behavior.

5.2.5 Hippocampal replays resemble policy rollouts

In our computational model, we designed policy rollouts to take the form of spatial trajectories that the agent could subsequently follow, and to occur only when the agent was stationary. These two properties are also important signatures of forward hippocampal replays – patterns of neural activity observed using electrophysiological recordings from rodents during spatial navigation (Gillespie et al., 2021; Pfeiffer and Foster, 2013; Widloski and Foster, 2022). Our model therefore allowed us to investigate whether forward replay in biological agents serve a similar function during decision making to the function of policy rollouts in our RL agent. Additionally, since we have direct access to the agent's policy and how it changes after a replay, our computational model can provide insights into the apparently conflicting data and contradictory viewpoints in the literature regarding the role of hippocampal replays. In particular, some studies have found a significant correlation between forward replay and subsequent behavior (Foster, 2017; Pfeiffer and Foster, 2013; Widloski and Foster, 2022), arguing that such a correlation suggests a role of forward replay for planning. On the contrary, other studies have found that forward replays do not always resemble subsequent behavior (Gillespie et al., 2021; Krause and Drugowitsch, 2022; Wu et al., 2017b), challenging the interpretation of forward replay as a form of planning. Our model offers a potentially conciliatory explanation, predicting that the correlation between forward replay and subsequent behavior can be positive or negative, depending on the replayed trajectory (Figure 5.3E; Yu and Frank, 2015; Antonov et al., 2022).

To investigate whether there is evidence for such replay-based modulation of animal behavior, we re-analyzed a recently published hippocampal dataset from rats navigating a dynamic maze very similar to the task in Figure 5.1B (Widloski and Foster, 2022). Our goal was to compare the recorded replay events to the policy rollouts exhibited by the RL agent, considering both the statistical properties of the replays themselves and how they relate to subsequent behavior. In this rodent experiment, animals had to repeatedly return to an initially unknown 'home' location, akin to the goal in our task (Figure D.7). Both this home location and the configuration of the maze changed between sessions. Whilst the behaving animals could not be 'teleported' between trials as in our task, rats instead had to navigate to an unknown rewarded 'away' location selected at random after each 'home' trial. These 'away' trials served as a useful control since the animals did not know the location of the rewarded well at the beginning of the trial.

We studied replay events detected in hippocampal recordings made with tetrode drives during the maze task ($n \in [187, 333]$ simultaneously recorded neurons per session; Figure D.7C). To detect replays, we followed Widloski and Foster (2022) and first trained a Bayesian decoder to estimate the animal's position on a discretized grid from the neural data during epochs when the animal was moving. We then applied this decoder during epochs when the animal was stationary at a reward location before initiating a new trial and defined replays as consecutive sequences of at least three adjacent decoded grid locations (Figure 5.4A; Figure D.7; see Appendix D.2 for details).

Similar to previous work (Widloski and Foster, 2022), we found that the hippocampal replays avoided passing through walls to a greater extent than expected by chance (Figure 5.4B; $p < 0.001$, permutation test). This finding suggests that hippocampal replays are shaped by a rapidly updated internal model of the environment, similar to how forward rollouts in our RL agent are shaped by its internal world model (Figure 5.1A). Additionally, the goal location was overrepresented in the hippocampal replays, consistent with the assumption of on-policy rollouts in the RL agent (Figure 5.4C; $p < 0.001$, permutation test; Widloski and Foster, 2022).

Inspired by our findings in the RL agent, we proceeded to investigate whether a replayed action was more likely to be taken by the animal if the replay was successful than if it was unsuccessful. Here, we defined a 'successful' replay as one which reached the goal location without passing through a wall (Figure 5.4A). Consistent with the RL model, we found that the first simulated action in the replay agreed with the next physical action more often for successful replays than for unsuccessful replays (Figure 5.4D, black; $p < 0.001$, permutation test). Such an effect was not observed in the 'away' trials (Figure 5.4D, gray; $p = 0.129$, permutation test), where the animals had no knowledge of the reward location and therefore could not know what constituted a successful replay. These findings are consistent with the hypothesis that successful replays should increase the probability of taking the replayed action, while unsuccessful replays should decrease this probability.

In the RL agent, we have direct access to the momentary policy and could therefore quantify the causal effect of a replay on behavior (Figure 5.3E). However, in the biological circuit, we cannot know whether the increased probability of following the first action of a successful replay is because the replay altered the policy (as in the RL agent), or whether the replay reflects a baseline policy that was already more likely to reach the goal prior to the replay. To circumvent this confound, we analyzed consecutive replays while the animal remained stationary. If our hypotheses hold, that (i) hippocampal replays resemble on-policy rollouts of an imagined action sequence, and (ii) performing a replay improves the policy, then consecutive replays should become increasingly successful even *in the absence* of any behavior between the replays.

To test this prediction, we considered trials where the animal performed a sequence of at least 3 replays at the 'away' location before moving to the 'home' location. We then quantified the

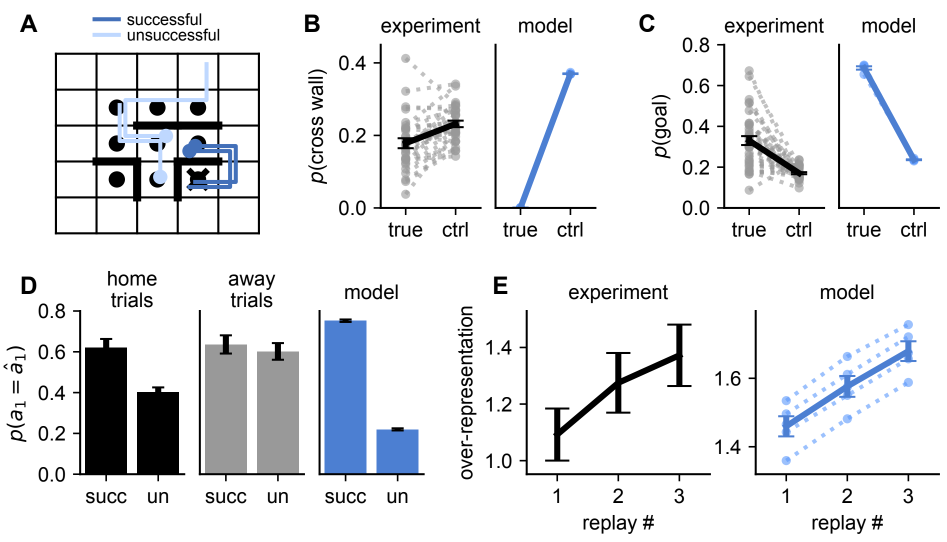

Figure 5.4 **Hippocampal replays resemble model rollouts. (A)** Illustration of experimental task structure and example replays (Widloski and Foster, 2022). Each episode had a different wall configuration and a randomly sampled home location (cross). Between each 'home' trial, the animal had to move to an 'away' location, which was sampled anew on each trial (black circles). Colored lines indicate example replay trajectories originating at the blue dots. Replays were detected during the stationary periods at the away locations before returning to the home location and classified according to whether they reached the home location (dark vs. light blue lines). **(B)** Fraction of replay transitions that pass through a wall in the experimental (black) and model (blue) data. Control values indicate the fraction of wall crossings in re-sampled environments with different wall configurations. Dashed lines indicate individual animals or RL agents, and solid lines indicate mean and standard error across animals or RL agents. **(C)** Fraction of replays that pass through the goal location in experimental (black) and model (blue) data. Control values indicate the average fraction of replays passing through a randomly sampled non-goal location. Dashed and solid lines are as in (B). See Figure D.8 for an analogous analysis of the away trials, where the goal was unknown. **(D)** Probability of taking the first replayed action, $p(a_1 = \hat{a}_1)$, for successful and unsuccessful replays during home trials (left; black), away trials (center; gray), and in the RL agent (right; blue). Bars and error bars indicate mean and standard error across sessions or RL agents. **(E)** Over-representation of successful replays during trials with at least three replays in the experimental data (left) and RL agents (right). The over-representation increased with replay number; an effect not seen in the away trials (Figure D.8). Over-representation was computed by dividing the success frequency by a reference frequency computed for randomly sampled alternative goal locations. Error bars indicate standard error across replays pooled from all animals (left) or standard error across five independently trained agents (right; dashed lines).

fraction of replays that were successful as a function of the replay index within the sequence, after regressing out the effect of time (Appendix D.2; Ólafsdóttir et al., 2017). We expressed this quantity as the degree to which the true goal was over-represented in the replay events by dividing the fraction of successful replays by a baseline calculated from the remaining non-goal locations, such that an over-representation of 1 implies that a replay was no more likely to be successful than expected by chance. Compellingly, this over-representation increased with each consecutive replay during the home trials (Figure 5.4E; left), and both the second and third replays exhibited substantially higher over-representation than the first replay ($p = 0.068$ and $p = 0.009$ respectively; permutation test; Appendix D.2). Such an effect was not seen during the away trials, where the rewarded location was not known to the animal (Figure D.8).

These findings are consistent with a theory in which replays represent on-policy rollouts that are used to iteratively refine the agent's policy, which in turn improves the quality of future replays – a phenomenon also observed in the RL agent (Figure 5.4E, right). In the RL agent, this effect could arise in part because the agent is less likely to perform an additional rollout after a successful rollout than after an unsuccessful rollout (Figure D.9). To eliminate this confound, we drew two samples from the policy each time the agent chose to perform a rollout, and we used one sample to update the hidden state of the agent, while the second sample was used to compute the goal over-representation (Appendix D.2). Such decoupling is not feasible in the experimental data, since we cannot read out the 'policy' of the animal. This leaves open the possibility that the increase in goal over-representation with consecutive biological replays is in part due to a reduced probability of performing an additional replay after a successful replay. However, we note that (i) the rodent task was not a 'reaction time task', since a 5-15 s delay was imposed between each trial. This makes a causal effect of replay success on the total number of replays less likely. (ii) if such an effect does exist, that is in itself consistent with a theory in which hippocampal replays guide planning.

5.2.6 RL agents use rollouts to optimize their hidden state

We have now seen that both biological and artificial agents appear to use policy rollouts to influence behavior in a way that depends on the *content* of the rollout. However, it remains to be understood (i) whether such an algorithm formally increases the expected reward, and (ii) how it is implemented mechanistically – a question we can address in the trained RL agent. In this section, we show that our theory has a firm theoretical grounding and makes quantitative predictions about the neural implementation of planning in PFC. Previously, we showed that the agent up- or downregulated the probability $p(\tau = \hat{\tau})$ of actually performing a rolled-out sequence $\hat{\tau}$ depending on the 'goodness' of the rollout (Figure 5.3E). This is reminiscent of canonical policy-gradient RL algorithms. These algorithms consider putative on-policy action sequences τ and apply *parameter updates* that cause $p(\tau)$ to increase under the agent's policy if τ led to more reward than expected, and to decrease otherwise. In our trained agent, adaptation to each new maze does not involve modifications of the fixed network parameters but instead occurs through changes to the *hidden state* $\boldsymbol{h}_k$. We therefore hypothesized that the performance improvements resulting from policy rollouts (Figure 5.3A; Figure 5.4E) were achieved through iterative modifications of $\boldsymbol{h}_k$ that approximated policy gradient ascent on the expected future reward in the episode as a function of $\boldsymbol{h}_k$ (Figure 5.5A).

To test this hypothesis, we considered each rollout performed by the RL agent and computed both (i) the actual hidden state update performed by the RL agent on the basis of this rollout, and (ii) the expected hidden state update computed by applying the policy gradient algorithm to the same rollout (Figure 5.5B; Appendix D.2). Our theory predicts that rollouts should change $\boldsymbol{h}_k$ in a way that increases $p(\tau = \hat{\tau})$ if the rollout is better than some baseline and decreases $p(\tau = \hat{\tau})$ otherwise. Since we do not know the baseline, we performed our analysis by taking the *derivative* of the hidden state change with respect to the expected reward from physically following $\hat{\tau}$, $R_{\hat{\tau}}$, which is independent of the baseline (Appendix D.2). This allowed us to define (i) a quantity $\boldsymbol{\alpha}^{\text{PG}} := \frac{\partial \Delta \boldsymbol{h}^{\text{PG}}}{\partial R_{\hat{\tau}}}$ that predicts how the hidden state *should* change as a function of $R_{\hat{\tau}}$ in the policy gradient formulation, and (ii) the corresponding quantity $\boldsymbol{\alpha}^{\text{RNN}} := \frac{\partial \Delta \boldsymbol{h}^{\text{RNN}}}{\partial R_{\hat{\tau}}}$ that indicates how the hidden state *actually* changed as a function of the content of the rollout. If the agent performs approximate policy gradient ascent in hidden state space, $\boldsymbol{\alpha}^{\text{RNN}}$ should be aligned with $\boldsymbol{\alpha}^{\text{PG}}$.

To investigate whether the response of the RNN to a rollout was consistent with this theory, we began by considering the effect on its hidden state of the first action in the rollout, $\hat{a}_1$. We did this by querying the alignment between (i) $\boldsymbol{\alpha}^{\text{RNN}}$ computed across rollouts from 1,000 episodes, and (ii) $\boldsymbol{\alpha}_1^{\text{PG}}$ computed from the same rollouts when considering only the probability of executing $\hat{a}_1$. To visualize this alignment, we performed PCA on $\{\boldsymbol{\alpha}_1^{\text{PG}}\}$ from all rollouts and projected both $\boldsymbol{\alpha}_1^{\text{PG}}$ and $\boldsymbol{\alpha}^{\text{RNN}}$ into this low-dimensional subspace. We then computed the average of each of these two quantities for each simulated action $\hat{a}_1 \in \{\text{left}, \text{right}, \text{up}, \text{down}\}$. We found that the average value of $\boldsymbol{\alpha}^{\text{RNN}}$ was strongly aligned with the average value of $\boldsymbol{\alpha}_1^{\text{PG}}$ for each action (Figure 5.5C), consistent with the theory outlined above. Importantly this means that $R_{\hat{\tau}}$ has *different* effects on the policy depending on the replayed trajectory $\hat{\tau}$. In other words, the spatial content of the rollout dynamically modulates the way in which the reward signal from the rollout affects the hidden state and policy of the agent.

To quantify the overlap between $\boldsymbol{\alpha}^{\text{RNN}}$ and $\boldsymbol{\alpha}_1^{\text{PG}}$ on a rollout-by-rollout basis, we computed the average cosine similarity d between $\boldsymbol{\alpha}^{\text{RNN}}$ and $\boldsymbol{\alpha}_1^{\text{PG}}$ across all rollouts. This overlap was substantially larger than zero ($d = 0.39 \pm 0.01$ mean $\pm$ sem; Figure 5.5D, left). When instead computing the overlap with $\boldsymbol{\alpha}_{\text{ctrl}}^{\text{RNN}}$ computed after changing the feedback input to falsely inform the agent that it simulated a different action $\hat{a}_{1,\text{ctrl}} \neq \hat{a}_1$, the corresponding value was $d = -0.11 \pm 0.004$. This confirms that $\boldsymbol{h}_k$ is optimized by incorporating the specific feedback input obtained from the rollout, and the negative sign reflects anti-correlations due to the policy being a normalized distribution over actions. For these analyses, we only considered the first simulated action $\hat{a}_1$. When instead querying the effect of the rollout on subsequent actions in $\hat{\tau}$, we found that the feedback input was also propagated through the network dynamics to

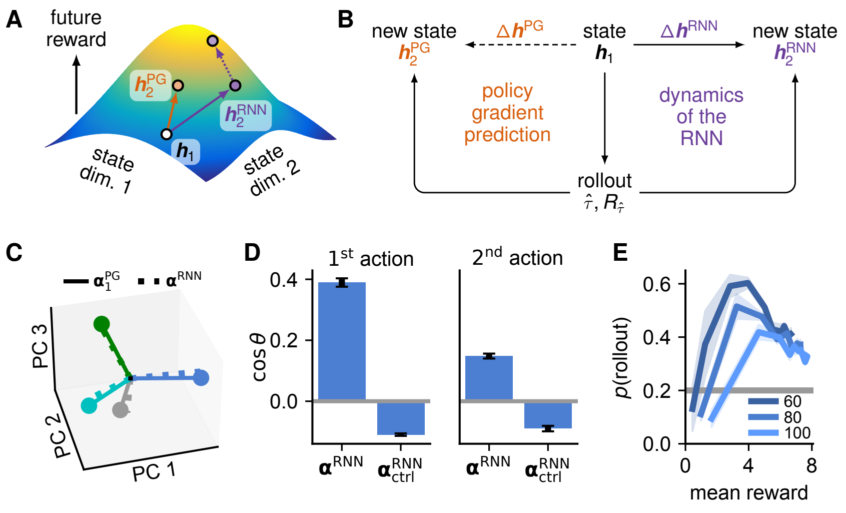

Figure 5.5 **Rollouts implement a hidden state optimization.** **(A)** The hidden state $\boldsymbol{h}_k$ of the RNN induces a policy with an expected future reward for the current episode. Rollouts can improve performance by shifting $\boldsymbol{h}$ to a region of state space with higher reward ($\boldsymbol{h}_1 \rightarrow \boldsymbol{h}_2^{\text{RNN}}$). The policy gradient algorithm estimates the direction of steepest ascent of the expected reward ($\boldsymbol{h}_1 \rightarrow \boldsymbol{h}_2^{\text{PG}}$). **(B)** We wanted to compare this theoretical hidden state update $\Delta\boldsymbol{h}^{\text{PG}} := \boldsymbol{h}_2^{\text{PG}} - \boldsymbol{h}_1$ to the empirical hidden state update $\Delta\boldsymbol{h}^{\text{RNN}} := \boldsymbol{h}_2^{\text{RNN}} - \boldsymbol{h}_1$ actually performed by the network dynamics on the basis of a rollout $\hat{\tau}$ and its associated reward $R_{\hat{\tau}}$. **(C)** A latent space was defined by performing PCA on $\boldsymbol{\alpha}_1^{\text{PG}}$ – the effect of $R_{\hat{\tau}}$ on $\boldsymbol{h}_k$ under the policy gradient algorithm. Solid lines and circles indicate the normalized average $\boldsymbol{\alpha}_1^{\text{PG}}$ for each of the four possible simulated actions ($\hat{a}_1$; colors). Dashed lines indicate the normalized average value of $\boldsymbol{\alpha}^{\text{RNN}}$ for the corresponding action, which is aligned with $\boldsymbol{\alpha}_1^{\text{PG}}$ in accordance with the theory. The first 3 PCs capture 100% of the variance in $\boldsymbol{\alpha}_1^{\text{PG}}$, since the policy is normalized and therefore only has three degrees of freedom. **(D)** Average cosine similarity between $\boldsymbol{\alpha}^{\text{RNN}}$ and $\boldsymbol{\alpha}_1^{\text{PG}}$, quantified in the space spanned by the top 3 PCs of $\boldsymbol{\alpha}_1^{\text{PG}}$ (see text for details). $\boldsymbol{\alpha}^{\text{RNN}}$ was computed using the true input, while $\boldsymbol{\alpha}^{\text{RNN}}_{\text{ctrl}}$ was computed after altering the feedback from the rollout to falsely inform the agent that it had simulated a different action $\hat{a}_{1,\text{ctrl}} \neq \hat{a}_1$. This confirms that the observed alignment is mediated by the input from the rollout. Left panel considers the effect of $R_{\hat{\tau}}$ on the first action ($\boldsymbol{\alpha}_1^{\text{PG}}$) and right panel considers the effect of $R_{\hat{\tau}}$ on the second action ($\boldsymbol{\alpha}_2^{\text{PG}}$). **(E)** We trained networks of different sizes (legend) and quantified both their performance (x-axis) and frequency of performing a rollout (y-axis) over the course of training (Figure D.10). To reach a given performance, we found that smaller networks relied *more* on rollouts, suggesting that the RL agents learn to plan in part because they are capacity limited. Additionally, the agents learned to rely less on rollouts late in training as they became increasingly good at the task, suggesting that they also plan because they are data limited.

these later actions (Figure 5.5D, right). These analyses confirm that policy rollouts consistently move the hidden state of the agent in the direction of the policy gradient.

5.3 Discussion

We have developed a new theory of planning in the prefrontal-hippocampal network, implemented as a recurrent neural network model and instantiated in a spatial navigation task requiring multi-step planning (Figure 5.1). Our model consists of a recurrent meta-reinforcement learning agent augmented with the explicit ability to plan using policy rollouts. We showed that this model provides a compelling account of human behavior in our task, where it explains the structure observed in human thinking times (Figure 5.2). These results suggest that planning using mental rollouts could constitute a major component underlying the striking human ability to adapt rapidly to new information and changing environments, where it allows agents to refine their behavior without incurring the potentially large cost of overtly executing suboptimal actions. Since mental simulation is generally faster and more efficient than physically interacting with the world (Vul et al., 2014), this allows agents to improve their overall performance despite the temporal opportunity cost of such simulation (Figure 5.3; Agrawal et al., 2022; Hamrick et al., 2017).

Our theory also suggests an important role of hippocampal replays during sequential decision making. By re-analyzing recordings from the rat hippocampus during a navigation task, we found that patterns of hippocampal replays and their relationship to behavior resembled the rollouts used by our model (Figure 5.4). These results suggest that hippocampal forward replays could be a manifestation of a planning process, and that the mechanistic insights derived from our model could generalize to biological circuits. In particular, we hypothesize that forward replays should have different effects on subsequent behavior depending on whether they lead to high-value or low-value states (Figure 5.3; Wu et al., 2017b). This hypothesis is consistent with previous models, where hippocampal replay is used to update state-action values that shape future behavior (Mattar and Daw, 2018). We suggest that forward replay implements planning through feedback to prefrontal cortex that drives a 'hidden state optimization' reminiscent of recent models of motor preparation (Figure 5.5; Kao et al., 2021b). This differs from prior work in the reinforcement learning literature, since our model does not involve arbitration between model-free and model-based policies computed separately (Daw et al., 2005; Geerts et al., 2020). Instead, model-based computations iteratively update a single policy that can be used for decision making at different stages of refinement.

5.3.1 Neural mechanisms of planning and decision making

Our model raises several interesting hypotheses about neural dynamics in hippocampus and prefrontal cortex and how these dynamics affect behavior. One is that hippocampal replays should causally affect the behavior of an animal as also suggested in previous work (Foster, 2017; Pfeiffer and Foster, 2013; Widloski and Foster, 2022). However, as noted previously

(Figure 5.4), this has been notoriously difficult to test in experiments due to the confound of how the behavioral intentions of the animal itself affect the content of hippocampal replays (Foster, 2017). Perhaps more interestingly, we predict that hippocampal forward replays should directly drive a change in PFC representations, consistent with previous work showing coordinated activity between hippocampus and PFC during sharp-wave ripples (Jadhav et al., 2016). Crucially, we also predict *how* PFC representations should change during planning depending on the spatial content and expected reward of a replay. These predictions could be investigated in experiments that record neural activity simultaneously from hippocampus and PFC, where both the timing and qualitative change in PFC representations can be related to the occurrence of replays in hippocampus.

To enable more detailed mechanistic predictions, our model could be extended in several ways. First, we have modeled the prefrontal network as a single fully connected network. In contrast, the brain relies on several connected but distinct circuits, all of which serve specialized functions that together give rise to the representations and dynamics driving human behavior. To understand these collective dynamics, it will therefore be interesting to extend our approach to modular models inspired by the architecture of multi-area networks. Second, our implementation of rollouts in the agent took the form of an abstract simulation process, where the underlying neural dynamics were not explicitly modeled. To better understand the mechanisms through which PFC interacts with other brain areas during planning, it will be important to model the whole rollout process as multi-area neural dynamics. Finally, while we propose a role of hippocampal replays in shaping immediate behavior via recurrent network dynamics, this is compatible with replays also having other functions, such as memory consolidation (Carr et al., 2011; van de Ven et al., 2016) or dopamine-driven synaptic plasticity over longer timescales (De Lavilléon et al., 2015; Gomperts et al., 2015).

5.3.2 Alternative planning algorithms

Planning in the RL agent was carried out explicitly in the space of observations. While this was already an abstract representation rather than pixel-level input, it could be interesting to explore planning in a latent space optimized e.g. to predict future observations (Zintgraf et al., 2019) or future policies and value functions (Ho et al., 2022; Schrittwieser et al., 2020). These ideas have proven useful in the machine learning literature, where they allow models to ignore details of the environment not needed to make good decisions, and it is plausible that the internal model of humans similarly does not include such task-irrelevant details. We also assumed that the planning process itself was ‘on policy’ – that is, the policy that was used to sample actions in the planning loop was identical to the policy used to act in the world. Although there is some support from the hippocampal replay data that forward replays are related to the ‘policy’ (e.g. wall avoidance and goal over-representation; Figure 5.4), there is in theory nothing that prevents the planning policy from differing arbitrarily from the action policy. In fact, the planning policy could even be explicitly optimized to yield good *plans* rather than re-using a policy optimized to yield good *behavior* (Pascanu et al., 2017). Such off-policy hippocampal sequence generation has also formed the basis of other recent theories of the role of hippocampus in planning and decision making (Mattar and Daw, 2018; McNamee et al., 2021). In this case, the policy gradient view of rollouts still provides a natural language for formalizing the planning process, since numerous off-policy extensions of the canonical policy gradient algorithm exist (Jie and Abbeel, 2010; Peshkin and Shelton, 2002).

5.3.3 Why do we spend time thinking?

Finally, while both humans and our RL agents made extensive use of planning, it is worth noting that mental simulation does not generate any new information about the world. In theory, it should therefore be possible to make equally good 'reflexive' decisions given enough computational power. This raises the question of why we rely on planning in the first place – in other words, what is the reason that decision making often takes time rather than being instantaneous? One possible reason could be that our decision making system is capacity limited, such that it does not have enough computational power to generate the optimal policy (Russek et al., 2022). In our computational model, this is supported by the observation that agents consisting of smaller RNNs tend to perform *more* rollouts than larger agents (Figure 5.5E). Alternatively, we could be data limited, meaning that we have not received enough training to learn the optimal policy. This also has support in our computational model, where networks of all sizes perform many rollouts early in training, when they have only seen a small amount of data, and gradually transition to a more reflexive policy that relies less on rollouts (Figure 5.5E; Figure D.10).

We hypothesize that data limitations are a major reason for the use of temporally extended planning in animals. In particular, we reason that learning the instantaneous mapping from states to actions needed for reflexive decisions would require a prohibitive amount of training data, which is generally not available for real-life scenarios. Indeed, training our meta-reinforcement learner required millions of episodes, while humans were immediately capable of solving the maze task from only a simple task description and demonstration. Such rapid learning could be due in part to the use of temporally extended planning algorithms as a form of 'canonical computation' that generalizes across tasks. If this is the case, we would be able to rely on generic planning algorithms acquired over the course of many previous tasks in order to solve a new task. When combined with a new task-specific transition function learned from relatively little experience or inferred from sensory inputs, planning would facilitate data-efficient reinforcement learning by allowing the agent to trade off processing time for a better policy (Schrittwieser et al., 2020). This is in contrast to our current model, which had to learn from scratch both the structure of the environment *and* how to use rollouts to shape its behavior. Importantly, planning as a canonical computation could generalize not just to other navigation tasks but also to other domains, such as compositional reasoning and sequence learning, where replay has recently been demonstrated in humans (Liu et al., 2019, 2021; Schwartenbeck et al., 2021). Further exploring these ideas will be an exciting avenue for future work.

Chapter 6

Discussion

Machine learning as a tool for systems neuroscience has undoubtedly come to stay. This toolbox has provided researchers with a wealth of new methods for analyzing, modelling, and interpreting the high-dimensional and noisy data that we are often faced with as neuroscientists. Bayesian approaches to machine learning have proven particularly influential in systems neuroscience, providing elegant methods for building inductive biases into our data analysis methods, and for capturing the biases of humans and other animals themselves.

Throughout this thesis, we have developed several such models that allow us to better understand the neural dynamics and computations underlying natural behaviours. First, we developed two new latent variable models that allow us to infer the dimensionality and topology of neural population recordings. This is useful since neural recordings are becoming increasingly high-dimensional, while it is hypothesized that the underlying quantities being represented in such high-dimensional data remain low-dimensional (Chaudhuri et al., 2019; Gallego et al., 2017; Humphries, 2020). While we have demonstrated applications of these methods to neural circuits involved in both motor control and navigation, we expect that they will also yield new insights in more cognitive domains, where low-dimensional representations of task-relevant variables have recently been demonstrated in e.g. the hippocampus of mice engaged in an evidence integration task (Nieh et al., 2021).

We then developed a new method for continual learning using approximate Bayesian inference and demonstrated how different algorithmic approaches to the continual learning problem yield different predictions for neural dynamics. We believe that this distinction between different classes of continual learning algorithms will be important in the field of neuroscience, where much uncertainty remains about the algorithmic solutions to the continual learning problem in biological systems (Clopath et al., 2017; Rule et al., 2019). In particular, we hope that comparisons with strong machine learning models can help disentangle the differences in representational drift between different neural circuits, and perhaps point to different mechanisms underlying the retention of memories across brain regions and organisms.

Finally, we developed a new reinforcement learning model, which was explicitly endowed with the ability to perform open-loop policy rollouts. We saw how this captures the important property of 'planning' in humans and demonstrated notable similarities with rodent hippocampal replays. The model learned to use such rollouts to perform a hidden state optimization, which can itself be considered a form of policy inference (Botvinick and Toussaint, 2012; Levine, 2018; Solway and Botvinick, 2012). In this case, however, the policy is parameterized by the hidden state of the network rather than the network parameters. This provides a new view of hippocampal replays as a form of inference process that allows for iterative updates of a policy stored in prefrontal cortex.

Together, these advances illustrate how approaches from Bayesian machine learning can be valuable both for analyzing the increasingly large datasets recorded in modern neuroscience, and for generating mechanistic and algorithmic hypotheses of how the brain solves the many challenging problems facing biological organisms. However, while probabilistic machine learning has been an essential tool for systems neuroscience in the past decade, it remains an open question whether this will continue to be the case in years to come. In particular, since deep learning has replaced Bayesian models for many applications in the machine learning literature, it is relevant to ask whether this will also be the case in neuroscience. As data sizes start increasing, Bayesian machine learning for data analysis may indeed become less important – similar to the recent trend towards more scalable machine learning applications in e.g. text and image processing (Dosovitskiy et al., 2020; Vaswani et al., 2017). In fact transformers – a modern architecture that is dominating machine learning – are already beginning to make their advances for neural data analysis (Ye and Pandarinath, 2021), where they can provide a powerful way of learning the relationships between large neural datasets and observed or unobserved regressors. Such methods may be particularly useful for application-driven domains such as brain-computer interfaces (Willett et al., 2021), where quantitative 'test performance' is paramount.

However, in data analysis for systems neuroscience, we often care as much about interpretability as we do about quantitative performance. This is because we fundamentally care about understanding *how* the brain solves computational problems – and this question is easier to answer with methods that are interpretable, e.g. by directly building in notions of topology or dimensionality. Additionally, strong models of neural data, such as those developed in Chapter 4 and Chapter 5, are not driven by considerations of quantitative benchmarking. Instead, they have been driven by a fundamental belief that biological brains perform approximately Bayesian computations, and that probability theory is therefore the right language with which to describe and understand neural dynamics. For these reasons, we believe that Bayesian machine learning will remain a critical tool in the toolbox of systems neuroscience, and that the insights gleaned from these approaches will continue to shed light on the neural underpinnings of natural behaviours.

References

Afshar, A., Santhanam, G., Byron, M. Y., Ryu, S. I., Sahani, M., and Shenoy, K. V. (2011). Single-trial neural correlates of arm movement preparation. *Neuron*, 71(3):555–564.

Agrawal, M., Mattar, M. G., Cohen, J. D., and Daw, N. D. (2022). The temporal dynamics of opportunity costs: A normative account of cognitive fatigue and boredom. *Psychological Review*, 129(3):564.

Aitchison, L., Jegminat, J., Menendez, J. A., Pfister, J.-P., Pouget, A., and Latham, P. E. (2021). Synaptic plasticity as Bayesian inference. *Nature Neuroscience*, 24(4):565–571.

Aljundi, R., Babiloni, F., Elhoseiny, M., Rohrbach, M., and Tuytelaars, T. (2018). Memory aware synapses: Learning what (not) to forget. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 139–154.

Allen, E., Baglama, J., and Boyd, S. (2000). Numerical approximation of the product of the square root of a matrix with a vector. *Linear Algebra and its Applications*, 310(1-3):167–181.

Alver, S. and Precup, D. (2021). What is going on inside recurrent meta reinforcement learning agents? *arXiv preprint arXiv:2104.14644*.

Amari, S.-I. (1998). Natural gradient works efficiently in learning. *Neural Computation*, 10(2):251–276.

Ames, K. C. and Churchland, M. M. (2019). Motor cortex signals for each arm are mixed across hemispheres and neurons yet partitioned within the population response. *eLife*, 8:e46159.

Antonov, G., Gagne, C., Eldar, E., and Dayan, P. (2022). Optimism and pessimism in optimised replay. *PLOS Computational Biology*, 18(1):e1009634.

Arvanitidis, G., Hansen, L. K., and Hauberg, S. (2017). Latent space oddity: on the curvature of deep generative models. *arXiv preprint arXiv:1710.11379*.

Ashwood, Z. C., Roy, N. A., Stone, I. R., Laboratory, I. B., Urai, A. E., Churchland, A. K., Pouget, A., and Pillow, J. W. (2022). Mice alternate between discrete strategies during perceptual decision-making. *Nature Neuroscience*, 25(2):201–212.

Azevedo, F. A., Carvalho, L. R., Grinberg, L. T., Farfel, J. M., Ferretti, R. E., Leite, R. E., Filho, W. J., Lent, R., and Herculano-Houzel, S. (2009). Equal numbers of neuronal and nonneuronal cells make the human brain an isometrically scaled-up primate brain. *Journal of Comparative Neurology*, 513(5):532–541.

Azouz, R. and Gray, C. M. (1999). Cellular mechanisms contributing to response variability of cortical neurons in vivo. *J. Neurosci.*, 19(6):2209–2223.

Banino, A., Balaguer, J., and Blundell, C. (2021). Pondernet: Learning to ponder. *arXiv preprint arXiv:2107.05407*.

Banino, A., Barry, C., Uria, B., Blundell, C., Lillicrap, T., Mirowski, P., Pritzel, A., Chadwick, M. J., Degris, T., Modayil, J., Wayne, G., Soyer, H., Viola, F., Zhang, B., Goroshin, R., Rabinowitz, N., Pascanu, R., Beattie, C., Petersen, S., Sadik, A., Gaffney, S., King, H., Kavukcuoglu, K., Hassabis, D., Hadsell, R., and Kumaran, D. (2018). Vector-based navigation using grid-like representations in artificial agents. *Nature*, 557(7705):429–433.

Bansal, A., Schwarzschild, A., Borgnia, E., Emam, Z., Huang, F., Goldblum, M., and Goldstein, T. (2022). End-to-end algorithm synthesis with recurrent networks: Logical extrapolation without overthinking. *arXiv preprint arXiv:2202.05826*.

Beck, J. M., Ma, W. J., Kiani, R., Hanks, T., Churchland, A. K., Roitman, J., Shadlen, M. N., Latham, P. E., and Pouget, A. (2008). Probabilistic population codes for Bayesian decision making. *Neuron*, 60(6):1142–1152.

Berkes, P., Orbán, G., Lengyel, M., and Fiser, J. (2011). Spontaneous cortical activity reveals hallmarks of an optimal internal model of the environment. *Science*, 331(6013):83–87.

Bernacchia, A., Lengyel, M., and Hennequin, G. (2018). Exact natural gradient in deep linear networks and its application to the nonlinear case. In *Advances in Neural Information Processing Systems*, volume 31.

Bishop, C. M. (1999). Bayesian PCA. In *Advances in Neural Information Processing Systems*.

Bjerke, M., Schott, L., Jensen, K. T., Battistin, C., Klindt, D. A., and Dunn, B. A. (2022). Understanding neural coding on latent manifolds by sharing features and dividing ensembles. *arXiv preprint arXiv:2210.03155*.

Bolkan, S. S., Stone, I. R., Pinto, L., Ashwood, Z. C., Iravedra Garcia, J. M., Herman, A. L., Singh, P., Bandi, A., Cox, J., Zimmerman, C. A., et al. (2022). Opponent control of behavior by dorsomedial striatal pathways depends on task demands and internal state. *Nature Neuroscience*, 25(3):345–357.

Borovitskiy, V., Terenin, A., Mostowsky, P., and Deisenroth, M. P. (2020). Matérn Gaussian processes on Riemannian manifolds. *arXiv preprint arXiv:2006.10160*.

Botvinick, M. and Toussaint, M. (2012). Planning as inference. *Trends in cognitive sciences*, 16(10):485–488.

Botvinick, M., Wang, J. X., Dabney, W., Miller, K. J., and Kurth-Nelson, Z. (2020). Deep reinforcement learning and its neuroscientific implications. *Neuron*, 107(4):603–616.

Botvinick, M. M. and Cohen, J. D. (2014). The computational and neural basis of cognitive control: charted territory and new frontiers. *Cognitive science*, 38(6):1249–1285.

Bui, T. D., Yan, J., and Turner, R. E. (2017). A unifying framework for Gaussian process pseudo-point approximations using power expectation propagation. *The Journal of Machine Learning Research*, 18(1):3649–3720.

Burda, Y., Grosse, R., and Salakhutdinov, R. (2015). Importance weighted autoencoders. *arXiv preprint arXiv:1509.00519*.

Callaway, F., van Opheusden, B., Gul, S., Das, P., Krueger, P. M., Griffiths, T. L., and Lieder, F. (2022). Rational use of cognitive resources in human planning. *Nature Human Behaviour*, 6(8):1112–1125.

Carmena, J. M., Lebedev, M. A., Henriquez, C. S., and Nicolelis, M. A. (2005). Stable ensemble performance with single-neuron variability during reaching movements in primates. *Journal of Neuroscience*, 25(46):10712–10716.

Carr, M. F., Jadhav, S. P., and Frank, L. M. (2011). Hippocampal replay in the awake state: a potential substrate for memory consolidation and retrieval. *Nature Neuroscience*, 14(2):147–153.

Challis, E. and Barber, D. (2013). Gaussian Kullback-Leibler approximate inference. *Journal of Machine Learning Research*, 14(8).

Chang, P. E., Wilkinson, W. J., Khan, M. E., and Solin, A. (2020). Fast variational learning in state-space Gaussian process models. In *2020 IEEE 30th International Workshop on Machine Learning for Signal Processing (MLSP)*, pages 1–6. IEEE.

Chaudhry, A., Dokania, P. K., Ajanthan, T., and Torr, P. H. (2018). Riemannian walk for incremental learning: Understanding forgetting and intransigence. *arXiv preprint arXiv:1801.10112*.

Chaudhuri, R., Gercek, B., Pandey, B., Peyrache, A., and Fiete, I. (2019). The intrinsic attractor manifold and population dynamics of a canonical cognitive circuit across waking and sleep. *Nature Neuroscience*, 22(9):1512–1520.

Chestek, C. A., Batista, A. P., Santhanam, G., Byron, M. Y., Afshar, A., Cunningham, J. P., Gilja, V., Ryu, S. I., Churchland, M. M., and Shenoy, K. V. (2007). Single-neuron stability during repeated reaching in macaque premotor cortex. *Journal of Neuroscience*, 27(40):10742–10750.

Cho, K., Van Merriënboer, B., Bahdanau, D., and Bengio, Y. (2014). On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*.

Churchland, M. M., Cunningham, J. P., Kaufman, M. T., Foster, J. D., Nuyujukian, P., Ryu, S. I., and Shenoy, K. V. (2012). Neural population dynamics during reaching. *Nature*, 487(7405):51–56.

Churchland, M. M. and Shenoy, K. V. (2007). Temporal complexity and heterogeneity of single-neuron activity in premotor and motor cortex. *Journal of neurophysiology*, 97:4235–4257.

Clopath, C., Bonhoeffer, T., Hübener, M., and Rose, T. (2017). Variance and invariance of neuronal long-term representations. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 372(1715):20160161.

Cong, Y., Zhao, M., Li, J., Wang, S., and Carin, L. (2020). GAN memory with no forgetting. In *Advances in Neural Information Processing Systems*, volume 33.

Constantinescu, A. O., O'Reilly, J. X., and Behrens, T. E. (2016). Organizing conceptual knowledge in humans with a gridlike code. *Science*, 352:1464–1468.

Cunningham, J. P. and Byron, M. Y. (2014). Dimensionality reduction for large-scale neural recordings. *Nature Neuroscience*, 17(11):1500–1509.

Cunningham, J. P. and Ghahramani, Z. (2015). Linear dimensionality reduction: Survey, insights, and generalizations. *The Journal of Machine Learning Research*, 16:2859–2900.

Dabney, W., Kurth-Nelson, Z., Uchida, N., Starkweather, C. K., Hassabis, D., Munos, R., and Botvinick, M. (2020). A distributional code for value in dopamine-based reinforcement learning. *Nature*, 577(7792):671–675.

Damianou, A. and Lawrence, N. D. (2013). Deep Gaussian processes. In *Artificial intelligence and statistics*, pages 207–215. PMLR.

Davidson, T. R., Falorsi, L., De Cao, N., Kipf, T., and Tomczak, J. M. (2018). Hyperspherical variational auto-encoders. *34th Conference on Uncertainty in Artificial Intelligence.*

Daw, N. D., Gershman, S. J., Seymour, B., Dayan, P., and Dolan, R. J. (2011). Model-based influences on humans' choices and striatal prediction errors. *Neuron*, 69(6):1204–1215.

Daw, N. D., Niv, Y., and Dayan, P. (2005). Uncertainty-based competition between prefrontal and dorsolateral striatal systems for behavioral control. *Nature Neuroscience*, 8(12):1704–1711.

Dayan, P., Hinton, G. E., Neal, R. M., and Zemel, R. S. (1995). The Helmholtz machine. *Neural computation*, 7(5):889–904.

de Jong, E. D. (2016). Incremental sequence learning. *arXiv preprint arXiv:1611.03068.*

De Lavilléon, G., Lacroix, M. M., Rondi-Reig, L., and Benchenane, K. (2015). Explicit memory creation during sleep demonstrates a causal role of place cells in navigation. *Nature Neuroscience*, 18(4):493–495.

de Saa, J. R. C. and Renart, A. (2022). Control limited perceptual decision making. *bioRxiv.*

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee.

Dhawale, A. K., Poddar, R., Wolff, S. B., Normand, V. A., Kopelowitz, E., and Ölveczky, B. P. (2017). Automated long-term recording and analysis of neural activity in behaving animals. *eLife*, 6:e27702.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. (2020). An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929.*

Doya, K., Ishii, S., Pouget, A., and Rao, R. P. (2007). *Bayesian brain: Probabilistic approaches to neural coding.* MIT press.

Driscoll, L. N., Pettit, N. L., Minderer, M., Chettih, S. N., and Harvey, C. D. (2017). Dynamic reorganization of neuronal activity patterns in parietal cortex. *Cell*, 170(5):986–999.

Drugowitsch, J., DeAngelis, G. C., Klier, E. M., Angelaki, D. E., and Pouget, A. (2014). Optimal multisensory decision-making in a reaction-time task. *Elife*, 3.

Duan, Y., Schulman, J., Chen, X., Bartlett, P. L., Sutskever, I., and Abbeel, P. (2016). RL2: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779.*

Duncker, L., Driscoll, L., Shenoy, K. V., Sahani, M., and Sussillo, D. (2020). Organizing recurrent network dynamics by task-computation to enable continual learning. In *Advances in Neural Information Processing Systems*, volume 33.

Duncker, L. and Sahani, M. (2018). Temporal alignment and latent Gaussian process factor inference in population spike trains. In *Advances in Neural Information Processing Systems*, volume 31.

Dunn, T. W., Marshall, J. D., Severson, K. S., Aldarondo, D. E., Hildebrand, D. G., Chettih, S. N., Wang, W. L., Gellis, A. J., Carlson, D. E., Aronov, D., et al. (2021). Geometric deep learning enables 3d kinematic profiling across species and environments. *Nature methods*, 18(5):564–573.

Echeveste, R., Aitchison, L., Hennequin, G., and Lengyel, M. (2020). Cortical-like dynamics in recurrent circuits optimized for sampling-based probabilistic inference. *Nature Neuroscience*, 23(9):1138–1149.

Ecker, A. S., Berens, P., Cotton, R. J., Subramaniyan, M., Denfield, G. H., Cadwell, C. R., Smirnakis, S. M., Bethge, M., and Tolias, A. S. (2014). State dependence of noise correlations in macaque primary visual cortex. *Neuron*, 82(1):235–248.

Ehret, B., Henning, C., Cervera, M. R., Meulemans, A., von Oswald, J., and Grewe, B. F. (2020). Continual learning in recurrent neural networks with hypernetworks. *arXiv preprint arXiv:2006.12109*.

Elgammal, A. and Lee, C.-S. (2008). Tracking people on a torus. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(3):520–538.

Elsayed, G. F., Lara, A. H., Kaufman, M. T., Churchland, M. M., and Cunningham, J. P. (2016). Reorganization between preparatory and movement population responses in motor cortex. *Nat. Commun.*, 7:1–15.

Ernst, M. O. and Banks, M. S. (2002). Humans integrate visual and haptic information in a statistically optimal fashion. *Nature*, 415(6870):429–433.

Failor, S. W., Carandini, M., and Harris, K. D. (2021). Learning orthogonalizes visual cortical population codes. *bioRxiv*.

Falorsi, L., de Haan, P., Davidson, T. R., and Forré, P. (2019). Reparameterizing distributions on Lie groups. *arXiv preprint arXiv:1903.02958*.

Falorsi, L. and Forré, P. (2020). Neural ordinary differential equations on manifolds. *arXiv preprint arXiv:2006.06663*.

Fenton, A. A. and Muller, R. U. (1998). Place cell discharge is extremely variable during individual passes of the rat through the firing field. *Proceedings of the National Academy of Sciences*, 95(6):3182–3187.

Feragen, A., Lauze, F., and Hauberg, S. (2015). Geodesic exponential kernels: When curvature and linearity conflict. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3032–3042.

Finkelstein, A., Derdikman, D., Rubin, A., Foerster, J. N., Las, L., and Ulanovsky, N. (2015). Three-dimensional head-direction coding in the bat brain. *Nature*, 517(7533):159–164.

Fiser, J., Berkes, P., Orbán, G., and Lengyel, M. (2010). Statistically optimal perception and learning: from behavior to neural representations. *Trends in Cognitive Sciences*, 14(3):119–130.

Flint, R. D., Scheid, M. R., Wright, Z. A., Solla, S. A., and Slutzky, M. W. (2016). Long-term stability of motor cortical activity: implications for brain machine interfaces and optimal feedback control. *Journal of neuroscience*, 36(12):3623–3632.

Fortuin, V., Garriga-Alonso, A., Ober, S. W., Wenzel, F., Rätsch, G., Turner, R. E., van der Wilk, M., and Aitchison, L. (2021). Bayesian neural network priors revisited. *arXiv preprint arXiv:2102.06571*.

Foster, D. J. (2017). Replay comes of age. *Annu. Rev. Neurosci*, 40(581-602):9.

Fu, M., Yu, X., Lu, J., and Zuo, Y. (2012). Repetitive motor learning induces coordinated formation of clustered dendritic spines in vivo. *Nature*, 483(7387):92–95.

Gallego, J. A., Perich, M. G., Chowdhury, R. H., Solla, S. A., and Miller, L. E. (2020). Long-term stability of cortical population dynamics underlying consistent behavior. *Nature Neuroscience*, 23(2):260–270.

Gallego, J. A., Perich, M. G., Miller, L. E., and Solla, S. A. (2017). Neural manifolds for the control of movement. *Neuron*, 94(5):978–984.

Ganguly, K. and Carmena, J. M. (2009). Emergence of a stable cortical map for neuroprosthetic control. *PLoS Biology*, 7(7):e1000153.

Gao, Y., Archer, E. W., Paninski, L., and Cunningham, J. P. (2016). Linear dynamical neural population models through nonlinear embeddings. In *Advances in Neural Information Processing Systems*, volume 29.

Gardner, R. J., Hermansen, E., Pachitariu, M., Burak, Y., Baas, N. A., Dunn, B. A., Moser, M.-B., and Moser, E. I. (2022). Toroidal topology of population activity in grid cells. *Nature*, 602(7895):123–128.

Geerts, J. P., Chersi, F., Stachenfeld, K. L., and Burgess, N. (2020). A general model of hippocampal and dorsal striatal learning and decision making. *Proceedings of the National Academy of Sciences*, 117(49):31427–31437.

Ghahramani, Z. (2015). Probabilistic machine learning and artificial intelligence. *Nature*, 521(7553):452–459.

Gillespie, A. K., Maya, D. A. A., Denovellis, E. L., Liu, D. F., Kastner, D. B., Coulter, M. E., Roumis, D. K., Eden, U. T., and Frank, L. M. (2021). Hippocampal replay reflects specific past experiences rather than a plan for subsequent choice. *Neuron*, 109(19):3149–3163.

Glaser, J. I., Benjamin, A. S., Chowdhury, R. H., Perich, M. G., Miller, L. E., and Kording, K. P. (2020). Machine learning for neural decoding. *Eneuro*, 7(4).

Gold, J. I. and Shadlen, M. N. (2001). Neural computations that underlie decisions about sensory stimuli. *Trends in cognitive sciences*, 5(1):10–16.

Gomperts, S. N., Kloosterman, F., and Wilson, M. A. (2015). VTA neurons coordinate with the hippocampal reactivation of spatial experience. *Elife*, 4:e05360.

Graves, A. (2016). Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*.

Griffiths, T. L., Lieder, F., and Goodman, N. D. (2015). Rational use of cognitive resources: Levels of analysis between the computational and the algorithmic. *Topics in cognitive science*, 7(2):217–229.

Hafting, T., Fyhn, M., Molden, S., Moser, M.-B., and Moser, E. I. (2005). Microstructure of a spatial map in the entorhinal cortex. *Nature*, 436(7052):801–806.

Hamrick, J. B., Ballard, A. J., Pascanu, R., Vinyals, O., Heess, N., and Battaglia, P. W. (2017). Metacontrol for adaptive imagination-based optimization. *arXiv preprint arXiv:1705.02670*.

Hardcastle, K., Maheswaranathan, N., Ganguli, S., and Giocomo, L. M. (2017). A multiplexed, heterogeneous, and adaptive code for navigation in medial entorhinal cortex. *Neuron*, 94(2):375–387.

Hastings, W. K. (1970). Monte Carlo sampling methods using Markov chains and their applications.

Heald, J. B., Lengyel, M., and Wolpert, D. M. (2021). Contextual inference underlies the learning of sensorimotor repertoires. *Nature*, 600(7889):489–493.

Hensman, J., Fusi, N., and Lawrence, N. D. (2013). Gaussian processes for big data. *arXiv preprint arXiv:1309.6835*.

Hensman, J., Matthews, A., and Ghahramani, Z. (2015a). Scalable variational Gaussian process classification. In *Artificial Intelligence and Statistics*, pages 351–360. PMLR.

Hensman, J., Matthews, A. G., Filippone, M., and Ghahramani, Z. (2015b). MCMC for variationally sparse Gaussian processes. In *Advances in Neural Information Processing Systems*, volume 28.

Ho, M. K., Abel, D., Correa, C. G., Littman, M. L., Cohen, J. D., and Griffiths, T. L. (2022). People construct simplified mental representations to plan. *Nature*, 606(7912):129–136.

Hochberg, L. R., Bacher, D., Jarosiewicz, B., Masse, N. Y., Simeral, J. D., Vogel, J., Haddadin, S., Liu, J., Cash, S. S., Van Der Smagt, P., et al. (2012). Reach and grasp by people with tetraplegia using a neurally controlled robotic arm. *Nature*, 485(7398):372–375.

Holtmaat, A. and Svoboda, K. (2009). Experience-dependent structural synaptic plasticity in the mammalian brain. *Nature Reviews Neuroscience*, 10(9):647–658.

Humphries, M. D. (2020). Strong and weak principles of neural dimension reduction. *arXiv preprint arXiv:2011.08088*.

Huszár, F. (2017). On quadratic penalties in elastic weight consolidation. *arXiv preprint arXiv:1712.03847*.

Innes, M., Saba, E., Fischer, K., Gandhi, D., Rudilosso, M. C., Joy, N. M., Karmali, T., Pal, A., and Shah, V. (2018). Fashionable modelling with Flux. *arXiv preprint arXiv:1811.01457*.

Jacob, P.-Y., Casali, G., Spieser, L., Page, H., Overington, D., and Jeffery, K. (2017). An independent, landmark-dominated head-direction signal in dysgranular retrosplenial cortex. *Nature Neuroscience*, 20(2):173–175.

Jaderberg, M., Mnih, V., Czarnecki, W. M., Schaul, T., Leibo, J. Z., Silver, D., and Kavukcuoglu, K. (2016). Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*.

Jadhav, S. P., Rothschild, G., Roumis, D. K., and Frank, L. M. (2016). Coordinated excitation and inhibition of prefrontal ensembles during awake hippocampal sharp-wave ripple events. *Neuron*, 90(1):113–127.

Jakob, W. (2012). Numerically stable sampling of the von Mises-Fisher distribution on S^2 (and other tricks).

Jayasumana, S., Hartley, R., Salzmann, M., Li, H., and Harandi, M. (2015). Kernel methods on Riemannian manifolds with Gaussian RBF kernels. *IEEE transactions on pattern analysis and machine intelligence*, 37(12):2464–2477.

Jensen, K. T., Hennequin, G., and Mattar, M. G. (2023). A recurrent network model of planning explains hippocampal replay and human behavior. *bioRxiv*.

Jensen, K. T., Kadmon Harpaz, N., Dhawale, A. K., Wolff, S. B., and Ölveczky, B. P. (2022a). Long-term stability of single neuron activity in the motor system. *Nature Neuroscience*, 25:1–11.

Jensen, K. T., Kao, T.-C., Stone, J., and Hennequin, G. (2021). Scalable Bayesian GPFA with automatic relevance determination and discrete noise models. In *Advances in Neural Information Processing Systems*, volume 34.

Jensen, K. T., Kao, T.-C., Tripodi, M., and Hennequin, G. (2020). Manifold GPLVMs for discovering non-Euclidean latent structure in neural data. In *Advances in Neural Information Processing Systems*, volume 33.

Jensen, K. T., Liu, D., Kao, T.-C., Lengyel, M., and Hennequin, G. (2022b). Beyond the Euclidean brain: inferring non-Euclidean latent trajectories from spike trains. *bioRxiv*.

Jiang, W.-C., Xu, S., and Dudman, J. T. (2022). Hippocampal representations of foraging trajectories depend upon spatial context. *Nature Neuroscience*, 25(12):1693–1705.

Jie, T. and Abbeel, P. (2010). On a connection between importance sampling and the likelihood ratio policy gradient. In *Advances in Neural Information Processing Systems*, volume 23.

Johnson, A. and Redish, A. D. (2007). Neural ensembles in CA3 transiently encode paths forward of the animal at a decision point. *Journal of Neuroscience*, 27(45):12176–12189.

Jordan, M. I. and Rumelhart, D. E. (1992). Forward models: Supervised learning with a distal teacher. *Cognitive science*, 16(3):307–354.

Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., Tunyasuvunakool, K., Bates, R., Žídek, A., Potapenko, A., et al. (2021). Highly accurate protein structure prediction with AlphaFold. *Nature*, 596(7873):583–589.

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

Kao, T.-C., Jensen, K. T., van de Ven, G., Bernacchia, A., and Hennequin, G. (2021a). Natural continual learning: success is a journey, not (just) a destination. In *Advances in Neural Information Processing Systems*, volume 34.

Kao, T.-C., Sadabadi, M. S., and Hennequin, G. (2021b). Optimal anticipatory control as a theory of motor preparation: A thalamo-cortical circuit model. *Neuron*, 109(9):1567–1581.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. (2020). Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*.

Katlowitz, K. A., Picardo, M. A., and Long, M. A. (2018). Stable sequential activity underlying the maintenance of a precisely executed skilled behavior. *Neuron*, 98(6):1133–1140.

Kaufman, M. T., Churchland, M. M., Ryu, S. I., and Shenoy, K. V. (2014). Cortical activity in the null space: permitting preparation without movement. *Nature Neuroscience*, 17(3):440–448.

Kawai, R., Markman, T., Poddar, R., Ko, R., Fantana, A. L., Dhawale, A. K., Kampff, A. R., and Ölveczky, B. P. (2015). Motor cortex is required for learning but not for executing a motor skill. *Neuron*, 86(3):800–812.

Keeley, S., Aoi, M., Yu, Y., Smith, S., and Pillow, J. W. (2020a). Identifying signal and noise structure in neural population activity with Gaussian process factor models. In *Advances in neural information processing systems*, volume 33.

Keeley, S., Zoltowski, D., Yu, Y., Smith, S., and Pillow, J. (2020b). Efficient non-conjugate Gaussian process factor models for spike count data using polynomial approximations. In *International Conference on Machine Learning*, pages 5177–5186. PMLR.

Keeley, S. L., Zoltowski, D. M., Aoi, M. C., and Pillow, J. W. (2020c). Modeling statistical dependencies in multi-region spike train data. *Current Opinion in Neurobiology*, 65:194–202.

Keshtkaran, M. R. and Pandarinath, C. (2019). Enabling hyperparameter optimization in sequential autoencoders for spiking neural data. *arXiv preprint arXiv:1908.07896*.

Keshtkaran, M. R., Sedler, A. R., Chowdhury, R. H., Tandon, R., Basrai, D., Nguyen, S. L., Sohn, H., Jazayeri, M., Miller, L. E., and Pandarinath, C. (2021). A large-scale neural network training framework for generalized estimation of single-trial population dynamics. *bioRxiv*.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Kirkpatrick, J., Pascanu, R., Rabinowitz, N., Veness, J., Desjardins, G., Rusu, A. A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., et al. (2017). Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526.

Knill, D. C. and Richards, W. (1996). *Perception as Bayesian inference*. Cambridge University Press.

Körding, K. P., Beierholm, U., Ma, W. J., Quartz, S., Tenenbaum, J. B., and Shams, L. (2007). Causal inference in multisensory perception. *PLoS one*, 2(9):e943.

Körding, K. P. and Wolpert, D. M. (2004). Bayesian integration in sensorimotor learning. *Nature*, 427(6971):244–247.

Körding, K. P. and Wolpert, D. M. (2006). Bayesian decision theory in sensorimotor control. *Trends in cognitive sciences*, 10(7):319–326.

Krakauer, J. W. and Shadmehr, R. (2006). Consolidation of motor memory. *Trends in neurosciences*, 29(1):58–64.

Krause, E. L. and Drugowitsch, J. (2022). A large majority of awake hippocampal sharp-wave ripples feature spatial trajectories with momentum. *Neuron*, 110(4):722–733.

Kunstner, F., Balles, L., and Hennig, P. (2019). Limitations of the empirical Fisher approximation for natural gradient descent. *arXiv preprint arXiv:1905.12558*.

Kurth-Nelson, Z., Economides, M., Dolan, R. J., and Dayan, P. (2016). Fast sequences of non-spatial state representations in humans. *Neuron*, 91(1):194–204.

Lai, L. and Gershman, S. J. (2021). Policy compression: An information bottleneck in action selection. In *Psychology of Learning and Motivation*, volume 74, pages 195–232. Elsevier.

Lakshmanan, K. C., Sadtler, P. T., Tyler-Kabara, E. C., Batista, A. P., and Yu, B. M. (2015). Extracting low-dimensional latent structure from time series in the presence of delays. *Neural computation*, 27(9):1825–1856.

Lara, A. H., Elsayed, G. F., Zimnik, A. J., Cunningham, J. P., and Churchland, M. M. (2018). Conservation of preparatory neural events in monkey motor cortex regardless of how movement is initiated. *eLife*, 7:e31826.

Lawrence, N. (2005). Probabilistic non-linear principal component analysis with Gaussian process latent variable models. *Journal of machine learning research*, 6:1783–1816.

Lawrence, N. D. (2004). Gaussian process latent variable models for visualisation of high dimensional data. In *Advances in neural information processing systems*.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444.

Levine, S. (2018). Reinforcement learning and control as probabilistic inference: Tutorial and review. *arXiv preprint arXiv:1805.00909*.

Li, Z. and Hoiem, D. (2017). Learning without forgetting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(12):2935–2947.

Liu, D. and Lengyel, M. (2021). A universal probabilistic spike count model reveals ongoing modulation of neural variability. In *Advances in Neural Information Processing Systems*, volume 34.

Liu, Y., Dolan, R. J., Kurth-Nelson, Z., and Behrens, T. E. (2019). Human replay spontaneously reorganizes experience. *Cell*, 178(3):640–652.

Liu, Y., Mattar, M. G., Behrens, T. E., Daw, N. D., and Dolan, R. J. (2021). Experience replay is associated with efficient nonlocal learning. *Science*, 372(6544):eabf1357.

Loo, N., Swaroop, S., and Turner, R. E. (2020). Generalized variational continual learning. *arXiv preprint arXiv:2011.12328*.

Lou, A., Lim, D., Katsman, I., Huang, L., Jiang, Q., Lim, S.-N., and De Sa, C. (2020). Neural manifold ordinary differential equations. *arXiv preprint arXiv:2006.10254*.

Low, R. J., Lewallen, S., Aronov, D., Nevers, R., and Tank, D. W. (2018). Probing variability in a cognitive map using manifold inference from neural dynamics. *bioRxiv*.

Lütcke, H., Margolis, D. J., and Helmchen, F. (2013). Steady or changing? long-term monitoring of neuronal population activity. *Trends in Neurosciences*, 36(7):375–384.

Ma, W. J., Beck, J. M., Latham, P. E., and Pouget, A. (2006). Bayesian inference with probabilistic population codes. *Nature Neuroscience*, 9(11):1432–1438.

Maaten, L. v. d. and Hinton, G. (2008). Visualizing data using t-SNE. *Journal of machine learning research*, 9:2579–2605.

MacKay, D. J. (1992). A practical Bayesian framework for backpropagation networks. *Neural computation*, 4(3):448–472.

MacKay, D. J. (1998). Introduction to Gaussian processes. *NATO ASI series. Series F: computer and system sciences*, pages 133–165.

MacKay, D. J. (2003). *Information theory, inference and learning algorithms.* Cambridge university press.

Macke, J. H., Buesing, L., Cunningham, J. P., Yu, B. M., Shenoy, K. V., and Sahani, M. (2012). Empirical models of spiking in neural populations. In *Advances in Neural Information Processing Systems.*

Makin, J. G., O'Doherty, J. E., Cardoso, M. M., and Sabes, P. N. (2018). Superior arm-movement decoding from cortex with a new, unsupervised-learning algorithm. *Journal of neural engineering*, 15(2):026010.

Mallasto, A. and Feragen, A. (2018). Wrapped Gaussian process regression on Riemannian manifolds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5580–5588.

Mallasto, A., Hauberg, S., and Feragen, A. (2019). Probabilistic Riemannian submanifold learning with wrapped Gaussian process latent variable models. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 2368–2377. PMLR.

Mante, V., Sussillo, D., Shenoy, K. V., and Newsome, W. T. (2013). Context-dependent computation by recurrent dynamics in prefrontal cortex. *Nature*, 503(7474):78–84.

Martens, J. (2014). New insights and perspectives on the natural gradient method. *arXiv preprint arXiv:1412.1193.*

Martens, J., Ba, J., and Johnson, M. (2018). Kronecker-factored curvature approximations for recurrent neural networks. In *International Conference on Learning Representations.*

Martens, J. and Grosse, R. (2015). Optimizing neural networks with Kronecker-factored approximate curvature. In *ICML*, pages 2408–2417.

Masullo, L., Mariotti, L., Alexandre, N., Freire-Pritchett, P., Boulanger, J., and Tripodi, M. (2019). Genetically defined functional modules for spatial orienting in the mouse superior colliculus. *Current Biology*, 29:2892–2904.

Mathieu, E. and Nickel, M. (2020). Riemannian continuous normalizing flows. *arXiv preprint arXiv:2006.10605.*

Mathis, A., Mamidanna, P., Cury, K. M., Abe, T., Murthy, V. N., Mathis, M. W., and Bethge, M. (2018). Deeplabcut: markerless pose estimation of user-defined body parts with deep learning. *Nature Neuroscience*, 21(9):1281–1289.

Mattar, M. G. and Daw, N. D. (2018). Prioritized memory access explains planning and hippocampal replay. *Nature Neuroscience*, 21(11):1609–1617.

Mattar, M. G. and Lengyel, M. (2022). Planning in the brain. *Neuron*, 110(6):914–934.

McNamee, D. C., Stachenfeld, K. L., Botvinick, M. M., and Gershman, S. J. (2021). Flexible modulation of sequence generation in the entorhinal–hippocampal system. *Nature Neuroscience*, 24(6):851–862.

Melnick, M. J. (1971). Effects of overlearning on the retention of a gross motor skill. *Research Quarterly. American Association for Health, Physical Education and Recreation*, 42(1):60–69.

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., and Teller, E. (1953). Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6):1087–1092.

Minxha, J., Adolphs, R., Fusi, S., Mamelak, A. N., and Rutishauser, U. (2020). Flexible recruitment of memory-based choice representations by the human medial frontal cortex. *Science*, 368(6498).

Morris, R. G. (1981). Spatial localization does not require the presence of local cues. *Learning and motivation*, 12(2):239–260.

Murray, I. and Adams, R. P. (2010). Slice sampling covariance hyperparameters of latent Gaussian models. *arXiv preprint arXiv:1006.0868*.

Naesseth, C. A., Ruiz, F. J., Linderman, S. W., and Blei, D. M. (2016). Reparameterization gradients through acceptance-rejection sampling algorithms. *arXiv preprint arXiv:1610.05683*.

Navarro, A. K., Frellsen, J., and Turner, R. E. (2017). The multivariate generalised von Mises distribution: inference and applications. In *Thirty-First AAAI Conference on Artificial Intelligence*.

Neal, R. M. (2012). *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media.

Nguyen, C. V., Li, Y., Bui, T. D., and Turner, R. E. (2017). Variational continual learning. *arXiv preprint arXiv:1710.10628*.

Nieh, E. H., Schottdorf, M., Freeman, N. W., Low, R. J., Lewallen, S., Koay, S. A., Pinto, L., Gauthier, J. L., Brody, C. D., and Tank, D. W. (2021). Geometry of abstract learned knowledge in the hippocampus. *Nature*, 595(7865):80–84.

Niv, Y. (2009). Reinforcement learning in the brain. *Journal of Mathematical Psychology*, 53(3):139–154.

Ólafsdóttir, H. F., Carpenter, F., and Barry, C. (2017). Task demands predict a dynamic switch in the content of awake hippocampal replay. *Neuron*, 96(4):925–935.

Opper, M. and Archambeau, C. (2009). The variational Gaussian approximation revisited. *Neural Computation*, 21(3):786–792.

Orbán, G., Berkes, P., Fiser, J., and Lengyel, M. (2016). Neural variability and sampling-based probabilistic representations in the visual cortex. *Neuron*, 92(2):530–543.

Orbán, G., Fiser, J., Aslin, R. N., and Lengyel, M. (2008). Bayesian learning of visual chunks by human observers. *Proceedings of the National Academy of Sciences*, 105(7):2745–2750.

Osawa, K., Swaroop, S., Jain, A., Eschenhagen, R., Turner, R. E., Yokota, R., and Khan, M. E. (2019). Practical deep learning with Bayesian principles. *arXiv preprint arXiv:1906.02506*.

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C. L., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., et al. (2022). Training language models to follow instructions with human feedback. *arXiv preprint arXiv:2203.02155*.

O'Doherty, J. E., Cardoso, M., Makin, J., and Sabes, P. (2017). Nonhuman primate reaching with multichannel sensorimotor cortex electrophysiology. *Zenodo http://doi. org/10.5281/zenodo*, 583331.

Pachitariu, M., Stringer, C., Dipoppa, M., Schröder, S., Rossi, L. F., Dalgleish, H., Carandini, M., and Harris, K. D. (2017). Suite2p: beyond 10,000 neurons with standard two-photon microscopy. *bioRxiv*.

Pan, P., Swaroop, S., Immer, A., Eschenhagen, R., Turner, R. E., and Khan, M. E. (2020). Continual deep learning by functional regularisation of memorable past. *arXiv preprint arXiv:2004.14070.*

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., et al. (2018). Inferring single-trial neural population dynamics using sequential auto-encoders. *Nature methods*, 15(10):805–815.

Pascanu, R., Li, Y., Vinyals, O., Heess, N., Buesing, L., Racanière, S., Reichert, D., Weber, T., Wierstra, D., and Battaglia, P. (2017). Learning model-based planning from scratch. *arXiv preprint arXiv:1707.06170.*

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. (2011). Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830.

Peshkin, L. and Shelton, C. R. (2002). Learning from scarce experience. *arXiv preprint cs/0204043.*

Peters, A. J., Chen, S. X., and Komiyama, T. (2014). Emergence of reproducible spatiotemporal activity during motor learning. *Nature*, 510(7504):263–267.

Peyrache, A. and Buzsáki, G. (2015). Extracellular recordings from multi-site silicon probes in the anterior thalamus and subicular formation of freely moving mice. *CRCNS.org.* Dataset. https://doi.org/10.6080/K0G15XS1.

Peyrache, A., Lacroix, M. M., Petersen, P. C., and Buzsáki, G. (2015). Internally organized mechanisms of the head direction sense. *Nature Neuroscience*, 18(4):569–575.

Pfeiffer, B. E. and Foster, D. J. (2013). Hippocampal place-cell sequences depict future paths to remembered goals. *Nature*, 497(7447):74–79.

Pillow, J. W., Shlens, J., Paninski, L., Sher, A., Litke, A. M., Chichilnisky, E., and Simoncelli, E. P. (2008). Spatio-temporal correlations and visual signalling in a complete neuronal population. *Nature*, 454(7207):995–999.

Piray, P. and Daw, N. D. (2021). Linear reinforcement learning in planning, grid fields, and cognitive control. *Nature communications*, 12(1):1–20.

Qin, Y.-L., McNaughton, B. L., Skaggs, W. E., and Barnes, C. A. (1997). Memory reprocessing in corticocortical and hippocampocortical neuronal ensembles. *Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences*, 352(1360):1525–1533.

Quinonero-Candela, J. and Rasmussen, C. E. (2005). A unifying view of sparse approximate Gaussian process regression. *The Journal of Machine Learning Research*, 6:1939–1959.

Rae, J. W., Borgeaud, S., Cai, T., Millican, K., Hoffmann, J., Song, F., Aslanides, J., Henderson, S., Ring, R., Young, S., et al. (2021). Scaling language models: Methods, analysis & insights from training Gopher. *arXiv preprint arXiv:2112.11446.*

Rao, R. P. (2004). Bayesian computation in recurrent neural circuits. *Neural computation*, 16(1):1–38.

Rasmussen, C. E. and Williams, C. K. (2006). *Gaussian processes for machine learning*, volume 2. MIT press Cambridge, MA.

Recanatesi, S., Ocker, G. K., Buice, M. A., and Shea-Brown, E. (2019). Dimensionality in recurrent spiking networks: global trends in activity and local origins in connectivity. *PLoS computational biology*, 15(7):e1006446.

Rey, L. A. P., Menkovski, V., and Portegies, J. W. (2019). Diffusion variational autoencoders. *arXiv preprint arXiv:1901.08991.*

Rezende, D. J., Mohamed, S., and Wierstra, D. (2014). Stochastic backpropagation and approximate inference in deep generative models. In *International conference on machine learning*, pages 1278–1286. PMLR.

Rezende, D. J., Papamakarios, G., Racanière, S., Albergo, M. S., Kanwar, G., Shanahan, P. E., and Cranmer, K. (2020). Normalizing flows on tori and spheres. *arXiv preprint arXiv:2022.02428.*

Ritter, H., Botev, A., and Barber, D. (2018). Online structured Laplace approximations for overcoming catastrophic forgetting. *arXiv preprint arXiv:1805.07810.*

Rokni, U., Richardson, A. G., Bizzi, E., and Seung, H. S. (2007). Motor learning with unstable neural representations. *Neuron*, 54(4):653–666.

Roweis, S. and Ghahramani, Z. (1999). A unifying review of linear Gaussian models. *Neural computation*, 11(2):305–345.

Rubin, A., Sheintuch, L., Brande-Eilat, N., Pinchasof, O., Rechavi, Y., Geva, N., and Ziv, Y. (2019). Revealing neural correlates of behavior without behavioral measurements. *Nature communications*, 10:1–14.

Rule, M. E., Loback, A. R., Raman, D. V., Driscoll, L. N., Harvey, C. D., and O'Leary, T. (2020). Stable task information from an unstable neural population. *Elife*, 9:e51121.

Rule, M. E., O'Leary, T., and Harvey, C. D. (2019). Causes and consequences of representational drift. *Current Opinion in Neurobiology*, 58:141–147.

Russek, E., Acosta-Kane, D., van Opheusden, B., Mattar, M. G., and Griffiths, T. (2022). Time spent thinking in online chess reflects the value of computation. *PsyArXiv.*

Rutten, V., Bernacchia, A., Sahani, M., and Hennequin, G. (2020). Non-reversible Gaussian processes for identifying latent dynamical structure in neural data. In *Advances in Neural Information Processing Systems*, volume 33.

Saha, G., Garg, I., and Roy, K. (2021). Gradient projection memory for continual learning. *arXiv preprint arXiv:2103.09762.*

Samborska, V., Butler, J. L., Walton, M. E., Behrens, T. E., and Akam, T. (2022). Complementary task representations in hippocampus and prefrontal cortex for generalizing the structure of problems. *Nature Neuroscience*, 25(10):1314–1326.

Santhanam, G., Ryu, S. I., Yu, B. M., Afshar, A., and Shenoy, K. V. (2006). A high-performance brain–computer interface. *Nature*, 442(7099):195–198.

Sauerbrei, B. A., Guo, J.-Z., Cohen, J. D., Mischiati, M., Guo, W., Kabra, M., Verma, N., Mensh, B., Branson, K., and Hantman, A. W. (2020). Cortical pattern generation during dexterous movement is input-driven. *Nature*, 577(7790):386–391.

Saxe, A., Nelli, S., and Summerfield, C. (2021). If deep learning is the answer, what is the question? *Nature Reviews Neuroscience*, 22(1):55–67.

Saxena, S. and Cunningham, J. P. (2019). Towards the neural population doctrine. *Current opinion in neurobiology*, 55:103–111.

Schimel, M., Kao, T.-C., Jensen, K. T., and Hennequin, G. (2021). iLQR-VAE: control-based learning of input-driven dynamics with applications to neural data. In *International Conference on Learning Representations*.

Schoonover, C. E., Ohashi, S. N., Axel, R., and Fink, A. J. (2021). Representational drift in primary olfactory cortex. *Nature*, 594(7864):541–546.

Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., et al. (2020). Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609.

Schultz, W., Dayan, P., and Montague, P. R. (1997). A neural substrate of prediction and reward. *Science*, 275(5306):1593–1599.

Schwartenbeck, P., Baram, A., Liu, Y., Mark, S., Muller, T., Dolan, R., Botvinick, M., Kurth-Nelson, Z., and Behrens, T. (2021). Generative replay for compositional visual understanding in the prefrontal-hippocampal circuit. *bioRxiv*.

Schwarz, J., Czarnecki, W., Luketina, J., Grabska-Barwinska, A., Teh, Y. W., Pascanu, R., and Hadsell, R. (2018). Progress & compress: A scalable framework for continual learning. In *International Conference on Machine Learning*, pages 4528–4537. PMLR.

Seelig, J. D. and Jayaraman, V. (2015). Neural dynamics for landmark orientation and angular path integration. *Nature*, 521(7551):186–191.

Shepard, R. N. and Metzler, J. (1971). Mental rotation of three-dimensional objects. *Science*, 171:701–703.

Shin, H., Lee, J. K., Kim, J., and Kim, J. (2017). Continual learning with deep generative replay. *arXiv preprint arXiv:1705.08690*.

Shin, J. D., Tang, W., and Jadhav, S. P. (2019). Dynamics of awake hippocampal-prefrontal replay for spatial learning and memory-guided decision making. *Neuron*, 104(6):1110–1125.

Sizemore, M. and Perkel, D. J. (2011). Premotor synaptic plasticity limited to the critical period for song learning. *Proceedings of the National Academy of Sciences*, 108(42):17492–17497.

Sohn, H., Narain, D., Meirhaeghe, N., and Jazayeri, M. (2019). Bayesian computation through cortical latent dynamics. *Neuron*, 103(5):934–947.

Sola, J., Deray, J., and Atchuthan, D. (2018). A micro Lie theory for state estimation in robotics. *arXiv preprint arXiv:1812.01537*.

Solway, A. and Botvinick, M. M. (2012). Goal-directed decision making as probabilistic inference: a computational framework and potential neural correlates. *Psychological review*, 119(1):120.

Steinmetz, N. A., Aydin, C., Lebedeva, A., Okun, M., Pachitariu, M., Bauza, M., Beau, M., Bhagat, J., Böhm, C., Broux, M., et al. (2021). Neuropixels 2.0: A miniaturized high-density probe for stable, long-term brain recordings. *Science*, 372(6539).

Straub, J. (2017). Bayesian inference with the von-Mises-Fisher distribution in 3d.

Stringer, C., Pachitariu, M., Steinmetz, N., Carandini, M., and Harris, K. D. (2019). High-dimensional geometry of population responses in visual cortex. *Nature*, 571(7765):361–365.

Sun, C., Shrivastava, A., Singh, S., and Gupta, A. (2017). Revisiting unreasonable effectiveness of data in deep learning era. In *Proceedings of the IEEE international conference on computer vision*, pages 843–852.

Sussillo, D. and Barak, O. (2013). Opening the black box: low-dimensional dynamics in high-dimensional recurrent neural networks. *Neural computation*, 25:626–649.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Teh, Y., Bapst, V., Czarnecki, W. M., Quan, J., Kirkpatrick, J., Hadsell, R., Heess, N., and Pascanu, R. (2017). Distral: Robust multitask reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 30.

Tikhonov, A. N. (1943). On the stability of inverse problems. In *Dokl. Akad. Nauk SSSR*, volume 39, pages 195–198.

Titsias, M. (2009). Variational learning of inducing variables in sparse Gaussian processes. In *Artificial intelligence and statistics*, pages 567–574. PMLR.

Titsias, M. and Lawrence, N. D. (2010). Bayesian Gaussian process latent variable model. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 844–851. JMLR Workshop and Conference Proceedings.

Titsias, M. K., Schwarz, J., Matthews, A. G. d. G., Pascanu, R., and Teh, Y. W. (2020). Functional regularisation for continual learning with Gaussian processes. In *International Conference on Learning Representations*.

Tomko, G. J. and Crapper, D. R. (1974). Neuronal variability: non-stationary responses to identical visual stimuli. *Brain research*, 79(3):405–418.

Tosi, A., Hauberg, S., Vellido, A., and Lawrence, N. D. (2014). Metrics for probabilistic geometries. *arXiv preprint arXiv:1411.7432*.

Tseran, H., Khan, M. E., Harada, T., and Bui, T. D. (2018). Natural variational continual learning. In *Continual Learning Workshop NeurIPS*, volume 2.

Turner-Evans, D. B. (2020). Kir.zip. *Janelia Research Campus*. Dataset. https://doi.org/10.25378/janelia.12490325.v1.

Turner-Evans, D. B., Jensen, K. T., Ali, S., Paterson, T., Sheridan, A., Ray, R. P., Wolff, T., Lauritzen, J. S., Rubin, G. M., Bock, D. D., and Jayaraman, V. (2020). The neuroanatomical ultrastructure and function of a biological ring attractor. *Neuron*, 108(1):145–163.

Ulrich, G. (1984). Computer generation of distributions on the M-sphere. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, 33(2):158–163.

Urtasun, R., Fleet, D. J., Geiger, A., Popović, J., Darrell, T. J., and Lawrence, N. D. (2008). Topologically-constrained latent variable models. In *Proceedings of the 25th international conference on Machine learning*, pages 1080–1087.

van de Ven, G. M., Siegelmann, H. T., and Tolias, A. S. (2020). Brain-inspired replay for continual learning with artificial neural networks. *Nature Communications*, 11(1):1–14.

van de Ven, G. M. and Tolias, A. S. (2018). Generative replay with feedback connections as a general strategy for continual learning. *arXiv preprint arXiv:1809.10635*.

van de Ven, G. M. and Tolias, A. S. (2019). Three scenarios for continual learning. *arXiv preprint arXiv:1904.07734*.

van de Ven, G. M., Trouche, S., McNamara, C. G., Allen, K., and Dupret, D. (2016). Hippocampal offline reactivation consolidates recently formed cell assembly patterns during sharp wave-ripples. *Neuron*, 92(5):968–974.

van Loan, C. F. and Pitsianis, N. (1993). Approximation with Kronecker products. In *Linear algebra for large scale and real-time applications*, pages 293–314. Springer.

van Opheusden, B., Galbiati, G., Kuperwajs, I., Bnaya, Z., Ma, W. J., et al. (2021). Revealing the impact of expertise on human planning with a two-player board game. *PsyArXiv*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30.

Vul, E., Goodman, N., Griffiths, T. L., and Tenenbaum, J. B. (2014). One and done? optimal decisions from very few samples. *Cognitive science*, 38(4):599–637.

Wainwright, M. J. and Jordan, M. I. (2008). *Graphical models, exponential families, and variational inference*. Now Publishers Inc.

Wang, J. X., Kurth-Nelson, Z., Kumaran, D., Tirumala, D., Soyer, H., Leibo, J. Z., Hassabis, D., and Botvinick, M. (2018). Prefrontal cortex as a meta-reinforcement learning system. *Nature Neuroscience*, 21(6):860–868.

Wang, J. X., Kurth-Nelson, Z., Tirumala, D., Soyer, H., Leibo, J. Z., Munos, R., Blundell, C., Kumaran, D., and Botvinick, M. (2016). Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763*.

Wang, P. Z. and Wang, W. Y. (2019). Riemannian normalizing flow on variational Wasserstein autoencoder for text modeling. *arXiv preprint arXiv:1904.02399*.

Wenzel, F., Roth, K., Veeling, B. S., Świątkowski, J., Tran, L., Mandt, S., Snoek, J., Salimans, T., Jenatton, R., and Nowozin, S. (2020). How good is the Bayes posterior in deep neural networks really? *arXiv preprint arXiv:2002.02405*.

Whittington, J. C., Muller, T. H., Mark, S., Chen, G., Barry, C., Burgess, N., and Behrens, T. E. (2020). The Tolman-Eichenbaum machine: Unifying space and relational memory through generalization in the hippocampal formation. *Cell*, 183(5):1249–1263.

Widloski, J. and Foster, D. J. (2022). Flexible rerouting of hippocampal replay sequences around changing barriers in the absence of global place field remapping. *Neuron*, 110(9):1547–1558.

Willett, F. R., Avansino, D. T., Hochberg, L. R., Henderson, J. M., and Shenoy, K. V. (2021). High-performance brain-to-text communication via handwriting. *Nature*, 593(7858):249–254.

Williams, C. and Rasmussen, C. (1995). Gaussian processes for regression. In *Advances in Neural Information Processing Systems*, volume 8.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256.

Wilson, A. and Nickisch, H. (2015). Kernel interpolation for scalable structured Gaussian processes (KISS-GP). In *International Conference on Machine Learning*, pages 1775–1784. PMLR.

Wilson, A. G., Dann, C., and Nickisch, H. (2015). Thoughts on massively scalable Gaussian processes. *arXiv preprint arXiv:1511.01870*.

Wilson, J. J., Alexandre, N., Trentin, C., and Tripodi, M. (2018). Three-dimensional representation of motor space in the mouse superior colliculus. *Current Biology*, 28(11):1744–1755.e12.

Wilson, M. A. and McNaughton, B. L. (1994). Reactivation of hippocampal ensemble memories during sleep. *Science*, 265(5172):676–679.

Wu, A., Pashkovski, S., Datta, S. R., and Pillow, J. W. (2018). Learning a latent manifold of odor representations from neural responses in piriform cortex. In *Advances in Neural Information Processing Systems*.

Wu, A., Roy, N. A., Keeley, S., and Pillow, J. W. (2017a). Gaussian process based nonlinear latent structure discovery in multivariate spike train data. In *Advances in Neural Information Processing Systems*, volume 30.

Wu, C.-T., Haggerty, D., Kemere, C., and Ji, D. (2017b). Hippocampal awake replay in fear memory retrieval. *Nature Neuroscience*, 20(4):571–580.

Xu, T., Yu, X., Perlik, A. J., Tobin, W. F., Zweig, J. A., Tennant, K., Jones, T., and Zuo, Y. (2009). Rapid formation and selective stabilization of synapses for enduring motor memories. *Nature*, 462(7275):915–919.

Yang, G., Pan, F., and Gan, W.-B. (2009). Stably maintained dendritic spines are associated with lifelong memories. *Nature*, 462(7275):920–924.

Yang, G. R., Joglekar, M. R., Song, H. F., Newsome, W. T., and Wang, X.-J. (2019). Task representations in neural networks trained to perform many cognitive tasks. *Nature Neuroscience*, 22(2):297–306.

Ye, J. and Pandarinath, C. (2021). Representation learning for neural population activity with neural data transformers. *arXiv preprint arXiv:2108.01210*.

Yu, B. M., Cunningham, J. P., Santhanam, G., Ryu, S., Shenoy, K. V., and Sahani, M. (2008). Gaussian-process factor analysis for low-dimensional single-trial analysis of neural population activity. In *Advances in Neural Information Processing Systems*, volume 21.

Yu, B. M., Cunningham, J. P., Santhanam, G., Ryu, S. I., Shenoy, K. V., and Sahani, M. (2009). Gaussian-process factor analysis for low-dimensional single-trial analysis of neural population activity. *Journal of Neurophysiology*, 102(1):614–635.

Yu, J. Y. and Frank, L. M. (2015). Hippocampal–cortical interaction in decision making. *Neurobiology of learning and memory*, 117:34–41.

Yuille, A. and Kersten, D. (2006). Vision as Bayesian inference: analysis by synthesis? *Trends in cognitive sciences*, 10(7):301–308.

Zeng, G., Chen, Y., Cui, B., and Yu, S. (2019). Continual learning of context-dependent processing in neural networks. *Nature Machine Intelligence*, 1(8):364–372.

Zenke, F., Poole, B., and Ganguli, S. (2017). Continual learning through synaptic intelligence. In *International Conference on Machine Learning*, pages 3987–3995. PMLR.

Zhao, Y. and Park, I. M. (2017). Variational latent Gaussian process for recovering single-trial dynamics from population spike trains. *Neural computation*, 29(5):1293–1316.

Zhao, Y., Yates, J. L., Levi, A. J., Huk, A. C., and Park, I. M. (2020). Stimulus-choice (mis) alignment in primate area MT. *PLoS computational biology*, 16(5):e1007614.

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

Zintgraf, L., Shiarlis, K., Igl, M., Schulze, S., Gal, Y., Hofmann, K., and Whiteson, S. (2019). VariBAD: A very good method for Bayes-adaptive deep RL via meta-learning. *arXiv preprint arXiv:1910.08348*.

Ziv, Y., Burns, L. D., Cocker, E. D., Hamel, E. O., Ghosh, K. K., Kitch, L. J., El Gamal, A., and Schnitzer, M. J. (2013). Long-term dynamics of CA1 hippocampal place codes. *Nature Neuroscience*, 16(3):264–266.

Appendix A

Bayesian GPFA

Further analyses of preparatory dynamics in the primate reaching task

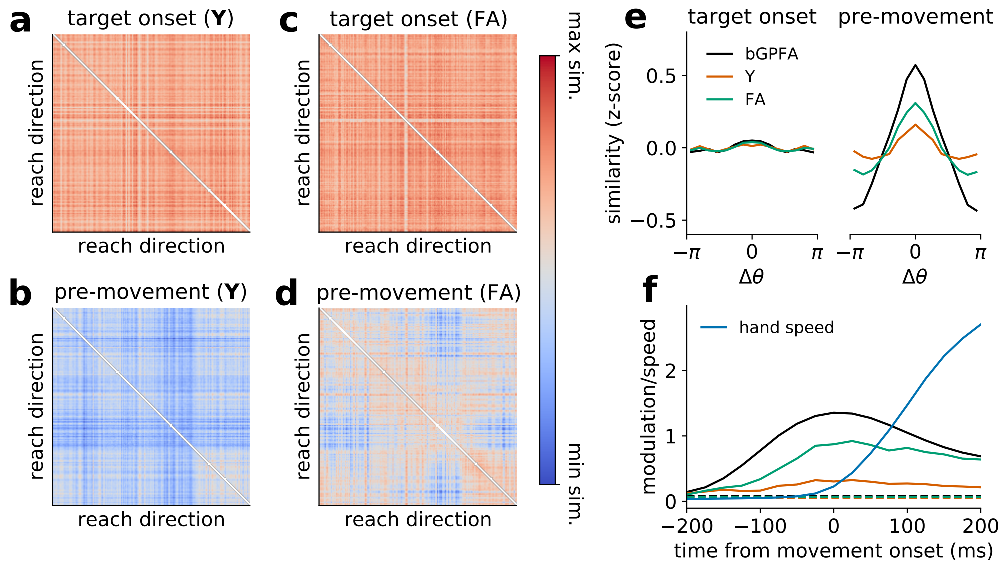

Figure A.1 **Further analyses of M1 preparatory dynamics. (a-d)** Similarity matrix of raw neural activity $\boldsymbol{Y}$ (a & b) and latent states found by FA (c & d) at target onset (a & c) and 75 ms prior to movement onset (b & d), with analyses performed as in Figure 3.3f. **(e)** z-scored similarity as a function of difference in reach direction; here, the mean similarity across pairs of reaches is shown at target onset (left) and 75 ms prior to movement onset (right). The bGPFA latent states show much stronger modulation than either raw neural activity ($\boldsymbol{Y}$) or latent states from FA. **(f)** Modulation of similarity by reach direction as a function of time from movement onset. Modulation was defined as the difference between maximum and minimum z-scored similarity as a function of difference in reach direction (peak-to-trough in panel e). Blue solid line indicates the z-scored hand speed, confirming the absence of premature movement relative to our definition of movement onset. bGPFA latent similarity increases well before hand speed and starts decreasing substantially before the hand speed peaks. Dashed lines indicate modulation at target onset for each method.

We performed analyses as in Figure 3.3f using the raw data ($\boldsymbol{Y}$) and using factor analysis (FA) with 20 latent dimensions instead of using the bGPFA latent states. The raw data $\boldsymbol{Y}$ showed a high degree of similarity at target onset compared to movement onset, but little discernable structure as a function of reach direction at either point in time (Figure A.1a-b).

While the FA latent distances exhibited no modulation by reach direction at target onset, FA did discover weak modulation at movement onset (Figure A.1a-b). This is qualitatively consistent with our results using bGPFA but with a lower signal to noise ratio. Here and in Section 3.1.3, we defined movement onset as the first time during a reach where the cursor velocity exceeded $0.025\,\mathrm{m\,s^{-1}}$, and we observed little to no quantifiable movement before this point (Figure A.1f). We also discarded 'trials' with premature movement for all analyses here and in Section 3.1.3, which we defined as reaches with a reaction time of 75 ms or less.

To quantify and compare how neural activity was modulated by the similarity of reach directions for different analysis methods, we first computed z-scores of the similarity matrices for both the bGPFA latent states, raw activity, and the latent states from FA. z-scores were calculated as $z = (\boldsymbol{S} - mean(\boldsymbol{S}))/std(\boldsymbol{S})$ for each similarity matrix $\boldsymbol{S}$, and the diagonal elements were excluded for this analysis. We then computed the mean of the z-scored pairwise similarities as a function of difference in reach direction across all pairs of 762 reaches. We found that none of the datasets exhibited notable modulation at target onset (Figure A.1e). In contrast, the neural data exhibited modulation by reach similarity 75 ms prior to movement onset. This modulation was strongest for the bGPFA latent states followed by the FA latents, and the modulation by reach similarity was very weak for the raw neural activity (Figure A.1e). To see how this modulation by reach direction varied as a function of time from movement onset, we computed the difference (δz) between the maximum and minimum of the modulation curves and repeated this analysis at different times prior to and during the reach process. We found that the modulation in neural activity space increased before any detectable movement, with bGPFA showing the strongest signal followed by factor analysis and then the raw activity (Figure A.1f). Indeed, the bGPFA latent modulation was maximized near movement onset, while the reach speed did not peak until several hundred milliseconds after movement onset where bGPFA latent trajectories have started to converge again. Taken together, these results confirm that our analyses of bGPFA preparatory states do not reflect premature movement onset, and that they are not artifacts of the temporal correlations introduced by our GP prior since noisier but qualitatively similar results arise from the use of factor analysis.

For further comparison with non-Bayesian Gaussian process factor analysis, we also fitted Poisson GPFA (P-GPFA) to the primate dataset using our variational inference approach for scalability but without a prior over $\boldsymbol{C}$ (Section 3.1.2). For this analysis, we used 16 latent dimensions as inferred by bGPFA, and we subselected latent processes with timescales ≤ 200 ms to study the putatively fast motor preparation as for bGPFA. We then orthogonalized the latent dimensions by performing an SVD on the loading matrix (see Yu et al., 2009 for details). Similar to our results from bGPFA, we found that the latent trajectories became modulated by movement direction prior to movement onset (Figure A.2a) with a similar degree of modulation to bGPFA ($\delta z = 1.01$ for bGPFA; $\delta z = 0.99$ for P-GPFA). When visualizing the latent trajectories for the example reaches considered in Figure 3.3e, we also found that these

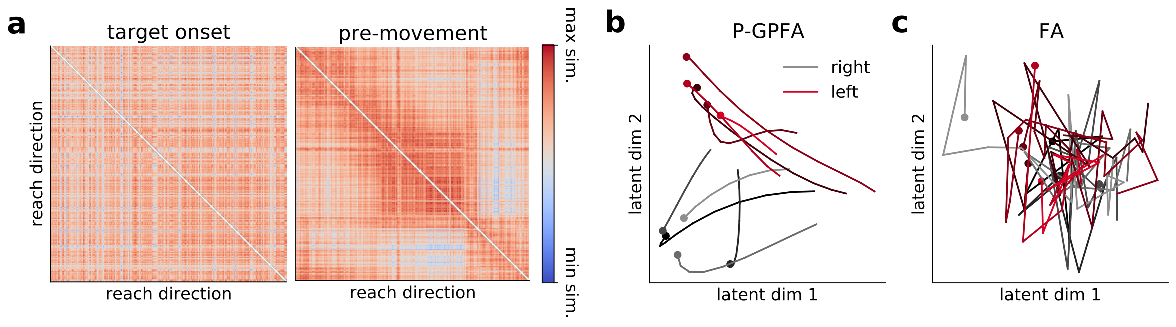

Figure A.2 **Analyses of M1 dynamics with GPFA and FA. (a)** P-GPFA was fitted to data recorded from M1 during the self-paced reaching task. We computed the similarity matrix of the latent state at stimulus onset, showing no obvious structure (left), and 75 ms prior to movement onset, showing modulation by reach direction (right). Reaches are sorted by reach direction along both axes. **(b)** Example P-GPFA latent trajectories in the two principal latent dimensions for five rightward reaches (grey) and five leftward reaches (red). Trajectories are plotted from the appearance of the stimulus until movement onset (circles; the trajectories shown are the same as in Figure 3.3e). **(c)** As in (b), now showing latent trajectories inferred by factor analysis. These exhibit less discernable structure due to the lack of an explicit smoothness prior.

diverged by reach direction (Figure A.2b), similar to the bGPFA latent trajectories and unlike vanilla factor analysis, which assumes temporal independence *a priori* (Figure A.2c)

S1 activity

In this section, we compare the latent processes inferred for M1 dynamics to those inferred by applying bGPFA to the recordings from S1. In contrast to the clustering by reach direction in M1, there was less obvious modulation by movement direction in S1 prior to movement onset (Figure A.3). To quantify this, we again computed the degree of modulation prior to movement onset, which was $\delta z = 0.38$ for S1 compared to $\delta z = 1.01$ for M1. This is also consistent with our decoding analyses in Figure 3.3b, which showed that activity in M1 predicts movement 100-150 ms into the future while S1 activity is not predictive of future kinematics to the same extent.

Further reaction time analyses

For analyses of correlations between latent distances and reaction times, we only considered reaches with a reaction time of at least 125 ms and at most 425 ms, which retained 712 of 762 reaches (Figure A.4a). This is because very long reaction times may reflect the monkey not being fully engaged with the task during those reaches, and very short reaction times may reflect

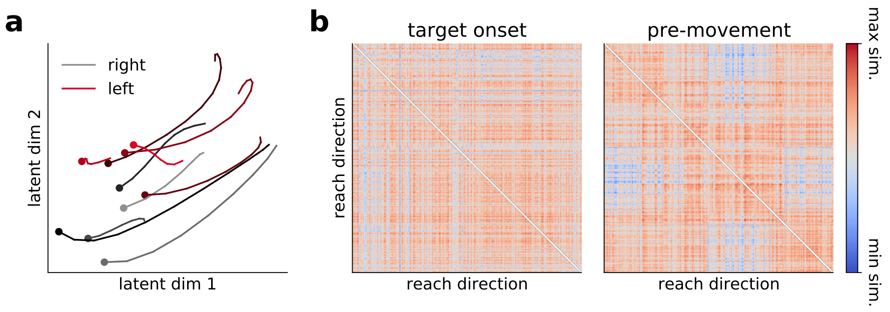

Figure A.3 **Analyses of S1 dynamics by reach direction.** **(a)** bGPFA was fitted to data recorded from S1 during the self-paced reaching task. The panel shows example latent trajectories in the two most informative latent dimensions for five rightward reaches (grey) and five leftward reaches (red). Trajectories are plotted from the appearance of the stimulus until movement onset (circles; the trajectories shown are the same as in Figure 3.3e). Unlike the M1 latent trajectories, there is no obvious clustering by reach direction during movement preparation. **(b)** Similarity matrix of the latent state at stimulus onset (left) and 75 ms prior to movement onset (right). Reaches are sorted by reach direction along both axes, and there is no obvious structure in either similarity matrix in contrast to the results for the M1 recordings.

spurious movement. To confirm that our finding of a strong correlation between latent distance and reaction time in Figure 3.3g is not an artifact of the temporal correlations introduced by the bGPFA generative model, we generated a synthetic control distribution. Here we drew 50,000 synthetic latent trajectories from our learned generative model with trajectory durations matched to those observed experimentally on each trial. We then computed mean preparatory states and latent distances to preparatory states as in the experimental data (Section 3.1.3) and computed correlations with the experimental reaction times. We found a mean correlation of 0.02 and a range of -0.14 to 0.18 in the synthetic data, suggesting that our generative model may introduce weak correlations between latent distances and reaction times. However, the experimentally observed correlation of 0.45 was much larger than what could be expected by chance. This verifies our finding that the distance from the latent state at target onset to the corresponding preparatory state has behavioral relevance, with better initial states leading to shorter reaction times.

Although we already find a fairly strong relationship between latent distances and reaction times, it is worth noting that several additional considerations may further improve such predictions. Notably, our naive Euclidean distance metric could be improved by instead defining a metric based on the probabilistic model itself (Tosi et al., 2014). Additionally, while we categorize reaches by reach direction, reaches in the same direction can still have different start and end points on the grid (Figure 3.3a), leading to different posture and muscle activations, which is likely to significantly affect neural activity. Our analysis by reach direction therefore only represents a coarse categorization of the rich behavioral space, and it remains to be seen how neural activity and latent trajectories are affected by e.g. posture during the task.

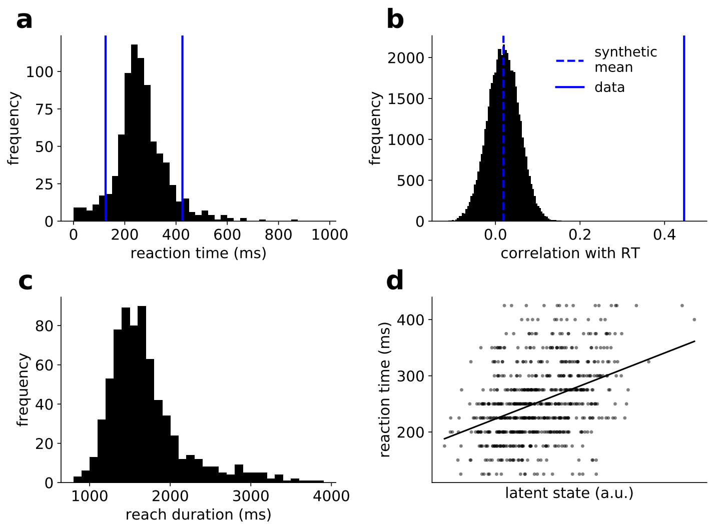

Figure A.4 **Further reaction time analyses.** **(a)** Histogram of reaction times across all succesful reaches. For our correlation analyses, we only considered reaches with a reaction time between 125 ms and 425 ms (blue vertical lines). **(b)** Pearson correlations between distance to prep state and reaction time in synthetic data. Histogram corresponds to correlations between the true reaction times and 50,000 draws from the learned generative model. Blue dashed line indicates mean across all synthetic datasets (0.02), which is much smaller than the observed correlation in the experimental data of 0.45 (blue solid line). **(c)** Histogram of reach durations for all reaches with a reaction time between 125 ms and 425 ms. **(d)** Plot of reaction time against the value of a long timescale latent dimension at target onset ($\tau = 1.4$ s, $\rho = 0.40$).

Finally we considered how the dynamics of long-timescale latent processes relate to the reaction time across trials (c.f. Section 3.1.3). Here we found that the two slowest dimensions had timescales of $\tau = 1.4$ s and $\tau = 1.7$ s, similar to the timescale of single reaches, which generally lasted between 1 and 2 seconds (Figure A.4c). Intriguingly, the latent state in these dimensions at target onset was predictive of reaction time, with Pearson correlations of $\rho = 0.40$ and $\rho = 0.36$ respectively (Figure A.4d). While the information about reaction time contained in these two dimensions was largely redundant, it was orthogonal to that encoded by the distance to preparatory state in the fast dimensions. In particular, a linear model had 19.9% variance explained from the distance to prep in fast dimensions, 15.7% variance explained from the slow latent dimension with the strongest correlation, and 28.7% when combining these two features, which corresponds to 80.7% of the additive value.

Latent dimensionality

In this section, we estimate the dimensionality of the primate data as a function of the offset between M1 and S1 spike times using both bGPFA and participation ratios computed on the basis of PCA (Recanatesi et al., 2019). The participation ratio is defined as

$$PR = \left(\sum_i \lambda_i\right)^2 / \sum_i \lambda_i^2, \tag{A.1}$$

where λ_i is the i^{th} eigenvalue of the covariance matrix $\boldsymbol{YY}^T$. When computing the participation ratio of the data as a function of the M1 spike time shift, we find that the dimensionality is minimized for a shift of 75-100 ms (Figure A.5). This suggests that the neural recordings can be explained more concisely when taking into account the offset in decoding between M1 and S1, which is consistent with the increased log likelihood after shifting the M1 spikes (Section 3.1.3).

This trend is not directly observable in the number of dimensions retained by bGPFA with and without a 100 ms shift of the M1 spike times (18.8 ± 0.19 vs 18.7 ± 0.20 respectively across 10 model fits). However, the discrete nature of this dimensionality measure makes it relatively insensitive to small effects since it relies on stochastic differences in the retention of a dimension with little information content. We therefore utilized the interpretation of the learned prior scale s_d^2 as a measure of variance explained (Appendix A) and defined a 'participation ratio' for bGPFA, similar to the PCA participation ratio considered above:

$$PR_{bGPFA} = \left(\sum_d s_d^2\right)^2 / \sum_d \left(s_d^2\right)^2. \tag{A.2}$$

Here we found a strong effect of shifting the M1 spike times, which reduced the dimensionality from $PR_{bGPFA} = 6.16 \pm 0.12$ to $PR_{bGPFA} = 5.62 \pm 0.06$ ($p = 0.001$). Additionally, we note that bGPFA explains the data with only a handful of latent dimensions (19 total dimensions; 6 when re-weighted as a participation ratio). This is much lower than the dimensionality of 127-129 estimated by the PCA participation ratio, which generally infers higher dimensionalities for noisier (more 'spherical') datasets.

Further validation of bGPFA on synthetic and biological data

In Figure 3.2a-b, we considered the performance of FA, GPFA and bGPFA on synthetic data with Gaussian noise. To further validate our method in a non-Gaussian setting relevant to the study of electrophysiological recordings, we also performed similar analyses on (i) synthetic data with Poisson noise and (ii) experimental recordings from the primate reaching task. In both cases, we compared FA, GPFA and bGPFA with Poisson noise, since such Poisson noise models are common in the neuroscience literature (Macke et al., 2012; Pandarinath et al., 2018; Wu et al., 2017a; Zhao and Park, 2017). We note that these non-conjugate models are all readily implemented within our inference framework as special cases of bGPFA.

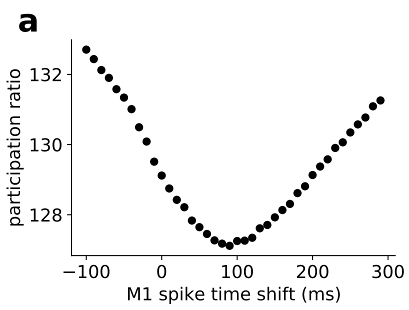

Figure A.5 **Neural dimensionality.** **(a)** Participation ratio (Equation A.1) as a function of temporal offset added to M1 spike times in the primate dataset.

Synthetic data In this section, we perform analyses similar to Figure 3.2b to validate bGPFA and its capacity for automatic relevance determination on synthetic data. We first generated a dataset drawn from the GPFA generative model but with a Poisson noise model after passing the activations through a softplus nonlinearity

$$\boldsymbol{Y} \sim Poisson\left(\log\left[1 + \exp \boldsymbol{C}\boldsymbol{X}\right]\right). \tag{A.3}$$

We then fitted Factor analysis, GPFA, and bGPFA with and without ARD to the resulting dataset, all with Poisson observation models and exponential non-linearities (Appendix A; we denote these Poisson models as 'P-FA' etc.). To quantify performance, we computed the cross-validated predictive log likelihood $\mathcal{L}_{pred} = \sum_{it} \log p_{Poisson}(y_{it}|f_{it})$, where i and t index neurons and time points in a held-out test set (results were similar when considering MSEs). When considering $\mathcal{L}_{pred}$ as a function of latent dimensionality, we found that both P-FA and P-GPFA exhibited a clear maximum at the true dimensionality of $D^* = 3$ while P-bGPFA without ARD was robust to overfitting, similar to our findings for the Gaussian models (Figure 3.2b). Finally, bGPFA with ARD was capable of automatically recovering this dimensionality as well as the maximum predictive performance achieved across the other models.

Experimental data We proceeded to perform an analysis as above on data from the self-paced monkey reaching dataset (O'Doherty et al., 2017). For this analysis, we used a smaller subset of the data for computational convenience, considering only 1000 timepoints but including all 200 neurons. We performed these analyses in 10-fold cross-validation, averaging performance over folds and repeating the entire analysis across 5 different random seeds. We again fitted P-FA and P-GPFA and found that these models exhibited a clear maximum in their predictive log likelihoods. As for the synthetic data, P-bGPFA without ARD was robust to overfitting, and the introduction of ARD allowed us to infer the optimal latent dimensionality of $D^* \approx 4$ as well as achieving optimal performance without *a priori* assumptions about the latent dimensionality. The robustness to overfitting of bGPFA both with and without ARD suggests that it could

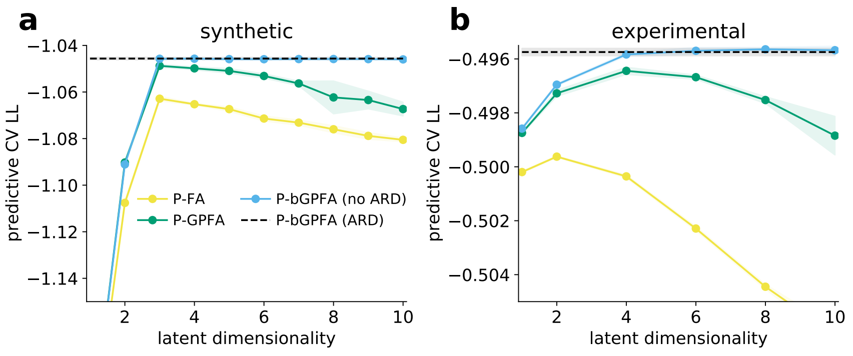

Figure A.6 **Bayesian GPFA applied to spike count data. (a)** Cross-validated predictive log likelihoods of factor analysis (yellow), GPFA (green), and Bayesian GPFA without ARD (blue) fitted to synthetic data with a ground truth dimensionality of three for different model dimensionalities. All methods used a Poisson observation model $p(y_{nt}|f_{nt})$. bGPFA with ARD recovered the performance of the optimal non-ARD models without requiring a search over latent dimensionalities (black). **(b)** As in (a), now for models applied to a subset of the monkey reaching data analyzed in Section 3.1.3.

also be a valuable tool in settings with large simultaneous recordings of thousands of neurons, which are becoming increasingly relevant with recent advances in neural recording technologies (Pachitariu et al., 2017; Steinmetz et al., 2021).

Taken together, these results further validate the utility of bGPFA on both synthetic and biological data with non-conjugate noise models. They also highlight the utility of automatic relevance determination in practice, where it obviates the need to perform extensive cross-validation to select an appropriate latent dimensionality for the experimental data.

Parameterizations of the approximate GP posterior

In this section, we compare different forms of the variational posterior $q(\boldsymbol{X})$ discussed in Section 3.1.2. For factorizing likelihoods, the optimal posterior takes the form

$$q(\boldsymbol{x}_d) \propto p(\boldsymbol{x}) \prod_t \mathcal{N}(x_t|g_t, v_t), \tag{A.4}$$

where g_t and v_t are variational parameters (Opper and Archambeau, 2009). Equation A.4 might therefore seem to be an appropriate form of the variational distribution $q(\boldsymbol{X})$. However, this formulation is computationally expensive.

Instead, we therefore consider approximate parameterizations of the form

$$q(\boldsymbol{x}_d) = \mathcal{N}(\boldsymbol{\mu}_d, \boldsymbol{\Sigma}_d) \tag{A.5}$$

$$\boldsymbol{\mu}_d = \boldsymbol{K}_d^{\frac{1}{2}} \boldsymbol{\nu}_d \tag{A.6}$$

$$\boldsymbol{\Sigma}_d = \boldsymbol{K}_d^{\frac{1}{2}} \boldsymbol{\Lambda}_d \boldsymbol{\Lambda}_d^T \boldsymbol{K}_d^{\frac{1}{2}}, \tag{A.7}$$

where $\boldsymbol{K}_d^{\frac{1}{2}}$ is a matrix square root of the prior covariance matrix $\boldsymbol{K}_d$, and $\boldsymbol{\nu}_d \in \mathbb{R}^T$ is a vector of variational parameters. This formulation simplifies the KL divergence term for each latent dimension in Equation 3.6 from

$$\mathrm{KL}[q(\boldsymbol{x}_d)||p(\boldsymbol{x}_d|\boldsymbol{t})] = \frac{1}{2}\left(\mathrm{Tr}(\boldsymbol{K}_d^{-1}\boldsymbol{\Sigma}_d) + \log|\boldsymbol{K}_d| - \log|\boldsymbol{\Sigma}_d| + \boldsymbol{\mu}_d^T \boldsymbol{K}_d^{-1}\boldsymbol{\mu}_d - T\right) \tag{A.8}$$

to

$$\mathrm{KL}[q(\boldsymbol{x}_d)||p(\boldsymbol{x}_d|\boldsymbol{t})] = \frac{1}{2}\left(||\boldsymbol{\Lambda}_d||_{\mathrm{F}}^2 - 2\log|\boldsymbol{\Lambda}_d| + ||\boldsymbol{\nu}_d||^2 - T\right). \tag{A.9}$$

In the following, we drop the $\cdot_d$ subscript to remove clutter, and we use the notation $\boldsymbol{\Psi} = \mathrm{diag}(\psi_1, ..., \psi_T)$ with positive elements $\psi_t > 0$, to denote a positive definite diagonal matrix.

Square root of the prior covariance

For a stationary prior covariance $\boldsymbol{K}$, we can directly parameterize $\boldsymbol{K}^{\frac{1}{2}}$ by taking the square root of $k(\cdot,\cdot)$ in the Fourier domain and computing the inverse Fourier transform. For the RBF kernel used in this work we get

$$k(t_i, t_j) = \exp\left(-\frac{(t_i - t_j)^2}{2\tau^2}\right) \tag{A.10}$$

$$k^{\frac{1}{2}}(t_i, t_j) = \left(\frac{2}{\pi}\right)^{\frac{1}{4}} \left(\frac{\delta t}{\tau}\right)^{\frac{1}{2}} \exp\left(-\frac{(t_i - t_j)^2}{\tau^2}\right). \tag{A.11}$$

In this expression, δt is the time difference between consecutive data points, we have assumed a signal variance of 1 in the prior kernel, and we note that our parameterization only gives rise to the exact matrix square root of the RBF kernel in the limit where $T \gg \tau$. Note that this is the case in the present work since $T \approx 30$ minutes is much larger than the longest timescales learned by bGPFA ($\tau \approx 2$ s). For most experiments in neuroscience, observations are binned such that time is on a regularly spaced grid and our parameterization can be applied directly. In other cases, kernel interpolation should first be used to construct a covariance matrix with Toeplitz structure (Wilson and Nickisch, 2015; Wilson et al., 2015).

Parameterization of the posterior covariance

We now proceed to describe the various parameterizations of $\boldsymbol{\Lambda}$ whose performance is compared in Figure A.7. Other parameterizations are explored in Challis and Barber (2013).

Diagonal $\boldsymbol{\Lambda}$ We parameterize each latent dimension with $\boldsymbol{\Lambda} = \boldsymbol{\Psi}$. This gives rise to a KL term:

$$2\text{KL}[q(\boldsymbol{x})||p(\boldsymbol{x})] = \sum_t \psi_t^2 + ||\boldsymbol{\nu}||^2 - T - 2\sum_t \log\psi_t. \tag{A.12}$$

We can compute $\boldsymbol{\Lambda v}$ in linear time since $\boldsymbol{\Lambda}$ is diagonal which allows for cheap differentiable sampling:

$$\boldsymbol{\eta} \sim \mathcal{N}(0, \boldsymbol{I}) \tag{A.13}$$

$$\text{sample} = \boldsymbol{K}^{\frac{1}{2}}(\boldsymbol{\Lambda\eta} + \boldsymbol{\nu}), \tag{A.14}$$

where the multiplication by $\boldsymbol{K}^{\frac{1}{2}}$ is done in $\mathcal{O}(T\log T)$ time in the Fourier domain.

Circulant $\boldsymbol{\Lambda}$ We parameterize each latent dimension with $\boldsymbol{\Lambda} = \boldsymbol{\Psi C}$. Here, $\boldsymbol{C} \in \mathbb{R}^{T\times T}$ is a positive definite circulant matrix with $1 + \frac{T}{2}$ (integer division) free parameters, which we parameterize directly in the Fourier domain as $\hat{\boldsymbol{c}} = \text{rfft}(\boldsymbol{c}) \in \mathbb{R}^{1+T/2}$, where $\boldsymbol{c}$ is the first column of $\boldsymbol{C}$ with $\hat{\boldsymbol{c}} \geq 0$ elementwise. We compute the KL as

$$2\text{KL}[q(\boldsymbol{x})||p(\boldsymbol{x})] = \left(\sum_t c_t^2\right)\left(\sum_t \psi_t^2\right) + ||\boldsymbol{\nu}||^2 - T - 2\sum_t \log\psi_t - 2\log|\boldsymbol{C}| \tag{A.15}$$

$$\log|\boldsymbol{C}| = \log\hat{c}_1 + \log\hat{c}_{\frac{T}{2}+1} + 2\sum_{i=2}^{\frac{T}{2}} \log\hat{c}_i \qquad \text{(even T)} \tag{A.16}$$

$$\log|\boldsymbol{C}| = \log\hat{c}_1 + 2\sum_{i=2}^{\frac{T+1}{2}} \log\hat{c}_i \qquad \text{(odd T)}, \tag{A.17}$$

where $\boldsymbol{c} = \text{irfft}(\hat{\boldsymbol{c}})$. We can sample differentiably in $\mathcal{O}(T\log T)$ time by computing

$$\boldsymbol{\eta} \sim \mathcal{N}(0, \boldsymbol{I}) \tag{A.18}$$

$$\boldsymbol{C\eta} = \text{irfft}(\hat{\boldsymbol{c}} \odot \text{rfft}(\boldsymbol{\eta})) \tag{A.19}$$

$$\text{sample} = \boldsymbol{K}^{\frac{1}{2}}(\boldsymbol{\Psi C\eta} + \boldsymbol{\nu}), \tag{A.20}$$

where $\odot$ denotes the complex element-wise product.

Low-rank $\boldsymbol{\Lambda}$ We let $\boldsymbol{Q} \in V_r(\mathbb{R}^T)$ such that $\boldsymbol{Q}^T\boldsymbol{Q} = \boldsymbol{I}_r$ and write

$$\boldsymbol{\Lambda} = \boldsymbol{I}_T - \boldsymbol{Q}\boldsymbol{\Psi}\boldsymbol{Q}^T, \tag{A.21}$$

where we now constrain $0 < \psi_i < 1$ to maintain the positive definiteness of $\boldsymbol{\Lambda}$. In practice, we keep $\boldsymbol{Q}$ on the Stiefel manifold (i.e. $\boldsymbol{Q}^T\boldsymbol{Q} = \boldsymbol{I}_r$) by (differentiably) computing the QR decomposition of a $T \times r$ matrix of free parameters.

Circulant inverse $\boldsymbol{\Lambda}$ We let $\boldsymbol{C}$ be a circulant positive definite matrix as above and parameterize

$$\boldsymbol{\Lambda} = (\boldsymbol{I} + \boldsymbol{\Psi}\boldsymbol{C}\boldsymbol{\Psi})^{-1}. \tag{A.22}$$

Computing $\boldsymbol{\Lambda v}$ products is done using the conjugate gradient algorithm, taking advantage of fast products with $\boldsymbol{\Psi}$ and $\boldsymbol{C}$; the same algorithm is also used to stochastically estimate $\log|\boldsymbol{\Lambda}|$ and its gradient (see the appendix of Rutten et al., 2020).

Toeplitz inverse $\boldsymbol{\Lambda}$ This proceeds just as for the circulant inverse form, with the circulant matrix $\boldsymbol{C}$ replaced by an arbitrary Toeplitz matrix (also exploiting fast $\boldsymbol{Tv}$ products):

$$\boldsymbol{\Lambda} = (\boldsymbol{I} + \boldsymbol{\Psi}\boldsymbol{T}\boldsymbol{\Psi})^{-1}. \tag{A.23}$$

Numerical comparisons between different parameterizations

To compare these parameterizations, we generated a synthetic dataset (Figure A.7a, orange dots) over $T = 1000$ time bins by drawing samples $\{y_1, \ldots, y_T\}$ as $y_t = x_t + \sigma_t\xi_t$. Here, $\xi(t) \sim \mathcal{N}(0,1)$ with non-stationary σ_t growing linearly from 0.1 to 0.5 over the whole range $0 \leq t < T$, and $x_i \sim \mathcal{N}(0, \boldsymbol{K}^{1/2}\boldsymbol{K}^{1/2})$ with $\boldsymbol{K}^{1/2}$ given by Equation A.11. We fixed the generative parameters to their ground truth and optimized the ELBO w.r.t. the variational parameters in this simple regression setting. We found that all of the parameterizations accurately recapitulated the GP posterior mean (Figure A.7a). However, the degree to which they captured the non-stationary posterior covariance and data log likelihood varied between methods (Figure A.7b-c). To quantify this, we computed the difference between the asymptotic ELBO of each method and the exact log marginal likelihood. This ELBO gap was small for the circulant parameterization, the inverse methods, and the low rank parameterization with sufficiently high r. Although the circulant parameterization did not fully capture the non-stationary aspect of the posterior variance, this did not affect the ELBO gap substantially. Importantly, however, the circulant parameterization was more than an order of magnitude faster per gradient evaluation than the other methods with comparable accuracy (Figure A.7c). For these reasons as well as the excellent performance in a latent variable setting (Section 3.1.3, Section 3.1.3, Appendix A), we

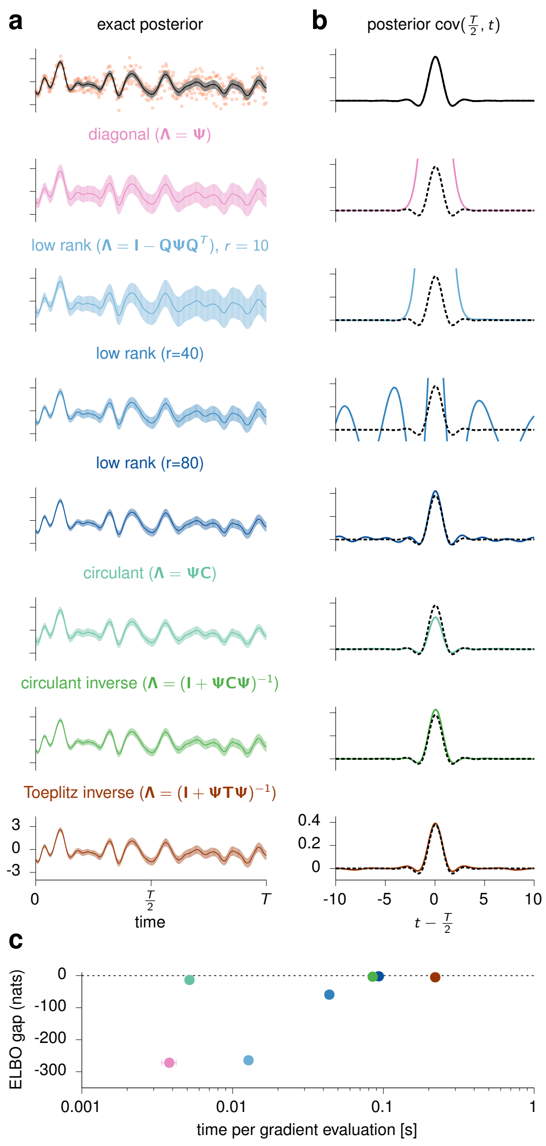

Figure A.7 **Comparisons of different forms of the approximate posterior $q(\boldsymbol{x})$.** **(a)** Synthetic data (orange dots) plotted together with the exact posterior (black) as well as the variational posteriors inferred by each whitened parameterization. The solid lines denote the (approximate) posterior means, and shaded areas indicate ± 1 posterior standard deviations. **(b)** Slice through the posterior covariance ($\text{Cov}_{x \sim q(x)}\left[x_{T/2}, x_t\right]$) for the true posterior (top and black dotted lines) and the approximate methods. Each method has different characteristics, and the circulant parameterization provides a good qualitative fit at very low computational cost. **(c)** We defined the 'ELBO gap' of each method as ELBO − LL, where LL is the true data log likelihood. We plotted this against the time per gradient evaluation and found that the circulant parameterization achieved high accuracy with cheap gradients.

used the circulant parameterization for all experiments. However, repeating all analyses with a simple diagonal parameterization also lead to good performance and qualitatively similar results.

Relation between variational posterior over $\boldsymbol{F}$ and true posterior

Here we show that our parameterization of $q(\boldsymbol{f}_n)$ includes the exact posterior in the case of Gaussian noise.

When the noise model is Gaussian (i.e., $p(\boldsymbol{y}_n|\boldsymbol{f}_n) = \mathcal{N}(\boldsymbol{y}|\boldsymbol{f}_n, \sigma_n^2 I)$), we can compute the posterior over $\boldsymbol{f}_n^* = f_n(\boldsymbol{X}^\star)$ at locations $\boldsymbol{X}^\star$ in closed form:

$$\boldsymbol{f}_n^*|\boldsymbol{X}^\star, \boldsymbol{X}, \boldsymbol{y}_n \sim \mathcal{N}(\boldsymbol{X}^{\star T}\boldsymbol{S}^2\boldsymbol{X}\hat{\boldsymbol{K}}^{-1}\boldsymbol{y}_n, \boldsymbol{X}^{\star T}\boldsymbol{S}(\boldsymbol{I} - \boldsymbol{X}\hat{\boldsymbol{K}}^{-1}\boldsymbol{X}^T)\boldsymbol{S}\boldsymbol{X}^\star) \tag{A.24}$$

where $\hat{\boldsymbol{K}} = \boldsymbol{X}^T\boldsymbol{S}^2\boldsymbol{X} + \sigma_n^2\boldsymbol{I}$. Note that the posterior is low-rank as the rank of $\boldsymbol{I} - \boldsymbol{X}\hat{\boldsymbol{K}}^{-1}\boldsymbol{X}^T$ is at most D. This means that when we do variational inference, we can parameterize our approximate posterior as:

$$q(\boldsymbol{f}_n^*) = \mathcal{N}(\boldsymbol{f}|\boldsymbol{X}^{\star T}\boldsymbol{S}\boldsymbol{\nu}_n, \boldsymbol{X}^{\star T}\boldsymbol{S}\boldsymbol{L}_n\boldsymbol{L}_n^T\boldsymbol{S}\boldsymbol{X}^\star) \tag{A.25}$$

where $\boldsymbol{\nu}_n \in \mathbb{R}^D$ and $\boldsymbol{L}_n \in \mathbb{R}^{D\times D}$ are the parameters of the approximate posterior (Section 3.1.2). We see that this parameterization is exact when:

$$\boldsymbol{\nu}_n = \boldsymbol{S}\boldsymbol{X}\hat{\boldsymbol{K}}^{-1}\boldsymbol{y}_n \tag{A.26}$$

$$\boldsymbol{L}_n\boldsymbol{L}_n^T = \boldsymbol{I} - \boldsymbol{X}\hat{\boldsymbol{K}}^{-1}\boldsymbol{X}^T. \tag{A.27}$$

Note that the right-hand side of Equation A.27 is guaranteed to be positive definite because the true posterior must be positive definite. Importantly, for this parameterization, the KL term in Equation 3.10 simplifies to

$$\mathrm{KL}(q(\boldsymbol{f}_n|\boldsymbol{X})||p(\boldsymbol{f}_n|\boldsymbol{X})) = \mathrm{KL}(\mathcal{N}(\boldsymbol{\nu}_n, \boldsymbol{L}_n\boldsymbol{L}_n^T)||\mathcal{N}(0, \boldsymbol{I})), \tag{A.28}$$

which is independent of $\boldsymbol{X}$ and allows us to do efficient inference due to the low dimensionality of $\boldsymbol{\nu}_n$ and $\boldsymbol{L}_n$.

Relation between variational posterior over $\boldsymbol{F}$ and SVGP

For general non-Gaussian noise models, the parameterization in Appendix A will no longer be exact. However, here we show that it is in this case equivalent to a stochastic variational Gaussian process (SVGP; Hensman et al., 2013). In SVGP, we choose a variational distribution:

$$q(\boldsymbol{u}) = \mathcal{N}(\boldsymbol{u}|\boldsymbol{Z}^T\boldsymbol{S}\boldsymbol{\mu}, \boldsymbol{Z}^T\boldsymbol{S}\boldsymbol{M}\boldsymbol{M}^T\boldsymbol{S}\boldsymbol{Z}) \tag{A.29}$$

at inducing points $\boldsymbol{Z} \in \mathbb{R}^{D\times m}$, where $\boldsymbol{\mu}$ and $\boldsymbol{M}$ are the “whitened” parameters (Hensman et al., 2015b). This gives an approximate posterior:

$$q(\boldsymbol{f}^*) = \mathbb{E}_{q(\boldsymbol{u})}\left[p(\boldsymbol{f}|\boldsymbol{u})\right] \tag{A.30}$$

$$= \mathcal{N}(\boldsymbol{f}|\boldsymbol{X}^{\star T}\boldsymbol{S}\boldsymbol{\Pi}_z\boldsymbol{\mu}; \boldsymbol{X}^{\star T}\boldsymbol{S}\boldsymbol{\Pi}_z(\boldsymbol{M}\boldsymbol{M}^T - \boldsymbol{I})\boldsymbol{\Pi}_z\boldsymbol{S}\boldsymbol{X}^\star) \tag{A.31}$$

where $\boldsymbol{\Pi}_{\boldsymbol{z}} = \boldsymbol{S}\boldsymbol{Z}(\boldsymbol{Z}^T\boldsymbol{S}^2\boldsymbol{Z})^{-1}\boldsymbol{Z}^T\boldsymbol{S}$. If we choose $m = D$ inducing points such that $\boldsymbol{Z} \in \mathbb{R}^{D\times D}$ and make sure $\boldsymbol{Z}$ has full rank, then $\boldsymbol{\Pi}_z = \boldsymbol{I}$ and thus

$$q(\boldsymbol{f}^*) = \mathcal{N}(\boldsymbol{f}|\boldsymbol{X}^{\star T}\boldsymbol{S}\boldsymbol{\mu}, \boldsymbol{X}^{\star T}\boldsymbol{S}(\boldsymbol{M}\boldsymbol{M}^T - \boldsymbol{I})\boldsymbol{S}\boldsymbol{X}^{\star}). \tag{A.32}$$

We recover the parameterization in Section 3.1.2 when

$$\boldsymbol{\mu} = \boldsymbol{\nu} \quad \text{and} \quad \boldsymbol{M}\boldsymbol{M}^T - \boldsymbol{I} = \boldsymbol{L}\boldsymbol{L}^T. \tag{A.33}$$

For these more general noise models, the whitened parameterization of $q(\boldsymbol{f})$ still gives rise to a computationally cheap KL divergence that is independent of $\boldsymbol{X}$ as in Equation A.28:

$$\mathrm{KL}(q(\boldsymbol{f}_n|\boldsymbol{X})||p(\boldsymbol{f}_n|\boldsymbol{X})) = \mathrm{KL}(\mathcal{N}(\boldsymbol{\nu}_n, \boldsymbol{L}_n\boldsymbol{L}_n^T)||\mathcal{N}(0, \boldsymbol{I})). \tag{A.34}$$

In summary, we have shown that (i) our parameterization of $q(\boldsymbol{f}_n)$ has sufficient flexibility to learn the true posterior when the noise model is Gaussian (Appendix A), and (ii) it is equivalent to performing SVGP where the locations of the inducing points do not matter provided that their rank is at least as high as the number of latent dimensions.

Automatic relevance determination

Here we briefly consider why introducing a prior over the factor matrix enables automatic relevance determination. These ideas reflect results by Bishop (1999) and our experiments in Section 3.1.3.

For simplicity, we will first consider the case of factor analysis where $p(\boldsymbol{X}) = \prod_{d,t}\mathcal{N}(x_{dt}; 0, 1)$. This gives rise to a marginal likelihood (with Gaussian noise) equal to

$$\log p(\boldsymbol{Y}) = \sum_t \log \mathcal{N}(\boldsymbol{y}_t; 0, \boldsymbol{C}\boldsymbol{C}^T + \boldsymbol{\Sigma}), \tag{A.35}$$

where $\boldsymbol{\Sigma} = \mathrm{diag}(\sigma_1^2, ..., \sigma_N^2)$ is a diagonal matrix of noise parameters. It is in this case quite clear that the optimal marginal likelihood is a monotonically increasing function of the latent dimensionality, since any marginal likelihood reachable with a certain rank D is also reachable with a larger rank $D' > D$; increasing D can only increase model flexibility. We could in this case threshold the magnitude of the columns of $\boldsymbol{C}$ to subselect more 'informative' dimensions, but this is not inherently different from putting an arbitrary cut-off on the variance explained in PCA, and there is no Bayesian "Occam's razor" built into the method (MacKay, 2003).

Consider now the case where we put a unit Gaussian prior on c_{nd}. In this case, $\{c_{nd}\}$ are no longer parameters of the model but rather latent variables to be inferred, which intuitively should reduce the risk of overfitting. To expand on this intuition, consider the ELBO (c.f. Section 3.1.2) that results from introducing such a prior over c_{nd}:

$$\log p(\boldsymbol{Y}) \geq \mathbb{E}_{q(\boldsymbol{X})}\left[\log p(\boldsymbol{Y}|\boldsymbol{X})\right] - \sum_{d,t} \mathrm{KL}\left[q(x_{dt})||\mathcal{N}(0,1)\right] \tag{A.36}$$

$$\log p(\boldsymbol{Y}|\boldsymbol{X}) = \sum_{n} \log \mathcal{N}(\boldsymbol{y}_n; 0, \boldsymbol{X}^T\boldsymbol{X} + \sigma_n \boldsymbol{I}). \tag{A.37}$$

Here we see that if a dimension d is truly uninformative, it should have $x_{dt} = 0\, \forall_t$ to avoid contributing noise to the likelihood term via $\boldsymbol{X}^T\boldsymbol{X}$. However, reducing this noise will increase the prior KL term, driving it to infinity in the limit of zero noise since the variational posterior over the d^{th} latent at time t, $q(x_{dt})$, is in this case a delta function at zero. Optimizing the ELBO therefore involves a balance between mitigating the noise induced by $\boldsymbol{X}^T\boldsymbol{X}$ and reducing the KL penalty, with both of these terms contributing to a decreased ELBO compared to the model without uninformative dimensions. Thus the prior over c_{nd} counteracts the overfitting that would normally occur when increasing the latent dimensionality in classical factor analysis, and this Bayesian treatment will lead to a decrease in the ELBO with increasing dimensionality beyond the optimal $D^\star$ that is needed to adequately explain the data.

Finally let us consider the case where we learn the prior scale of the factor matrix, such that $c_{nd} \sim \mathcal{N}(0, s_d^2)$ with s_d optimized w.r.t. the ELBO. Critically, the likelihood term now becomes:

$$\log p(\boldsymbol{Y}|\boldsymbol{X}) = \sum_{n} \log \mathcal{N}(\boldsymbol{y}_n; 0, \boldsymbol{X}^T\boldsymbol{S}^2\boldsymbol{X} + \sigma_n \boldsymbol{I}). \tag{A.38}$$

with $\boldsymbol{S} = \mathrm{diag}(s_1, \ldots, s_D)$. In this case, adding uninformative dimensions beyond the optimal $D^\star$ still cannot increase the ELBO (in the limit of large N). However, letting $s_d \to 0$ for these superfluous dimensions will prevent them from contributing to $p(\boldsymbol{Y}|\boldsymbol{X})$, thus allowing $q(x_{dt}) \to \mathcal{N}(0,1)$ to drive the prior KL term to zero for these dimensions. In this limit, we recover both the ELBO and the posteriors associated with the $D^\star$- dimensional model. We thus have a built-in Occam's razor which will shave off any uninformative latent dimensions, and these will be identifiable as dimensions for which $s_d \approx 0$ and $q(x_{dt}) \approx \mathcal{N}(0,1)$.

These ideas generalize to GPFA where the posterior over latents will instead approach the GP prior $q(\boldsymbol{x}_d) \approx \mathcal{N}(0, \boldsymbol{K})$ for uninformative dimensions. This corresponds to the limit of $\boldsymbol{\nu} \to \boldsymbol{0}$, $\boldsymbol{C} \to \boldsymbol{I}$, and $\boldsymbol{\Psi} \to \boldsymbol{I}$ in our circulant parameterization in Section 3.1.2 and Appendix A. In all of our simulations, we found a clear clustering of dimensions after training with some clustered near zero s_d, and others clustered with much larger s_d (Figure 3.2c and Figure 3.3b). Note that in practice we do not actively truncate the model by discarding dimensions with $s_d \approx 0$ but merely use the terminology to indicate that these dimensions have negligible contributions to the posterior predictive $q(\boldsymbol{y}_n)$, as well as to the latent posteriors $q(\boldsymbol{x}_d)$ for the dimensions with large s_d.

Most informative dimensions

In this work, we refer to the latent dimensions with the highest values of s_d as the 'most informative dimensions'. We do this because (i) observing the value of the corresponding latent x_d decreases the variance of the expected distribution of neural activity more as s_d increases, and (ii) the Fisher information of x_d increases as s_d increases.

To show this, we consider how the distribution over f_n (the activity of neuron n) given $\boldsymbol{c}_n$ (the n^{th} row of $\boldsymbol{C}$) changes when x_d (the value of the d^{th} latent) is known, and how this varies with s_d. In the following, we omit the $\cdot_n$ subscript for notational simplicity, and we note that f, x_d and c_d are all scalar values. With unknown x_d, f is Gaussian with zero mean and variance $\mathbb{E}_{p(\boldsymbol{x})}\left[\boldsymbol{c}^T\boldsymbol{x}\boldsymbol{x}^T\boldsymbol{c}\right] = \boldsymbol{c}^T\boldsymbol{c}$. Thus,

$$p(f|\boldsymbol{c}) = \mathcal{N}(f;0,\boldsymbol{c}^T\boldsymbol{c}) \tag{A.39}$$

In contrast, for known x_d, we have

$$p(f|\boldsymbol{c},x_d) = \mathcal{N}(f;c_d x_d,\boldsymbol{c}_{-d}^T\boldsymbol{c}_{-d}), \tag{A.40}$$

where $\boldsymbol{c}_{-d}$ is $\boldsymbol{c}$ with the d^{th} element removed. We thus see that the decrease in variance of f from observing x_d is c_d^2. Finally, we can approximate the process of averaging this quantity over neurons by noting that $c_d \sim \mathcal{N}(0,s_d^2)$ and marginalising out $\boldsymbol{c}$:

$$\mathbb{E}_{p(\boldsymbol{c})}[\sigma^2_{f|\boldsymbol{c}} - \sigma^2_{f|\boldsymbol{c},x_d}] = \mathbb{E}_{p(\boldsymbol{c})}[c_d^2] = s_d^2, \tag{A.41}$$

where $\sigma^2_{f|\boldsymbol{c}}$ is the variance of $p(f|\boldsymbol{c})$. Thus, s_d^2 can be interpreted as the expected decrease in the variance of the denoised neural activity f when learning the value of the d^{th} latent.

This can also be understood in information-theoretic terms by considering the Fisher information of the d^{th} latent dimension which is given by

$$\mathcal{I}(x_d|\boldsymbol{c}) = -\mathbb{E}_{p(f|x_d,\boldsymbol{c})}\left[\frac{\partial^2}{\partial x_d^2}\log p(f|x_d,\boldsymbol{c})\right] \tag{A.42}$$

$$= \left[\sum_{d'\neq d} c_{d'}^2\right]^{-1}. \tag{A.43}$$

To relate this quantity to our prior scale parameters $\{s_d\}$, we consider the expectation of the inverse Fisher information:

$$\mathbb{E}_{p(\boldsymbol{c})}[\mathcal{I}(x_d|\boldsymbol{c})^{-1}] = \sum_{d'\neq d} s_{d'}^2. \tag{A.44}$$

For a given set of latent dimensions $[1,D]$ with corresponding $\{s_d\}_1^D$, we thus see that the expected *inverse* Fisher information is *minimized* for the dimension with the highest value of

s_d. In Figure 3.2 and Figure 3.3 we use s_d together with the posterior latent mean parameters $\boldsymbol{\nu}_d$ to identify 'discarded' dimensions.

Noise models and evaluation of their expectations

Gaussian The Gaussian noise model is given by

$$\log p(y_{nt}|f_{nt}) = -\frac{1}{2}\log(2\pi) - \frac{1}{2}(y_{nt} - f_{nt})^2/\sigma_n^2, \tag{A.45}$$

where σ_n is a learnable parameter. In this case we can easily compute the expected log-density under the approximate posterior analytically:

$$\mathbb{E}_{q(f_{nt}|\boldsymbol{X})}\left[\log p(y_{nt}|f_{nt})\right] = -\frac{1}{2}\left(\log(2\pi) + \frac{(y_{nt} - \mu_{nt})^2 + \Sigma_{ntt}}{\sigma_n^2}\right), \tag{A.46}$$

where $q(\boldsymbol{f}_n|\boldsymbol{X}) = \mathcal{N}(\boldsymbol{f}_n; \boldsymbol{\mu}_n, \boldsymbol{\Sigma}_n)$ and Σ_{ntt} is the approximate posterior variance of neuron n at time t (i.e., the t^{th} diagonal element of $\boldsymbol{\Sigma}_n$).

Poisson The Poisson noise model is given by

$$\log p(y_{nt}|f_{nt}) = y_{nt}\log g(f_{nt}) - g(f_{nt}) - \log(y_{nt}!), \tag{A.47}$$

where g is a link function. If we choose an exponential link function (i.e., $g(x) = \exp(x)$), we can compute in closed-form the expected log-density of the approximate posterior as:

$$\begin{aligned}\mathbb{E}_{q(f_{nt}|\boldsymbol{X})}\left[\log p(y_{nt}|f_{nt})\right] &= \mathbb{E}_{q(f_{nt}|\boldsymbol{X})}\left[y_{nt}f_{nt} - \exp(f_{nt}) - \log(y_{nt}!)\right] & \text{(A.48)}\\ &= y_{nt}\mu_{nt} - \exp\left(\mu_{nt} + \frac{1}{2}\Sigma_{ntt}\right) - \log(y_{nt}!). & \text{(A.49)}\end{aligned}$$

For the analyses shown in Figure 3.2c-d, we use the exponential link function.

For general link functions g, we may not be able to evaluate the expected log-density in closed-form. In this case, we approximate it with Gauss-Hermite quadrature:

$$\mathbb{E}_{q(f_{nt}|\boldsymbol{X})}\left[\log p(y_{nt}|f_{nt})\right] \approx \frac{1}{\sqrt{\pi}}\sum_{i=1}^{k_{\text{GH}}} \omega_i \log p(y_{nt}|f_{nt}^{(i)}) \tag{A.50}$$

where

$$\begin{aligned}\omega_i &= \frac{2^{k_{\text{GH}}-1}k_{\text{GH}}!\sqrt{\pi}}{{k_{\text{GH}}}^2[H_{k_{\text{GH}}-1}(r_i)]^2}, & \text{(A.51)}\\ f_{nt}^{(i)} &= \left(\sqrt{2\Sigma_{ntt}}\right)r_i + \mu_{nt}, & \text{(A.52)}\end{aligned}$$

$H_k(r)$ are the physicist's Hermite polynomials, and r_i with $i = 1,\dots,k$ are roots of $H_k(r)$. For a given order of approximation k_{GH}, we can evaluate both ω_i and r_i using standard numerical software packages such as Numpy. In practice, we find that $k_{\text{GH}} = 20$ gives an accurate approximation to the expected log-density under the approximate posterior. Note that we could also estimate the expectation over $q(f_{nt})$ for general link functions g using a Monte Carlo estimate, but we use Gauss-Hermite quadrature in this work since it has a lower computational cost and lower variance.

Negative binomial The negative binomial noise model is given by

$$\log p(y_{nt}|f_{nt}) = \log \binom{y_{nt} + \kappa_n - 1}{y_{nt}} + \kappa_n \log(1 - g(f_{nt})) + y \log(g(f_{nt})), \tag{A.53}$$

where $g(f_{nt})$ denotes the probability of success in a Bernoulli trial. Here, each success corresponds to the emission of one spike in bin t, and thus $p(y_{nt}|f_{nt})$ is the distribution over the number of successful trials (spikes) before reaching κ_n failed trials. The link function $g(x) : \mathbb{R} \to [0,1)$ maps f_{nt} to a real number between 0 and 1. In practice we use a sigmoid link-function $g(x) = 1/(1+\exp(-x))$.

In this model, κ_n is a learnable parameter, which modulates the overdispersion of the distribution since the mean and variance of $p(y_{nt}|f_{nt})$ are given by:

$$\mu_{NB} = \frac{g(f_{nt})\kappa_n}{1 - g(f_{nt})} \tag{A.54}$$

$$\sigma^2_{NB} = \mu_{NB}\left(1 + \frac{\mu_{NB}}{\kappa_n}\right). \tag{A.55}$$

This is the parameter which we compare between the ground truth and trained models in Figure 3.2, and we see that the Poisson model is recovered for neuron n as $\kappa_n \to \infty$.

For the negative binomial noise model we cannot compute the expected log-density in closed-form. We instead approximate this expectation using Gauss-Hermite quadrature as described above.

Implementation

In this section, we provide pseudocode for bGPFA (Algorithm 1) with the circulant parameterization for $q(\boldsymbol{X})$ and discuss other implementation details.

Note that we need to sample the full trajectory $\boldsymbol{x}_d$ before subsampling for each batch due to the correlations introduced by $\boldsymbol{K}$. In practice, we run the optimization for 2000 passes over the full data which we found empirically lead to convergence of the ELBO. We used $M = 20$ Monte

Algorithm 1: Bayesian GPFA with automatic relevance determination

```
input: data $\boldsymbol{Y} \in \mathbb{R}^{N \times T}$, maximum latent dimensionality $D$, # of Monte Carlo samples $M$,
  learning rate $\gamma$
parameters: $\theta = \{\{s_d\}_1^D, \{\tau_d\}_1^D, \{\boldsymbol{\nu}_d\}_1^D, \{\tilde{\boldsymbol{c}}_d\}_1^D, \{\boldsymbol{\Psi}_d\}_1^D, \{\boldsymbol{L}_n\}_1^N, \{\hat{\boldsymbol{\nu}}_n\}_1^N, \{\hat{\sigma}_n \text{ or } \kappa_n\}_1^N\}$

while not converged do
    $\nabla\mathcal{L} \leftarrow 0$
    for batch in batches do

        %For each of $M$ Monte Carlo samples
        for $m = 1 : M$ do

            % sample from approximate posterior $q(\boldsymbol{X})$
            for $d = 1 : D$ do
                $\boldsymbol{\eta}_d^{(m)} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}_T)$
                $\boldsymbol{k}_d^{\frac{1}{2}} = \sigma_{\frac{1}{2},d} \exp\left(-\frac{(\boldsymbol{t}-t_0)^2}{2\tau^2_{\frac{1}{2},d}}\right)$        // single column of $\boldsymbol{K}$
                $\boldsymbol{x}_d^{(m)} = \text{Toeplitz\_mult}(\boldsymbol{k}_d^{\frac{1}{2}}, \boldsymbol{\nu}_d + \boldsymbol{C}\boldsymbol{\eta}_d^{(m)})$        // Appendix A
            $\boldsymbol{X}_m = [\boldsymbol{x}_1^{(m)}; \ldots; \boldsymbol{x}_d^{(m)}]$

            % compute $q(\boldsymbol{F})$ and $\mathbb{E}_{q(\boldsymbol{F})}[p(\boldsymbol{Y}|\boldsymbol{F})]$
            $\hat{\boldsymbol{\mu}}_n = \boldsymbol{X}_m^\top \hat{\boldsymbol{\nu}}_n$        // variational mean
            $\hat{\boldsymbol{\sigma}}_n^2 = \text{diag}\left(\boldsymbol{X}_m^T \boldsymbol{S} \boldsymbol{L}_n \boldsymbol{L}_n^\top \boldsymbol{S} \boldsymbol{X}_m\right)$
            $\log p_{YF}^{(m)} = \sum_{n,t \in batch} \mathbb{E}_{\mathcal{N}(f_{nt}; \hat{\mu}_{nt}, \hat{\sigma}^2_{nt})}[\log p(y_{nt}|f_{nt})]$        // Appendix A

        % compute KL terms
        $\text{KL}_x = \frac{\text{size}(batch)}{\text{size}(data)} \sum_d \text{KL}[q(\boldsymbol{x}_d)||p(\boldsymbol{x}_d)]$        // Appendix A
        $\text{KL}_f = \frac{\text{size}(batch)}{\text{size}(data)} \sum_n \text{KL}[q(\boldsymbol{f}_n)||p(\boldsymbol{f}_n)]$        // Appendix A

        % update gradient with batch gradient
        $\tilde{\mathcal{L}} = \frac{1}{M} \sum_m \log p_{YF}^{(m)} - \text{KL}_x - \text{KL}_f$
        $\nabla\mathcal{L} \leftarrow \nabla\mathcal{L} + \nabla\tilde{\mathcal{L}}$

    % update parameters based on total gradients (we use Adam in practice)
    $\theta \leftarrow \theta + \gamma\nabla\mathcal{L}$
```

Carlo samples for each update step when fitting synthetic data and $M = 10$ for the primate data. For all models, $q(\boldsymbol{X})$ was initialized at the prior $p(\boldsymbol{X})$. The prior scale parameters were initialized as $s_d = \rho||\boldsymbol{c}_d||_2^2$ where $\boldsymbol{c}_d$ is the d^{th} row of the factor matrix $\boldsymbol{C}$ found by factor analysis (Pedregosa et al., 2011), and $\rho = 3$ was found empirically to give good convergence on the primate data. When using a Gaussian noise model, noise variances were initialized as the σ_n^2 found by factor analysis. For negative binomial noise models, we initialized $\kappa_n = \frac{1}{T}\sum_t y_{nt}$, which matches the mean of the distribution to the data for $f = 0$. Length scales τ were initialized at 200 ms for all latent dimensions for the primate data and at $\approx 80\%$ of the ground truth value for the synthetic data. Synthetic data was fitted on a single GPU with 8GB RAM. Primate data was fitted on a single GPU with 12GB RAM and took approximately 30 hours for a single model fit to the full dataset at 25 ms resolution. We also note that when fitting data with a Gaussian noise model, we mean-subtracted the original data, whereas we include explicit mean parameters in the Poisson and negative binomial noise models since they are non-linear.

Code availability A PyTorch implementation of bGPFA can be found at https://github.com/tachukao/mgplvm-pytorch.

Cross-validation and kinematic decoding

In this section, we describe the procedure for computing cross-validated errors in Figure 3.2, and performing kinematic decoding analyses in Figure 3.3. In these analyses, expectations over $\boldsymbol{X}$ were computed using the posterior mean of $q(\boldsymbol{X})$ and expectations over $\boldsymbol{F}$ were computed using Monte Carlo samples from $q(\boldsymbol{F})$.

Prediction errors To compute cross-validated errors, we divide the time points into a training and a test set, $\mathcal{T}_{train} = \{t_1, t_2, ..., t_{T_{train}}\}$ and $\mathcal{T}_{test} = \{t_{T_{train}+1}, ..., T\}$, and similarly for the neurons $\mathcal{N}_{train}$ and $\mathcal{N}_{test}$. We also define $\mathcal{T}_{tot} = \mathcal{T}_{train} \bigcup \mathcal{T}_{test}$ and $\mathcal{N}_{tot} = \mathcal{N}_{train} \bigcup \mathcal{N}_{test}$. We first fit the generative parameters θ_{gen} of each model to data from all the neurons at the training time points using variational inference (taking θ to include the variational parameters ϕ of $q_\phi(\boldsymbol{F})$):

$$\theta_{gen} = \text{argmax}_{\theta_{gen}} \left[p(\boldsymbol{Y}_{\mathcal{N}_{tot}, \mathcal{T}_{train}} | \theta_{gen}) \right]. \tag{A.56}$$

We then fix the generative parameters and infer a distribution over latents from the training neurons recorded at all time points using a second pass of variational inference:

$$q(\boldsymbol{X}_{1:D, \mathcal{T}_{tot}} | \boldsymbol{Y}_{\mathcal{N}_{train}, \mathcal{T}_{tot}}, \theta_{gen}) \approx p(\boldsymbol{X}_{1:D, \mathcal{T}_{tot}} | \boldsymbol{Y}_{\mathcal{N}_{train}, \mathcal{T}_{tot}}, \theta_{gen}). \tag{A.57}$$

Finally we use the inferred latent states and generative parameters to predict the activity of the test neurons at the test time points

$$\hat{\boldsymbol{Y}}_{\mathcal{N}_{test}, \mathcal{T}_{test}} = \int \boldsymbol{Y} p(\boldsymbol{Y}_{\mathcal{N}_{test}, \mathcal{T}_{test}} | \boldsymbol{X}_{1:D, \mathcal{T}_{test}}, \theta_{gen}) q(\boldsymbol{X}_{1:D, \mathcal{T}_{test}} | \boldsymbol{Y}_{\mathcal{N}_{train}, \mathcal{T}_{tot}}, \theta_{gen}) d\boldsymbol{X}_{1:D, \mathcal{T}_{test}} \tag{A.58}$$

This allows us to compute a cross-validated predictive mean squared error as

$$\epsilon = \frac{1}{|\mathcal{N}_{test}|\,|\mathcal{T}_{test}|} ||\hat{\boldsymbol{Y}}_{\mathcal{N}_{test},\mathcal{T}_{test}} - \boldsymbol{Y}_{\mathcal{N}_{test},\mathcal{T}_{test}}||_2^2. \tag{A.59}$$

Kinematic decoding For kinematic decoding analyses, we only considered the latents and behavior prior to a period of approximately 5 minutes where the monkey disengaged from the task (the first 1430 seconds; Section 3.1.3). Cursor positions in the x and y directions were first fitted with cubic splines and velocities extracted as the first derivative of these splines. To evaluate kinematic decoding performance, we followed Keshtkaran et al. (2021) and computed the expected activity of all neurons at all time points under our model:

$$\hat{\boldsymbol{Y}} = \int \boldsymbol{Y} p(\boldsymbol{Y}|\boldsymbol{F}) q(\boldsymbol{F}|\boldsymbol{X}) q(\boldsymbol{X}|\boldsymbol{t}) d\boldsymbol{X} d\boldsymbol{F}. \tag{A.60}$$

For non-Gaussian noise models, this can be viewed as the first non-linear step of a decoding model from the latent states $\boldsymbol{X}$. We then performed 10-fold cross-validation where 90% of the data was used to fit a ridge regression model which was tested on the held-out 10% of the data. The regularization strength was determined using 10-fold cross-validation on the 90% training data. The predictive performance was computed as the mean across the 10 folds. Models were fitted and evaluated independently for the hand x and y velocities, and the final performance was computed as the mean variance accounted for across these two dimensions. Results in Section 3.1.3 are reported as mean ± standard error across 10 different splits of the data into folds used for cross-validation.

Appendix B

Manifold GPLVMs

The mouse head direction circuit

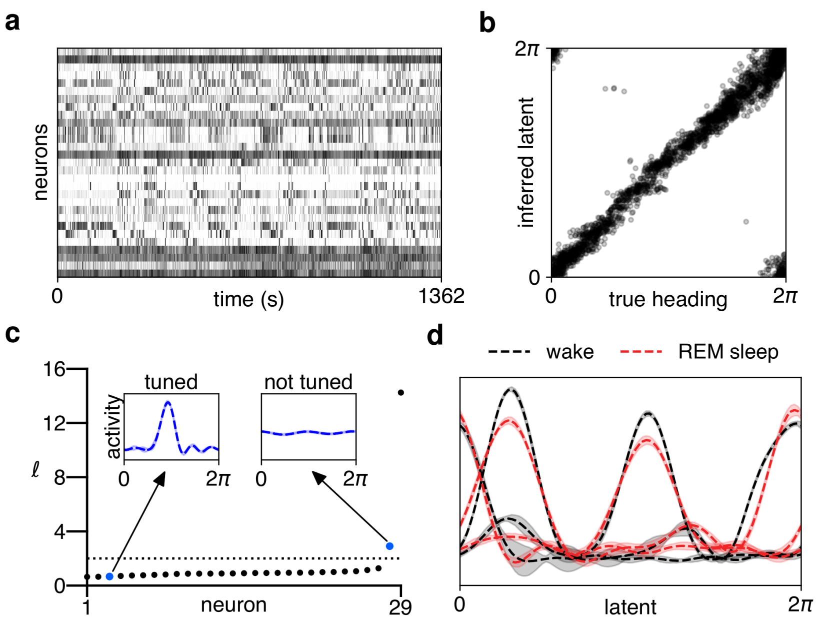

Figure B.1 **The mouse head direction circuit.** **(a)** Population activity recorded from mouse ADn during foraging. **(b)** Variational mean inferred by T^1-mGPLVM plotted against the true mouse head direction. **(c)** Kernel length scales for the 29 neurons recorded. Dashed line: $\ell^2 = 4$ (maximum d in the T^1-kernel). Insets: example neurons with low and high ℓ. **(d)** Tuning curves for three example neurons inferred during wake (black) and REM sleep (red).

To highlight the importance of unsupervised non-Euclidean learning methods in neuroscience and to illustrate the interpretability of the learned GP parameters, we consider a dataset from Peyrache and Buzsáki (2015) recorded from the mouse anterodorsal thalamic nucleus (ADn; Figure B.1a). This data has also been analyzed in Peyrache et al. (2015), Chaudhuri et al. (2019) and Rubin et al. (2019). We consider the same example session shown in Figure 2 of Chaudhuri et al. (2019) (Mouse 28, session 140313) and bin spike counts in 500 ms time bins for analysis with mGPLVM. When comparing cross-validated log likelihoods for T^1- and $\mathbb{R}^1$-mGPLVM fitted to the data, T^1 consistently outperformed $\mathbb{R}^1$ with a test log likelihood ratio of 127 ± 30 (mean $\pm$ sem) across 10 partitions of the data.

Fitting T^1-mGPLVM to the binned spike data, we found that the inferred latent state was highly correlated with the true head direction (Figure B.1b). However, in contrast to the data considered in Section 3.2.3 and Section 3.2.3, this mouse dataset contains neurons with more heterogeneous baseline activities and tuning properties. This is reflected in the learned GP parameters, which converge to small kernel length scales for neurons that contribute to the heading representation (Figure B.1c, 'tuned') and large length scales for those that do not (Figure B.1c, 'not tuned'). Finally, since mGPLVM does not require knowledge of behaviour, we also fitted mGPLVM to data recorded from the same neurons during a period of rapid eye movement (REM) sleep. Here we found that the representation of subconscious heading during REM sleep was similar to the representation of heading when the animal was awake after matching the offset between the two sets of tuning curves (Figure B.1d), similar to results by Peyrache et al. (2015). However, their analyses relied on recordings from two separate brain regions to align the activity from neurons in ADn to a subconscious head direction decoded from the postsubiculum and vice versa. In contrast, mGPLVM allows for fully unsupervised Bayesian analyses across both wake and sleep using recordings from a single brain area.

Priors on manifolds

For all manifolds, we use priors that factorize over conditions, $p^{\mathcal{M}}(\{g_j\}) = \prod_j p^{\mathcal{M}}(g_j)$. As described in Section 3.2.2, we use a Gaussian prior $p^{R^n}(g) = \mathcal{N}(g; 0, \mathbf{I}_n)$ over latent states in $\mathbb{R}^n$, and uniform priors for the spheres, tori, and $SO(3)$. These uniform priors have a density which is the inverse volume of the manifold:

$$p^{S^n}(g) = \left[\frac{2\pi^{\frac{n+1}{2}}}{\Gamma(\frac{n+1}{2})}\right]^{-1} \tag{B.1}$$

$$p^{T^n}(g) = [2\pi]^{-n} \tag{B.2}$$

$$p^{SO(3)}(g) = \left[\frac{2\pi^{\frac{4}{2}}}{2\Gamma(\frac{4}{2})}\right]^{-1}. \tag{B.3}$$

Note that the volume of S^n is the surface area of the n-sphere, and the volume of $SO(3)$ is half the volume of S^3.

Lie groups and their exponential maps

For simplicity of exposition, we have skimmed over the details of how the 'capitalized' Exponential map $\text{Exp}_G : \mathbb{R}^n \to G$ is defined in Section 3.2.2, particularly in relation to the group's Lie algebra $\mathfrak{g}$. Here we make this connection more explicit. As described in the main text, the Lie algebra $\mathfrak{g}$ of a group G is a vector space tangent to G at its identity element. The exponential map $\exp_G : \mathfrak{g} \to G$ maps elements from the Lie algebra to the group, and is conceptually distinct from the "capitalised" Exponential map defined in Section 3.2.2, which maps from $\mathbb{R}^n$ to G. However, because the Lie algebra is isomorphic to $\mathbb{R}^n$, we have found it convenient in both our exposition and our implementation to work directly with the pair $(\mathbb{R}^n, \mathrm{Exp}_G)$, instead of $(\mathfrak{g}, \exp_G)$. To expand on the connection between the two, note that we can define as in Sola et al. (2018) the isomorphism $\mathrm{Hat} : \mathbb{R}^n \to \mathfrak{g}$, which maps every element in $\mathbb{R}^n$ to a distinct element in the Lie algebra $\mathfrak{g}$. Therefore, $\mathrm{Exp}_G : \mathbb{R}^n \to G$ is in fact the composition $\exp_G \circ \mathrm{Hat}$.

Manifold-specific parameterizations

Here we provide some further justification for the forms of $\tilde{q}_\phi(\tilde{g})$ provided in Equations 3.32 and 3.33 as well as the exponential maps which are used to derive these densities and are needed for optimization in Equation 3.25. For both T^n and $SO(3)$, we use Equation 3.22 from Falorsi et al. (2019), which we repeat here for reference:

$$\tilde{q}_\phi(\tilde{g}) = \sum_{\boldsymbol{x} \in \mathbb{R}^n : \mathrm{Exp}_G(\boldsymbol{x}) = \tilde{g}} r_\phi(\boldsymbol{x}) |\boldsymbol{J}(\boldsymbol{x})|^{-1}. \tag{B.4}$$

In what follows, we will use $\boldsymbol{g}$ to indicate a vector representation of group element g to avoid conflicts of notation.

Note that the expressions in this section largely follow Falorsi et al. (2019), but we re-write them in a different basis for ease of computational implementation.

Tori

The n-Torus T^n is the direct product of n circles, such that we can parameterize members of this group as $\boldsymbol{g} \in \mathbb{R}^n$ whose elements are all angles between 0 and 2π. Note that this is equivalent to the parameterization in Equation 3.30, except that here we denote an element on the circle by its angle, while in Equation 3.30 we denote it by a unit 2-vector for notational consistency with the other kernels. Because 1-dimensional rotations are commutative, the parameterization of the torus as a list of angles allows us to perform group operations by simple addition modulo 2π. We therefore slightly abuse notation and write the exponential map $\mathrm{Exp}_{T^n} : \mathbb{R}^n \to T^n$ as an element-wise modulo operation:

$$\mathrm{Exp}_{T^n} \boldsymbol{x} = \boldsymbol{x} \bmod 2\pi. \tag{B.5}$$

Equation B.5 has inverse Jacobian $|\boldsymbol{J}(x)|^{-1} = 1$. Moreover, since $\mathrm{Exp}_{T^n}(\boldsymbol{x}) = \mathrm{Exp}_{T^n}(\boldsymbol{x} + 2\pi \boldsymbol{k})$ for any integer vector $\boldsymbol{k} \in \mathbb{Z}^n$, the change-of-variable formula in Equation B.4 yields the following density on T^n:

$$\tilde{q}_\phi(\mathrm{Exp}_{T^n}\boldsymbol{x}) = \sum_{\boldsymbol{k}\in\mathbb{Z}^n} r_\phi(\boldsymbol{x}+2\pi\boldsymbol{k}). \tag{B.6}$$

For ease of implementation, it is also convenient to rewrite the kernel distance function Equation 3.30 as

$$d_{T^n}(\boldsymbol{g},\boldsymbol{g}') = 2\cdot\mathbf{1}_n\cdot(1-\cos(\boldsymbol{g}-\boldsymbol{g}')) \tag{B.7}$$

where $\mathbf{1}_n$ is the n-vector full of ones, and $\cos(\cdot)$ is applied element-wise to $\boldsymbol{g}-\boldsymbol{g}'$.

Special orthogonal group

We use quaternions $\boldsymbol{g}\in\mathbb{R}^4$ to represent elements $g\in SO(3)$ as indicated in Equation 3.31. For a rotation of φ radians around axis $\boldsymbol{u}\in\mathbb{R}^3$ with $\|\boldsymbol{u}\|=1$,

$$\boldsymbol{g} = \left(\cos\frac{\varphi}{2}, \boldsymbol{u}\sin\frac{\varphi}{2}\right)\in\mathbb{R}^4. \tag{B.8}$$

The exponential map $\mathrm{Exp}_{SO(3)}:\mathbb{R}^3\to SO(3)$ is

$$\mathrm{Exp}_{SO(3)}\boldsymbol{x} = (\cos\|\boldsymbol{x}\|, \hat{\boldsymbol{x}}\sin\|\boldsymbol{x}\|), \tag{B.9}$$

where $\hat{\boldsymbol{x}}=\boldsymbol{x}/\|\boldsymbol{x}\|$ and $\varphi=2\|\boldsymbol{x}\|$ is the angle of rotation. This gives rise to an inverse Jacobian

$$|\boldsymbol{J}(\boldsymbol{x})|^{-1} = \varphi^2/(2(1-\cos\varphi)). \tag{B.10}$$

Using Equation B.4 we get the density on the group

$$\tilde{q}_\phi(\mathrm{Exp}_{SO(3)}\boldsymbol{x}) = \sum_{k\in\mathbb{Z}}\left[r_\phi(\boldsymbol{x}+\pi k\hat{\boldsymbol{x}})\,\frac{2\|\boldsymbol{x}+\pi k\hat{\boldsymbol{x}}\|^2}{1-\cos(2\|\boldsymbol{x}+\pi k\hat{\boldsymbol{x}}\|)}\right], \tag{B.11}$$

where the sum over k stems from the fact that a rotation of $\varphi+2k\pi$ around axis $\hat{\boldsymbol{x}}$ is equivalent to a rotation of φ around the same axis.

mGPLVM on spheres

In this section, we discuss how to fit mGPLVMs on spheres. We first consider spheres that are also Lie groups and then discuss a general framework for all n-spheres.

Lie group spheres

We begin by noting that S^n is not a Lie group unless $n=1$ or $n=3$. We can therefore only apply the ReLie framework to S^1 and S^3. S^1 is equivalent to T^1 and is most easily treated using

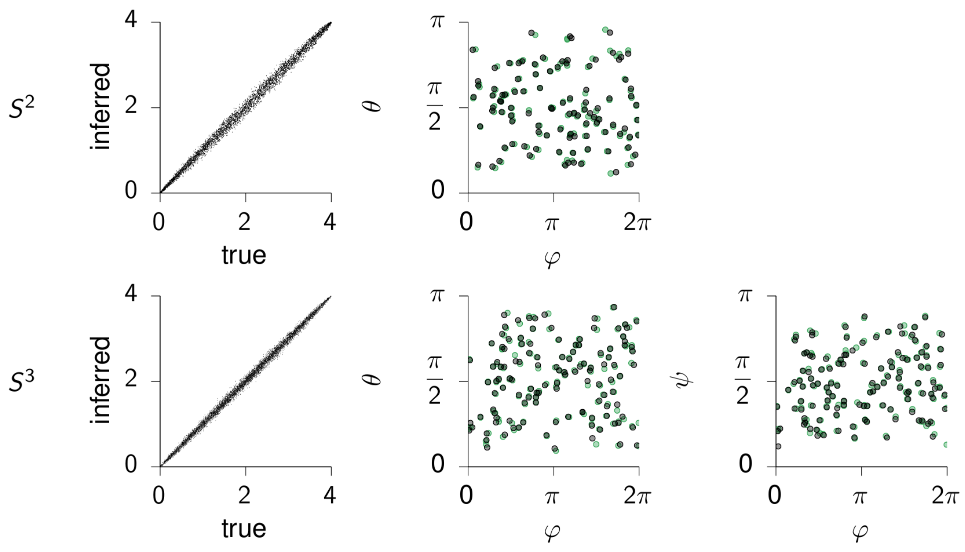

Figure B.2 **Applying mGPLVM to synthetic data on S^2 (top) and S^3 (bottom).** Pairwise distances between the variational means $\{g_j^\mu\}$ are plotted against the corresponding pairwise distances between the true latent states $\{g_j\}$ for S^2 (top left) and S^3 (bottom left). Since the log likelihood is a function of these pairwise distances through the kernel (Equation 3.29), this illustrates that mGPLVM recovers the important features of the true latents. Inferred (black) and true (green) latent states in spherical coordinates for S^2 (top middle) and S^3 (bottom middle and bottom right). For S^2, we are showing the latent states in spherical polar coordinates $\boldsymbol{g} = (\sin\theta\cos\varphi, \sin\theta\sin\varphi, \cos\theta)$ with $\theta \in [0,\pi]$ and $\varphi \in [0,2\pi]$. For S^3, we use hyperspherical coordinates $\boldsymbol{g} = (\sin\psi\sin\theta\cos\varphi, \sin\psi\sin\theta\sin\varphi, \sin\theta\cos\psi, \cos\theta)$ with $\theta, \psi \in [0,\pi]$ and $\varphi \in [0,2\pi]$.

the torus formalism above. For S^3, we note that $SO(3)$ is simply S^3 with double coverage. This is because quaternions $\boldsymbol{g}$ and $-\boldsymbol{g}$ represent the same element of $SO(3)$, while they correspond to distinct elements of S^3. The Jacobian and exponential maps of S^3 are therefore identical to those of $SO(3)$. The expression for the density on S^3 also mirrors Equation B.11 except that the sum is over $\boldsymbol{x} + 2\pi k\hat{\boldsymbol{x}}$ instead of $\boldsymbol{x} + \pi k\hat{\boldsymbol{x}}$:

$$\tilde{q}_\phi(\mathrm{Exp}_{S^3}\boldsymbol{x}) = \sum_{k\in\mathbb{Z}} \left[r_\phi(\boldsymbol{x} + 2\pi k\hat{\boldsymbol{x}}) \frac{2\|\boldsymbol{x} + 2\pi k\hat{\boldsymbol{x}}\|^2}{1 - \cos(2\|\boldsymbol{x} + 2\pi k\hat{\boldsymbol{x}}\|)} \right]. \tag{B.12}$$

We demonstrate S^3-mGPLVM on synthetic data from S^3 in Figure B.2 (bottom).

non-Lie group spheres

The ReLie framework does not directly apply to distributions defined on non-Lie groups. Nevertheless, we can still apply mGPLVM to an n-sphere embedded in $\mathbb{R}^{n+1}$ by taking each latent variational distribution q_{ϕ_j} to be a von Mises-Fisher distribution (VMF), whose entropy is known analytically. Parameterizing group element $g \in S^n$ by a unit-norm vector $\boldsymbol{g} \in \mathbb{R}^{n+1}$,

$\|\boldsymbol{g}\| = 1$, this density is given by:

$$q_\phi(\boldsymbol{g}; \boldsymbol{g}^\mu, \kappa) = \frac{\kappa^{n/2-1}}{(2\pi)^{n/2} I_{n/2-1}(\kappa)} \exp(\kappa \boldsymbol{g}^\mu \cdot \boldsymbol{g}), \tag{B.13}$$

where $\cdot$ denotes the dot product. Here, I_v is the modified Bessel function of the first kind at order v, $\boldsymbol{g}^\mu$ is the mean direction of the distribution on the hypersphere, and $\kappa \geq 0$ is a concentration parameter – the larger κ, the more concentrated the distribution around $\boldsymbol{g}^\mu$.

Using a VMF distribution as the latent distribution, we can easily evaluate the ELBO in Equation 3.19 because (i) there are well-known algorithms for sampling from the distribution using rejection-sampling (Ulrich, 1984), and (ii) both the entropy term $H(q_\phi)$ and its gradient can be derived analytically (Davidson et al., 2018). For details of how to differentiate through rejection sampling, we refer to Naesseth et al. (2016) and Davidson et al. (2018).

In the following, we provide details for applying mGPLVM to S^2, for which we do not need to use rejection sampling and instead use inverse transform sampling (Jakob, 2012). For S^2, the VMF distribution simplifies to (Straub, 2017)

$$q_\phi(\boldsymbol{g}; \boldsymbol{g}^\mu, \kappa) = \frac{\kappa}{2\pi(\exp(\kappa) - \exp(-\kappa))} \exp(\kappa \boldsymbol{g}^\mu \cdot \boldsymbol{g}), \tag{B.14}$$

and its entropy is

$$\begin{aligned} H(q_\phi) &= -\int_{S^2} q_\phi(\boldsymbol{g}; \boldsymbol{g}^\mu, \kappa) \log q_\phi(\boldsymbol{g}; \boldsymbol{g}^\mu, \kappa) d\boldsymbol{g} && \text{(B.15)} \\ &= -\log\left(\frac{\kappa}{4\pi \sinh\kappa}\right) - \frac{\kappa}{\tanh\kappa} + 1. && \text{(B.16)} \end{aligned}$$

These equations allow us to apply mGPLVM to S^2 by optimizing the ELBO as described in the main text; this is illustrated for synthetic data on S^2 in Figure B.2 (top).

Posterior over tuning curves

We can derive the posterior over tuning curves in Equation 3.26 as follows:

$$\begin{aligned} p(\boldsymbol{f}_i^\star | \boldsymbol{Y}, \mathcal{G}^\star) &= \int p(\boldsymbol{f}_i^\star, \mathcal{G} | \mathcal{G}^\star, \boldsymbol{Y}) \; d\mathcal{G} && \text{(B.17)} \\ &= \int p(\boldsymbol{f}_i^\star | \mathcal{G}^\star, \{\mathcal{G}, \boldsymbol{Y}\}) p(\mathcal{G} | \boldsymbol{Y}) \; d\mathcal{G} && \text{(B.18)} \\ &\approx \int p(\boldsymbol{f}_i^\star | \mathcal{G}^\star, \{\mathcal{G}, \boldsymbol{Y}\}) q_\phi(\mathcal{G}) \; d\mathcal{G} && \text{(B.19)} \\ &\approx \frac{1}{K} \sum_{k=1}^{K} p(\boldsymbol{f}_i^\star | \mathcal{G}^\star, \{\mathcal{G}_k, \boldsymbol{Y}\}) && \text{(B.20)} \end{aligned}$$

Here, each $\mathcal{G}_k$ is a set of M latents (one for each of the M conditions in the data $\boldsymbol{Y}$) sampled from the variational posterior $q_\phi(\mathcal{G})$. The standard deviation around the mean tuning curves in all figures are estimated from 1000 independent samples from this posterior, with each draw involving the following two steps: (i) draw a sample $\mathcal{G}_k$ from q_ϕ and (ii) conditioned on this sample, draw from the predictive distribution $p(\boldsymbol{f}_i^\star | \mathcal{G}^\star, \{\mathcal{G}_k, \boldsymbol{Y}\})$. Together, these two steps correspond to a single draw from the posterior. Note that we make a variational sparse GP approximation (Section 3.2.2) and therefore approximate the predictive distribution $p(\boldsymbol{f}_i^\star | \mathcal{G}^\star, \{\mathcal{G}_k, \boldsymbol{Y}\})$ as described in Titsias (2009).

Alignment for visualization

The mGPLVM solutions for non-Euclidean spaces are degenerate because the ELBO depends on the sampled latents through (i) their uniform prior density, (ii) their entropy, and (iii) the GP marginal likelihood, and all three quantities are invariant to transformations that preserve pairwise distances. For example, the application of a common group element g to *all* the variational means leaves pairwise distances unaffected and therefore does not affect the ELBO. Additionally, pairwise distances are invariant to reflections along any axis of the coordinate system we have chosen to represent each group. Therefore, to plot comparisons between true and fitted latents, we use numerical optimization to find a single distance-preserving transformation that minimizes the average geodesic distance between the variational means $\{g_j^\mu\}$ and the true latents $\{g_j\}$.

For the n-dimensional torus (Figures 3.6 and 3.7) which we parameterize as

$$\boldsymbol{g} \in \{(g_1, \cdots, g_n); \forall k : g_k \in [0, 2\pi]\},$$

the distance metric depends on $\cos(g_k - g_k')$ and is invariant to any translation and reflection of all latents along each dimension

$$g_k \to (\alpha_k g_k + \beta_k) \mod 2\pi$$

where $\alpha_k \in \{1, -1\}$ and $\beta_k \in [0, 2\pi]$. We optimize discretely over the $\{\alpha_k\}$ by trying every possible combination, and continuously over β_k for each combination of $\{\alpha_k\}$.

In the case of S^2, S^3 and $SO(3)$ (Figures B.2 and 3.7), the distance metrics are invariant to unitary transformations $\boldsymbol{g} \to \boldsymbol{R}\boldsymbol{g}$ where $\boldsymbol{R}\boldsymbol{R}^T = \boldsymbol{R}^T\boldsymbol{R} = \boldsymbol{I}$ for the parameterizations used in this work. For visualization of these groups, we align the inferred latents with the true latents by optimizing over $\boldsymbol{R}$ on the manifold of orthogonal matrices.

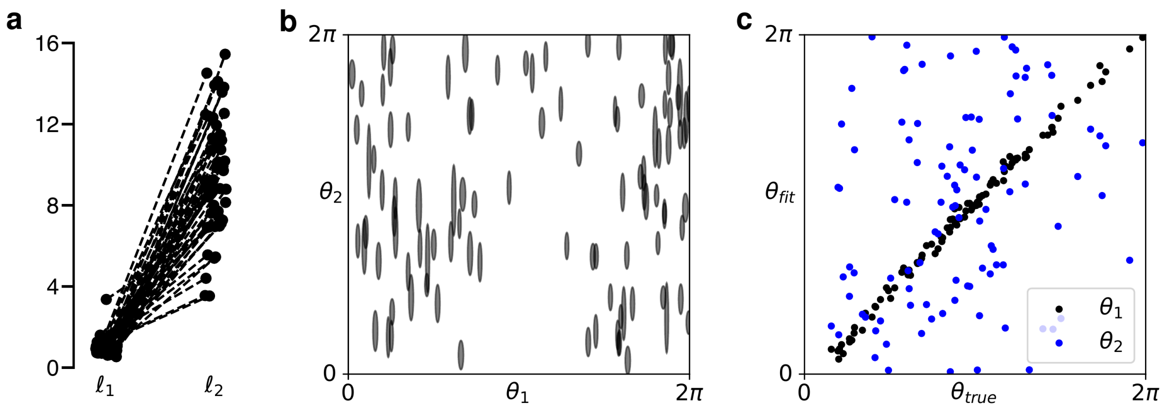

Figure B.3 **Automatic relevance determination (ARD) in T^2-mGPLVM.** A T^2 model with ARD was fitted to the T^1 data in Figure 3.6. **(a)** Length scales along each of the two dimensions for each neuron. **(b)** Posterior variational distributions. Shading indicates ±1 s.t.d. around the posterior mean in each dimension. **(c)** Variational mean plotted against the true latent state for each dimension.

Automatic relevance determination

As we mention in Section 3.2.4, it is possible to exploit automatic relevance determination (ARD) for automatic selection of the dimensionality of groups with additive distance metrics such as the T^n-distance in Equation B.7. While we have not investigated this in detail, we illustrate the idea here on a simple example. We consider the same synthetic data as in Figure 3.6 and fit a T^2-mGPLVM with a kernel on T^2 that has separate lengthscales ℓ_1 and ℓ_2 for each dimension:

$$k_{T^2_{\text{ARD}}}(\boldsymbol{g},\boldsymbol{g}') = \alpha^2 \exp\left(\frac{\cos(g_1 - g_1') - 1}{\ell_1^2}\right) \exp\left(\frac{\cos(g_2 - g_2') - 1}{\ell_2^2}\right). \tag{B.21}$$

Additionally, we assume the variational distribution to factorize across latent dimensions:

$$q_{\phi_j}(\cdot) = q_{\phi_j^1}(\cdot)\, q_{\phi_j^2}(\cdot), \tag{B.22}$$

such that their entropies add up to the total entropy:

$$H(q_{\phi_j}) = H(q_{\phi_j^1}) + H(q_{\phi_j^2}). \tag{B.23}$$

This corresponds to assuming that each variational covariance matrix $\boldsymbol{\Sigma}_j$ (Section 3.2.2) is diagonal.

When fitting this model, we find that one length parameter goes to large values while the other remains on the order of the size of the space (Figure B.3a; note that $d_{T^1} \in [0,4]$). This indicates that neurons are only tuned to one of the two torus dimensions. Additionally, posterior variances become very large in the non-contributing dimension, i.e. the data does not constrain the other angular dimension (Figure B.3b). This further indicates that the model has effectively shrunk from a 2-torus to a single circle. We note that the entropy of the factor in the variational posterior that corresponds to the discarded dimension becomes $\log 2\pi$ as the variance goes to infinity in this direction. This exactly offsets the increased complexity penalty of the prior for T^2 compared to T^1, such that the two models have the same ELBO. The model thus reduces to a T^1 model, demonstrating how ARD can be exploited to automatically infer the dimensionality of the latent space.

Direct products of Lie groups

Here, we elaborate slightly on the extension of mGPLVM to direct products of Lie groups, briefly mentioned in the discussion (Section 3.2.4). Assuming additive distance metrics and factorized variational distributions, direct product kernels become multiplicative and entropies become additive – very much as in our illustration of ARD in Appendix B. That is, for a group product $\mathcal{M} = \mathcal{M}_1 \times \ldots \times \mathcal{M}_L$, we can write

$$k^{\mathcal{M}}(g, g') = \prod_l k^{\mathcal{M}_l}(g, g'), \tag{B.24}$$

$$H(q_{\phi_j}^{\mathcal{M}}) = \sum_l H(q_{\phi_j}^{\mathcal{M}_l}). \tag{B.25}$$

As a simple example, we consider a $(T^1 \times \mathbb{R}^1)$-mGPLVM, which we fit to the *Drosophila* data from Section 3.2.3. Here we find that the T^1 dimension of the group product, which we denote by $\theta^{(T^1 \times \mathbb{R}^1)}$, captures the angular component of the data since it is very strongly correlated with the latent state θ^{T^1} inferred by the simpler T^1-mGPLVM (Figure B.4a). It is somewhat harder to predict what features of the data will be captured by the $\mathbb{R}^1$ dimension $x^{(T^1 \times \mathbb{R}^1)}$ of the $(T^1 \times \mathbb{R}^1)$-mGPLVM, but we hypothesize that it might capture a global temporal modulation of the neural activity. We therefore plot the mean instantaneous activity $\bar{y}$ across neurons against $x^{(T^1 \times \mathbb{R}^1)}$ and find that these quantities are indeed positively correlated (Figure B.4b). This exemplifies how an mGPLVM on a direct product of groups can capture qualitatively different components of the data by combining representations with different topologies.

This direct product model is very closely related to the ARD model in Appendix B, and the two can also be combined in a direct product of ARD kernels. For example, we can imagine constructing a $(T^n \times \mathbb{R}^n)$ direct product ARD kernel, which automatically selects the appropriate number of both periodic and scalar dimensions that best, and most parsimoniously, explains the data.

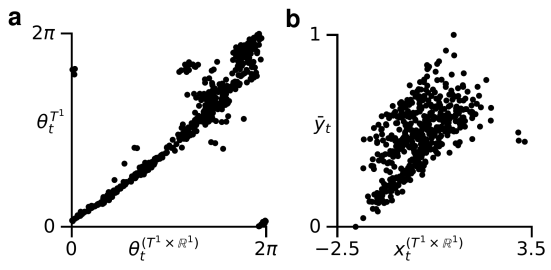

Figure B.4 $(T^1 \times \mathbb{R}^1)$**-mGPLVM.** **(a)** Latent states inferred by T^1-mGPLVM (Figure 3.8a) against the periodic coordinate of a $(T^1 \times \mathbb{R}^1)$-mGPLVM fitted to the *Drosophila* data. **(b)** Momentary average population activity $\bar{y}_t$ against the scalar Euclidean component of the $(T^1 \times \mathbb{R}^1)$ latent representation.

Implementation

Scaling As mentioned in Section 3.2.2, approximating the GP likelihood term $\mathbb{E}_{q_\phi}[\log p(\boldsymbol{Y}|\{g_j\})]$ in the mGPLVM ELBO scales as $\mathcal{O}(m^2MNK)$ with m inducing points, M latent states, N neurons, and K Monte Carlo samples. Estimating the entropy term is $\mathcal{O}(MKd)$ for a d-dimensional Euclidean latent space, $\mathcal{O}(MK(2k_{max}+1)^d)$ for a d-dimensional torus, and $\mathcal{O}(MK(2k_{max}+1))$ for $SO(3)$ and S^3, where k_{max} is the maximum value of k used in Equation 3.22. For all manifolds considered in this work, we can compute a closed-form $\text{Exp}(\cdot)$ while for general matrix Lie groups, approximating Exp as a power series is $\mathcal{O}(d^3)$ (Falorsi et al., 2019), further increasing the complexity of mGPLVM for such groups.

For our manifolds of interest, computing the likelihood term tends to be the main computational bottleneck, although the entropy term can become prohibitive for high-dimensional periodic latents (Rezende et al., 2020). When computing $\mathbb{E}_{q_\phi}[\log p(\boldsymbol{Y}|\{g_j\})]$, most of the complexity is due to inverting NK matrices of size $(Mm^2)\times(Mm^2)$, which can be performed in parallel for each Monte Carlo sample and neuron. Using PyTorch for parallelization across neurons and MC samples, we can train T^1-mGPLVM with $N = 300$ and $M = 1000$ in $\sim$100 seconds on an NVIDIA GeForce RTX 2080 GPU with 8GB RAM.

Initialization For all simulations, we initialized the system with variational means at the identity element of the manifold, but with large variational variances to reflect the lack of prior information about the true latent states. Inducing points were initialized according to the prior on each manifold (Equation 3.15). To avoid variational distributions collapsing to the uniform distribution early during learning, we ran a preliminary ‘warm up’ optimization phase during which some of the parameters were held fixed. Specifically, we fixed the variational covariance matrices as well as the kernel variance parameters (α in Equation 3.27), and prioritized a better data fit by setting the entropy term to zero in Equation 3.19. Learning proceeded as normal thereafter.

Entropy approximation When evaluating Equation 3.22, we used values of $k_{max} = 3$ for the tori and S^3 as in Falorsi et al. (2019) and $k_{max} = 5$ for $SO(3)$ since the sum takes steps of

π instead of 2π. In theory, the finite k_{max} can lead to an overestimation of the ELBO for large variational uncertainties, as $\tilde{q}$ is systematically underestimated, leading to overestimation of the entropy. To mitigate this, we capped the approximate entropy for non-Euclidean manifolds at the maximum entropy corresponding to a uniform distribution on the manifold.

Code A python package implementing mGPLVM can be found at https://github.com/tachukao/mgplvm-pytorch.

Appendix C

Natural continual learning

Derivation of the NCL learning rule

In this section, we provide further details of how the NCL learning rule in Section 4.1.2 is derived and also provide an alternative derivation of the algorithm.

NCL learning rule As discussed in Section 4.1.2, we derive NCL as the solution of a trust region optimization problem. That is, we maximize the posterior loss $\mathcal{L}_k(\theta)$ within a region of radius r centered around θ with a distance metric of the form $d(\theta, \theta+\boldsymbol{\delta}) = \sqrt{\boldsymbol{\delta}^\top \boldsymbol{\Lambda}_{k-1}\boldsymbol{\delta}/2}$. This distance metric was chosen to take into account the curvature of the prior via its precision matrix $\boldsymbol{\Lambda}_{k-1}$ and encourage parameter updates that do not affect performance on previous tasks. Formally, we solve the optimization problem

$$\boldsymbol{\delta} = \arg\min_{\boldsymbol{\delta}} \mathcal{L}_k(\theta) + \nabla_\theta \mathcal{L}_k(\theta)^\top \boldsymbol{\delta} \quad \text{subject to } \frac{1}{2}\boldsymbol{\delta}^\top \boldsymbol{\Lambda}_{k-1}\boldsymbol{\delta} \leq r^2, \tag{C.1}$$

where $\mathcal{L}_k(\theta+\boldsymbol{\delta}) \approx \mathcal{L}_k(\theta) + \nabla_\theta \mathcal{L}_k(\theta)^\top \boldsymbol{\delta}$ is a first-order approximation to the updated Laplace objective. Here we recall from Equation 2.31 that

$$\mathcal{L}_k(\theta) = \ell_k(\theta) - \frac{1}{2}(\theta - \boldsymbol{\mu}_{k-1})^T \boldsymbol{\Lambda}_{k-1}(\theta - \boldsymbol{\mu}_{k-1}) \tag{C.2}$$

from which we get

$$\nabla_\theta \mathcal{L}_k(\theta)^\top \boldsymbol{\delta} = \nabla_\theta \ell_k(\theta)^\top \boldsymbol{\delta} - (\theta - \boldsymbol{\mu}_{k-1})^\top \boldsymbol{\Lambda}_{k-1}\boldsymbol{\delta} \tag{C.3}$$

The optimization in Equation C.1 is carried out by introducing a Lagrange multiplier η to construct a Lagrangian $\tilde{\mathcal{L}}$:

$$\tilde{\mathcal{L}}(\boldsymbol{\delta}, \eta) = \mathcal{L}_k(\theta) + \nabla_\theta \ell_k(\theta)^\top \boldsymbol{\delta} - (\theta - \boldsymbol{\mu}_{k-1})^\top \boldsymbol{\Lambda}_{k-1}\boldsymbol{\delta} + \eta(r^2 - \frac{1}{2}\boldsymbol{\delta}^\top \boldsymbol{\Lambda}_{k-1}\boldsymbol{\delta}). \tag{C.4}$$

We then take the derivative of $\tilde{\mathcal{L}}$ w.r.t. $\boldsymbol{\delta}$ and set it to zero:

$$\nabla_{\boldsymbol{\delta}} \tilde{\mathcal{L}}(\boldsymbol{\delta}, \eta) = \nabla_\theta \ell_k(\theta) - \boldsymbol{\Lambda}_{k-1}(\theta - \boldsymbol{\mu}_{k-1}) - \eta \boldsymbol{\Lambda}_{k-1}\boldsymbol{\delta}' = 0. \tag{C.5}$$

Rearranging this equation gives

$$\boldsymbol{\delta} = \frac{1}{\eta}\left[\boldsymbol{\Lambda}_{k-1}^{-1}\nabla_\theta \ell_k(\theta) - (\theta - \boldsymbol{\mu}_{k-1}),\right]. \tag{C.6}$$

where η itself depends on r^2 implicitly. Finally we define a learning rate parameter $\gamma = 1/\eta$ and arrive at the NCL learning rule:

$$\theta \leftarrow \theta + \gamma\left[\boldsymbol{\Lambda}_{k-1}^{-1}\nabla_\theta \ell_k(\theta) - (\boldsymbol{\theta} - \boldsymbol{\mu}_{k-1})\right]. \tag{C.7}$$

Alternative derivation Here, we present an alternative derivation of the NCL learning rule. In this formulation, we seek to update the parameters of our model on task k by maximizing $\mathcal{L}_k(\theta)$ subject to a constraint on the allowed change in the prior term. To find our parameter updates $\boldsymbol{\delta}$, we again solve a constrained optimization problem:

$$\boldsymbol{\delta} = \arg\min_{\boldsymbol{\delta}} \mathcal{L}_k(\theta) + \nabla_\theta \mathcal{L}_k(\theta)^\top \boldsymbol{\delta} \quad \text{such that} \quad \mathcal{C}(\boldsymbol{\delta}) \leq r^2. \tag{C.8}$$

Here we define $\mathcal{C}(\boldsymbol{\delta})$ as the approximate change in log probability under the prior

$$\mathcal{C}(\boldsymbol{\delta}) = (\theta + \boldsymbol{\delta} - \boldsymbol{\mu}_{k-1})^\top \boldsymbol{\Lambda}_{k-1}(\theta + \boldsymbol{\delta} - \boldsymbol{\mu}_{k-1}) - (\theta - \boldsymbol{\mu}_{k-1})^\top \boldsymbol{\Lambda}_{k-1}(\theta - \boldsymbol{\mu}_{k-1}). \tag{C.9}$$

Following a similar derivation to above, we find the solution to this optimization problem as

$$\eta\boldsymbol{\delta} = \boldsymbol{\Lambda}_{k-1}^{-1}\nabla_\theta \mathcal{L}_k(\theta) - \eta(\theta - \boldsymbol{\mu}_{k-1}) = \boldsymbol{\Lambda}_{k-1}^{-1}\nabla_\theta \ell_k(\theta) - (1+\eta)(\theta - \boldsymbol{\mu}_{k-1}) \tag{C.10}$$

for some Lagrange multiplier η. This gives rise to the update rule

$$\theta \leftarrow \theta + \gamma\left[\boldsymbol{\Lambda}_{k-1}^{-1}\nabla_\theta \ell_k(\theta) - \lambda(\theta - \boldsymbol{\mu}_{k-1})\right] \tag{C.11}$$

for a learning rate parameter γ and some choice of the parameter λ that depends on both η and γ. We recover the learning rule derived in Section 4.1.2 with the choice of $\lambda = 1$. In practice, λ can also be treated as a hyperparameter to be optimized.

Task details

Split MNIST The split MNIST benchmark involves 5 tasks, each corresponding to the pairwise classification of two digits. The 10 digits of the MNIST dataset are randomly divided over the 5 tasks (i.e., for each random seed, this division can be different). During the incremental training protocol, these tasks are visited one after the other, followed by testing on all tasks. The original 28×28 pixel grey-scale images and the standard train/test-split are used, giving 60,000 training (∼6,000 per digit) and 10,000 test images (∼1,000 per digit).

Split CIFAR-100 The split CIFAR-100 benchmark consists of 10 tasks, with each task corresponding to a ten-way classification problem. The 100 classes of the CIFAR-100 dataset are randomly divided over the 10 tasks. Each network is trained on these tasks one after the other followed by testing on all tasks. The 32×32 pixel RGB-colour images are normalised by z-scoring each channel (using means and standard deviations calculated over the training set). We use the standard train/test-split, giving 500 training and 100 test images for each class.

Stimulus-response tasks Here, we provide a brief overview of the six stimulus-response (SR) tasks. Detailed descriptions of the stimulus-response tasks used in this work can be found in the appendix of Yang et al. (2019). All tasks are characterized by a stimulus period and a response period, and some tasks include an additional delay period between the two. The duration of the stimulus and delay periods are variable across trials and drawn uniformly at random within an allowed range. During the stimulus period, the input to the network takes the form of $\boldsymbol{x} = (\cos\theta_{in}, \sin\theta_{in})$, where $\theta_{in} \in [0, 2\pi]$ is some stimulus drawn uniformly at random for each trial. An additional tonic input is provided to the network, which indicates the identity of the task using a one-hot encoding. A constant input to a 'fixation channel' during the stimulus and delay periods signifies that the network output should be 0 in the response channels and 1 in a 'fixation channel'. During the response period, the fixation input is removed and the output should be 0 in the fixation channel. The target output in the response channels takes the form $\boldsymbol{y} = (\cos\theta_{out}, \sin\theta_{out})$ where θ_{out} is some target output direction described for each task below:

- **task 1 (fdgo)** During this task $\theta_{out} = \theta_{in}$ and there is no delay period.
- **task 2 (fdanti)** During this task $\theta_{out} = \pi + \theta_{in}$ and there is no delay period.
- **task 3 (delaygo)** During this task $\theta_{out} = \theta_{in}$ and there is a delay period separating the stimulus and response periods.
- **task 4 (delayanti)** During this task $\theta_{out} = \pi + \theta_{in}$ and there is a delay period separating the stimulus and response periods.
- **task 5 (dm1)** During this task, two stimuli are drawn from $[0, 2\pi]$ with different input magnitudes such that $\boldsymbol{x} = (m_1 \cos\theta_1 + m_2 \cos\theta_2, m_1 \sin\theta_1 + m_2 \sin\theta_2)$. θ_{out} is then the element in (θ_1, θ_2) corresponding to the largest m.
- **task 6 (dm2)** As in 'dm1', but where the input is now provided through a separate input channel.

The loss for each task was computed as a mean squared error from the target output.

SMNIST For this task set, we use the stroke MNIST dataset created by de Jong (2016). This consists of a series of digits, each of which is represented as a sequence of vectors $\{\boldsymbol{x}_t \in \mathbb{R}^4\}$. The first two columns take values in $[-1, 0, 1]$ and indicate the discretized displacement in the x and y direction at each time step. The last two columns are used for special 'end-of-line' inputs when the virtual pen is lifted from the paper for a new stroke to start, and an 'end-of-digit' input when the digit is finished. See de Jong (2016) for further details about how the dataset was generated and formatted. In addition to the standard digits 0-9, we include two additional sets of digits:

- the digits 0-9 where the x and y directions have been swapped (i.e. the first two elements of $\boldsymbol{x}_t$ are swapped),
- the digits 0-9 where the x and y directions have been inverted (i.e. the first two elements of $\boldsymbol{x}_t$ are negated).

Furthermore, we omitted the initial entry of each digit corresponding to the 'start' location to increase task difficulty. We turned this dataset into a continual learning task by constructing five binary classification tasks for each set of digits: $\{[2,3],[4,5],[1,7],[8,9],[0,6]\}$. Note that we have swapped the '1' and '6' from a standard split MNIST task to avoid including the 0 vs 1 classification task, which we found to be too easy. For each trial, a digit was sampled at random from the corresponding dataset, and $\boldsymbol{x}_t$ was provided as an input to the network at each time step, corrupted by Gaussian noise with $\sigma = 1$. After the 'end-of-digit' input, a response period with a duration of 5 time steps followed. During this response period only, a cross-entropy loss was applied to the output units $\boldsymbol{y}$ to train the network. During testing, digits were sampled from the separate test dataset and classification performance was quantified as the fraction of digits for which the correct class was assigned the highest probability in the last timestep of the response period. Task identity was provided to the network, which was used in the form of a multi-head output layer.

Network architectures

Feedforward network archictecture For split MNIST, all methods are compared using a fully-connected network with 2 hidden layers containing 400 units with ReLU non-linearities, followed by a softmax output layer.

For split CIFAR-100, the network consists of 5 pre-trained convolutional layers, 2 fully-connected layers with 2000 ReLU units each, and a softmax output layer. The architecture of the convolutional layers and their pre-training protocol on the CIFAR-10 dataset are described by van de Ven et al. (2020). The only difference is that here we pre-train a new set of convolutional layers for each random seed, while van de Ven et al. (2020) used the same set of pre-trained convolutional layers for all random seeds. For all compared methods, the pre-trained convolutional layers are frozen during the incremental training protocol.

The softmax output layer of the feedforward networks is treated differently depending on the continual learning setting (van de Ven and Tolias, 2019). In the task-incremental learning setting, there is a separate output layer for each task and only the output layer of the task under consideration is used at any given time (i.e., a multi-head output layer). In the domain-incremental learning setting, there is a single output layer that is shared between all tasks. In the class-incremental learning setting, there is one large output layer that spans all tasks and contains a separate output unit for each class.

Recurrent network architecture The dynamics of the RNN used in Section 4.1.3 can be described by the following equations:

$$\boldsymbol{h}_t = \boldsymbol{H}\boldsymbol{r}_{t-1} + \boldsymbol{G}\boldsymbol{x}_t + \boldsymbol{\xi}_t = \boldsymbol{W}\boldsymbol{z}_t + \boldsymbol{\xi}_t \tag{C.12}$$

$$\boldsymbol{y}_t \sim p(\boldsymbol{y}_t | \boldsymbol{C}\boldsymbol{r}_t) \tag{C.13}$$

where we define $\boldsymbol{r}_t = \phi(\boldsymbol{h}_t)$, $\boldsymbol{z}_t = (\boldsymbol{r}_{t-1}^\top, \boldsymbol{x}_t^\top)^\top$, $\boldsymbol{W} = (\boldsymbol{H}^\top, \boldsymbol{G}^\top)^\top$, and time is indexed by t. Here, $\boldsymbol{r} \in \mathbb{R}^{N_{rec} \times 1}$ are the network activations, $\boldsymbol{x} \in \mathbb{R}^{n_{in} \times 1}$ are the inputs, $\boldsymbol{y} \in \mathbb{R}^{n_{out} \times 1}$ are the network outputs, and we refer to $\boldsymbol{W}\boldsymbol{z}_t$ as the 'recurrent inputs' to the network. The noise model $p(\boldsymbol{y}_t | \boldsymbol{C}\boldsymbol{r}_t)$ may be a Gaussian distribution for a regression task or a categorical distribution for a classification task, and $\phi(\boldsymbol{h})$ is a nonlinearity that is applied to $\boldsymbol{h}$ element-wise (in this work the ReLU function). The parameters of the RNN are given by $\theta = (\boldsymbol{W}, \boldsymbol{C})$. The process noise $\{\boldsymbol{\xi}_t\}$ are zero-mean Gaussian random variables with covariance matrices $\boldsymbol{\Sigma}_t^{\boldsymbol{\xi}}$. In this model, the log-likelihood of observing a sequence of outputs $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_T$ given inputs $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T$ and $\boldsymbol{\xi}_1, \ldots, \boldsymbol{\xi}_T$ is given by

$$\ell(\theta) = \log p_\theta(\{\boldsymbol{y}\} | \{\boldsymbol{x}\}, \{\boldsymbol{\xi}\}) = \log p(\{\boldsymbol{y}\} | \{\boldsymbol{C}\boldsymbol{r}\}), \tag{C.14}$$

where $p(\boldsymbol{y}|\boldsymbol{C}\boldsymbol{r})$ may be a Gaussian distribution for a regression task or a categorical distribution for a classification task.

KFAC approximation to the Fisher matrix

For all experiments in this work, we make a Kronecker-factored approximation to the FIM of each task k in Equation 2.33. Concretely, we use the block-wise Kronecker-factored approximation to the FIM proposed in Section 3 of Martens and Grosse (2015) for feedforward neural networks. For recurrent neural networks, we use the approximation presented in Section 3.4 of Martens et al. (2018). Both approximations allow us to write the FIM on task k as the Kronecker product $\boldsymbol{F}_k \approx \hat{\boldsymbol{A}}_k \otimes \hat{\boldsymbol{G}}_k$. For completeness, we derive the approximation for RNNs below. We refer the readers to Martens and Grosse (2015) for details on derivations for feedforward networks.

KFAC approximation for RNNs Recall from Appendix C that the log likelihood of observing a sequence of outputs $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_T$ given inputs $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T$ and $\boldsymbol{\xi}_1, \ldots, \boldsymbol{\xi}_T$ is

$$\ell(\boldsymbol{W}, \boldsymbol{C}) = \sum_{t=1}^{T} \log p(\boldsymbol{y}_t | \boldsymbol{C}\boldsymbol{r}_t), \tag{C.15}$$

where $\boldsymbol{r}_t$ is completely determined by the dynamics of the network and the inputs. With a slight abuse of notation, we use $\overline{\boldsymbol{x}}$ to denote both $\partial\ell/\partial\boldsymbol{x}$ for vectors $\boldsymbol{x}$ and $\partial\ell/\partial\text{vec}(\boldsymbol{X})$ for matrices $\boldsymbol{X}$. In this section, it should be clear given the context whether $\overline{\boldsymbol{x}}$ is representing the gradient of $\mathcal{L}$ with respect to a vector or a vectorized matrix. Using these notations, we can write the gradient of $\mathcal{L}$ with respect to vec($\boldsymbol{W}$) as :

$$\overline{\boldsymbol{w}} = \sum_{t=1}^{T} \overline{\boldsymbol{h}}_t \frac{\partial \boldsymbol{h}_t}{\partial \text{vec}(\boldsymbol{W})} = \sum_{t=1}^{T} \overline{\boldsymbol{h}}_t \boldsymbol{z}_t^\top = \sum_{t=1}^{T} \boldsymbol{z}_t \otimes \overline{\boldsymbol{h}}_t \tag{C.16}$$

which can be easily derived fom the backpropagation through time (BPTT) algorithm and the definition of a Kronecker product. Using this expression for $\overline{\boldsymbol{w}}$, we can write the FIM of $\boldsymbol{W}$ as:

$$\boldsymbol{F}_{\boldsymbol{W}} = \mathbb{E}_{\{(\boldsymbol{\xi}, \boldsymbol{x}, \boldsymbol{y})\} \sim \mathcal{M}} \left[\overline{\boldsymbol{w}}\, \overline{\boldsymbol{w}}^\top \right] \tag{C.17}$$

$$= \mathbb{E} \left[\left(\sum_{t=1}^{T} \boldsymbol{z}_t \otimes \overline{\boldsymbol{h}}_t \right) \left(\sum_{s=1}^{T} \boldsymbol{z}_s \otimes \overline{\boldsymbol{h}}_s \right)^\top \right] \tag{C.18}$$

$$= \sum_{t=1}^{T} \sum_{s=1}^{T} \mathbb{E} \left[\left(\boldsymbol{z}_t \boldsymbol{z}_s^\top \right) \otimes \left(\overline{\boldsymbol{h}}_t \overline{\boldsymbol{h}}_s^\top \right) \right]. \tag{C.19}$$

Here the expectations are taken with respect to the model distribution. Unfortunately, computing $\boldsymbol{F}_{\boldsymbol{W}}$ can be prohibitively expensive. First, the number of computations scales quadratically with the length of the input sequence T. Second, for networks of dimension n, there are n^4 entries in the Fisher matrix which can therefore be too large to store in memory, let alone perform any useful computations with it. For this reason, we follow Martens et al. (2018) and make the following three assumptions in order to derive a tractable Kronecker-factored approximation to the Fisher. The first assumption we make is that the input and recurrent activty $\boldsymbol{z}_t$ is uncorrelated with the adjoint activations $\overline{\boldsymbol{h}}_t$:

$$\boldsymbol{F_W} \approx \sum_{t=1}^{T}\sum_{s=1}^{T} \mathbb{E}\left[\boldsymbol{z}_t \boldsymbol{z}_s^\top\right] \otimes \mathbb{E}_{\{(\boldsymbol{\xi},\boldsymbol{x},\boldsymbol{y})\}\sim\mathcal{M}}\left[\overline{\boldsymbol{h}}_t \overline{\boldsymbol{h}}_s^\top\right]. \tag{C.20}$$

Note that this approximation is exact when the network dynamics are linear (i.e., $\phi(\boldsymbol{x}) = \boldsymbol{x}$). The second assumption that we make is that both the forward activity $\boldsymbol{z}_t$ and adjoint activity $\overline{\boldsymbol{h}}_t$ are temporally homogeneous. That is, the statistical relationship between $\boldsymbol{z}_t$ and $\boldsymbol{z}_s$ only depends on the difference $\tau = s - t$, and similarly for that between $\overline{\boldsymbol{h}}_t$ and $\overline{\boldsymbol{h}}_s$. Defining $\mathcal{A}_\tau = \mathbb{E}\left[\boldsymbol{z}_s \boldsymbol{z}_{s+\tau}^\top\right]$ and similarly $\mathcal{G}_\tau = \mathbb{E}\left[\overline{\boldsymbol{h}}_s \overline{\boldsymbol{h}}_{s+\tau}^\top\right]$, we have $\mathcal{A}_{-\tau} = \mathcal{A}_\tau^\top$ and $\mathcal{G}_{-\tau} = \mathcal{G}_\tau$. Using these expressions, we can further approximate the Fisher as:

$$\boldsymbol{F_W} \approx \sum_{\tau=-T}^{T} (T - |\tau|)\mathcal{A}_\tau \otimes \mathcal{G}_\tau. \tag{C.21}$$

The third and final approximation we make is that $\mathcal{A}_\tau \approx 0$ and $\mathcal{G}_\tau \approx 0$ for $\tau \neq 0$. In other words, we assume the forward activity $\boldsymbol{z}_t$ and adjoint activity $\overline{\boldsymbol{h}}_t$ are approximately indendent across time. This gives the final expression:

$$\boldsymbol{F_W} \approx \mathbb{E}[T]\,\mathbb{E}\left[\boldsymbol{z}\boldsymbol{z}^\top\right] \otimes \mathbb{E}\left[\overline{\boldsymbol{h}}\,\overline{\boldsymbol{h}}^\top\right] = \hat{\boldsymbol{A}}_{\boldsymbol{W}} \otimes \hat{\boldsymbol{G}}_{\boldsymbol{W}}, \tag{C.22}$$

where we have also taken an expectation over the sequence length T to account for variable sequence lengths in the data. Following a similar derivation, we can approximate the Fisher of $\boldsymbol{C}$ as:

$$\boldsymbol{F_C} \approx \mathbb{E}[T]\,\mathbb{E}\left[\boldsymbol{r}\boldsymbol{r}^\top\right] \otimes \mathbb{E}\left[\overline{\boldsymbol{y}}\,\overline{\boldsymbol{y}}^\top\right] = \hat{\boldsymbol{A}}_{\boldsymbol{C}} \otimes \hat{\boldsymbol{G}}_{\boldsymbol{C}}. \tag{C.23}$$

Implementation

Algorithm 2: NCL with momentum

```
1  input: $f$ (network), $\{\mathcal{D}_k\}_{k=1}^{K}$, $\alpha$, $p_w$ (prior), $B$ (batch size), $\gamma$ (learning rate), $\theta_0$, $\rho$
2  initialize: $\boldsymbol{A}_\theta \leftarrow p_w \boldsymbol{I}$, $\boldsymbol{G}_\theta \leftarrow p_w \boldsymbol{I}$,
3  initialize: $\theta_1 \leftarrow \theta_0$, initialize: $\boldsymbol{M}_\theta \leftarrow$ zeros_like($\theta_0$),        // Gradient momentum
4  for $k = 1 \dots K$ do
5      $\widetilde{\boldsymbol{A}}, \widetilde{\boldsymbol{G}} \leftarrow$ nearest_kf_sum$(\boldsymbol{A}_\theta \otimes \boldsymbol{G}_\theta, \alpha \boldsymbol{I} \otimes \alpha \boldsymbol{I})$
6      $\boldsymbol{P}_L \leftarrow \widetilde{\boldsymbol{G}}^{-1}$
7      $\boldsymbol{P}_R \leftarrow \widetilde{\boldsymbol{A}}^{-1}$
8      while not converged do
9          $\{\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}\}_{i=1}^{B} \sim \mathcal{D}_k$                // Input and target output
10         for $i = 1, \dots, B$ do
11             $\hat{\boldsymbol{y}}^{(i)} = f(\boldsymbol{x}^{(i)}, \theta_k)$                 // Empirical output
12         $\ell = \sum_i^B \log p(\boldsymbol{y}^{(i)} | \hat{\boldsymbol{y}}^{(i)}) / B$                         // Loss
13
14         % Build up momentum
15         $\boldsymbol{M}_\theta \leftarrow \rho \boldsymbol{M}_\theta + \nabla_\theta \ell + \boldsymbol{G}_\theta (\theta_k - \theta_{k-1}) \boldsymbol{A}_\theta$
16
17         % Update model parameters
18         $\theta_k \leftarrow \theta_k - \gamma p_w^2 \boldsymbol{P}_L \boldsymbol{M}_\theta \boldsymbol{P}_R$
19     % Update Fisher matrix components
20     Compute $\hat{\boldsymbol{A}}_k$ and $\hat{\boldsymbol{G}}_k$
21     $\boldsymbol{A}_\theta, \boldsymbol{G}_\theta \leftarrow$ nearest_kf_sum$(\boldsymbol{A}_\theta \otimes \boldsymbol{G}_\theta, \hat{\boldsymbol{A}}_k \otimes \hat{\boldsymbol{G}}_k)$
```

In this section we discuss various implementation details for NCL. Algorithm 2 provides an overview of the algorithm in the form of pseudocode. For numerical stability, we add $\alpha^2 \boldsymbol{I}$ to the precision matrix $\boldsymbol{\Lambda}_{k-1}$ before computing the projection matrices $\boldsymbol{P}_L$ and $\boldsymbol{P}_R$. In general, we set the prior over the parameters θ when learning the first task as $p(\theta) = \mathcal{N}(\boldsymbol{0}; p_w^{-2} \boldsymbol{I})$.

Feedforward networks By default, we set p_w^{-2} to be approximately the number of samples that the learner sees in each task, corresponding to a unit Gaussian prior before normalizing our precision matrices by the amount of data seen in each task (here, $p_w^{-2} = 12000$ for split MNIST and $p_w^{-2} = 5000$ for split CIFAR-100). We also consider hyperparameter optimizations over p_w^{-2} by trying different values on a log scale from 10^2 to 10^{11} with a random seed not included during the evaluation. We use $\alpha = 10^{-10}$ and $\lambda = 1$ for all experiments.

For all experiments with feedforward networks, we use a batch size of 256 and we train for either 2000 iterations per task (split MNIST) or 5000 iterations per task (split CIFAR-100). For NCL and OWM, we train with momentum ($\rho = 0.9$) and a learning rate of $\gamma = 0.05$. For SI, EWC and KFAC, we train using the Adam optimizer ($\beta_1 = 0.9$, $\beta_2 = 0.999$) with learning rate of $\gamma = 0.001$ (split MNIST) or $\gamma = 0.0001$ (split CIFAR-100). All models were trained on single GPUs with training times of 10-100 minutes.

RNNs We again set p_w^{-2} approximately equal to the number of samples that the learner sees in each task, corresponding to a unit Gaussian prior before normalizing our precision matrices by the amount of data seen in each task (here, $p_w^{-2} = 10^6$ for the stimulus-response task and $p_w^{-2} = 6000$ for SMNIST).

We used momentum ($\rho = 0.9$) in all our experiments involving NCL, OWM and DOWM, as is also done in Duncker et al. (2020). We found that the use of momentum greatly speeds up convergence in practice.

All models were trained on single GPUs with training times of 10-100 minutes depending on the task set and model size. We used a training batch size of 32 for the stimulus-response tasks and 256 for the SMNIST tasks. In all cases, we used a test batch size of 2048 for evaluation and for computing projection and Fisher matrices. We used a learning rate of $\gamma = 0.01$ for SMNIST and $\gamma = 0.005$ for the stimulus-response tasks across all projection-based methods. We used a learning rate of $\gamma = 0.001$ for KFAC with the Adam optimizer. All models were trained on 10^6 data samples per task. A hyperparameter optimization over α for the projection-based methods and λ for KFAC with Adam is provided in Figure C.6.

Relation to projection-based continual learning

In this section, we further elaborate on the intuition that projection-based continual learning methods such as Orthogonal Weight Modification (OWM; Zeng et al., 2019) may be viewed as variants of NCL with particular approximations to the prior Fisher matrix. These approaches are typically motivated as a way to restrict parameter changes in a neural network that is learning a new task to subspaces orthogonal to those used in previous tasks.

For example, to solve the continual learning problem in RNNs as described in Appendix C, Duncker et al. (2020) proposed a projected gradient algorithm (DOWM) that restricts modifications to the recurrent/input weight matrix $\boldsymbol{W}$ on task $k+1$ to column and row spaces of $\boldsymbol{W}$ that are not heavily "used" in the first k tasks. Specifically, they concatenate input and recurrent activity $\boldsymbol{z}_t$ across the first k tasks into a matrix $\boldsymbol{Z}_{1:k}$. They use $\boldsymbol{Z}_{1:k}$ and $\boldsymbol{W}\boldsymbol{Z}_{1:k}$ as estimates of the row and column spaces of $\boldsymbol{W}$ that are important for the first k tasks. They proceed to construct the following projection matrices:

$$\boldsymbol{P}_z^{1:k} = \boldsymbol{Z}_{1:k}(\boldsymbol{Z}_{1:k}\boldsymbol{Z}_{1:k}^\top + \alpha \boldsymbol{I})^{-1}\boldsymbol{Z}_{1:k}{}^\top \tag{C.24}$$

$$\approx k\alpha \left(\mathbb{E}\left[\boldsymbol{z}\boldsymbol{z}^\top\right] + \alpha \boldsymbol{I}\right)^{-1} \tag{C.25}$$

$$\boldsymbol{P}_{wz}^{1:k} = \boldsymbol{W}\boldsymbol{Z}_{1:k}(\boldsymbol{W}\boldsymbol{Z}_{1:k}\boldsymbol{Z}_{1:k}^\top \boldsymbol{W}^\top + \alpha \boldsymbol{I})^{-1}(\boldsymbol{W}\boldsymbol{Z}_{1:k})^\top \tag{C.26}$$

$$\approx k\alpha \left(\boldsymbol{W}\mathbb{E}\left[\boldsymbol{z}\boldsymbol{z}^\top\right]\boldsymbol{W}^\top + \alpha \boldsymbol{I}\right)^{-1}, \tag{C.27}$$

which are used to derive update rules for $\boldsymbol{W}$ as:

$$\text{vec}(\Delta \boldsymbol{W}) \propto \left(\boldsymbol{P}_z^{1:k} \otimes \boldsymbol{P}_{wz}^{1:k}\right)\overline{\boldsymbol{w}} \tag{C.28}$$

$$\propto \left(\mathbb{E}\left[\boldsymbol{z}\boldsymbol{z}^\top\right] + \alpha \boldsymbol{I}\right)^{-1} \otimes \left(\boldsymbol{W}\mathbb{E}\left[\boldsymbol{z}\boldsymbol{z}^\top\right]\boldsymbol{W}^\top + \alpha \boldsymbol{I}\right)^{-1}\overline{\boldsymbol{w}} \tag{C.29}$$

where $\overline{\boldsymbol{w}} = \text{vec}(\nabla_{\boldsymbol{W}}\ell_{k+1}(\boldsymbol{W},\boldsymbol{C}))$. These projection matrices restrict changes in the row and column space of $\boldsymbol{W}$ to be orthogonal to $\boldsymbol{Z}_{1:k}$ and $\boldsymbol{W}\boldsymbol{Z}_{1:k}$ respectively. Similar update rules can be defined for $\boldsymbol{C}$. Zeng et al. (2019) propose a similar projection-based learning rule (OWM) in feedforward networks, which only restricts changes in the row-space of the weight parameters (i.e., $\boldsymbol{P}_{wz} = \boldsymbol{I}$).

With a scaled additive approximation to the sum of Kronecker products, the NCL update rule on task $k+1$ is given by

$$\text{vec}(\Delta \boldsymbol{W}) \propto \left(\mathbb{E}\left[\boldsymbol{z}\boldsymbol{z}^\top\right] + \pi\alpha \boldsymbol{I}\right)^{-1} \otimes \left(\mathbb{E}\left[\overline{\boldsymbol{h}}\,\overline{\boldsymbol{h}}^\top\right] + \frac{1}{\pi}\alpha \boldsymbol{I}\right)^{-1}\overline{\boldsymbol{w}} + (\text{vec}(\boldsymbol{W}_k) - \text{vec}(\boldsymbol{W})). \tag{C.30}$$

We see that this NCL update rule looks similar to the OWM and DOWM update steps, and that they share the same projection matrix in the row-space $\boldsymbol{P}_z$ when $\pi = 1$. The methods proposed by Duncker et al. (2020) and Zeng et al. (2019) can thus be seen as approximations to NCL with a Kronecker structured Fisher matrix. However, we also note that OWM and DOWM do not include the regularization term $(\text{vec}(\boldsymbol{W}_k) - \text{vec}(\boldsymbol{W}))$. This implies that while OWM and DOWM encourage parameter updates along flat directions of the prior, the performance of these methods may deteriorate in the limit of infinite training duration if a local minimum of task k is not found in a flat subspace of previous tasks (c.f. Figure 4.1).

To further emphasize the relationship between OWM, DOWM and NCL, we compared the approximations to the Fisher matrix $\boldsymbol{F}_{\text{approx}} = \boldsymbol{P}_R^{-1} \otimes \boldsymbol{P}_L^{-1}$ implied by the projection matrices of these methods (Figure C.1). Here we found that OWM and DOWM provided reasonable approximations to the true Fisher matrix with both Gaussian (Figure C.1) and categorical (Figure C.2) observation models. This motivates a Bayesian interpretation of these methods as using an approximate prior precision matrix to project gradients, similar to the derivation of NCL in Appendix C. Here it is also worth noting that while we use an optimal sum of

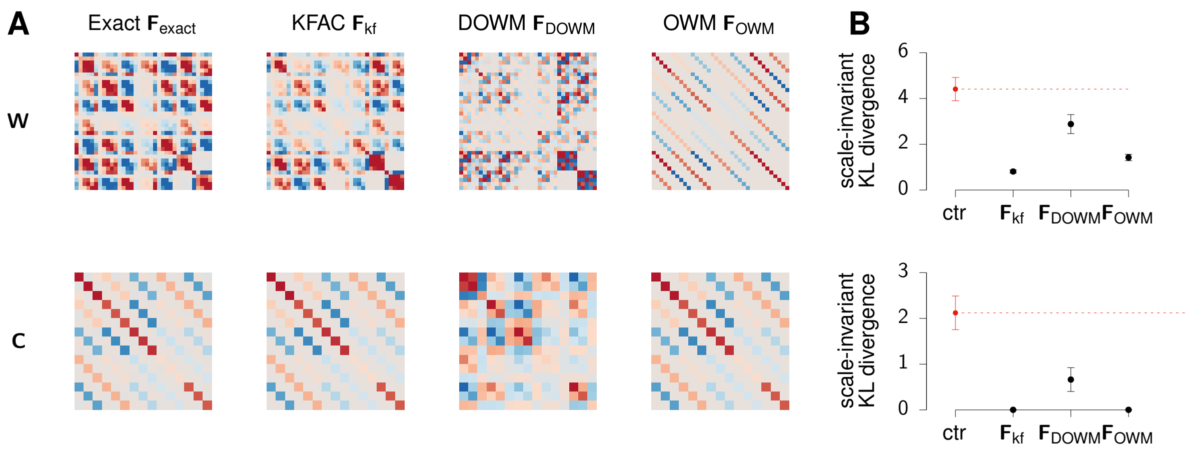

Figure C.1 **Comparison of projection matrices.** In a Bayesian framework, we can formalize what is meant by directions 'important for previous tasks' as those that are strongly constrained by the prior $p(\theta|\mathcal{D}_{1:k-1})$. To see how this compares with OWM and DOWM, we considered the Kronecker-structured precision matrices $\boldsymbol{F}_{\text{approx}}$ implied by the projection matrices $\boldsymbol{P}_R$ and $\boldsymbol{P}_L$ for each method and related them to the exact Fisher matrix $\boldsymbol{F}_{\text{exact}}$ in a linear recurrent network. **(A; top)** $\boldsymbol{F}_{\text{exact}}$ (left) for $\boldsymbol{W}$ as well as the approximations to $\boldsymbol{F}_{\text{exact}}$ provided by our Kronecker-factored approximation (KFAC; $\boldsymbol{F}_{\text{kf}}$), DOWM ($\boldsymbol{F}_{\text{DOWM}}$), and OWM ($\boldsymbol{F}_{\text{OWM}}$). **(B; top)** Scale-invariant KL-divergence (Equation C.51) between $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{F}_{\text{exact}}^{-1})$ and $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{F}_{\text{approx}}^{-1})$ for each approximation. Red horizontal line indicates the mean value obtained from $\boldsymbol{F}_{\text{approx}} = \boldsymbol{R}\boldsymbol{F}_{\text{exact}}\boldsymbol{R}^\top$ where $\boldsymbol{R}$ is a random rotation matrix (averaged over 500 random samples). **(Bottom)** Same as (A–B) but for the readout matrix $\boldsymbol{C}$.

Kronecker factors to update the prior precision after each task in NCL, OWM and DOWM simply sum their Kronecker factors. In the case of OWM, this is in fact an exact approximation to the sum of the Kronecker products since the right Kronecker factor is in this case a constant matrix $\boldsymbol{I}$. For DOWM, summing the individual Kronecker factors does not provide an optimal approximation to the sum of the Kronecker products, but our results suggest that it is a fairly reasonable approximation up to a scale factor which can be absorbed into the learning rate.

Another recent projection-based approach to continual learning developed by Saha et al. (2021) restricts parameter updates to occur in a subspace of the full parameter space deemed important for previous tasks. This method, known as 'Gradient projection memory' (GPM), is similar to OWM but with a hard cut-off separating 'important' from 'unimportant' directions of parameter space. The important subspace is in this case determined by thresholding the singular values of the activity matrix $\boldsymbol{Z}_k$. GPM can thus be seen as a discretized version of OWM with a projection matrix constituting a binary approximation to the prior Fisher matrix.

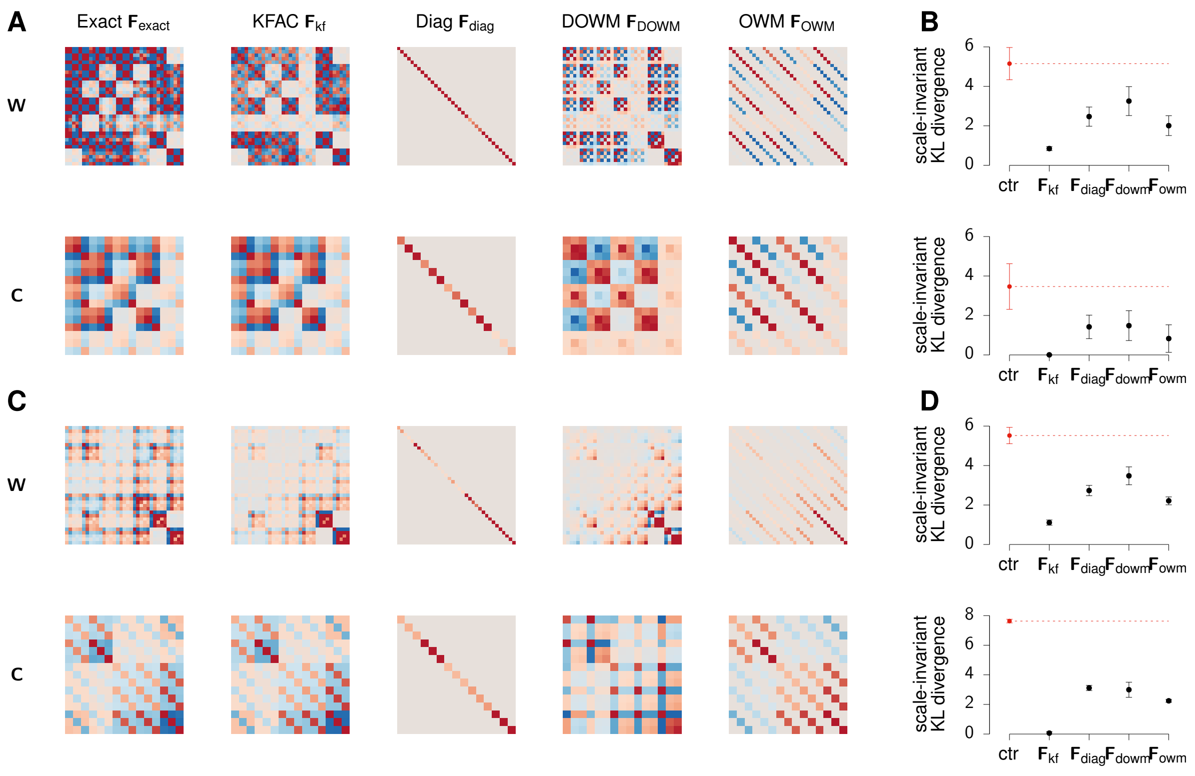

Figure C.2 **Comparison of Fisher Approximations in a Linear RNN with rotated Gaussian and categorical likelihoods. (A)** Exact and approximations to the Fisher information matrix of the recurrent and input weight matrix $\boldsymbol{W}$ (left) and the linear readout $\boldsymbol{C}$ (bottom) of a linear recurrent neural network with Gaussian noise and non-diagonal noise covariance $\boldsymbol{\Sigma}$. From the left: exact Fisher information matrix $\boldsymbol{F}_{\text{exact}}$, Kronecker-Factored approximation ($\boldsymbol{F}_{\text{kf}}$; KFAC), Diagonal ($\boldsymbol{F}_{\text{diag}}$), DOWM ($\boldsymbol{F}_{\text{DOWM}}$), and OWM ($\boldsymbol{F}_{\text{OWM}}$). **(B)** Scale-invariant KL-divergence between $\mathcal{N}(\mathbf{0}, \boldsymbol{F}_{\text{exact}}^{-1})$ and $\mathcal{N}(\mathbf{0}, \boldsymbol{F}^{-1})$ for $\boldsymbol{F} \in \{\boldsymbol{F}_{\text{kf}}, \boldsymbol{F}_{\text{diag}}, \boldsymbol{F}_{\text{DOWM}}, \boldsymbol{F}_{\text{OWM}}\}$. Red horizontal lines indicate the mean value obtained from $\boldsymbol{F}_{\text{approx}} = \boldsymbol{R}\boldsymbol{F}_{\text{exact}}\boldsymbol{R}^{\top}$ where $\boldsymbol{R}$ is a random rotation matrix (averaged over 500 random samples). **(C-D)** As in (A-B), now for a categorical noise model $p(\boldsymbol{y}|\boldsymbol{C}\boldsymbol{r}) = \text{Cat}\left(\text{softmax}(\boldsymbol{C}\boldsymbol{r})\right)$.

Kronecker-factored approximation to the sums of Kronecker Products

In this section, we consider three different Kronecker-factored approximations to the sum of two Kronecker products:

$$\boldsymbol{X} \otimes \boldsymbol{Y} \approx \boldsymbol{Z} = \boldsymbol{A} \otimes \boldsymbol{B} + \boldsymbol{C} \otimes \boldsymbol{D}. \tag{C.31}$$

In particular, we consider the special case where $\boldsymbol{A} \in \mathbb{R}^{n \times n}$, $\boldsymbol{B} \in \mathbb{R}^{m \times m}$, $\boldsymbol{C} \in \mathbb{R}^{n \times n}$, and $\boldsymbol{D} \in \mathbb{R}^{m \times m}$ are symmetric positive-definite. $\boldsymbol{Z}$ will not in general be a Kronecker product, but for computational reasons it is desirable to approximate it as one to avoid computing or storing a full-sized precision matrix.

Scaled additive approximation The first approximation we consider was proposed by Martens and Grosse (2015). They propose to approximate the sum with

$$\boldsymbol{Z} \approx (\boldsymbol{A} + \pi \boldsymbol{C}) \otimes (\boldsymbol{B} + \frac{1}{\pi}\boldsymbol{D}), \tag{C.32}$$

where π is a scalar parameter. Using the triangle inequality, Martens and Grosse (2015) derived an upper-bound to the norm of the approximation error

$$\|\boldsymbol{Z} - (\boldsymbol{A} + \pi \boldsymbol{C}) \otimes (\boldsymbol{B} + \frac{1}{\pi}\boldsymbol{D})\| \tag{C.33}$$

$$= \|\frac{1}{\pi}\boldsymbol{A} \otimes \boldsymbol{D} + \pi \boldsymbol{C} \otimes \boldsymbol{B}\| \tag{C.34}$$

$$\leq \frac{1}{\pi}\|\boldsymbol{A} \otimes \boldsymbol{D}\| + \pi \|\boldsymbol{C} \otimes \boldsymbol{B}\| \tag{C.35}$$

for any norm $\|\cdot\|$. They then minimize this upper-bound with respect to π to find the optimal π:

$$\pi = \sqrt{\frac{\|\boldsymbol{C} \otimes \boldsymbol{B}\|}{\|\boldsymbol{A} \otimes \boldsymbol{D}\|}}. \tag{C.36}$$

As in (Martens and Grosse, 2015), we use a trace norm in bounding the approximation error, and noting that $\mathrm{Tr}(\boldsymbol{X} \otimes \boldsymbol{Y}) = \mathrm{Tr}(\boldsymbol{X})\mathrm{Tr}(\boldsymbol{Y})$, we can compute the optimal π as:

$$\pi = \sqrt{\frac{\mathrm{Tr}(\boldsymbol{B})\mathrm{Tr}(\boldsymbol{C})}{\mathrm{Tr}(\boldsymbol{A})\mathrm{Tr}(\boldsymbol{D})}}. \tag{C.37}$$

Minimal mean-squared error The second approximation we consider was originally proposed by van Loan and Pitsianis (1993). In this case, we approximate the sum of Kronecker products by minimizing a mean squared loss:

$$\boldsymbol{X}, \boldsymbol{Y} = \arg\min_{\boldsymbol{X}, \boldsymbol{Y}} \|\boldsymbol{Z} - \boldsymbol{X} \otimes \boldsymbol{Y}\|_F^2 \tag{C.38}$$

$$= \arg\min_{\boldsymbol{X}, \boldsymbol{Y}} \|\mathcal{R}(\boldsymbol{A} \otimes \boldsymbol{B}) + \mathcal{R}(\boldsymbol{C} \otimes \boldsymbol{D}) - \mathcal{R}(\boldsymbol{X} \otimes \boldsymbol{Y})\|_F^2 \tag{C.39}$$

$$= \arg\min_{\boldsymbol{X}, \boldsymbol{Y}} \|\mathrm{vec}(\boldsymbol{A})\mathrm{vec}(\boldsymbol{B})^\top + \mathrm{vec}(\boldsymbol{C})\mathrm{vec}(\boldsymbol{D})^\top - \mathrm{vec}(\boldsymbol{X})\mathrm{vec}(\boldsymbol{Y})^\top\|_F^2, \tag{C.40}$$

where $\mathcal{R}(\boldsymbol{A} \otimes \boldsymbol{B}) = \mathrm{vec}(\boldsymbol{A})\mathrm{vec}(\boldsymbol{B})^\top$ is the rearrangement operator (van Loan and Pitsianis, 1993). The optimization problem thus involves finding the best rank-one approximation to a rank-2 matrix. This can be solved efficiently using a singular value decomposition (SVD) without ever constructing an $n^2 \times m^2$ matrix (see Algorithm 3 for details).

Algorithm 3: Mean-squared error approximation of the sum of Kronecker products

```
1 input: $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$, $\boldsymbol{D}$
2 $\boldsymbol{a} \leftarrow \mathrm{vec}(\boldsymbol{A})$, $\boldsymbol{b} \leftarrow \mathrm{vec}(\boldsymbol{B})$, $\boldsymbol{c} \leftarrow \mathrm{vec}(\boldsymbol{C})$, $\boldsymbol{d} \leftarrow \mathrm{vec}(\boldsymbol{D})$        // Vectorize $\boldsymbol{A},\boldsymbol{B},\boldsymbol{C},\boldsymbol{D}$
3 $\boldsymbol{Q}, \_ \leftarrow \mathrm{QR}(\left[\boldsymbol{a};\boldsymbol{c}\right])$        // Orthogonal basis for $\boldsymbol{a}$ and $\boldsymbol{c}$ in $\mathbb{R}^{n^2 \times 2}$
4 $\boldsymbol{H} \leftarrow (\boldsymbol{Q}^\top \boldsymbol{a})\boldsymbol{b}^\top + (\boldsymbol{Q}^\top \boldsymbol{c})\boldsymbol{d}^\top$
5 $\boldsymbol{U}, \boldsymbol{s}, \boldsymbol{V}^\top \leftarrow \mathrm{SVD}(\boldsymbol{H})$
6 $\boldsymbol{y} \leftarrow$ first column of $\sqrt{\boldsymbol{s}_1}\boldsymbol{V}$
7 $\boldsymbol{x} \leftarrow$ first column of $\sqrt{\boldsymbol{s}_1}\boldsymbol{Q}\boldsymbol{U}$
8 $\boldsymbol{X} \leftarrow \mathrm{reshape}(\boldsymbol{x}, (n,n))$, $\boldsymbol{Y} \leftarrow \mathrm{reshape}(\boldsymbol{y}, (m,m))$
```

Minimal KL-divergence In this work, we propose an alternative approximation to $\boldsymbol{Z}$ motivated by the fact that $\boldsymbol{X} \otimes \boldsymbol{Y}$ is meant to approximate the precision matrix of the approximate posterior after learning task k. We thus define two multivariate Gaussian distributions $q(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}; \boldsymbol{\mu}, \boldsymbol{X} \otimes \boldsymbol{Y})$ and $p(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w}; \boldsymbol{\mu}, \boldsymbol{Z})$ (note that the mean of these distributions are found in NCL by gradient-based optimization). We are interested in finding the matrices $\boldsymbol{X}$ and $\boldsymbol{Y}$ that minimize the KL-divergence between the two distributions

$$\begin{aligned}
2D_{\mathrm{KL}}(q||p) &= \log|\boldsymbol{X} \otimes \boldsymbol{Y}| - \log|\boldsymbol{Z}| + \mathrm{Tr}(\boldsymbol{Z}(\boldsymbol{X} \otimes \boldsymbol{Y})^{-1}) - d && \text{(C.41)}\\
&= m\log|\boldsymbol{X}| + n\log|\boldsymbol{Y}| + \mathrm{Tr}(\boldsymbol{A}\boldsymbol{X}^{-1} \otimes \boldsymbol{B}\boldsymbol{Y}^{-1}) + \mathrm{Tr}(\boldsymbol{C}\boldsymbol{X}^{-1} \otimes \boldsymbol{D}\boldsymbol{Y}^{-1}) - d && \text{(C.42)}\\
&= -m\log|\boldsymbol{X}^{-1}| - n\log|\boldsymbol{Y}^{-1}| + \mathrm{Tr}(\boldsymbol{A}\boldsymbol{X}^{-1})\mathrm{Tr}(\boldsymbol{B}\boldsymbol{Y}^{-1}) && \text{(C.43)}\\
&\qquad + \mathrm{Tr}(\boldsymbol{C}\boldsymbol{X}^{-1})\mathrm{Tr}(\boldsymbol{D}\boldsymbol{Y}^{-1}) - d && \text{(C.44)}
\end{aligned}$$

where $d = nm$. Differentiating with respect to $\boldsymbol{X}^{-1}$, and $\boldsymbol{Y}^{-1}$ and setting the result to zero, we get

$$\begin{aligned}
0 &= \frac{\partial D_{\mathrm{KL}}(q||p)}{\partial \boldsymbol{X}^{-1}} = \frac{1}{2}\left[-m\boldsymbol{X} + \mathrm{Tr}(\boldsymbol{B}\boldsymbol{Y}^{-1})\boldsymbol{A} + \mathrm{Tr}(\boldsymbol{D}\boldsymbol{Y}^{-1})\boldsymbol{C}\right] && \text{(C.45)}\\
0 &= \frac{\partial D_{\mathrm{KL}}(q||p)}{\partial \boldsymbol{Y}^{-1}} = \frac{1}{2}\left[-n\boldsymbol{Y} + \mathrm{Tr}(\boldsymbol{A}\boldsymbol{X}^{-1})\boldsymbol{B} + \mathrm{Tr}(\boldsymbol{C}\boldsymbol{X}^{-1})\boldsymbol{D}\right]. && \text{(C.46)}
\end{aligned}$$

Rearranging these equations, we find the self-consistency equations:

$$\begin{aligned}
\boldsymbol{X} &= \frac{1}{m}\left[\mathrm{Tr}(\boldsymbol{B}\boldsymbol{Y}^{-1})\boldsymbol{A} + \mathrm{Tr}(\boldsymbol{D}\boldsymbol{Y}^{-1})\boldsymbol{C}\right] && \text{(C.47)}\\
\boldsymbol{Y} &= \frac{1}{n}\left[\mathrm{Tr}(\boldsymbol{A}\boldsymbol{X}^{-1})\boldsymbol{B} + \mathrm{Tr}(\boldsymbol{D}\boldsymbol{X}^{-1})\boldsymbol{D}\right]. && \text{(C.48)}
\end{aligned}$$

This shows that the optimal $\boldsymbol{X}$ ($\boldsymbol{Y}$) is a linear combination of $\boldsymbol{A}$ and $\boldsymbol{C}$ ($\boldsymbol{B}$ and $\boldsymbol{D}$). It is unclear whether we can solve for $\boldsymbol{X}$ and $\boldsymbol{Y}$ analytically in Equation C.47 and Equation C.48. However,

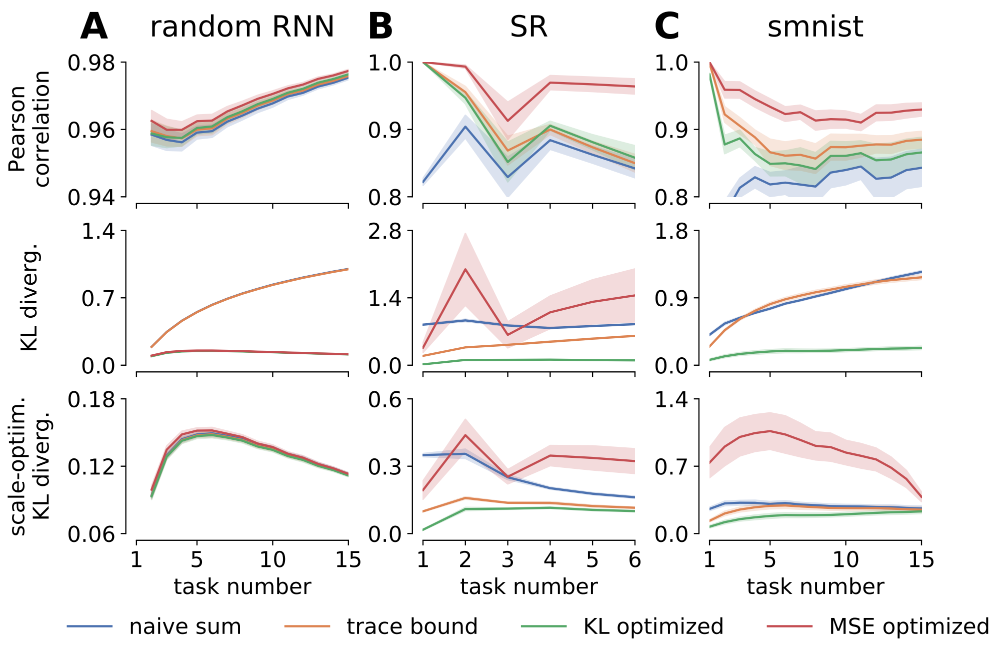

Figure C.3 **Comparison of different Kronecker approximations to consecutive sums of two Kronecker products. (A)** Comparison of approximations for Fisher matrices computed from random RNNs with dynamics as described in Section 4.1.3. All similarity/distance measures are computed between the true sum $\sum_{k'}^{k} \boldsymbol{F}_{k'}$ and each iterative approximation. **(B)** As in (A) for the Fisher matrices from the stimulus-response tasks, here trained with 50 hidden units to make the computation of the true sum tractable. **(C)** As in (A) for the Fisher matrices from the SMNIST tasks. Note that the KL divergence for the MSE-minimizing approximation is not shown in panel 2 as it is an order of magnitude larger than the alternatives and thus does not fit on the axis.

we can find $\boldsymbol{X}$ and $\boldsymbol{Y}$ numerically by iteratively applying the following update rules:

$$\boldsymbol{X}_{k+1} = (1-\beta)\boldsymbol{X}_k + \frac{\beta}{m}\left(\mathrm{Tr}(\boldsymbol{B}\boldsymbol{Y}_k^{-1})\boldsymbol{A} + \mathrm{Tr}(\boldsymbol{D}\boldsymbol{Y}_k^{-1})\boldsymbol{C}\right) \tag{C.49}$$

$$\boldsymbol{Y}_{k+1} = (1-\beta)\boldsymbol{Y}_k + \frac{\beta}{n}\left(\mathrm{Tr}(\boldsymbol{A}\boldsymbol{X}_k^{-1})\boldsymbol{C} + \mathrm{Tr}(\boldsymbol{C}\boldsymbol{X}_k^{-1})\boldsymbol{D}\right) \tag{C.50}$$

for initial guesses $\boldsymbol{X}_0$ and $\boldsymbol{Y}_0$. In practice, we initialize using the scaled additive approximation and find that the algorithm converges with $\beta = 0.3$ after tens of iterations.

Comparisons To compare different approximations of the precision matrix to the posterior, we consider Kronecker structured Fisher matrices from (i) a random RNN model, (ii) the Fishers learned in the stimulus-response tasks, and (iii) the Fishers learned in the SMNIST tasks. We then iteratively update $\boldsymbol{\Lambda}_k \approx \boldsymbol{\Lambda}_{k-1} + \boldsymbol{F}_k$, approximating this sum using each of the approaches described above as well as a naive unweighted sum of the pairs of Kronecker factors. We compare these approximations using three different metrics: the correlation with the true sum of Kronecker products $\sum_{k'}^{k} \boldsymbol{F}_{k'}$ (Figure C.3, top row), the KL divergence from the true sum (Figure C.3, middle row), and the scale-optimized KL divergence from the true sum (Figure C.3, bottom row). Here we define the scale-optimized KL divergence as

$$\mathrm{KL}_\lambda[\boldsymbol{\Lambda}_1||\boldsymbol{\Lambda}_2] = \min_\lambda \mathrm{KL}[\lambda\boldsymbol{\Lambda}_1||\boldsymbol{\Lambda}_2] \tag{C.51}$$

$$= \frac{1}{2}\left(\log\frac{|\boldsymbol{\Lambda}_1|}{|\boldsymbol{\Lambda}_2|} + d\log\frac{\mathrm{Tr}(\boldsymbol{\Lambda}_1^{-1}\boldsymbol{\Lambda}_2)}{d}\right), \tag{C.52}$$

where d is the dimensionality of the precision matrices $\boldsymbol{\Lambda}_1$ and $\boldsymbol{\Lambda}_2$ and we take $\mathrm{KL}[\boldsymbol{\Lambda}_1, \boldsymbol{\Lambda}_2] = D_{\mathrm{KL}}(\mathcal{N}(\mathbf{0}, \boldsymbol{\Lambda}_1^{-1})||\mathcal{N}(\mathbf{0}, \boldsymbol{\Lambda}_2^{-1}))$. This is a useful measure since a scaling of the approximate prior does not change the subspaces that are projected out in the weight projection methods but merely scales the learning rate. By contrast in NCL, having an appropriate scaling is useful for a consistent Bayesian interpretation.

We find that all the methods yield reasonable correlations and scale-optimized KL divergences between the true sum of Kronecker products and the approximate sum, although the L2-optimized approximation tends to have a slightly better correlation and slightly worse scaled KL (Figure C.3, red). However, the KL-optimized Kronecker sum greatly outperforms the other methods as quantified by the regular KL divergence and is the method used in this work since it is relatively cheap to compute and only needs to be computed once per task (Figure C.3, green).

Further results

Performance with different prior scalings

Here we consider the performance of KFAC and NCL for different values of λ on the stimulus-response task set with 256 recurrent units. We start by recalling that λ is a parameter that is used to define a modified Laplace loss function with a rescaling of the prior term (c.f. Section 4.1.2):

$$\mathcal{L}_k^{(\lambda)}(\theta) = \log p(\mathcal{D}_k|\theta) - \lambda(\theta - \boldsymbol{\mu}_{k-1})^\top \boldsymbol{\Lambda}_{k-1}(\theta - \boldsymbol{\mu}_{k-1}). \tag{C.53}$$

In this context, it is worth noting that KFAC and NCL have the same stationary points when they share the same value of λ. Despite this, the performance of NCL was robust across different values of λ (Figure C.4A), while learning was unstable and performance generally poor for KFAC with small values of $\lambda \in [1, 10]$. However, as we increased λ for KFAC, learning stabilized and catastrophic forgetting was mitigated (Figure C.4B). A similar pattern was observed for the SMNIST task set (Figure C.6).

We hypothesize that the improved performance of KFAC for high values of λ is due in part to the gradient preconditioner of KFAC becoming increasingly similar to NCL's precondi-

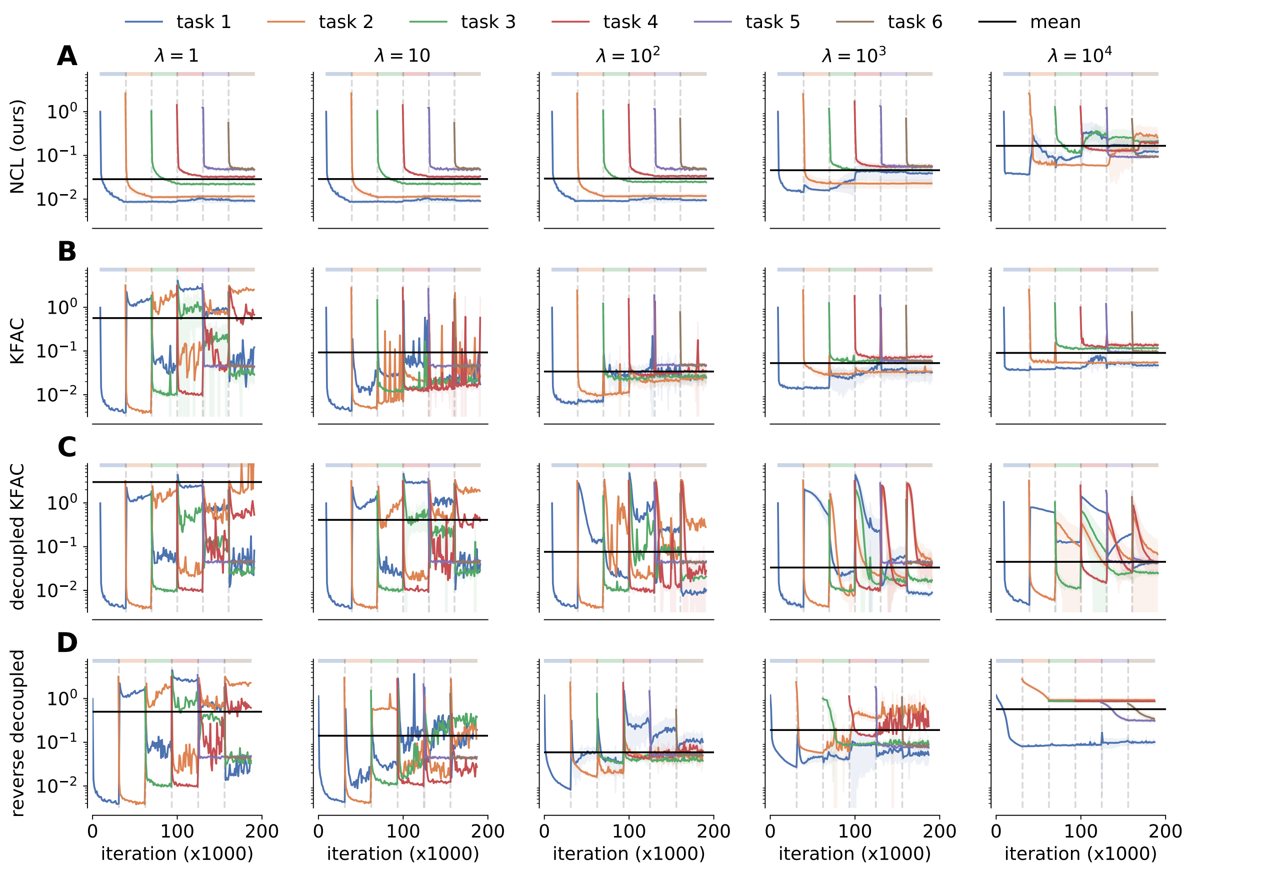

Figure C.4 **Continual learning on SR tasks with different λ. (A)** Evolution of the loss during training for each of the six stimulus-response tasks for NCL with different values of λ. The performance of NCL is generally robust across different choices of λ until it starts overfitting too heavily on early tasks. **(B)** As in (A), now for KFAC with Adam which performs poorly for small λ. **(C)** As in (B), now with "decoupled Adam" where we fix $\lambda_m = 1$ for the gradient estimate and vary $\lambda = \lambda_v$ for the preconditioner. Interestingly, this is sufficient to overcome the catastrophic forgetting observed for KFAC with $\lambda_m = \lambda_v = 1$. The transient forgetting observed at the beginning of a new task is likely due to the time it takes to gradually update the preconditioner for the new task as more data is observed. **(D)** As in (C), now fixing $\lambda_v = 1$ for the preconditioner and varying $\lambda = \lambda_m$ for the gradient estimate. For higher values of λ_m, this performs worse than both KFAC and decoupled KFAC.

tioner $\mathbf{\Lambda}_{k-1}^{-1}$ as λ increases (Section 4.1.2). To test this hypothesis, we modified the Adam optimizer (Kingma and Ba, 2014) to use different values of λ when computing the Adam momentum and preconditioner. Specifically, we computed the momentum and preconditioner of some scalar parameter θ as:

$$m^{(i)} \leftarrow \beta_1 m^{(i-1)} + (1-\beta_1)\nabla_\theta \mathcal{L}^{(\lambda_m)} \tag{C.54}$$

$$v^{(i)} \leftarrow \beta_2 v^{(i-1)} + (1-\beta_2)\left(\nabla_\theta \mathcal{L}^{(\lambda_v)}\right)^2 \tag{C.55}$$

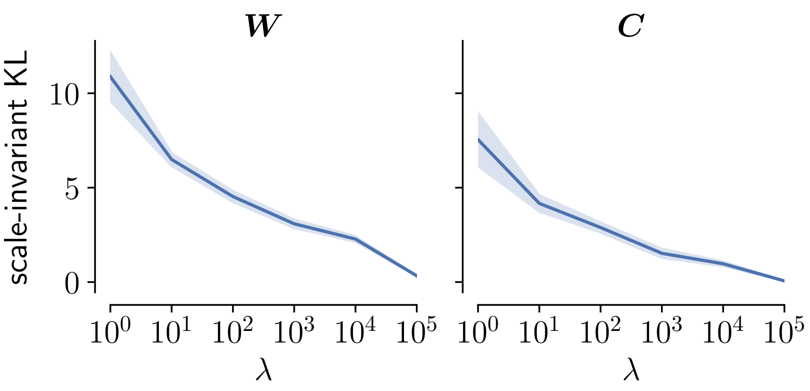

Figure C.5 **Similarity of the Adam preconditioner and diagonal Fisher matrix.** Scale-invariant KL divergence (Equation C.51) between the diagonal of $\mathbf{\Lambda}_{k-1}$ and the preconditioner used by Adam ($\sqrt{\boldsymbol{v}}$; Kingma and Ba, 2014) at the end of training on task k. Results are averaged over the five first stimulus-response tasks, and the figure indicates mean and standard error across 5 seeds for the state matrix $\boldsymbol{W}$ (left) and the output matrix $\boldsymbol{C}$ (right).

where $\mathcal{L}^{(\lambda)}$ is defined in Equation C.53 and importantly λ_m may not be equal to λ_v. As in vanilla Adam, we used m and v to update the parameter θ according to the following update equations at the i^{th} iteration:

$$\hat{m}^{(i)} \leftarrow m^{(i)}/(1-\beta_1^i) \tag{C.56}$$
$$\hat{v}^{(i)} \leftarrow v^{(i)}/(1-\beta_2^i) \tag{C.57}$$
$$\theta^{(i)} \leftarrow \theta^{(i-1)} + \gamma \hat{m}^{(i)}/(\sqrt{\hat{v}^{(i)}} + \epsilon), \tag{C.58}$$

where γ is a learning rate, and β_1, β_2, and ϵ are standard parameters of the Adam optimizer (see Kingma and Ba, 2014 for further details). Using this modified version of Adam, which we call "decoupled Adam", we considered two variants of KFAC: (i) "decoupled KFAC", where we fix $\lambda_m = 1$ and vary λ_v (Figure C.4C), and (ii) "reverse decoupled", where we fix $\lambda_v = 1$ and vary λ_m (Figure C.4D). We found that "decoupled KFAC" performed well for large λ_v, suggesting that it is sufficient to overcount the prior in the Adam preconditioner without changing the gradient estimate (Figure C.4C). "Reverse decoupled" also partly overcame the catastrophic forgetting for high λ_m, but performance was worse than for either NCL, vanilla Adam, or decoupled Adam (Figure C.4D). These results support our hypothesis that the increased performance of KFAC for high λ is due in part to the changes in the gradient preconditioner. To further highlight how the preconditioning in Adam relates to the trust region optimization employed by NCL, we computed the scaled KL divergence between the Adam preconditioner and the diagonal of the Kronecker-factored prior precision matrix $\mathbf{\Lambda}_{k-1}$ at the end of training on task k. We found that the Adam preconditioner increasingly resembled $\mathbf{\Lambda}_{k-1}$, the preconditioner used by NCL, as λ increased (Figure C.5).

In summary, our results suggest that preconditioning with $\mathbf{\Lambda}_{k-1}$ in NCL may mitigate the need to overcount the prior when using weight regularization for continual learning. Additionally, such preconditioning to encourage parameter updates that retain good performance on previous tasks also appears to be a major contributing factor to the success of weight regularization with a high value of λ when using Adam for optimization.

Hyperparameter optimizations

Feedforward networks For the experiments with feedforward networks, we performed hyperparameter optimizations by searching over the following parameter ranges (all on a log-scale): c in SI from 10^{-5} to 10^{8}, λ in EWC and KFAC from 10^{-4} to 10^{14}, α in OWM from 10^{-12} to 10^{6}, and p_w^{-2} in NCL from 10^{2} to 10^{11}. The hyperparameter grid searches were performed using a random seed not included during the evaluation. The selected hyperparameter values for each experiment are reported in Table C.1.

	Split MNIST			**Split CIFAR-100**		
	Task	**Domain**	**Class**	**Task**	**Domain**	**Class**
SI (c)	10^0	10^5	10^7	10^2	10^3	10^6
EWC (λ)	10^8	10^9	10^{13}	10^7	10^5	10^{-3}
KFAC (λ)	10^{10}	10^5	10^4	10^5	10^3	10^{10}
OWM (α)	10^{-2}	10^{-5}	10^{-4}	10^{-2}	10^{-2}	10^{-4}
NCL (p_w^{-2})	10^3	10^7	10^9	10^3	10^7	10^8

Table C.1 Selected hyperparameter values for all compared methods on the experiments with feedforward networks.

RNNs For the experiments with RNNs, we optimized over the parameter α used to invert the approximate Fisher matrices in the projection-based methods (NCL, OWM and DOWM) or over the parameter λ used to scale the importance of the prior for weight regularization (KFAC).

For KFAC, we found that the performance was very sensitive to the value of λ across all tasks sets, and in particular that $\lambda = 1$ performed poorly. In the projection-based methods, α can be seen as evening out the learnings rates between directions that are otherwise constrained by the projection matrices, and indeed standard gradient descent is recovered as $\alpha \to \infty$ (on the Laplace objective for NCL and on ℓ_k for OWM/DOWM). We found that NCL in general outperformed the other projection-based methods with less sensitivity to the regularization parameter α. DOWM was particularly sensitive to α and required a relatively high value of this parameter to balance its otherwise conservative projection matrices. Here it is also worth noting that there is an extensive literature on how a parameter equivalent to α can be dynamically adjusted when doing standard natural gradient descent using the Fisher matrix for the current loss (see Martens, 2014 for an overview). While this has not been explored in the context of projection-based continual learning, it could be interesting to combine these projection based

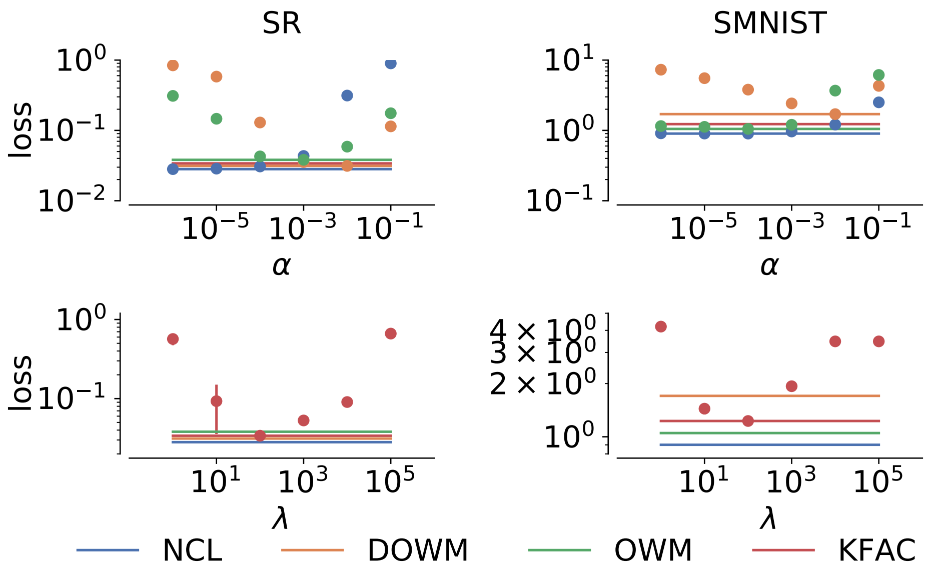

Figure C.6 **Hyperparameter optimization for RNNs. (left)** Comparison of the average loss across tasks on the stimulus-response task set as a function of α for the projection-based methods (top panel) and as a function of λ for KFAC (bottom panel). Circles and error bars indicate mean and s.e.m. across 5 random seeds. Horizontal lines indicate the optimal value for each method. **(right)** As before, now for the SMNIST task set.

methods with Tikhonov dampening (Tikhonov, 1943) in future work to automatically adjust α.

We generally report results in the main text and appendix using the optimal hyperparameter settings for each method unless otherwise noted. However, $\alpha = 10^{-5}$ was used for both NCL and Laplace-DOWM in Figure 4.3C to compare the qualitative behavior of the two different Fisher approximations without the confound of a large learning rate in directions otherwise deemed "important" by the approximation.

Numerical results of experiments with feedforward networks

To facilitate comparison to our results, here we provide a table with the numerical results (Table C.2) of the experiments with feedforward networks reported in Figure 4.2 of the main text.

Table C.2 Numerical results for the experiments with feedforward networks, corresponding to Figure 4.2 in the main text. Reported is the test accuracy (as %, averaged over all tasks) after training on all tasks. Each experiment was performed either 20 (split MNIST) or 10 (split CIFAR-100) times with different random seeds, and we report the mean (± standard error) across seeds.

	Split MNIST			**Split CIFAR-100**		
Method	**Task**	**Domain**	**Class**	**Task**	**Domain**	**Class**
None	*81.58 ±1.64*	*59.47 ±1.71*	*19.88 ±0.02*	*61.43 ±0.36*	*18.42 ±0.33*	*7.71 ±0.18*
Joint	*99.69 ±0.02*	*98.69 ±0.04*	*98.32 ±0.05*	*78.78 ±0.25*	*46.85 ±0.51*	*49.78 ±0.21*
SI	97.24 ±0.55	65.20 ±1.48	21.40 ±1.30	74.84 ±0.39	22.58 ±0.37	7.02 ±1.04
EWC	98.67 ±0.22	63.44 ±1.70	20.08 ±0.16	75.38 ±0.24	19.97 ±0.44	7.74 ±0.18
KFAC	99.04 ±0.10	67.86 ±1.33	19.99 ±0.04	76.61 ±0.23	26.57 ±0.66	7.59 ±0.17
OWM	99.36 ±0.05	87.46 ±0.74	80.73 ±1.11	77.07 ±0.27	28.51 ±0.30	29.23 ±0.51
NCL (no opt)	99.53 ±0.03	84.9 ±1.06	47.49 ±0.84	77.88 ±0.26	32.81 ±0.38	16.63 ±0.34
NCL	99.55 ±0.03	91.48 ±0.64	69.31 ±1.65	78.38 ±0.27	38.79 ±0.24	26.36 ±1.09

SMNIST dynamics with DOWM

In this section, we investigate the latent dynamics of a network trained by DOWM with $\alpha = 0.001$ (c.f. the analysis in Section 4.1.3 for NCL). Here we found that the task-associated recurrent dynamics for a given task were more stable after learning the corresponding task than in networks trained with NCL. Indeed, the DOWM networks exhibited near-zero drift for early tasks even after learning all 15 tasks (Figure C.7). However, DOWM also learned representations that were less well-separated after the first 1-2 classification tasks (Figure C.7, bottom) than those learned by NCL. This is consistent with our results in Section 4.1.3, where DOWM exhibited high performance on the first task even after learning all 15 tasks, but performed less well on later tasks (Figure C.7). These results may be explained by the observation that DOWM tends to overestimate the number of dimensions that are important for learned tasks (Section 4.1.3) and thus projects out too many dimensions in the parameter updates when learning new tasks.

In the context of biological networks, it is unlikely that the brain remembers previous tasks in a way that causes it to lose the capacity to learn new tasks. However, it is also not clear how the balance between capacity and task complexity plays out in the mammalian brain, which on the one hand has many orders of magnitude more neurons than the networks analyzed here, but on

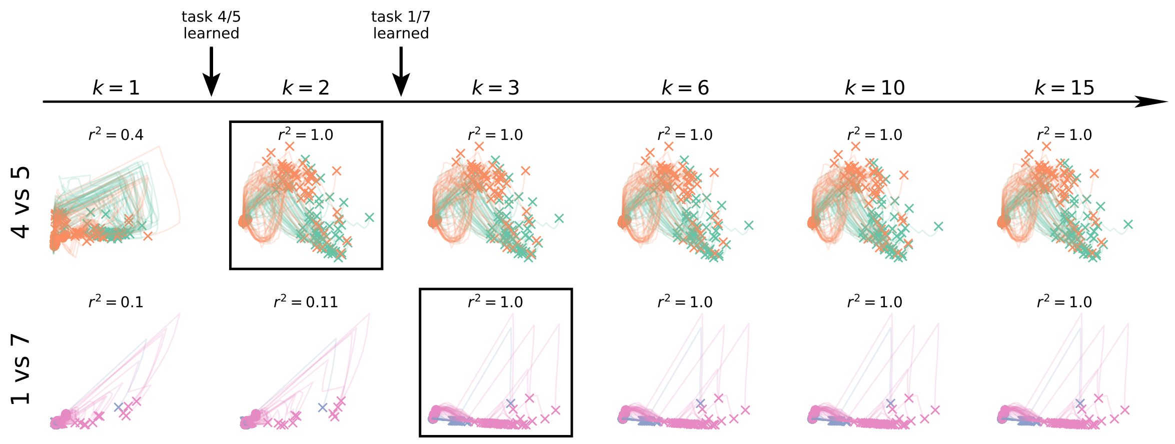

Figure C.7 **Latent dynamics during SMNIST.** We considered two example tasks, 4 vs 5 (top) and 1 vs 7 (bottom). For each task, we simulated the response of a network trained by DOWM to 100 digits drawn from that task distribution at different times during learning. We then fitted a factor analysis model for each example task to the response of the network right after the correponding task had been learned (squares; $k = 2$ and $k = 3$ respectively). We used this model to project the responses at different times during learning into a common latent space for each example task. For both example tasks, the network initially exhibited variable dynamics with no clear separation of inputs and subsequently acquired stable dynamics after learning to solve the task. The r^2 values above each plot indicate the similarity of neural population activity with that collected immediately after learning the corresponding task, quantified across all neurons (not just the 2D projection).

the other hand also learns more behaviors that are more complex than the problems studied in this work. In networks where capacity is not a concern, it may in fact be desirable to employ a strategy similar to that of DOWM — projecting out more dimensions in the parameter updates than is strictly necessary — so as to avoid forgetting in the face of the inevitable noise and turnover of e.g. synapses and cells in biological systems.

Details of toy example in schematic

In Figure 4.1A, we consider two regression tasks with losses defined as:

$$\ell_1(\theta) = \frac{1}{2}(\theta - \theta_1)^T \boldsymbol{Q}_1(\theta - \theta_1) \tag{C.59}$$

$$\ell_2(\theta) = \frac{1}{2}(\theta - \theta_2)^T \boldsymbol{Q}_2(\theta - \theta_2), \tag{C.60}$$

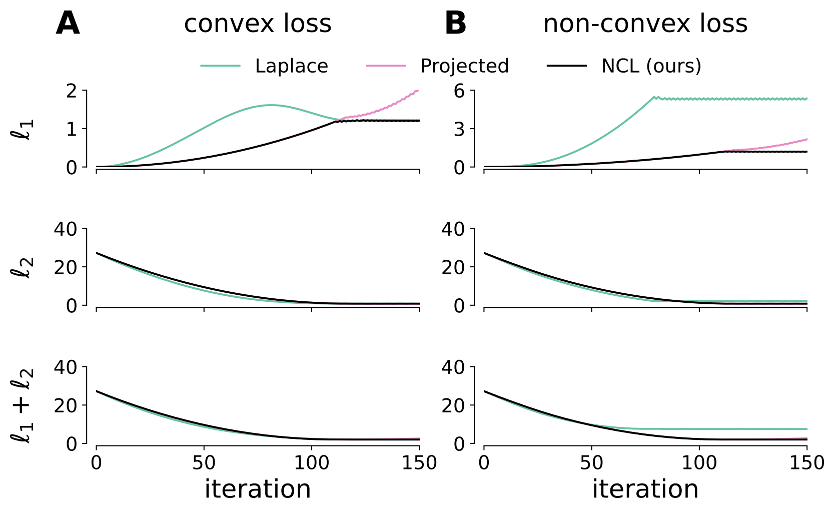

Figure C.8 **Losses on toy optimization problems. (A)** Loss as a function of optimization step on task 1 (top), task 2 (middle) and the combined loss (bottom) on the convex toy continual learning problem for different optimization methods. **(B)** As in (A), now for the non-convex problem.

where $\theta_1 = (3, -6)^\top$, $\theta_2 = (3, 6)^\top$,

$$\boldsymbol{Q}_1 = \boldsymbol{R}(\phi_1) \begin{bmatrix} 1 & 0 \\ 0 & \zeta \end{bmatrix} \boldsymbol{R}(\phi_1)^T, \tag{C.61}$$

$$\boldsymbol{Q}_2 = \boldsymbol{R}(\phi_2) \begin{bmatrix} 2 & 0 \\ 0 & \zeta \end{bmatrix} \boldsymbol{R}(\phi_2)^T, \tag{C.62}$$

$$\boldsymbol{R}(\phi) = \begin{bmatrix} \cos(\phi) & -\sin(\phi) \\ \sin(\phi) & \cos(\phi) \end{bmatrix}, \tag{C.63}$$

and $\zeta = 5.5$. We 'train' on task 1 first by setting $\theta = \theta_1$. We then construct a Laplace approximation to the posterior after learning task 1 to find the posterior precision $\boldsymbol{Q}_1$ (which is in this case exact since the loss is quadratic in θ). Now we proceed to train on task 2 by maximizing the posterior (see Equation 2.31):

$$\mathcal{L}_2(\theta) = \ell_2(\theta) + \frac{1}{2}(\theta - \theta_1)^T \boldsymbol{Q}_1 (\theta - \theta_1) \tag{C.64}$$

$$= \ell_2(\theta) + \ell_1(\theta) \tag{C.65}$$

The gradient of $\mathcal{L}_2(\theta)$ with respect to θ is given by:

$$\nabla_\theta \mathcal{L} = \boldsymbol{Q}_1(\theta - \theta_1) + \boldsymbol{Q}_2(\theta - \theta_2). \tag{C.66}$$

We can optimize $\ell(\theta)$ using the following three methods:

$$\text{Laplace:} \quad \Delta\theta \propto \boldsymbol{Q}_1(\theta-\theta_1)+\boldsymbol{Q}_2(\theta-\theta_2) \tag{C.67}$$

$$\text{NCL:} \quad \Delta\theta \propto (\theta-\theta_1)+\boldsymbol{Q}_1^{-1}\boldsymbol{Q}_2(\theta-\theta_2) \tag{C.68}$$

$$\text{Projected:} \quad \Delta\theta \propto \boldsymbol{Q}_1^{-1}\boldsymbol{Q}_2(\theta-\theta_2), \tag{C.69}$$

where γ is the learning rate. Note that in 'projected', we optimize on task 2 only rather than on the Laplace posterior.

In Figure 4.1B, we consider a slight modification to ℓ_2 such that the loss is no longer convex:

$$\ell_2(\boldsymbol{w}) = \frac{1}{2}(\theta-\theta_2)^T\boldsymbol{Q}_2(\theta-\theta_2)+a-a\exp\left(-\frac{1}{2}(\theta-\boldsymbol{v})^T\boldsymbol{Q}_v(\theta-\boldsymbol{v})\right), \tag{C.70}$$

where we have added a Gaussian with covariance $\boldsymbol{Q}_v$ to the second loss. The NCL preconditioner from task 1 remains unchanged ($\boldsymbol{Q}_1^{-1}$) since ℓ_1 is unchanged. Denoting $G := a\exp\left(-\frac{1}{2}(\theta-\boldsymbol{v})^T\boldsymbol{Q}_v(\theta-\boldsymbol{v})\right)$, we thus have the following updates when learning task 2:

$$\text{Laplace:} \quad \boldsymbol{\Delta}\theta \propto \boldsymbol{Q}_1(\theta-\theta_1)+\boldsymbol{Q}_2(\theta-\theta_2)+\boldsymbol{Q}_v(\theta-\boldsymbol{v})G \tag{C.71}$$

$$\text{NCL:} \quad \boldsymbol{\Delta}\theta \propto (\theta-\theta_1)+\boldsymbol{Q}_1^{-1}\boldsymbol{Q}_2(\theta-\theta_2)+\boldsymbol{Q}_1^{-1}\boldsymbol{Q}_v(\theta-\boldsymbol{v})G \tag{C.72}$$

$$\text{Projected:} \quad \boldsymbol{\Delta}\theta \propto \boldsymbol{Q}_1^{-1}\boldsymbol{Q}_2(\theta-\theta_2)+\boldsymbol{Q}_1^{-1}\boldsymbol{Q}_v(\theta-\boldsymbol{v})G. \tag{C.73}$$

In this non-convex case, the different methods can converge to different local minima (c.f. Figure 4.1B).

The losses on both tasks as well as the combined loss as a function of optimization step are illustrated in Figure C.8 for the convex and non-convex settings.

Appendix D

Reinforcement learning

D.1 Supplementary figures

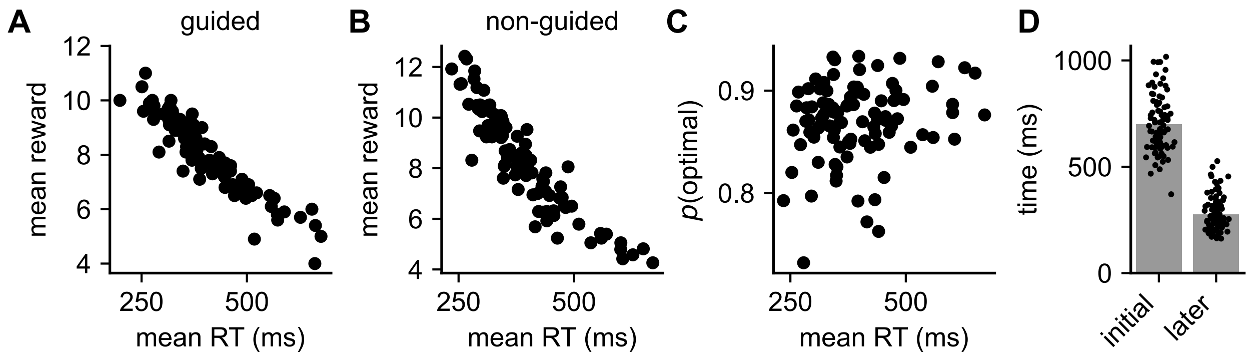

Figure D.1 **Overview of human data for all participants. (A)** Mean reward per episode as a function of the average response time during the guided trials (Appendix D.2). Each data point corresponds to a single participant. **(B)** Mean reward per episode as a function of the average response time during the non-guided trials. The strong negative correlation implies that participants on average got more reward when they acted faster, confirming that participants who acted faster were not simply making random key presses. **(C)** Fraction of actions that were consistent with an optimal policy as a function of mean response time, plotted for all participants during the non-guided trials. There was a significant positive Pearson correlation between these two quantities ($r = 0.20$; $p < 0.024$, permutation test). This correlation confirms that participants who thought for longer were not simply disengaged with the task, but that they instead invested the time to make higher-quality decisions. **(D)** Mean of the log-normal distribution of perception-action delays fitted to data from the guided episodes for each participant (dots) using either the first action within each trial (left) or all other actions (right). These prior distributions were used to infer the thinking times in Figure 5.2.

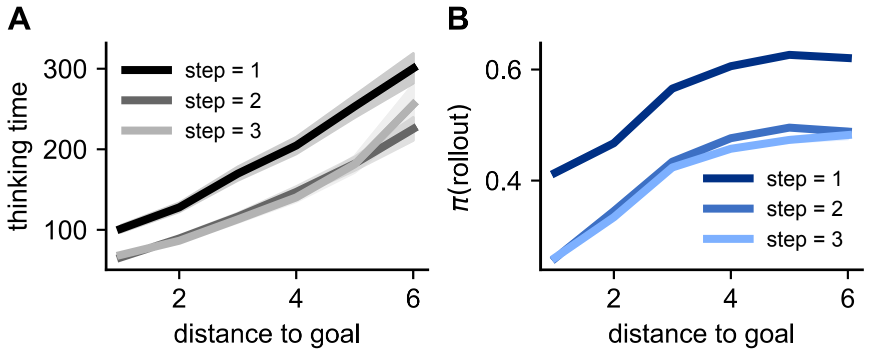

Figure D.2 **Thinking time and π(rollout) by distance to goal and step within trial.** **(A)** Figure illustrating the average thinking time across human participants as a function of distance to goal (x-axis), conditioned on different steps within the trial (lines, legend). Subjects generally spent longer thinking before the first action of each trial, after controlling for the distance to goal, while subsequent actions were associated with similar thinking times. Lines and shadings indicate mean and standard error when repeating the analysis across human participants ($n = 94$). **(B)** π(rollout) for the agent clamped to the human trajectory as a function of distance to goal and for different steps within the trial. Similar to the human participants, the agent had a higher probability of performing a rollout on the first step of each trial. Subsequent steps were associated with similar rollout probabilities after controlling for the distance to goal. When conditioning on both distance to goal and step within trial, the residual correlation between π(rollout) and thinking time remained at a significantly positive value of $r = 0.026 \pm 0.004$ (mean $\pm$ sem).

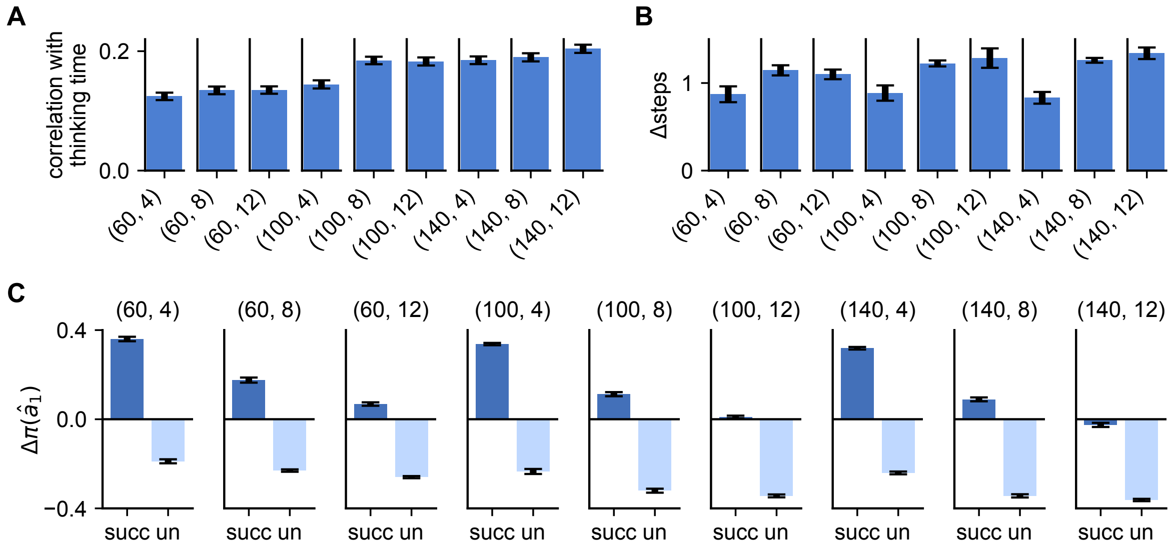

Figure D.3 **Properties of networks with different hyperparameters.** To investigate the robustness of our results to the choice of network size (N) and planning horizon (L), we trained networks with each combination of $N \in \{60, 100, 140\}$ and $L \in \{4, 8, 12\}$ and repeated some of our key analyses. For all analyses, we report mean and standard error across 5 networks with each set of hyperparameters. The results in the main text are all reported for a network with $N = 100$ and $L = 8$. **(A)** We quantified the correlation between the network π(rollout) and human response times across different networks (c.f. Figure 5.2F). x-ticks indicate network size and planning horizon as (N, L). **(B)** We computed the improvement in the network policy from performing 5 rollouts compared to the policy in the absence of rollouts (c.f. Figure 5.3A). The policy improvement was quantified as the average number of steps needed to reach the goal on trial 2 in the absence of rollouts, minus the average number of steps needed with 5 rollouts enforced at the beginning of the trial and no rollouts during the rest of the trial. Positive values indicate that rollouts improved the policy. **(C)** We investigated how rollouts changed the policy (c.f. Figure 5.3E). For each network, we computed the average change in $\pi(\hat{a}_1)$ from before a rollout to after a rollout and report this change separately for successful ('succ') and unsuccessful ('un') rollouts. Positive values indicate that $\hat{a}_1$ became *more* likely and negative values that $\hat{a}_1$ became *less* likely after the rollout. We observe that networks with longer planning horizons tend to have less positive $\Delta\pi(\hat{a}_1)$ for successful rollouts and more negative $\Delta\pi(\hat{a}_1)$ for unsuccessful rollouts. This is consistent with a policy gradient-like algorithm with a baseline that approximates the probability of success, which increases with planning horizon. In other words, since longer rollouts are more likely to reach the goal, we should expect them to be successful and not strongly update our policy when it occurs. On the contrary, an unsuccessful rollout is less likely and should lead to a large policy change. The converse is true for shorter planning horizons.

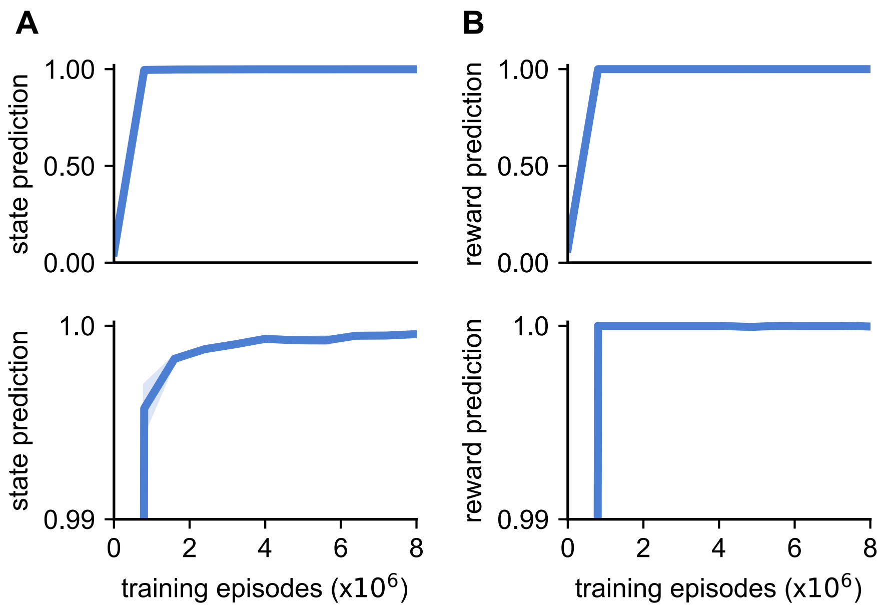

Figure D.4 **Accuracy of the internal world model.** **(A)** Accuracy of the internal transition model over the course of training. Accuracy was computed as the probability that the predicted next state was the true state reached by the agent, ignoring all teleportation steps where the transition cannot be predicted. The accuracy was averaged across all timesteps from 1,000 episodes, and the line and shading indicate mean and standard error across 5 RL agents. The upper panel considers the full range of [0, 1] while the lower panel considers the range [0.99, 1.0]. We see that the transition model rapidly approaches ceiling performance, although it continues to improve slightly throughout training. **(B)** Accuracy of the internal reward model over the course of training. Accuracy was computed as the probability that the predicted reward location was the true reward location during the exploitation phase of the task (see Figure D.5 for an analysis of the model accuracy during exploration). Lines and shadings indicate mean and standard error across 5 RL agents.

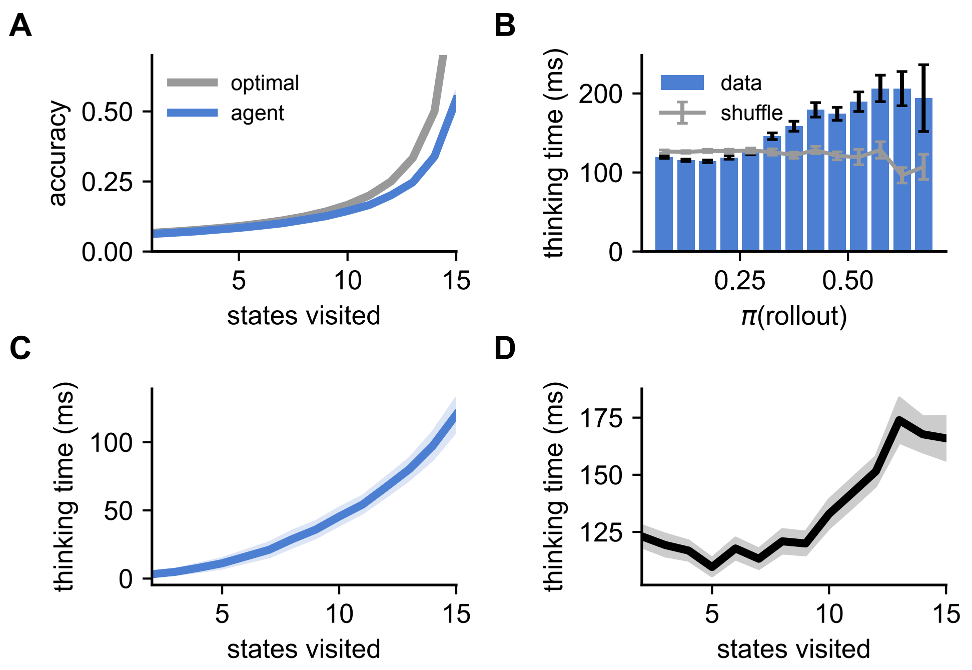

Figure D.5 **Analyses of the exploration period in humans and RL agents. (A)** At each point in time, the agent outputs its belief over where the goal is located under its internal model, which was trained using a cross-entropy loss (Appendix D.2). The figure shows the average probability assigned to the true goal, plotted as a function of the number of unique states visited during the exploration phase of the task. As more states are explored, the posterior over possible goals becomes narrower and prediction accuracy increases. When the model chooses to perform a rollout, the imagined goal is chosen as the maximum likelihood location from this posterior to predict the 'success' of the rollout. The figure illustrates that this imagined goal becomes increasingly likely to be the true goal as the agent explores more of the environment. **(B)** Thinking time of human participants during exploration, plotted as a function of π(rollout) for RL agents clamped to the human trajectory. Bars and error bars indicate mean and standard error of the human thinking time across all states where π(rollout) fell in the corresponding bin. Gray line indicates a control where human thinking times have been shuffled. The Pearson correlation between π(rollout) and human thinking times is $r = 0.097 \pm 0.008$, suggesting that the model captures some of the structure in human thinking during exploration and not just during the exploitation phase. Note that the very first action of the episode was not included in this or subsequent analyses of the human data. **(C)** Model thinking time as a function of the number of unique states visited during the exploration phase of the task, with each rollout assumed to take 120 ms as specified in the main text and Appendix D.2. Line and shading indicate mean and standard error across RL agents. The increase in thinking time with visited states mirrors the predictive performance from panel (A) and suggests that the agent increasingly chooses to engage in 'model-based' planning as its uncertainty over possible goal locations decreases. **(D)** Human thinking time as a function of the number of unique states visited during the exploration phase of the task. Line and shading indicate mean and standard error across participants. The increase in thinking time with states visited suggests that humans may also transition to more model based behavior as the posterior over possible goal locations becomes narrower. A notable difference from the computational model is found early in the exploration phase, where human thinking times tend to decrease slightly over the first few unique state visits.

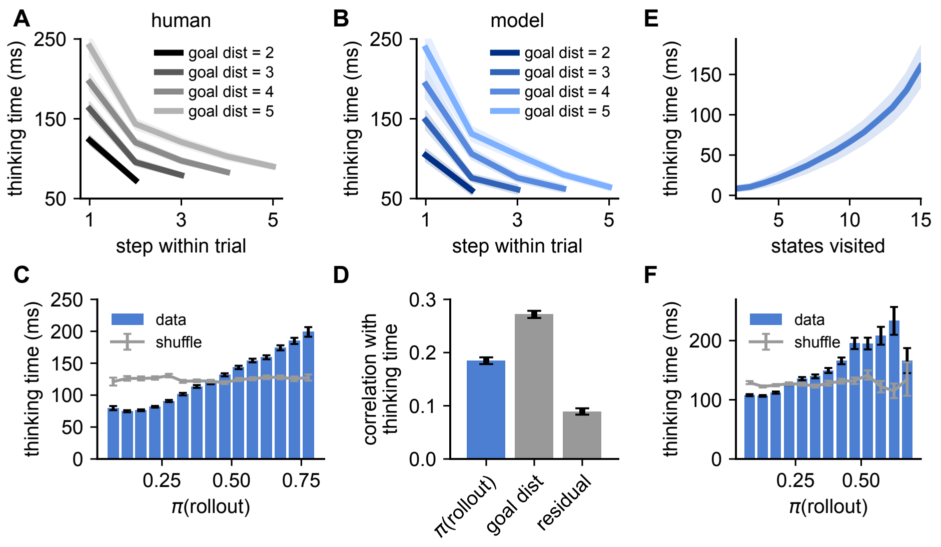

Figure D.6 **Model thinking times with a Euclidean prior.** We retrained our model with a prior that favors Euclidean actions, $p(a_k|s_k) = 0.2\mathcal{U}_{all}(a_k) + 0.8\mathcal{U}_{Euclidean}(a_k|s_k)$. Here, $\mathcal{U}_{all}$ is a uniform distribution over *all* actions, and $\mathcal{U}_{Euclidean}(a_k|s_k)$ is a uniform distribution over those actions that do not cross a boundary from state s_k. We then compared the thinking times of this 'Euclidean' model to human thinking times, similar to the analyses in Figure 5.2 and Figure D.5. **(A)** Human thinking time as a function of the step-within-trial (x-axis) for different initial distances to the goal at the beginning of the trial (lines, legend; Figure 5.2C). Shading indicates standard error of the mean across 94 participants. **(B)** Model 'thinking times' separated by time-within-trial and distance-to-goal, exhibiting a similar pattern to human participants. Note that the Euclidean agents spend more time thinking than the non-Euclidean agents in Figure 5.2D, and that this leads to a closer match to human thinking times. **(C)** Binned human thinking time as a function of the probability that the Euclidean agent chooses to perform a rollout, π(rollout). Error bars indicate standard error of the mean within each bin. Gray horizontal line indicates a shuffled control, where human thinking times were randomly permuted before the analysis. The Pearson corrrelation between these two quantities was $r = 0.185 \pm 0.006$ (mean $\pm$ standard error). **(D)** Correlation between human thinking time and the regressors (i) π(rollout) under the model, (ii) distance-to-goal, and (iii) π(rollout) after conditioning on distance-to-goal ('residual'; Appendix D.2). Bars and error bars indicate mean and standard error across human participants ($n = 94$). The residual correlation was $r = 0.089 \pm 0.006$. **(E)** Model thinking time as a function of the number of unique states visited during the exploration phase of the task. Line and shading indicate mean and standard error across RL agents. **(F)** Thinking time of human participants during exploration, plotted as a function of π(rollout) for RL agents clamped to the human trajectory. Bars and error bars indicate mean and standard error of the human thinking time across all states where π(rollout) fell in the corresponding bin. Gray line indicates a control where human thinking times have been shuffled. The Pearson correlation between π(rollout) and human thinking times was $r = 0.146 \pm 0.007$.

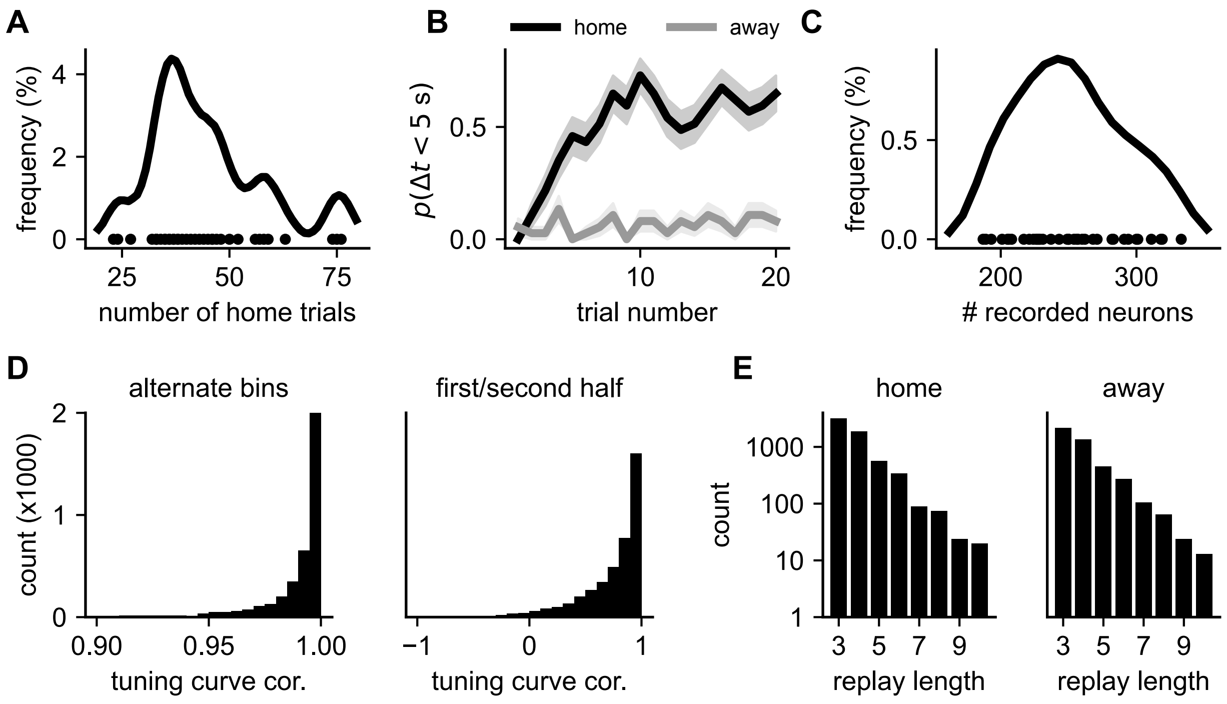

Figure D.7 **Overview of rodent data.** **(A)** Kernel density estimate ($\sigma = 3$ trials) of the distribution of the number of 'home' trials in each session across all animals (an equivalent number of away trials was performed between the home trials). Dots indicate individual sessions. **(B)** Fraction of trials where the animal reached the correct goal location and started licking within 5 seconds of the trial starting, separated by home and away trials. Reaching the goal within 5 seconds was used as a success criterion by Widloski and Foster (2022) since the goal is never explicitly cued at this time (Appendix D.2). Line and shading indicate mean and standard error across sessions. The animals learn the location of the home well within a few trials and consistently return to this location on the home trials. **(C)** Distribution of the number of recorded neurons in each session. Line indicates a convolution with a Gaussian filter (15 neuron std) and dots indicate individual sessions. Note that consecutive sessions on the same day (2-3 sessions per day) involved recording from the same neurons, so there are fewer distinct data points than there are sessions. **(D)** Consistency of spatial tuning curves of hippocampal neurons. Consistency was quantified by constructing two tuning curves on the 5x5 spatial grid (Figure 5.4A) for each neuron and computing the Pearson correlation between the two tuning curves. The data was split into either even/odd time bins in a session (left plot) or first/second half of the session (right plot) to compute a pair of tuning curves. **(E)** Distribution of replay lengths, measured as the number of states visited in a replay, for all replays during home (left) or away (right) trials. Note the log scale on the y-axis.

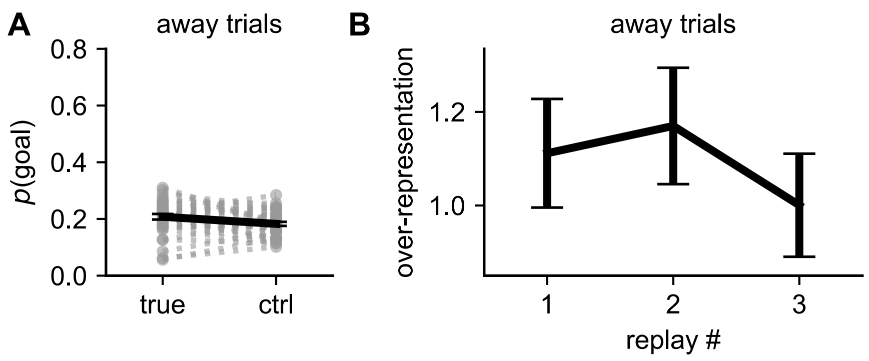

Figure D.8 **Analysis of replays during away trials. (A)** Fraction of replays reaching either the true goal (left) or a randomly sampled alternative goal location (right) during away trials. In contrast to the home trials (Figure 5.4C), the goal is not over-represented during away trials, where the goal location is unknown. **(B)** Over-representation of replay success as a function of replay number within sequences of replays containing at least 3 distinct replay events (c.f. Figure 5.4E). In contrast to the home trials, there is no increase in over-representation with replay number during these away trials.

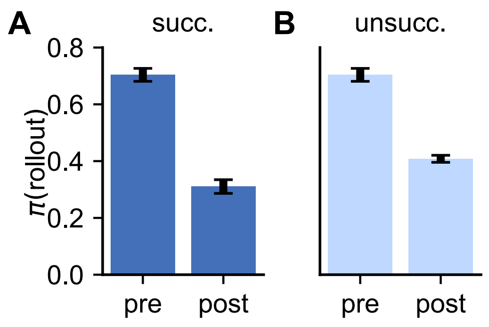

Figure D.9 **Change in π(rollout) for successful and unsuccessful rollouts. (A)** π(rollout) before (left) and after (right) successful rollouts. Bars and error bars indicate mean and standard error across 5 RL agents. The data used for this analysis was the same data used in Figure 5.3E. **(B)** As in (A), now for unsuccessful rollouts. π^{post}(rollout) was substantially larger after unsuccessful than successful rollouts ($\Delta\pi(\text{rollout}) = 0.10 \pm 0.01$ mean $\pm$ sem).

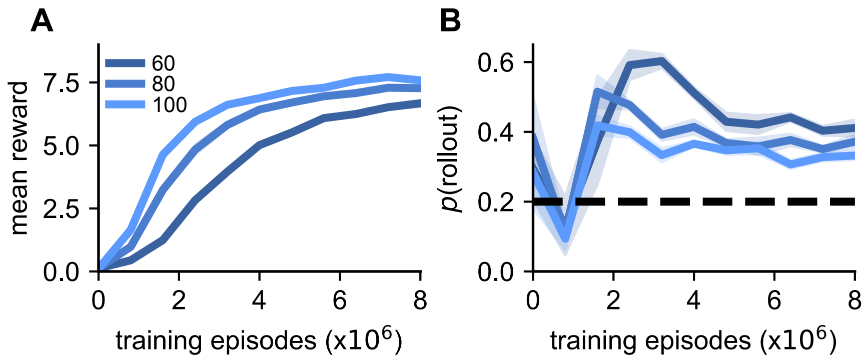

Figure D.10 **Performance and rollouts as a function of network size.** **(A)** We trained networks of different sizes (legend; $N \in [60, 80, 100]$) and quantified their performance over the course of training. **(B)** Fraction of timesteps where the agent chose to perform a rollout over the course of training for different network sizes. Note that the agents perform rollouts at chance level but with high variance at initialization, and this data point was therefore not included in the analysis in Figure 5.5E, where we only considered the learned rollout frequency from episode 800,000 onwards. It is interesting to note that the agents first learn to *suppress* the rollout frequency below chance before increasing it to levels above chance. This is consistent with a theory where rollouts only become useful when (i) an internal world model has been learned, and (ii) the agent has learned *how* to use rollouts to improve its policy. Finally, rollouts become less frequent again later in training as the base policy improves.

D.2 Methods

Software

All models were trained in Julia version 1.7 using Flux and Zygote for automatic differentiation (Innes et al., 2018). Human behavioral experiments were written in OCaml, with the front-end transpiled to javascript for running in the participants' browsers. All analyses of the models and human data were performed in Julia version 1.8. All analyses of hippocampal replay data were performed in Python 3.8.

Statistics

Unless otherwise stated, all plots are reported as mean and standard error across human participants ($n = 94$), independently trained RL agents ($n = 5$), or experimental sessions in rodents ($n = 37$).

Environment

We generated mazes using Algorithm 4.

For each environment, a goal location was sampled uniformly at random. When subjects took an action leading to the goal, they transitioned to this location before being teleported to a random location. In the computational model, this was achieved by feeding the agent an input at this location before teleporting the agent to the new location. The policy of the agent at this iteration of the network dynamics was ignored, since the agent was teleported rather than taking an action.

Reinforcement learning model

We trained our agent to maximize the expected reward, with the expectation taken both over environments $\mathcal{E}$ and the agent's policy π:

$$\mathcal{U} = \mathbb{E}_{\mathcal{E}}\left[J(\theta)\right] \tag{D.1}$$

$$= \mathbb{E}_{\mathcal{E}}\left[\mathbb{E}_{\pi}\left(\sum_{k=1}^{K} r_k\right)\right]. \tag{D.2}$$

Here, $\mathcal{U}$ is the *utility function*, k indicates the iteration within an episode, and r_k indicates the instantaneous reward at each iteration. We additionally introduced the following auxiliary

Algorithm 4: Maze generating algorithm

```
$\mathcal{A} \leftarrow$ 4x4 arena with walls everywhere.
$\mathcal{V} \leftarrow \{\}$ % empty initial set of visited states.
$s \leftarrow$ random starting location.

% Define function to walk through the maze and remove walls
Function walk_maze($s$, $\mathcal{A}$, $\mathcal{V}$)
    $\mathcal{V}$.add($s$) % Add $s$ to set of visited states
    $\mathcal{N} \leftarrow$ neighbors($s$) % Neighbors of $s$, including those through the periodic boundaries
    % Iterate through all neighboring states in random order
    for $n \in$ randomize($\mathcal{N}$) do
        % If we reached a state we have not seen before
        if $n \notin \mathcal{V}$ then
            $\mathcal{A}$.remove_wall($s$, $n$) % Remove wall between $s$ and $n$ from arena
            $\mathcal{A}$, $\mathcal{V}$ = walk_maze($n$, $\mathcal{A}$, $\mathcal{V}$) % Continue from new state
    return $\mathcal{A}$, $\mathcal{V}$

$\mathcal{A}$, $\mathcal{V}$ = walk_maze($s$, $\mathcal{A}$, $\mathcal{V}$) % Construct maze using our recursive algorithm

%Remove 3 additional walls at random to increase the degeneracy of the tasks.
%This increases the number of decision points with multiple routes to the goal.
for $i$ = 1:3 do
    $w$ = random_wall($\mathcal{A}$) % Select one of the remaining walls at random
    $\mathcal{A}$.remove_wall($w$) % Remove from set of walls

return $\mathcal{A}$ % Return the maze we constructed
```

losses at each iteration:

$$\mathcal{L}_V = 0.5(V_k - R_k)^2 \quad \text{value function} \tag{D.3}$$

$$\mathcal{L}_H = \mathbb{E}_{\pi_k} \log \pi_k \quad \text{entropy regularization} \tag{D.4}$$

$$\mathcal{L}_P = -\sum_i \left[s_{k+1}^{(i)} \log \hat{s}_{k+1}^{(i)} + g^{(i)} \log \hat{g}_k^{(i)} \right] \quad \text{internal world model.} \tag{D.5}$$

Here, $\hat{\boldsymbol{g}}_k$, and $\hat{\boldsymbol{s}}_{k+1}$ are additional network outputs representing the agent's estimate of the current reward location and upcoming state. $\boldsymbol{g}$ and $\boldsymbol{s}_{k+1}$ are the corresponding ground truth quantities, represented as one-hot vectors. $R_k := \sum_{k'=k}^{K} r_{k'}$ is the empirical cumulative future reward from iteration k onwards, and V_k is the value function of the agent.

To maximize the utility and minimize the losses, we trained the RL agent on-policy using a policy gradient algorithm with a baseline (Sutton and Barto, 2018) and parameter updates of the form

$$\Delta\theta \propto \sum_{a_k \sim \pi} \Big[(\underbrace{\nabla_\theta \log \pi_k(a_k)}_{\text{actor}} + \underbrace{\beta_v \nabla_\theta V_k}_{\text{critic}})\delta_k - \beta_e \nabla_\theta \sum_a \underbrace{\pi_{k,a} \log \pi_{k,a}}_{\text{entropy}} + \underbrace{\beta_p \Delta\theta_p}_{\text{predictive}} \Big] \tag{D.6}$$

Here, $\delta_k := -V_k + R_k$ is the 'advantage function', and $\Delta\theta_p = \nabla_\theta \mathcal{L}_P$ is the derivative of the predictive loss $\mathcal{L}_P$, which was used to train the 'internal model' of the agent. $\beta_p = 0.5$, $\beta_v = 0.05$ and $\beta_e = 0.05$ are hyperparameters controlling the importance of the three auxiliary losses. While we use the predictive model explicitly in the planning loop, similar auxiliary losses are also commonly used to speed up training by encouraging the learning of useful representations (Jaderberg et al., 2016).

Our model consisted of a GRU network with 100 hidden units (Cho et al., 2014). The policy was computed as a linear function of the hidden state followed by a softmax normalization. The value function was computed as a linear function of the hidden state. The predictions of the next state and reward location were computed with a neural network that received as input a concatenation of the current hidden state $\boldsymbol{h}_k$ and the action a_k sampled from the policy (as a one-hot representation). The output layer of this feedforward network was split into a part that encoded a distribution over the predicted next state (a vector of 16 grid locations with softmax normalization), and a part that encoded the predicted reward location in the same way. This network had a single hidden layer with 33 units and a ReLU nonlinearity.

The model was trained using ADAM (Kingma and Ba, 2014) on 200,000 batches, each consisting of 40 episodes, for a total of 8×10^6 training episodes. These episodes were sampled independently from a total task space of $(273 \pm 13) \times 10^6$ tasks (mean $\pm$ standard error). The total task space was estimated by sampling 50,000 wall configurations and computing the fraction of the resulting 1.25×10^9 pairwise comparisons that were identical, divided by 16 to account for the possible reward locations. This process was repeated 10 times to estimate a mean and confidence interval. These considerations suggest that the task coverage during training was $\approx 2.9\%$, which confirms that the majority of tasks seen at test time are novel (although we do not enforce this explicitly).

For all evaluations of the model, actions were sampled greedily rather than on-policy unless otherwise stated. This was done since the primary motivation for using a stochastic policy is to explore the space of policies to improve learning, and performance was better under the greedy policy at test time.

To train the model with a Euclidean prior over actions (Figure D.6; Section 2.5.2), we replaced the entropy regularization term $\mathcal{L}_H = \mathbb{E}_{\pi_k}[\log \pi_k]$ with a KL regularization of the form

$$\mathcal{L}_H = \mathrm{KL}[\pi(a_k|s_k)||p(a_k|s_k)]. \tag{D.7}$$

As mentioned in Figure D.6, $p(a_k|s_k) = 0.2\mathcal{U}_{all}(a_k) + 0.8\mathcal{U}_{Euclidean}(a_k|s_k)$, where $\mathcal{U}_{all}$ is a uniform distribution over *all* actions, and $\mathcal{U}_{Euclidean}(a_k|s_k)$ is a uniform distribution over those actions that do not cross a boundary from state s_k. Note that both here and for the standard entropy regularization, we take gradient steps of the form $\mathbb{E}_{\{s\sim\pi_\theta\}}\left[\sum_k \nabla_\theta \text{KL}[\pi_\theta(a_k|s_k)||p(a_k|s_k)]\right]$ (with a uniform prior in the case of standard entropy regularization). By doing this, we ignore the dependency of the state sequence $\{s\}$ on θ, which is a common approach in the literature (Levine, 2018). This results in the agent learning to follow the prior conditioned on the current state, but it does not learn to move to states where the prior probability of selecting an action is expected to be high (Levine, 2018). However, in future work it would be interesting to systematically compare this approach to the use of the log derivative trick for estimating $\nabla_\theta \mathcal{L}_H$ as discussed in Section 2.5.

Planning

Our implementation of ‘planning’ in the form of policy rollouts is described in Algorithm 5. This routine was invoked whenever a ‘rollout’ was sampled from the policy instead of a physical action.

Algorithm 5: Planning routine for the RL agent

```
1  input: maximum planning depth ($n_{max}$), current hidden state ($\boldsymbol{h}_k$), and agent location
   ($\boldsymbol{s}_k$)
2  parameters: network parameters $\theta$, defining $\phi(\cdot)$, $\zeta(\cdot)$, $p(\hat{\boldsymbol{g}}|\boldsymbol{h}_k)$, and $p(\hat{\boldsymbol{s}}|a,\boldsymbol{h})$
3
4  $\tilde{\boldsymbol{g}} \leftarrow \text{argmax}\, p(\hat{\boldsymbol{g}}|\boldsymbol{h}_k)$ % predicted goal location
5  $\tilde{\boldsymbol{h}}_k, \tilde{\pi}_k, \tilde{\boldsymbol{s}}_k \leftarrow \boldsymbol{h}_k, \pi_k, \boldsymbol{s}_k$ % simulated hidden state, policy, and agent location, initialized to
   true values
6  $n \leftarrow 0$ % planning iteration
7
8  while $n < n_{max}$ and $\tilde{\boldsymbol{s}}_{k+n} \neq \tilde{\boldsymbol{g}}$ do
9      $\tilde{a}_{k+n} \sim \tilde{\pi}_{k+n}[\{a\}_{\text{no\_plan}}]$ % imagined action sampled on-policy but from physical actions
       only
10     $\tilde{\boldsymbol{s}}_{k+n+1} \leftarrow \text{argmax}\, p(\hat{\boldsymbol{s}}_{k+n+1}|\tilde{a}_{k+n}, \tilde{\boldsymbol{h}}_{k+n})$ % predicted next state from current imagined
       state and action
11     $\tilde{\boldsymbol{x}}_{k+n+1} \leftarrow O(\tilde{\boldsymbol{s}}_{k+n+1}, \tilde{\boldsymbol{g}})$ % expected observations on next iteration (assuming access to
       the function $O(\cdot)$)
12     $\tilde{\boldsymbol{h}}_{k+n+1} \leftarrow \phi(\tilde{\boldsymbol{x}}_{k+n+1}, \tilde{\boldsymbol{h}}_{k+n})$ % simulate agent dynamics
13     $\tilde{\pi}_{k+n+1} = \zeta(\tilde{\boldsymbol{h}}_{k+n+1})$ % generate new policy
14     $n \leftarrow n+1$ % update planning iteration
15
16 % return action sequence and whether the rollout reached the expected goal
17 return: $\{\tilde{a}_{k'}\}_k^{k+n}$, $\delta(\tilde{\boldsymbol{s}}_{k+n}, \tilde{\boldsymbol{g}})$
```

For the network update following a rollout, the input $\boldsymbol{x}_{k+1}$ was augmented with an additional 'rollout input' consisting of (i) the sequence of simulated actions, each as a 1-hot vector, and (ii) a binary input indicating whether the imagined sequence of states reached the imagined goal location. Additionally, the time within the session was only updated by 120 ms after a rollout in contrast to the 400 ms update after a physical action or teleportation step.

Note that while both an imagined 'physical state' $\tilde{\boldsymbol{s}}_k$ and 'hidden state' $\tilde{\boldsymbol{h}}_k$ are updated during the rollout, the agent continues from the *original* location $\boldsymbol{s}_k$ and hidden state $\boldsymbol{h}_k$ after the rollout, but with an augmented input. Additionally, gradients were not propagated through the rollout process, which was considered part of the 'environment'. This means that there was no explicit gradient signal that encouraged the policy to drive useful or informative rollouts. Instead, the rollout process simply relied on the utility of the base policy optimized for acting in the environment.

Performance by number of rollouts

To quantify the performance as a function of the number of planning steps in the RL agent (Figure 5.3A), we simulated each agent in 1,000 different mazes until it first found the goal and was teleported to a random location. We then proceeded to enforce a particular number of rollouts before the agent was released in trial 2. During this release phase, no more rollouts were allowed – in other words, the policy was re-normalized over the physical actions, and the probability of performing a rollout was set to zero. Performance was then quantified as the average number of steps needed to reach the goal during this test phase. The optimal reference value was computed as the average optimal path length for the corresponding starting states. When performing more than one sequential rollout prior to taking an action, the policy of the agent can continue to change through two potential mechanisms. The first is that the agent can explicitly 'remember' the action sequences from multiple rollouts and somehow arbitrate between them. The second is to progressively update the hidden state in a way that leads to a better expected policy with each rollout, since the feedback from a rollout is incorporated into the hidden state that induces the policy used to draw the next rollout. On the basis of the analysis in Figure 5.5, we expect the second mechanism to be dominant, although we did not explicitly test the ability of the agent to 'remember' multiple action sequences from sequential rollouts. For this and all other RNN analyses, the agent executed the most likely action under the policy during 'testing' in contrast to the sampling performed during training, where such stochasticity is necessary for exploring the space of possible actions. All results were qualitatively similar if actions were sampled during the test phase, although average performance was slightly worse.

Performance in the absence of rollouts and with shuffled rollout times

To quantify the performance of the RL agent in the absence of rollouts, we let the agent receive inputs and produce outputs as normal. However, we set the probability of performing a rollout under the policy to zero and re-normalized the policy over the physical actions before choosing an action from the policy. We compared the average performance of the agent (number of rewards collected) in this setting to the performance of the default agent in the same environments.

To compare the original performance to an agent with randomized rollout times, we counted the number of rollouts performed by the default agent in each environment. We then re-sampled a new set of network iterations at which to perform rollouts, matching the size of this new set to the original number of rollouts performed in the corresponding environment. Finally, we let the agent interact with the environment again, while enforcing a rollout on these network iterations, and preventing rollouts at all other timesteps. It is worth noting that we could not predict *a priori* the iterations on which the agent would find the goal, at which point rollouts were not possible. If a rollout had been sampled at such an iteration, we re-sampled this rollout from the set of remaining network iterations.

Rollouts by network size

To investigate how the frequency of rollouts depended on network size (Figure 5.5E; Figure D.10), we trained networks with either 60, 80, or 100 hidden units (GRUs). Five networks were trained with each size. At regular intervals during training, we tested the networks on a series of 5,000 mazes and computed (i) the average reward per episode, and (ii) the fraction of actions that were rollouts rather than physical actions. We then plotted the rollout fraction as a function of average reward to see how frequently an agent of a given size performed rollouts for a particular performance.

Effect of rollouts on agent policy

To quantify the effect of rollouts on the policy of the agent, we simulated each agent in 1,000 different mazes until it first found the goal and was teleported to a random location. We then resampled rollouts until both (i) a successful rollout and (ii) an unsuccessful rollout had been sampled. Finally, we quantified $\pi^{pre}(\hat{a}_1)$ and $\pi^{post}(\hat{a}_1)$ separately for the two scenarios and plotted the results in Figure 5.3E. Importantly, this means that each data point in the 'successful' analysis had a corresponding data point in the 'unsuccessful' analysis with the exact same maze, location, and hidden state. In this way, we could query the effect of rollouts on the policy without the confound of how the policy itself affects the rollouts. For this analysis, we discarded episodes where the first 100 sampled rollouts did not result in both a successful and an unsuccessful rollout.

For Figure D.9, we used the same episodes and instead quantified π(rollout) before and after the rollout, repeating the analysis for both successful and unsuscessful rollouts.

Overlap between hidden state updates and policy gradients

Using a single rollout ($\hat{\tau}$) to approximate the expectation over trajectories of the gradient of the expected future reward for a given episode, $\nabla_{\boldsymbol{h}} J_{fut}(\boldsymbol{h})$, the policy gradient update in $\boldsymbol{h}$ takes the form $\Delta \boldsymbol{h} \propto (R_{\hat{\tau}} - b)\nabla_{\boldsymbol{h}} \log p(\hat{\tau})$. Here, $\Delta \boldsymbol{h}$ is the change in hidden state resulting from the rollout, $R_{\hat{\tau}}$ is the 'reward' of the simulated trajectory, b is a constant or state-dependent baseline, and $\nabla_{\boldsymbol{h}} \log p(\hat{\tau})$ is the gradient with respect to the hidden state of the log probability of $\hat{\tau}$ under the policy induced by $\boldsymbol{h}$. This implies that the *derivative* of the hidden state update w.r.t. $R_{\hat{\tau}}$, $\boldsymbol{\alpha}^{\text{RNN}} := \frac{\partial \Delta \boldsymbol{h}}{\partial R_{\hat{\tau}}}$, should be proportional to $\boldsymbol{\alpha}^{\text{PG}} := \nabla_{\boldsymbol{h}} \log p(\hat{\tau})$.

For these analyses, we divided $\hat{\tau}$ into its constituent actions, defining $\boldsymbol{\alpha}_k^{\text{PG}} := \nabla_{\boldsymbol{h}} \log p(\hat{a}_k | \hat{a}_{1:k-1})$ as the derivative w.r.t. the hidden state of the log probability of taking the simulated action at step k, conditioned on the actions at all preceding steps (1 to $k-1$) being consistent with the rollout. To compute $\boldsymbol{\alpha}^{\text{RNN}}$, we also needed to take derivatives w.r.t. $R_{\hat{\tau}}$ – the 'reward' of a rollout. A naive choice here would be to simply consider $R_{\hat{\tau}}$ to be the input specifying whether the rollout reached the reward or not. However, we hypothesized that the agent would also use information about e.g. how long the simulated trajectory was in its estimate of the 'goodness' of a rollout (since a shorter rollout implies that the goal was found faster). We therefore determined the direction in planning input state space that was most predictive of the time-to-goal of the agent. We did this by using linear regression to predict the (negative) time-to-next-reward as a function of the planning feedback $\boldsymbol{x}_f$ across episodes and rollouts. This defines the (normalized) direction $\hat{\boldsymbol{\nu}}$ in planning input space that maximally increases the expected future reward. Finally, we defined $R_{\hat{\tau}}$ as the magnitude of the planning input in direction $\hat{\boldsymbol{\nu}}$, $R_{\hat{\tau}} := \boldsymbol{x}_f \cdot \hat{\boldsymbol{\nu}}$. We could then compute $\boldsymbol{\alpha}^{\text{RNN}}$ with this definition of $R_{\hat{\tau}}$ using automatic differentiation.

In Figure 5.5C, we computed $\boldsymbol{\alpha}^{\text{RNN}}$ and $\boldsymbol{\alpha}_1^{\text{PG}}$ across 1,000 episodes. We then performed PCA on the set of $\boldsymbol{\alpha}_1^{\text{PG}}$ and projected both $\boldsymbol{\alpha}^{\text{RNN}}$ and $\boldsymbol{\alpha}_1^{\text{PG}}$ into the space spanned by the top 3 PCs. Finally, we computed the mean value of both quantities conditioned on $\hat{a}_1$ to visualize the alignment. In Figure 5.5D, we considered the same $\boldsymbol{\alpha}^{\text{RNN}}$ and $\boldsymbol{\alpha}_1^{\text{PG}}$ vectors, now computing the cosine similarity between each pair of vectors before taking an average. This cosine similarity was still computed in the space spanned by the top 3 PCs since we were primarily interested in changes in $\boldsymbol{h}$ within the subspace that would affect $\log p(\hat{\tau})$. As a control, we repeated the analysis after altering the planning input $\boldsymbol{x}_f$ to falsely inform the agent that it had simulated some other action $\hat{a}_{1,\text{ctrl}} \neq \hat{a}_1$. Finally, we also repeated this analysis using $\boldsymbol{\alpha}_2^{\text{PG}}$ to characterize how the effects of the planning input propagated through the recurrent network dynamics to modulate future action probabilities.

Human data collection

The human behavioral experiment used in this study has been certified as exempt from IRB review by the UC San Diego Human Research Protection Program. We collected data from 100 human participants (50 male, 50 female) recruited on Prolific to perform the task described in Figure 5.1B. All participants provided informed consent prior to commencing the experiment. Subjects were asked to complete (i) 6 'guided' episodes where the optimal path was shown explicitly, followed by (ii) 40 non-guided episodes, and (iii) 12 guided episodes. The task can be found online. During data collection, a subject was deemed 'disengaged', and the trial repeated, if one of three conditions were met: (i) the same key was pressed 5 times in a row, (ii) the same key pair was pressed four times in a row, or (iii) no key was pressed for 7 seconds. Participants were paid a fixed rate of \$3 plus a performance-dependent bonus of \$0.002 for each completed trial across both guided and non-guided episodes. The experiment took approximately 22 minutes to complete, and the average pay across participants was \$10.5 per hour including the performance bonus.

The data from 6 participants with a mean response time greater than 690 ms during the guided episodes were excluded to avoid including participants who were not sufficiently engaged with the task. For the guided episodes, only the last 10 episodes were used for further analyses. For the non-guided episodes, we discarded the first two episodes and used the last 38 episodes. This was done to give participants two episodes to get used to the task for each of the two conditions, and the first set of guided episodes was intended as an instruction in how to perform the task.

Performance as a function of trial number

We considered all episodes where the humans or RL agents completed at least four trials, evaluating the RL agents across 50,000 episodes. We then computed the average across these episodes of the number of steps to goal as a function of trial number separately for all subjects. Figure 5.2A illustrates the mean and standard error across subjects (human participants or RL agents). The optimal value during the exploitation phase was computed by using dynamic programming to find the shortest path between each possible starting location and the goal location, averaged across all environments seen by the RL agent. To compute the exploration baseline, brute force search was used to identify the path that explored the full environment as fast as possible. The optimal exploration performance was then computed as the expected time-to-first-reward under this policy, averaged over all possible goal locations.

Estimation of thinking times

In broad strokes, we assumed that for each action, the response time t_r is the sum of a thinking time t_t and some perception-action delay t_d, both subject to independent variability:

$$t_\mathrm{r} = t_\mathrm{t} + t_\mathrm{d} \quad \text{with} \quad t_\mathrm{t} \sim p_\mathrm{t} \quad \text{and} \quad t_\mathrm{d} \sim p_\mathrm{d}. \tag{D.8}$$

Here, $\{t_\mathrm{r}, t_\mathrm{t}, t_\mathrm{d}\} \geq 0$ since elapsed time cannot be negative. We assumed that the prior distribution over perception-action delays, p_d, was identical during guided and non-guided trials. For each subject, we obtained a good model of p_d (see below) by considering the distribution of response times measured during guided trials. This was possible because guided trials involved no 'thinking' by definition, such that $t_\mathrm{d} \equiv t_\mathrm{r}$ was directly observed. Finally, for any non-guided trial with observed response t_r, we formed a point estimate of the thinking time by computing the mean of the posterior $p(t_\mathrm{t}|t_\mathrm{r})$:

$$\hat{t}_{\mathrm{t}|t_\mathrm{r}} = \mathbb{E}_{p(t_\mathrm{t}|t_\mathrm{r})}[t_\mathrm{t}]. \tag{D.9}$$

In more detail, we took p_t during non-guided trials to be uniform between 0 and 7 s – the maximum response time allowed, beyond which subjects were considered disengaged, and the trial was discarded and reset. For $p_\mathrm{d}(t_\mathrm{d})$, we assumed a shifted log-normal distribution,

$$p_\mathrm{d}(t_\mathrm{d}; \mu, \sigma, \delta) = \begin{cases} \frac{1}{(t_\mathrm{d}-\delta)\sigma\sqrt{2\pi}} \exp\left[-\frac{(\log(t_\mathrm{d}-\delta)-\mu)^2}{2\sigma^2}\right] & \text{if } t_d > \delta \\ 0 & \text{otherwise} \end{cases} \tag{D.10}$$

with parameters μ, σ, and δ obtained via maximum likelihood estimation based on the collection of response times $t_\mathrm{r} \equiv t_\mathrm{d}$ observed during guided trials. For a given δ, the maximum likelihood values of μ and σ are simply given by the mean and standard deviation of the logarithm of the shifted observations. To fit this shifted log-normal model, we thus performed a grid search over $\delta \in [0, \min(t_r^{\text{guided}}) - 1]$ at 1 ms resolution and selected the value under which the optimal (μ, σ) gave the largest likelihood. This range of δ was chosen to ensure that (i) only positive values of t_r^{guided} had positive probability, and (ii) all observed t_r^{guided} had non-zero probability. We then retained the optimal μ, σ, and δ to define the prior over $p_\mathrm{d}(t_d)$ on non-guided trials for each subject.

According to Bayes' rule, the posterior is proportional to

$$p(t_\mathrm{t}|t_\mathrm{r}) \propto p(t_\mathrm{r}|t_\mathrm{t})p(t_\mathrm{t}) \tag{D.11}$$

where

$$p(t_\mathrm{r}|t_\mathrm{t}) = \int_0^\infty dt_\mathrm{d}\, p_\mathrm{d}(t_\mathrm{d})\, p(t_\mathrm{r}|t_\mathrm{t}, t_\mathrm{d}) \tag{D.12}$$

$$= \int_0^\infty dt_\mathrm{d}\, p_\mathrm{d}(t_\mathrm{d})\, \delta(t_\mathrm{d} - (t_\mathrm{r} - t_\mathrm{t})) \tag{D.13}$$

$$= p_\mathrm{d}(t_\mathrm{r} - t_\mathrm{t}) \tag{D.14}$$

Therefore, the posterior is given by

$$p(t_\mathrm{t}|t_\mathrm{r}) \propto \begin{cases} p_\mathrm{d}(t_\mathrm{r} - t_\mathrm{t}) & \text{if } t_\mathrm{t} > 0 \\ 0 & \text{otherwise,} \end{cases} \tag{D.15}$$

resulting in the following posterior mean:

$$\hat{t}_{\mathrm{t}|t_\mathrm{r}} := \mathbb{E}_{p(t_\mathrm{t}|t_\mathrm{r})}[t_\mathrm{t}] = t_\mathrm{r} - \int_\delta^{t_\mathrm{r}} t_\mathrm{d}\, p_\mathrm{d}(t_\mathrm{d}|t_\mathrm{d} < t_\mathrm{r}; \mu, \sigma, \delta)\, dt_\mathrm{d}. \tag{D.16}$$

Here, $p_\mathrm{d}(t_\mathrm{d}|t_\mathrm{d} < t_\mathrm{r})$ denotes $p_\mathrm{d}(t_\mathrm{d})$ re-normalized over the interval $t_\mathrm{d} < t_\mathrm{r}$, and the condition $(t_\mathrm{d} < t_\mathrm{r})$ is equivalent to $(t_\mathrm{t} > 0)$. We note that the integral runs from δ to t_r since $p_\mathrm{d}(t_\mathrm{d}) = 0$ for $t_\mathrm{d} < \delta$. As δ simply shifts the distribution over t_d, we can rewrite this as

$$\hat{t}_{\mathrm{t}|t_\mathrm{r}} = t_\mathrm{r} - \delta - \int_0^{t_\mathrm{r}-\delta} x\, p_\mathrm{d}(x|x < t_\mathrm{r} - \delta; \mu, \sigma, \delta = 0)\, dx. \tag{D.17}$$

This is useful since the conditional expectation of a log-normally distributed random variable with $\delta = 0$ is given in closed form by

$$\mathbb{E}_{\mu,\sigma}[x|x < k] = \int_0^k x\, p(x|x < k; \mu, \sigma, \delta = 0)\, dx \tag{D.18}$$

$$= \exp[\mu + 0.5\sigma^2] \frac{\Phi\left(\frac{\log(k) - \mu - \sigma^2}{\sigma}\right)}{\Phi\left(\frac{\log(k) - \mu}{\sigma}\right)}, \tag{D.19}$$

where $\Phi(\cdot)$ is the cumulative density function of the standard Gaussian, $\mathcal{N}(0,1)$. This allows us to compute the posterior mean thinking time for an observed response time t_r in closed form as

$$\hat{t}_{t|t_\mathrm{r}} = t_\mathrm{r} - \delta - \mathbb{E}_{\mu,\sigma}[x|x < t_\mathrm{r} - \delta]. \tag{D.20}$$

We note that the support of $p_\mathrm{d}(t_\mathrm{d}|t_\mathrm{d} < t_\mathrm{r}; \mu, \sigma, \delta)$ is $t_\mathrm{d} \in [\delta, t_\mathrm{r}]$. For 0.6% of the non-guided decisions, the value of t_r was *lower* than the estimated δ for the corresponding participant, in which case $p(t_\mathrm{t}|t_\mathrm{r})$ is undefined. In such cases, we defined the thinking time to be $\hat{t}_{\mathrm{t}|t_\mathrm{r}} = 0$, since the response time was shorter than our estimated minimum perception-action delay. A necessary (but not sufficient) condition for $t_\mathrm{r} < \delta$ is that t_r is smaller than the smallest response time in the guided trials.

The whole procedure of fitting and inference described above was repeated separately for actions that immediately followed a teleportation step (i.e. the first action in each trial) and for all other actions. This is because we expected the first action in each trial to be associated with an additional perceptual delay compared to actions that followed a predictable transition.

All results were qualitatively similar using other methods for estimating thinking time, including (i) a log-normal prior over t_d with no shift ($\delta = 0$), (ii) using the posterior mode instead of the posterior mean, (iii) estimating a constant t_d from the guided trials, and (iv) estimating a constant t_d as the 0.1 or 0.25 quantile of t_r from the non-guided trials.

Thinking times in different situations

To investigate how the thinking time varied in different situations, we considered only exploitation trials and computed for every action (i) the minimum distance to the goal *at the beginning of the corresponding trial*, and (ii) what action number this was within the trial. We then computed the mean thinking time as a function of action number separately for each distance-to-goal. This analysis was repeated across experimental subjects and results reported as mean and standard error across subjects.

We repeated this analysis for the RL agents, where 'thinking time' was now defined based on the average number of rollouts performed, conditioned on action-within-trial and original distance to goal.

Comparison of human and model thinking times

For each subject and each RL agent, we clamped the trajectory of the agent to that taken by the subject (i.e. we used the human actions instead of sampling from the policy). After taking an action, we recorded $\pi(\text{rollout})$ under the model on the first timestep of the new state for comparison with human thinking times. We then sampled a rollout with probability $\pi(\text{rollout})$ and took an action (identical to the next human action) with probability $1 - \pi(\text{rollout})$, repeating this process until the next state was reached. Finally, we computed the average $\pi(\text{rollout})$ across 20 iterations of each RL agent for comparison with the human thinking time in each state. Figure 5.2E shows the human thinking time as a function of $\pi(\text{rollout})$, with the bars and error bars illustrating the mean and standard error in each bin. For this analysis, data was aggregated across all participants. Results were similar if we compared human thinking times with the average number of rollouts performed rather than the initial $\pi(\text{rollout})$.

In Figure 5.2F, we computed the correlation between thinking time and various regressors on a participant-by-participant basis and report the result as mean and standard error across participants ($n = 94$). For the 'residual' correlation, we first computed the mean thinking time for each distance-to-goal for each participant and the corresponding mean π(rollout) for the RL agents. We then subtracted the appropriate mean values from the thinking times and π(rollout) in the human participants and RL agents. In other words, we subtracted the average thinking time for situations 5 steps from the goal from all data points where the participant was 5 steps from the goal etc. Finally, we computed the correlation between the residual π(rollout) and the residual thinking times. This analysis was repeated across all participants and the result reported as mean and standard error across participants. Note that all 'distance-to-goal' measures refer to the shortest path to goal rather than the number of steps actually taken by the participant to reach the goal.

Analysis of hippocampal replays

For our analyses of hippocampal replays in rats, we used data recently recorded by Widloski and Foster (2022). This dataset consisted of a total of 37 sessions from 3 rats (n = 17, 12, 8 sessions for each rat) as they performed a dynamic maze task. This task was carried out in a square arena with 9 putative reward locations. In each session, six walls were placed in the arena, and a single reward location was randomly selected as the 'home' well. The task involved alternating between moving to this home well and a randomly selected 'away' well. Importantly, a delay of 5-15 s was imposed between the animal leaving the previous rewarded well before reward (chocolate milk) became available at the next rewarded well. On the away trials, the emergence of reward was also accompanied by a visual cue at the rewarded well, informing the animal that this was the reward location. Note that we only considered replays at the previous well before this visual cue and reward became available. In a given session, the animals generally performed around 80 trials (40 home trials and 40 away trials; Figure D.7). For further task details, we refer to Widloski and Foster (2022).

For our analyses, we only included trials which lasted less than 40 seconds. We did this to discard time periods where the animals were not engaged with the task. Additionally, we discarded the first home trial of each session, where the home location was unknown, since we wanted to compare the hippocampal replays with model rollouts during the exploitation phase of the maze task. For all analyses, we discretized the environment into a 5x5 grid (the 3x3 grid of wells and an additional square of states around these) in order to facilitate more direct comparisons with our human and RNN task. Following Widloski and Foster (2022), we defined 'movement epochs' as times where the animal had a velocity greater than 2 cm/s and 'stationary epochs' as times there the animal had a velocity less than 2 cm/s.

Replay detection

To detect replays, we followed Widloski and Foster (2022) and fitted a Bayesian decoder to neural activity as a function of position during movement epochs in each session, assuming Poisson noise statistics and considering only neurons with an average firing rate of at least 0.1 Hz over the course of the session. This decoder was trained on a rolling window of neural activity spanning 75 ms and sampled at 5 ms intervals (Widloski and Foster, 2022). We then detected replays during stationary epochs by classifying each momentary hippocampal state as the maximum likelihood state under the Bayesian decoder, again using neural activity in 75 ms windows at 5 ms intervals. Forward replays were defined as sequences of states which included 2 consecutive transitions to an adjacent state (i.e. a temporally and spatially contiguous sequence of three or more states), and which originated at the true animal location. For all animals, we only analyzed replays where the animal was at the previous reward location *before* it initiated the new trial (c.f. Widloski and Foster, 2022). To increase noise robustness, we allowed for short 'lapses' in a replay, defined as periods with a duration less than or equal to 20 ms, where the decoded location moved to a distant location before returning to the previously decoded location. These lapses were ignored for downstream analyses.

Wall avoidance

To compute the wall avoidance of replays (Figure 5.4B), we calculated the fraction of state transitions that passed through a wall. This was done across all replays preceding a 'home' trial (i.e. when the animal knew the next goal). As a control, we computed the same quantity averaged over 7 control conditions, which corresponded to the remaining non-identical rotations and reflections of the walls from the corresponding session. We repeated this analysis for all sessions and report the results in Figure 5.4 as mean and standard error across sessions. To test for significance, we randomly permuted the 'true' and 'control' labels independently for each session and computed the fraction of permutations (out of 10,000), where the difference between 'control' and 'true' was larger than the experimentally observed value.

This analysis was also repeated in the RL agent, where the control value was computed with respect to 50,000 other wall configurations sampled from the maze generating algorithm (Algorithm 4).

Reward enrichment

To compute the reward enrichment in hippocampal replays (Figure 5.4C), we computed the fraction of all replays preceding a 'home' trial that passed through the reward location. As a control, we repeated this analysis for the remaining 7 locations that were neither the reward location nor the current agent location (for each replay). Control values are reported as the average across these 7 control locations across all replays. This analysis was repeated for all sessions. To test for significance, we randomly permuted the ‘goal’ and ‘control’ labels independently for each session and computed the fraction of permutations (out of 10,000) where the difference between ‘goal’ and ‘control’ was larger than the experimentally observed value.

This analysis was also repeated in the RL agent, where the control value was computed across the remaining 14 possible goal locations (that were not the current location or true goal).

Behavior by replay type

To investigate how the animal behavior depended on the type of replay (Figure 5.4D), we analyzed home trials and away trials separately. We constructed a list of all the ‘first’ replayed actions $\hat{a}_1$, defined as the cardinal direction corresponding to the first state transition in each replay. We then constructed a corresponding list of the first physical action following the replay, corresponding to the cardinal direction of the first physical state transition after the replay. Finally, we computed the overlap between these two vectors to arrive at the probability of ‘following’ a replay. This overlap was computed separately for ‘successful’ and ‘unsuccessful’ replays, where successful replays were defined as those that reached the goal without passing through a wall. For the unsuccessful replays, we considered the 7 remaining locations that were not the current animal location or current goal. We then computed the average overlap under the assumption that each of these locations were the goal, while discarding replays that were successful for the ‘true’ goal. This analysis was performed independently across all sessions and results reported as mean and standard error across sessions. To test for significance, we randomly permuted the ‘successful’ and ‘unsuccessful’ labels independently for each session and computed the fraction of permutations (out of 10,000) where the difference between successful and unsuccessful replays was larger than the experimentally observed value.

This analysis was also repeated in the RL agent, where we considered all exploitation trials together since they were not divided into ‘home’ or ‘away’ trials. In this case, the control was computed with respect to all 14 locations that were not the current location or current goal location.

Effect of consecutive replays

To compute how the probability of a replay being ‘successful’ depended on replay number (Figure 5.4E), we considered all trials where an animal performed at least 3 replays. We then computed a binary vector indicating whether each replay was successful or not. From this vector, we subtracted the expected success frequency from a linear model predicting success from (i) the time since arriving at the current well, and (ii) the time until departing the current well. We did this to account for any effect of time that was separate from the effect of replay number, since such an effect has previously been reported by Ólafsdóttir et al. (2017). However, this work also notes that many of what they denote 'disengaged' replays are non-local and would automatically be filtered out by our focus on local replays. When fitting this linear model, we capped all time differences at a maximum value of $|\Delta t| = 15$ s to avoid the analysis being dominated by outliers, and because Ólafsdóttir et al. (2017) only observe an effect for time differences in this range. Our results were not sensitive to altering or removing this threshold. We then conditioned on replay number and computed the probability of success (after regressing out time) as a function of replay number. Finally, we repeated this analysis for all 7 control locations for each replay and divided the true values by control values defined as the average across replays of the average across control locations. A separate correction factor was subtracted from these control locations, which was computed by fitting a linear model to predict the average probability of successfully reaching a control location as a function of the predictors described above. The normalization by control locations was done to account for changes in replay statistics that might affect the results, such as systematically increasing or decreasing replay durations with replay number. To compute the statistical significance of the increase in goal over-representation, we also performed this analysis after independently permuting the order of the replays in each trial to break any temporal structure. This permutation was performed *after* regressing out the effect of time. We repeated this analysis across 10,000 independent permutations and computed statistical significance as the number of permutations for which the increase in over-representation was greater than or equal to the experimental value.

For the corresponding analysis in the RL agents, we did not regress out time since there is no separability between time and replay number. Additionally, the RL agent cannot alter its policy in the absence of explicit network updates – which in our model are always tied to either a rollout or an action. As noted in the main text, an increase in the probability of 'success' with replay number in the RL agent could also arise from the fact that performing further replays is less likely after a successful replay than after an unsuccessful replay (Figure D.9). We therefore performed the analysis of consecutive replays in the RL agent in a 'crossvalidated' manner at the level of the policy. In other words, every time the agent performed a rollout, we drew two samples from the rollout generation process. The first of these samples was used as normal by the agent to update $\boldsymbol{h}_k$ and drive future behavior. The second sample was never used by the agent, but was instead used to compute the 'success frequency' for our analyses. This was done to break the correlation between the choice of performing a replay and the assessment of how good the policy was, which allowed us to compute an unbiased estimate of the quality of the policy as a function of replay number. As mentioned in the main text, such an analysis is not possible in the biological data. However, since the biological task was not a reaction time task, we expect less of a causal effect of replay success on the number of replays. Additionally, as noted in the text, if some of the effect in the biological data is in fact driven by a decreased propensity for further replays after a successful replay, that is in itself supporting evidence for a theory of replay as a form of planning.