Natural continual learning: success is a journey, not (just) a destination

Ta-Chu Kao[*@1], **Kristopher T. Jensen**[*1], **Alberto Bernacchia**[2], **and Guillaume Hennequin**[1]

[1] Computational and Biological Learning Lab, Department of Engineering, University of Cambridge, Cambridge, UK
[2] MediaTek Research, Cambridge, UK

[*] These authors contributed equally [@] Corresponding author (tck29@cam.ac.uk)

Abstract

Biological agents are known to learn many different tasks over the course of their lives, and to be able to revisit previous tasks and behaviors with little to no loss in performance. In contrast, artificial agents are prone to 'catastrophic forgetting' whereby performance on previous tasks deteriorates rapidly as new ones are acquired. This shortcoming has recently been addressed using methods that encourage parameters to stay close to those used for previous tasks. This can be done by (i) using specific parameter regularizers that map out suitable destinations in parameter space, or (ii) guiding the optimization journey by projecting gradients into subspaces that do not interfere with previous tasks. However, parameter regularization has been shown to be relatively ineffective in recurrent neural networks (RNNs), a setting relevant to the study of neural dynamics supporting biological continual learning. Similarly, projection based methods can reach capacity and fail to learn any further as the number of tasks increases. To address these limitations, we propose Natural Continual Learning (NCL), a new method that unifies weight regularization and projected gradient descent. NCL uses Bayesian weight regularization to encourage good performance on all tasks at convergence and combines this with gradient projections designed to prevent catastrophic forgetting during optimization. NCL formalizes gradient projection as a trust region algorithm based on the Fisher information metric, and achieves scalability via a novel Kronecker-factored approximation strategy. Our method outperforms both standard weight regularization techniques and projection based approaches when applied to continual learning problems in RNNs. The trained networks evolve task-specific dynamics that are strongly preserved as new tasks are learned, similar to experimental findings in biological circuits.

1 Introduction

Catastrophic forgetting is a common feature of 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 in recurrent neural networks (RNNs) which are important due to their practical and biological relevance. However, continual learning in RNNs has recently proven challenging for many existing algorithms (Ehret et al., 2020; Duncker et al., 2020). For these reasons, we focus our experiments on RNNs in the present work although our algorithm and theoretical considerations are applicable to any continual learning setting that can be formalized as a probabilistic model, whether the problem is supervised or unsupervised and the architecture recurrent or feedforward.

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 (Kirkpatrick et al., 2017; Huszár, 2017; Nguyen et al., 2017; Ritter et al., 2018). 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 (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 (Kaufman et al., 2014; Ames and Churchland, 2019; Failor et al., 2021; Jensen et al., 2021). While these methods have been found to perform well in many continual learning settings, they also suffer from various shortcomings. In particular, while Bayesian weight regularization provides a natural way to weigh 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 a new method for continual learning, natural continual learning (NCL), by combining Bayesian continual learning using weight regularization with 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 maxima of the Bayesian approximate posterior. We show that NCL outperforms previous continual learning algorithms, and that our principled Bayesian treatment is particularly important when task number and complexity increases or network size decreases. 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.

2 Method

Notations We use $\boldsymbol{X}^\top$, $\boldsymbol{X}^{-1}$, $\text{Tr}(\boldsymbol{X})$ and $\text{vec}(\boldsymbol{X})$ to denote the transpose, inverse, trace, and column-wise vectorization of a matrix $\boldsymbol{X}$. 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 $(\boldsymbol{X} \otimes \boldsymbol{Y})_{mi+k,mj+l} = \boldsymbol{X}_{ij}\boldsymbol{Y}_{kl}$. We use bold lower-case letters $\boldsymbol{x}$ to denote column vectors. We use 'OWM' to refer to orthogonal weight modification (Zeng et al., 2019) and 'DOWM' for 'doubly orthogonal weight modification' to refer to the method proposed by Duncker et al. (2020). $\mathcal{D}_k$ refers to a 'dataset' corresponding to task k which in this work generally consists of a set of input-output pairs $\{\boldsymbol{x}_k^{(i)}, \boldsymbol{y}_k^{(i)}\}$ such that $\ell_k(\boldsymbol{\theta}) := \log p(\mathcal{D}_k|\boldsymbol{\theta}) = \sum_i \log p_{\boldsymbol{\theta}}(\boldsymbol{y}_k^{(i)}|\boldsymbol{x}_k^{(i)})$ is the task-related performance on task k for a model with parameters $\boldsymbol{\theta}$. Finally, we use $\hat{\mathcal{D}}_k$ to refer to a dataset generated by inputs from the k^{th} task where $\{\hat{\boldsymbol{y}}_k^{(i)} \sim p_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{x}_k^{(i)})\}$ are drawn from the model distribution.

2.1 Bayesian continual learning

Problem statement In continual learning, we train a model on a set of K datasets $\{\mathcal{D}_1, \ldots, \mathcal{D}_K\}$ that arrive sequentially. The aim is to learn a probabilistic model $p(\mathcal{D}|\boldsymbol{\theta})$ 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" datasets (i.e., $\mathcal{D}_j$ for $j < k$) when learning task k. While we enforce this stringent condition in this paper, our approach may be easily combined with memory-based techniques such as coresets and generative replay which allow for storage of a subset of past data or an increase in model parameters with each task (Ehret et al., 2020; von Oswald et al., 2019; Nguyen et al., 2017; Pan et al., 2020).

Bayesian approach 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 $p(\boldsymbol{\theta})$ on the model parameters and compute the posterior after observing k datasets according to Bayes' rule:

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

where $\mathcal{D}_{1:k}$ is a concatenation of the first k datasets $(\mathcal{D}_1, \ldots, \mathcal{D}_k)$. In theory, it is thus possible to compute the exact posterior $p(\boldsymbol{\theta}|\mathcal{D}_{1:k})$ after k datasets, while only observing $\mathcal{D}_k$, by using the posterior $p(\boldsymbol{\theta}|\mathcal{D}_{1:k-1})$ 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 $p(\boldsymbol{\theta}|\mathcal{D}_{1:k-1})$ is approximated by a parametric distribution with parameters $\boldsymbol{\phi}_{k-1}$. The approximate posterior $q(\boldsymbol{\theta}; \boldsymbol{\phi}_{k-1})$ is then used as a prior for task k.

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

$$\boldsymbol{\mu}_k = \arg\max_{\boldsymbol{\theta}} \ \log p(\boldsymbol{\theta}|\mathcal{D}_k, \boldsymbol{\phi}_{k-1}) \tag{2}$$

$$= \arg\max_{\boldsymbol{\theta}} \ \log p(\mathcal{D}_k|\boldsymbol{\theta}) + \log q(\boldsymbol{\theta}|\boldsymbol{\phi}_{k-1}) \tag{3}$$

$$= \arg\max_{\boldsymbol{\theta}} \ \underbrace{\ell_k(\boldsymbol{\theta}) - \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\mu}_{k-1})^\top \boldsymbol{\Lambda}_{k-1}(\boldsymbol{\theta} - \boldsymbol{\mu}_{k-1})}_{:= \mathcal{L}_k(\boldsymbol{\theta})} \tag{4}$$

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

$$\boldsymbol{\Lambda}_k = -\nabla^2_{\boldsymbol{\theta}} \log p(\boldsymbol{\theta}|\mathcal{D}_k, \boldsymbol{\phi}_{k-1})\big|_{\boldsymbol{\theta}=\boldsymbol{\mu}_k} = H(\mathcal{D}_k, \boldsymbol{\mu}_k) + \boldsymbol{\Lambda}_{k-1}, \tag{5}$$

where $H(\mathcal{D}_k, \boldsymbol{\mu}_k) = -\nabla^2_{\boldsymbol{\theta}} \log p(\mathcal{D}_k|\boldsymbol{\theta})\big|_{\boldsymbol{\theta}=\boldsymbol{\mu}_k}$ 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 dataset $\mathcal{D}_k$. First, given $\mathcal{D}_k$ and the previous posterior $q(\boldsymbol{\theta}|\boldsymbol{\mu}_{k-1}, \boldsymbol{\Lambda}_{k-1}^{-1})$ (i.e. the new prior), $\boldsymbol{\mu}_k$ is found using gradient-based optimization (Equation 4). This step can be interpreted as optimizing the likelihood of $\mathcal{D}_k$ while penalizing changes in the parameters $\boldsymbol{\theta}$ 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 5.

Approximating the Hessian In practice, computing $\boldsymbol{\Lambda}_k$ presents two major difficulties. First, because $q(\boldsymbol{\theta}; \boldsymbol{\phi}_k)$ 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; Martens, 2014; Huszár, 2017; Ritter et al., 2018):

$$\boldsymbol{F}_k = \mathbb{E}_{p(\hat{\mathcal{D}}_k|\boldsymbol{\theta})}\left[\nabla_{\boldsymbol{\theta}} \log p(\hat{\mathcal{D}}_k|\boldsymbol{\theta}) \nabla_{\boldsymbol{\theta}} \log p(\hat{\mathcal{D}}_k|\boldsymbol{\theta})^\top\right]\Big|_{\boldsymbol{\theta}=\boldsymbol{\mu}_k} \approx H(\mathcal{D}_k, \boldsymbol{\mu}_k) \tag{6}$$

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

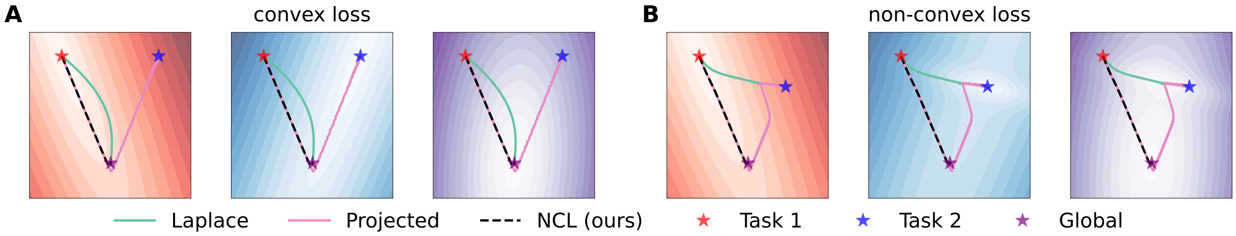

Figure 1: **Continual learning in a toy problem. (A)** Loss landscapes of task 1 (ℓ_1; left), task 2 (ℓ_2; middle) and the combined loss $\ell_{1+2} = \ell_1 + \ell_2$ (right). Stars indicate the global optima for ℓ_1 (red), ℓ_2 (blue), and ℓ_{1+2} (purple). We assume that $\boldsymbol{\theta}$ has been 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 J for further mathematical details.

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 3 as a comparison for our own Kronecker-factored method.

2.2 Natural continual learning

While the online Laplace approximation 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 (Van de Ven and Tolias, 2018; Duncker et al., 2020). 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). We illustrate the shortcomings of weight regularization on a simple continual regression problem in Figure 1, where gradient descent on the Laplace posterior produces an indirect optimization path along which the first task is transiently forgotten as the second task is being learned. In addition, this can lead to catastrophic forgetting when the loss is non-convex (Figure 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 (Zeng et al., 2019; Duncker et al., 2020). 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 1; pink), unless the learning rate is exactly zero in directions that interfere with previous tasks.

To address these shortcomings, 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 $\boldsymbol{\Lambda}$. 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 maximizing the posterior loss $\mathcal{L}_k(\boldsymbol{\theta})$ within a region of radius r centered around $\boldsymbol{\theta}$ with a distance metric of the form $d(\boldsymbol{\theta}, \boldsymbol{\theta} + \boldsymbol{\Delta}) = \sqrt{\boldsymbol{\Delta}^\top \boldsymbol{\Lambda}_{k-1} \boldsymbol{\Delta}/2}$ that takes into account the curvature of the prior via its precision matrix $\boldsymbol{\Lambda}_{k-1}$:

$$\boldsymbol{\Delta} = \arg\min_{\boldsymbol{\Delta}} \mathcal{L}_k(\boldsymbol{\theta}) + \nabla_{\boldsymbol{\theta}} \mathcal{L}_k(\boldsymbol{\theta})^\top \boldsymbol{\Delta} \quad \text{subject to} \quad \frac{1}{2} \boldsymbol{\Delta}^\top \boldsymbol{\Lambda}_{k-1} \boldsymbol{\Delta} \leq r^2, \tag{7}$$

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

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \gamma \left[\boldsymbol{\Lambda}_{k-1}^{-1} \nabla_{\boldsymbol{\theta}} \ell_k(\boldsymbol{\theta}) - (\boldsymbol{\theta} - \boldsymbol{\mu}_{k-1}) \right] \tag{8}$$

for a learning rate parameter γ (which is implicitly a function of r in Equation 8). To get some intuition for this learning rule, we note that $\boldsymbol{\Lambda}_{k-1}$ 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 $\boldsymbol{\theta}$ to stay close to $\boldsymbol{\mu}_{k-1}$, the optimal parameters for the previous task. As we illustrate in Figure 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 (Figure 1B).

Implementation In this work, we use a Kronecker-factored approximation to each Fisher matrix $\boldsymbol{F}_k$ in Equation 6 (Martens and Grosse, 2015; Ritter et al., 2018), although we note that the general NCL framework can be applied with other approximations to $\boldsymbol{F}_k$ such as the diagonal approximation of Kirkpatrick et al. (2017). A major challenge in implementing NCL lies in the computation of $\boldsymbol{\Lambda}_{k-1}^{-1}$, which is generally intractable for large models. 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 5). To address this challenge, we derived an efficient algorithm for making a Kronecker-factored approximation to $\boldsymbol{\Lambda}_k = \boldsymbol{F}_k + \boldsymbol{\Lambda}_{k-1} \approx \boldsymbol{L}_k \otimes \boldsymbol{R}_k$ when $\boldsymbol{\Lambda}_{k-1} = \boldsymbol{L}_{k-1} \otimes \boldsymbol{R}_{k-1}$ and $\boldsymbol{F}_k$ are also Kronecker products. This approximation minimizes the KL-divergence between $\mathcal{N}(\boldsymbol{\mu}_k, (\boldsymbol{L}_k \otimes \boldsymbol{R}_k)^{-1})$ and $\mathcal{N}(\boldsymbol{\mu}_k, (\boldsymbol{\Lambda}_{k-1} + \boldsymbol{F}_k)^{-1})$ (see Appendix C for details). The NCL algorithm is described in pseudocode in Appendix B together with additional implementation and computational details. Finally, while we have derived NCL with a Laplace approximation in this section for simplicity, it can similarly be applied in the variational continual learning framework of Nguyen et al. (2017) (Appendix I).

2.3 Related work

As discussed in Section 2.1, our method is derived from prior work that relies on Bayesian inference to perform weight regularization for continual learning (Kirkpatrick et al., 2017; Nguyen et al., 2017; Huszár, 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 (Appendix F).

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

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \gamma \boldsymbol{P}_L \nabla_{\boldsymbol{\theta}} \ell_k(\boldsymbol{\theta}) \boldsymbol{P}_R, \tag{9}$$

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 9 as

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \gamma (\boldsymbol{P}_R \otimes \boldsymbol{P}_L) \nabla_{\boldsymbol{\theta}} \ell_k(\boldsymbol{\theta}). \tag{10}$$

This resembles the NCL update rule in Equation 8 where we identify $\boldsymbol{P}_R \otimes \boldsymbol{P}_L$ with the approximate inverse prior precision matrix used for gradient preconditioning in NCL $\boldsymbol{\Lambda}_{k-1}^{-1} = \boldsymbol{L}_{k-1}^{-1} \otimes \boldsymbol{R}_{k-1}^{-1}$. Indeed, we note that for a Kronecker-structured approximation to $\boldsymbol{F}_k$, the matrix $\boldsymbol{L}_{k-1}$ is the empirical covariance matrix of the network activations experienced during all tasks up to $k-1$ (Martens and Grosse, 2015; Bernacchia et al., 2018, Appendix D) which is exactly the inverse of the projection matrix 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 right Kronecker factor $\boldsymbol{R}_{k-1}$ designed to approximate the posterior precision of previous tasks (Appendix D) while Duncker et al. (2020) use the covariance of recurrent inputs and Zeng et al. (2019) use the identity matrix $\boldsymbol{I}$. Notably, both of these choices of $\boldsymbol{R}_{k-1}$ still provide reasonable approximations to the prior Fisher matrix which can be used to motivate OWM and DOWM as projecting out steep directions of the prior (Appendix E).

(ii) NCL includes an additional regularization term $(\boldsymbol{\theta}-\boldsymbol{\mu}_{k-1})$ 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 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):

$$\mathcal{L}_k^{(\lambda)}(\boldsymbol{\theta}) = \log p(\mathcal{D}_k|\boldsymbol{\theta}) - \lambda(\boldsymbol{\theta}-\boldsymbol{\mu}_{k-1})^\top \boldsymbol{\Lambda}_{k-1}(\boldsymbol{\theta}-\boldsymbol{\mu}_{k-1}). \tag{11}$$

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 11. Interestingly, this Hessian is given by $\boldsymbol{H}_k = -H(\mathcal{D}_k, \boldsymbol{\theta}) - \lambda\boldsymbol{\Lambda}_{k-1}$, 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).

3 Experiments and results

3.1 NCL in recurrent neural networks

As discussed in Section 1, we consider experiments in recurrent neural networks (RNNs), a setting that has recently proven challenging for continual learning (Duncker et al., 2020; Ehret et al., 2020). Here, we briefly describe the network dynamics and how the NCL algorithm (Section 2.2) is implemented.

Network dynamics The dynamics of the RNN used in this work can be described by the following equations:

$$\boldsymbol{h}_t = \boldsymbol{A}\boldsymbol{r}_{t-1} + \boldsymbol{B}\boldsymbol{x}_t + \boldsymbol{\xi}_t = \boldsymbol{W}\boldsymbol{z}_t + \boldsymbol{\xi}_t \tag{12}$$

$$\boldsymbol{y}_t \sim p(\boldsymbol{y}_t|\boldsymbol{C}\boldsymbol{r}_t) \tag{13}$$

where we define $\boldsymbol{r}_t = \phi(\boldsymbol{h}_t)$, $\boldsymbol{z}_t = (\boldsymbol{r}_{t-1}^\top, \boldsymbol{x}_t^\top)^\top$, $\boldsymbol{W} = (\boldsymbol{A}^\top, \boldsymbol{B}^\top)^\top$, 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, and $\boldsymbol{y} \in \mathbb{R}^{n_{out}\times 1}$ are the network outputs. 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 $\boldsymbol{\theta} = (\boldsymbol{W}, \boldsymbol{C})$. 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

$$\ell(\boldsymbol{\theta}) = \log p_\theta(\{\boldsymbol{y}\}|\{\boldsymbol{x}\}, \{\boldsymbol{\xi}\}) = \log p(\{\boldsymbol{y}\}|\{\boldsymbol{C}\boldsymbol{r}\}), \tag{14}$$

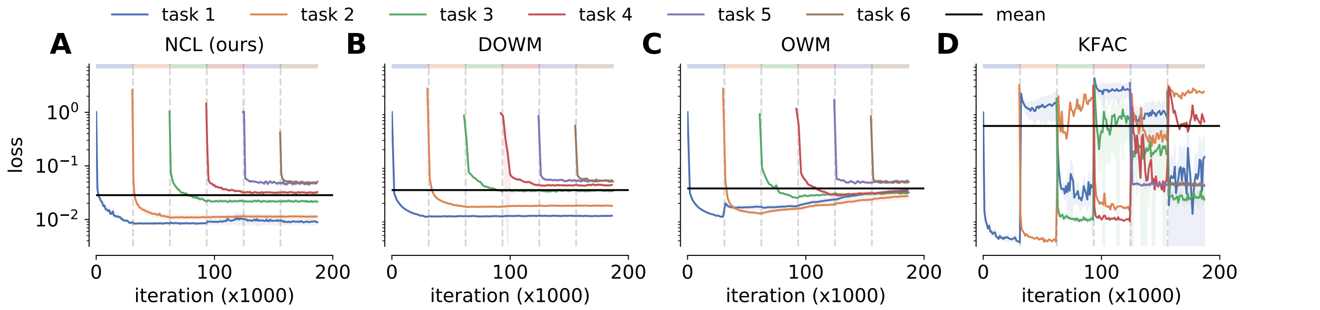

Figure 2: **Continual learning on stimulus-response tasks.** Evolution of the loss during training for each of the six stimulus-response tasks for NCL (A), DOWM (B), OWM (C), and KFAC with $\lambda = 1$ (D). Coloured lines and shadings indicate mean and stdev for each task across 5 random seeds. Black lines indicate the mean loss over all tasks at the end of training. Task order matches that of Duncker et al. (2020). Note that the earlier tasks are also 'easier' in a non-continual setting (Yang et al., 2019).

Kronecker-factored approximation to the FIMs We use a Kronecker-factored approximation to the Fisher information matrix of $\ell(\boldsymbol{\theta})$ with respect to the parameters $\boldsymbol{C}$ and $\boldsymbol{W}$ (see Appendix D or section 3.4 of Martens et al., 2018 for details). Defining $\overline{\boldsymbol{x}} := \partial \ell / \partial \text{vec}(\boldsymbol{X})$, we approximate the FIMs of $\boldsymbol{C}$ and $\boldsymbol{W}$ as:

$$\boldsymbol{F_C} = \mathbb{E}_{\{(\boldsymbol{\xi},\boldsymbol{x},\boldsymbol{y})\}\sim\mathcal{M}}\left[\overline{\boldsymbol{c}}\,\overline{\boldsymbol{c}}^\top\right] \approx \mathbb{E}\left[T\right]\mathbb{E}_{\{(\boldsymbol{\xi},\boldsymbol{x},\boldsymbol{y})\}\sim\mathcal{M}}\left[\boldsymbol{r}\boldsymbol{r}^\top\right] \otimes \mathbb{E}_{\{(\boldsymbol{\xi},\boldsymbol{x},\boldsymbol{y})\}\sim\mathcal{M}}\left[\overline{\boldsymbol{y}}\,\overline{\boldsymbol{y}}^\top\right] \tag{15}$$

$$\boldsymbol{F_W} = \mathbb{E}_{\{(\boldsymbol{\xi},\boldsymbol{x},\boldsymbol{y})\}\sim\mathcal{M}}\left[\overline{\boldsymbol{w}}\,\overline{\boldsymbol{w}}^\top\right] \approx \mathbb{E}\left[T\right]\mathbb{E}_{\{(\boldsymbol{\xi},\boldsymbol{x},\boldsymbol{y})\}\sim\mathcal{M}}\left[\boldsymbol{z}\boldsymbol{z}^\top\right] \otimes \mathbb{E}_{\{(\boldsymbol{\xi},\boldsymbol{x},\boldsymbol{y})\}\sim\mathcal{M}}\left[\overline{\boldsymbol{h}}\,\overline{\boldsymbol{h}}^\top\right], \tag{16}$$

where $\mathbb{E}\left[T\right]$ is the expected length of each input-output sequence, and we take $\{(\boldsymbol{\xi},\boldsymbol{x},\boldsymbol{y})\} \sim \mathcal{M}$ to mean that $\{\boldsymbol{\xi}\}$, $\{\boldsymbol{x}\}$ and $\{\boldsymbol{y}\}$ are drawn from the model distribution defined in Equation 12.

3.2 Stimulus-response tasks

In this section, we compare different methods for continual learning on a set of neuroscience inspired 'stimulus-response' (SR) tasks (Yang et al., 2019; details in Appendix G).

Following previous work, we first considered RNNs with 256 units (Yang et al., 2009; Duncker et al., 2020). While NCL, OWM and DOWM all managed to learn the six tasks without catastrophic forgetting (Figure 2A–C), we found that NCL achieved superior average performance across all tasks after training (Figure 3A). We then compared NCL, OWM, and DOWM to KFAC, an online Laplace algorithm that uses Adam (Kingma and Ba, 2014) to optimize the objective in Equation 4 with a Kronecker-factored approximation to the precision matrix in Equation 5 (Section 2.1; Ritter et al., 2018). Consistent with the results shown in Duncker et al. (2020), we found that NCL, OWM, and DOWM outperformed KFAC (Figure 2D; 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 4) and approximate the posterior precision matrix in the same way, but differ in the way they precondition the gradient of the objective. Our results thus demonstrate empirically that the choice of optimization algorithm is important to prevent forgetting, consistent with the intuition provided by Figure 1.

In previous work, poor performance with weight regularization approaches such as EWC and KFAC has been mitigated by introducing a hyperparameter λ that increases the importance of the prior term compared to a standard Bayesian treatment (Equation 11; Kirkpatrick et al., 2017; Ritter et al., 2018; Loo et al., 2020). We confirmed this here by performing a grid search over λ, which showed that KFAC with $\lambda \in [100, 1000]$ could perform almost as well as NCL (Section H.1; Figure 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 $\boldsymbol{\Lambda}_{k-1}$ (Section 2.3). In support of this hypothesis, we found that the KL divergence between the Adam preconditioner and the approximate prior precision $\boldsymbol{\Lambda}_{k-1}$ 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 (Section H.1).

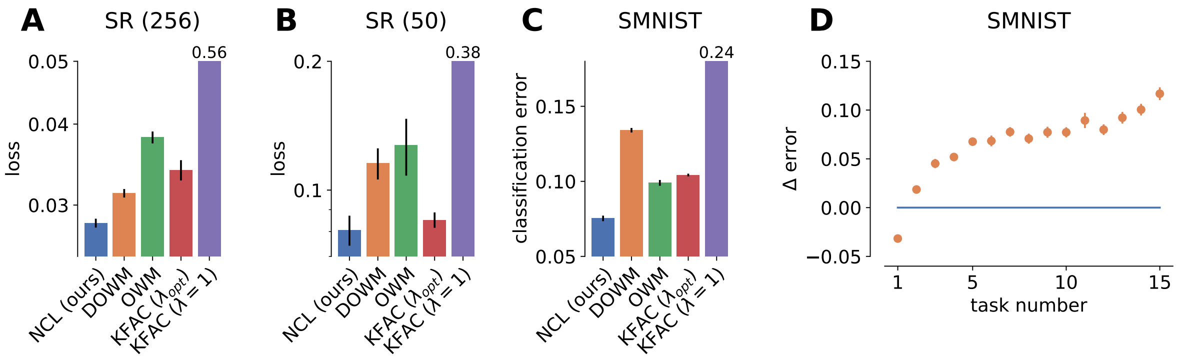

Figure 3: **Performance on SR and SMNIST tasks. (A)** Mean loss of NCL, DOWM, OWM, KFAC (optimal λ), and KFAC ($\lambda = 1$) across stimulus-response tasks for RNNs with 256 units. Here and in (B) and (C), KFAC with $\lambda = 1$ failed catastrophically and its performance is indicated in text as it does not fit on the axes. **(B)** As in (A), now for networks with 50 units. Error bars in (A) and (B) indicate standard error (s.e.m) across 5 random seeds. **(C)** Mean classification error across SMNIST tasks for networks with 30 hidden units. **(D)** Difference between the mean classification error of Laplace-DOWM and NCL as a function of task number. Error bars in (C) and (D) indicate s.e.m. across 100 random task permutations.

Duncker et al. have previously shown that the dynamics of RNNs trained to perform the SR tasks of Figure 2 are low dimensional, suggesting that 256 units may not be needed to learn the tasks in a continual fashion. We therefore reduced the network size to 50 units to further challenge the continual learning algorithms. We found that NCL outperformed both DOWM and OWM, suggesting that a good approximation to the Fisher matrix is particularly important in more challenging settings (Figure 3B; Section H.2).

3.3 Stroke MNIST

Another way to challenge the CL 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 inputs to the network. Similar to Ehret et al. (2020), we constructed a continual learning problem by considering consecutive binary classification tasks inspired by the canonical 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 G for details).

As for the SR task set in Section 3.2, we found that NCL outperformed previous projection based methods (Figure 3C). 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 3C). To investigate how the difference in performance between NCL and DOWM was affected by their different approximations to the Fisher matrix (Appendix E), 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 3D). 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 E). 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.

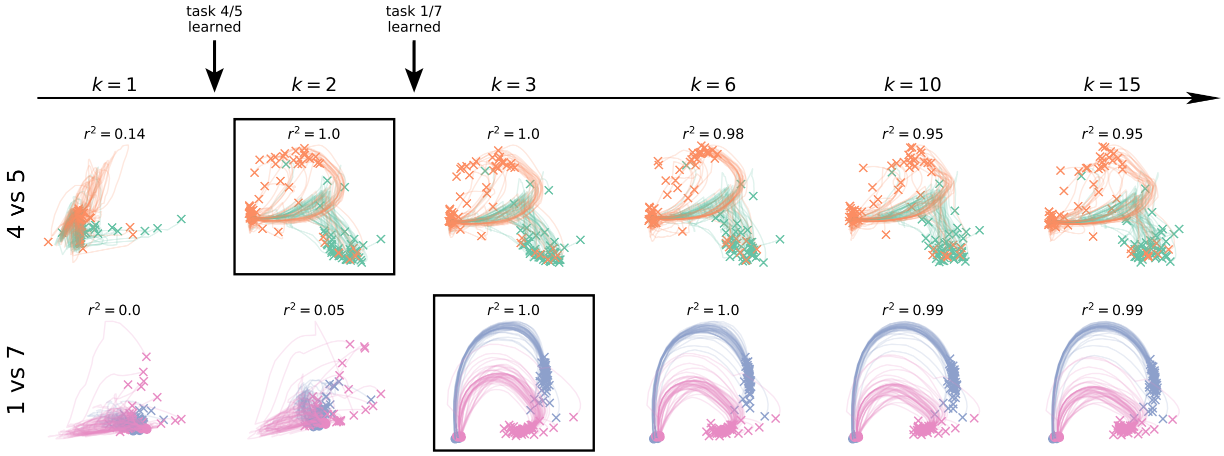

Figure 4: **Latent dynamics during SMNIST.** We considered two example tasks, 4 vs 5 (top) and 1 vs 7 (bottom). For each task, we simulated the response of a network trained by NCL to 100 digits drawn from that task distribution at different times during learning. We then fitted a factor analysis model 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).

3.4 Dissecting the dynamics of networks trained on the SMNIST task set

To further investigate how the 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 (see Section H.4 for an equivalent analysis with DOWM). 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 (Yu et al., 2009; Gallego et al., 2020; Jensen et al., 2020; Mante et al., 2013; Jensen et al., 2021). 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).

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). 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 contention in biological circuits (Clopath et al., 2017; Lütcke et al., 2013). 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 (Peters et al., 2014; Katlowitz et al., 2018; Dhawale et al., 2017; Chestek et al., 2007; Ganguly and Carmena, 2009).

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 in a recurrent neural network setting which has previously been shown to be challenging for various continual learning algorithms (Duncker et al., 2020; Ehret et al., 2020). 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 networks 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). 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 use NCL with other weight regularization approaches such as EWC (Kirkpatrick et al., 2017), and to extend its use to feedforward neural networks. Finally, a separate branch of continual learning utilizes generative replay or functional regularization based on previous data and models (Van de Ven and Tolias, 2018; Pan et al., 2020; Li and Hoiem, 2017; Shin et al., 2017). While our work has focused on weight regularization, such regularization and replay is 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).

Acknowledgements

We are grateful to Gido van de Ven, Siddharth Swaroop, Lea Duncker, Laura Driscoll, Naama Kadmon Harpaz, and Yashar Ahmadian for insightful discussions. We thank Gido van de Ven, Siddharth Swaroop, and Robert Pinsler for useful comments on the manuscript.

References

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.

Bernacchia, A., Lengyel, M., and Hennequin, G. (2018). Exact natural gradient in deep linear networks and its application to the nonlinear case. *Advances in Neural Information Processing Systems*, 31:5941–5950.

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.

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.

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

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.

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

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*.

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

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.

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

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

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

Jensen, K. T., Kao, T.-C., Stone, J. T., and Hennequin, G. (2021). Scalable Bayesian GPFA with automatic relevance determination and discrete noise models. *bioRxiv*.

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.

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

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.

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

Li, Z. and Hoiem, D. (2017). Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):2935–2947.

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

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.

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.

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

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*.

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*.

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

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

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

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

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*.

von Oswald, J., Henning, C., Sacramento, J., and Grewe, B. F. (2019). Continual learning with hypernetworks. *arXiv preprint arXiv:1906.00695*.

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.

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.

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.

Appendix – Natural continual learning

A Derivation of the NCL learning rule

In this section, we provide further details of how the NCL learning rule in Section 2.2 is derived and also provide an alternative derivation of the algorithm.

NCL learning rule As discussed in Section 2.2, we derive NCL as the solution of a trust region optimization problem. That is, we maximize the posterior loss $\mathcal{L}_k(\boldsymbol{\theta})$ within a region of radius r centered around $\boldsymbol{\theta}$ with a distance metric of the form $d(\boldsymbol{\theta}, \boldsymbol{\theta}+\boldsymbol{\Delta}) = \sqrt{\boldsymbol{\Delta}^\top \boldsymbol{\Lambda}_{k-1}\boldsymbol{\Delta}/2}$. 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

$$\boldsymbol{\Delta} = \arg\min_{\boldsymbol{\Delta}} \mathcal{L}_k(\boldsymbol{\theta}) + \nabla_{\boldsymbol{\theta}}\mathcal{L}_k(\boldsymbol{\theta})^\top \boldsymbol{\Delta} \quad \text{subject to} \quad \frac{1}{2}\boldsymbol{\Delta}^\top \boldsymbol{\Lambda}_{k-1}\boldsymbol{\Delta} \leq r^2, \tag{17}$$

where $\mathcal{L}_k(\boldsymbol{\theta}+\boldsymbol{\Delta}) \approx \mathcal{L}_k(\boldsymbol{\theta}) + \nabla_{\boldsymbol{\theta}}\mathcal{L}_k(\boldsymbol{\theta})^\top \boldsymbol{\Delta}$ is a first-order approximation to the updated Laplace objective. Here we recall from Equation 4 that

$$\mathcal{L}_k(\boldsymbol{\theta}) = \ell_k(\boldsymbol{\theta}) - \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\mu}_{k-1})^T \boldsymbol{\Lambda}_{k-1}(\boldsymbol{\theta} - \boldsymbol{\mu}_{k-1}) \tag{18}$$

from which we get

$$\nabla_{\boldsymbol{\theta}}\mathcal{L}_k(\boldsymbol{\theta})^\top \boldsymbol{\Delta} = \nabla_{\boldsymbol{\theta}}\ell_k(\boldsymbol{\theta})^\top \boldsymbol{\Delta} - (\boldsymbol{\theta} - \boldsymbol{\mu}_{k-1})^\top \boldsymbol{\Lambda}_{k-1}\boldsymbol{\Delta} \tag{19}$$

The optimization in Equation 17 is carried out by introducing a Lagrange multiplier η to construct a Lagrangian $\tilde{\mathcal{L}}$:

$$\tilde{\mathcal{L}}(\boldsymbol{\Delta}, \eta) = \mathcal{L}_k(\boldsymbol{\theta}) + \nabla_{\boldsymbol{\theta}}\ell_k(\boldsymbol{\theta})^\top \boldsymbol{\Delta} - (\boldsymbol{\theta} - \boldsymbol{\mu}_{k-1})^\top \boldsymbol{\Lambda}_{k-1}\boldsymbol{\Delta} + \eta(r^2 - \frac{1}{2}\boldsymbol{\Delta}^\top \boldsymbol{\Lambda}_{k-1}\boldsymbol{\Delta}). \tag{20}$$

We then take the derivative of $\tilde{\mathcal{L}}$ w.r.t. $\boldsymbol{\Delta}$ and set it to zero:

$$\nabla_{\boldsymbol{\Delta}}\tilde{\mathcal{L}}(\boldsymbol{\Delta}, \eta) = \nabla_{\boldsymbol{\theta}}\ell_k(\boldsymbol{\theta}) - \boldsymbol{\Lambda}_{k-1}(\boldsymbol{\theta} - \boldsymbol{\mu}_{k-1}) - \eta\boldsymbol{\Lambda}_{k-1}\boldsymbol{\Delta}' = 0. \tag{21}$$

Rearranging this equation gives

$$\boldsymbol{\Delta} = \frac{1}{\eta}\left[\boldsymbol{\Lambda}_{k-1}^{-1}\nabla_{\boldsymbol{\theta}}\ell_k(\boldsymbol{\theta}) - (\boldsymbol{\theta} - \boldsymbol{\mu}_{k-1}),\right]. \tag{22}$$

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:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \gamma\left[\boldsymbol{\Lambda}_{k-1}^{-1}\nabla_{\boldsymbol{\theta}}\ell_k(\boldsymbol{\theta}) - (\boldsymbol{\theta} - \boldsymbol{\mu}_{k-1})\right]. \tag{23}$$

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 $\mathcal{L}_k(\boldsymbol{\theta})$ 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:

$$\boldsymbol{\Delta} = \arg\min_{\boldsymbol{\Delta}} \mathcal{L}_k(\boldsymbol{\theta}) + \nabla_{\boldsymbol{\theta}}\mathcal{L}_k(\boldsymbol{\theta})^\top \boldsymbol{\Delta} \quad \text{such that} \quad \mathcal{C}(\boldsymbol{\Delta}) \leq r^2. \tag{24}$$

Here we define $\mathcal{C}(\boldsymbol{\Delta})$ as the approximate change in log probability under the prior

$$\mathcal{C}(\boldsymbol{\Delta}) = (\boldsymbol{\theta} + \boldsymbol{\Delta} - \boldsymbol{\mu}_{k-1})^\top \boldsymbol{\Lambda}_{k-1}(\boldsymbol{\theta} + \boldsymbol{\Delta} - \boldsymbol{\mu}_{k-1}) - (\boldsymbol{\theta} - \boldsymbol{\mu}_{k-1})^\top \boldsymbol{\Lambda}_{k-1}(\boldsymbol{\theta} - \boldsymbol{\mu}_{k-1}). \tag{25}$$

Following a similar derivation to above, we find the solution to this optimization problem as

$$\eta\boldsymbol{\Delta} = \boldsymbol{\Lambda}_{k-1}^{-1}\nabla_{\boldsymbol{\theta}}\mathcal{L}_k(\boldsymbol{\theta}) - \eta(\boldsymbol{\theta} - \boldsymbol{\mu}_{k-1}) = \boldsymbol{\Lambda}_{k-1}^{-1}\nabla_{\boldsymbol{\theta}}\ell_k(\boldsymbol{\theta}) - (1+\eta)(\boldsymbol{\theta} - \boldsymbol{\mu}_{k-1}) \tag{26}$$

for some Lagrange multiplier η. This gives rise to the update rule

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \gamma\left[\boldsymbol{\Lambda}_{k-1}^{-1}\nabla_{\boldsymbol{\theta}}\ell_k(\boldsymbol{\theta}) - \lambda(\boldsymbol{\theta} - \boldsymbol{\mu}_{k-1})\right] \tag{27}$$

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 2.2 with the choice of $\lambda = 1$. In practice, λ can also be treated as a hyperparameter to be optimized (Section H.1).

B Implementation

In this section we discuss various implementation details for NCL and provide an overview of the algorithm in the form of pseudocode (Algorithm 1). In particular, we use momentum $\rho = 0.9$ in all our experiments involving NCL as is done in Duncker et al. (2020). We found that the use of momentum greatly speeds up convergence in practice.

Algorithm 1: Natural continual learning with momentum

```
input: $\{\mathcal{D}_k\}_{k=1}^K$, $\alpha$, $p_w$ (prior), $n_b$ (batch size), $\gamma$ (learning rate), $\boldsymbol{W}_0$, $\boldsymbol{C}_0$, $\rho$
initialize: $L_{\boldsymbol{W}} \leftarrow p_w\boldsymbol{I}$, $R_{\boldsymbol{W}} \leftarrow p_w\boldsymbol{I}$, $L_{\boldsymbol{C}} \leftarrow p_w\boldsymbol{I}$, $R_{\boldsymbol{C}} \leftarrow p_w\boldsymbol{I}$
initialize: $\boldsymbol{W}_1 \leftarrow \boldsymbol{W}_0$, $\boldsymbol{C}_1 \leftarrow \boldsymbol{C}_0$
initialize: $\boldsymbol{M}_{\boldsymbol{W}} \leftarrow$ zeros_like($\boldsymbol{W}_0$), $\boldsymbol{M}_{\boldsymbol{C}} \leftarrow$ zeros_like($\boldsymbol{C}_0$)        // Gradient momentum
for $k = 1 \ldots K$ do
    while not converged do
        $\{\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}\}_{i=1}^{n_b} \sim \mathcal{D}_k$        // Input and target output
        for $i = 1, \ldots, n_b$ do
            $\hat{\boldsymbol{y}}^{(i)} = RNN(\boldsymbol{x}^{(i)}, \boldsymbol{W}_k, \boldsymbol{C}_k)$        // Empirical output
        $\ell = \frac{1}{n_b}\sum_i^{n_b}\log p(\boldsymbol{y}_t^{(i)}|\hat{\boldsymbol{y}}_t^{(i)})$        // Loss

        % Build up momentum
        $\boldsymbol{M}_{\boldsymbol{W}} \leftarrow \rho\boldsymbol{M}_{\boldsymbol{W}} + \nabla_{\boldsymbol{W}}\ell + R_{\boldsymbol{W}}(\boldsymbol{W}_k - \boldsymbol{W}_{k-1})L_{\boldsymbol{W}}$
        $\boldsymbol{M}_{\boldsymbol{C}} \leftarrow \rho\boldsymbol{M}_{\boldsymbol{C}} + \nabla_{\boldsymbol{C}}\ell + R_{\boldsymbol{C}}(\boldsymbol{C}_k - \boldsymbol{C}_{k-1})L_{\boldsymbol{C}}$

        % Update model parameters
        $\boldsymbol{W}_k \leftarrow \boldsymbol{W}_k - \gamma p_w^2\ \left[(R_{\boldsymbol{W}} + \alpha\boldsymbol{I})^{-1}\ \boldsymbol{M}_{\boldsymbol{W}}(L_{\boldsymbol{W}} + \alpha\boldsymbol{I})^{-1}\right]$
        $\boldsymbol{C}_k \leftarrow \boldsymbol{C}_k - \gamma p_w^2\ \left[(R_{\boldsymbol{C}} + \alpha\boldsymbol{I})^{-1}\ \boldsymbol{M}_{\boldsymbol{C}}(L_{\boldsymbol{C}} + \alpha\boldsymbol{I})^{-1}\right]$
    % Update Fisher matrix components
    compute $\mathbb{E}\left[\bar{\boldsymbol{h}}\,\bar{\boldsymbol{h}}^\top\right]$ and $\mathbb{E}\left[\bar{\boldsymbol{y}}\,\bar{\boldsymbol{y}}^\top\right]$ using Algorithm 2
    $L_{\boldsymbol{W}}, R_{\boldsymbol{W}} \leftarrow$ nearest_kf($L_{\boldsymbol{W}} \otimes R_{\boldsymbol{W}} + \mathbb{E}[T]\,\mathbb{E}\left[\boldsymbol{z}\boldsymbol{z}^\top\right] \otimes \mathbb{E}\left[\bar{\boldsymbol{h}}\,\bar{\boldsymbol{h}}^\top\right]$)        // Appendix C
    $L_{\boldsymbol{C}}, R_{\boldsymbol{C}} \leftarrow$ nearest_kf($L_{\boldsymbol{C}} \otimes R_{\boldsymbol{C}} + \mathbb{E}[T]\,\mathbb{E}\left[\boldsymbol{r}\boldsymbol{r}^\top\right] \otimes \mathbb{E}\left[\bar{\boldsymbol{y}}\,\bar{\boldsymbol{y}}^\top\right]$)
```

For NCL, we set the prior over the parameters $\boldsymbol{W}$ and $\boldsymbol{C}$ when learning the first task as $p(\text{vec}(\boldsymbol{W})) = \mathcal{N}(\boldsymbol{0}; p_w^{-2}\boldsymbol{I})$ and $p(\text{vec}(\boldsymbol{C})) = \mathcal{N}(\boldsymbol{0}; p_w^{-2}\boldsymbol{I})$ respectively. In particular, 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, $p_w^{-2} = 10^6$ for the stimulus-response task and $p_w^{-2} = 6000$ for SMNIST).

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 0.001 for KFAC with Adam. 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 Section H.3.

Algorithm 2: Estimating $\mathbb{E}\left[\overline{\boldsymbol{h}}\,\overline{\boldsymbol{h}}^\top\right], \mathbb{E}\left[\overline{\boldsymbol{y}}\,\overline{\boldsymbol{y}}^\top\right]$

```
input: $\{\{(\boldsymbol{x}_t, \boldsymbol{\xi}_t)\}_{t=1}^{T_i}\}_{i=1}^{n_\mathrm{b}}$, $\boldsymbol{W}$, $\boldsymbol{C}$ , $n, m$

initialize:
$\hat{\boldsymbol{h}}_t^{(i)} \leftarrow \mathrm{zeros}(n, 1)\ \forall t, k$
$\hat{\boldsymbol{y}}_t^{(i)} \leftarrow \mathrm{zeros}(m, 1)\ \forall t, k$

% Sample targets from the model
for $i = 1 \ldots n_b$ do
    for $t = 1 \ldots T_i$ do
        $\boldsymbol{h}_t = \boldsymbol{W}[\boldsymbol{r}_{t-1}; \boldsymbol{x}_t^{(i)}]$
        $\boldsymbol{r}_t = \phi(\boldsymbol{h}_t + \boldsymbol{\xi}_t^{(i)})$
        $\boldsymbol{y}_t^{(i)} \sim p(\boldsymbol{y}_t | \boldsymbol{C}\boldsymbol{r}_t)$          // sample from observation model
% Compute adjoints
for $i = 1 \ldots n_b$ do
    for $t = 1 \ldots T_i$ do
        $\boldsymbol{h}_t = \boldsymbol{W}[\boldsymbol{r}_{t-1}; \boldsymbol{x}_t^{(i)}] + \hat{\boldsymbol{h}}_t^{(i)}$
        $\boldsymbol{r}_t = \phi(\boldsymbol{h}_t + \boldsymbol{\xi}_t^{(i)})$
        $\ell_t^{(i)} = \log p(\boldsymbol{y}_t^{(i)} | \boldsymbol{C}\boldsymbol{r}_t + \hat{\boldsymbol{y}}_t^{(i)})$
$\ell = \sum_{i=1}^{n_\mathrm{b}} \sum_{t=1}^{T_i} \ell_t^{(i)} / n_\mathrm{b}$
% Compute adjoints of $\hat{\boldsymbol{h}}_t^{(i)}$ and $\hat{\boldsymbol{y}}_t^{(i)}$ with respect to $\ell$ via automatic differentiation
$\overline{\boldsymbol{h}}_t^{(i)} \leftarrow$ adjoint of $\hat{\boldsymbol{h}}_t^{(i)}$,   $\overline{\boldsymbol{y}}_t^{(i)} \leftarrow$ adjoint of $\hat{\boldsymbol{y}}_t^{(i)}$
$\mathbb{E}\left[\overline{\boldsymbol{h}}\,\overline{\boldsymbol{h}}^\top\right] \approx \sum_{i=1}^{n_\mathrm{b}} \sum_{t=1}^{T_i} \overline{\boldsymbol{h}}_t^{(i)} (\overline{\boldsymbol{h}}_t^{(i)})^\top / (n_\mathrm{b}\, \mathbb{E}[T])$
$\mathbb{E}\left[\overline{\boldsymbol{y}}\,\overline{\boldsymbol{y}}^\top\right] \approx \sum_{i=1}^{n_\mathrm{b}} \sum_{t=1}^{T_i} \overline{\boldsymbol{y}}_t^{(i)} (\overline{\boldsymbol{y}}_t^{(i)})^\top / (n_\mathrm{b}\, \mathbb{E}[T])$
```

C 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:

$$\boldsymbol{X} \otimes \boldsymbol{Y} \approx \boldsymbol{Z} = \boldsymbol{A} \otimes \boldsymbol{B} + \boldsymbol{C} \otimes \boldsymbol{D}. \tag{28}$$

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

$$\boldsymbol{Z} \approx (\boldsymbol{A} + \pi \boldsymbol{C}) \otimes (\boldsymbol{B} + \frac{1}{\pi}\boldsymbol{D}), \tag{29}$$

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

$$\begin{aligned} &\|\boldsymbol{Z} - (\boldsymbol{A} + \pi \boldsymbol{C}) \otimes (\boldsymbol{B} + \frac{1}{\pi}\boldsymbol{D})\| && (30)\\ &= \|\frac{1}{\pi}\boldsymbol{A} \otimes \boldsymbol{D} + \pi \boldsymbol{C} \otimes \boldsymbol{B}\| && (31)\\ &\leq \frac{1}{\pi}\|\boldsymbol{A} \otimes \boldsymbol{D}\| + \pi\|\boldsymbol{C} \otimes \boldsymbol{B}\| && (32)\end{aligned}$$

for any norm $\| \cdot \|$. They then minimize this upper-bound with respect to π to find the optimal π:

$$\pi = \sqrt{\frac{\|\boldsymbol{C} \otimes \boldsymbol{B}\|}{\|\boldsymbol{A} \otimes \boldsymbol{D}\|}}. \tag{33}$$

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:

$$\pi = \sqrt{\frac{\mathrm{Tr}(\boldsymbol{B})\mathrm{Tr}(\boldsymbol{C})}{\mathrm{Tr}(\boldsymbol{A})\mathrm{Tr}(\boldsymbol{D})}}. \tag{34}$$

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:

$$\begin{aligned} \boldsymbol{X}, \boldsymbol{Y} &= \arg\min_{\boldsymbol{X},\boldsymbol{Y}} \|\boldsymbol{Z} - \boldsymbol{X} \otimes \boldsymbol{Y}\|_F^2 && (35)\\ &= \arg\min_{\boldsymbol{X},\boldsymbol{Y}} \|\mathcal{R}(\boldsymbol{A} \otimes \boldsymbol{B}) + \mathcal{R}(\boldsymbol{C} \otimes \boldsymbol{D}) - \mathcal{R}(\boldsymbol{X} \otimes \boldsymbol{Y})\|_F^2 && (36)\\ &= \arg\min_{\boldsymbol{X},\boldsymbol{Y}} \|\mathrm{vec}(\boldsymbol{A})\mathrm{vec}(\boldsymbol{B})^\top + \mathrm{vec}(\boldsymbol{C})\mathrm{vec}(\boldsymbol{D})^\top - \mathrm{vec}(\boldsymbol{X})\mathrm{vec}(\boldsymbol{Y})^\top\|_F^2, && (37)\end{aligned}$$

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

```
input: $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{C}$, $\boldsymbol{D}$
$\boldsymbol{a} \leftarrow \mathrm{vec}(\boldsymbol{A})$, $\boldsymbol{b} \leftarrow \mathrm{vec}(\boldsymbol{B})$, $\boldsymbol{c} \leftarrow \mathrm{vec}(\boldsymbol{C})$, $\boldsymbol{d} \leftarrow \mathrm{vec}(\boldsymbol{D})$          // Vectorize $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{D}$
$\boldsymbol{Q}, \_ \leftarrow \mathrm{QR}(\left[\boldsymbol{a}; \boldsymbol{c}\right])$          // Orthogonal basis for $\boldsymbol{a}$ and $\boldsymbol{c}$ in $\mathbb{R}^{n^2 \times 2}$
$\boldsymbol{H} \leftarrow (\boldsymbol{Q}^\top \boldsymbol{a})\boldsymbol{b}^\top + (\boldsymbol{Q}^\top \boldsymbol{c})\boldsymbol{d}^\top$
$\boldsymbol{U}, \boldsymbol{s}, \boldsymbol{V}^\top \leftarrow \mathrm{SVD}(\boldsymbol{H})$
$\boldsymbol{y} \leftarrow$ first column of $\sqrt{\boldsymbol{s}_1}\boldsymbol{V}$
$\boldsymbol{x} \leftarrow$ first column of $\sqrt{\boldsymbol{s}_1}\boldsymbol{Q}\boldsymbol{U}$
$\boldsymbol{X} \leftarrow \mathrm{reshape}(\boldsymbol{x}, (n, n))$, $\boldsymbol{Y} \leftarrow \mathrm{reshape}(\boldsymbol{y}, (m, m))$
```

Minimal KL-divergence In this paper, 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

$$\begin{aligned}
2D_{\mathrm{KL}}(q||p) &= \log|\boldsymbol{X} \otimes \boldsymbol{Y}| - \log|\boldsymbol{Z}| + \mathrm{Tr}(\boldsymbol{Z}(\boldsymbol{X} \otimes \boldsymbol{Y})^{-1}) - d && (38)\\
&= m\log|\boldsymbol{X}| + n\log|\boldsymbol{Y}| + \mathrm{Tr}(\boldsymbol{A}\boldsymbol{X}^{-1} \otimes \boldsymbol{B}\boldsymbol{Y}^{-1}) + \mathrm{Tr}(\boldsymbol{C}\boldsymbol{X}^{-1} \otimes \boldsymbol{D}\boldsymbol{Y}^{-1}) - d && (39)\\
&= -m\log|\boldsymbol{X}^{-1}| - n\log|\boldsymbol{Y}^{-1}| + \mathrm{Tr}(\boldsymbol{A}\boldsymbol{X}^{-1})\mathrm{Tr}(\boldsymbol{B}\boldsymbol{Y}^{-1}) && (40)\\
&\qquad + \mathrm{Tr}(\boldsymbol{C}\boldsymbol{X}^{-1})\mathrm{Tr}(\boldsymbol{D}\boldsymbol{Y}^{-1}) - d && (41)
\end{aligned}$$

where $d = nm$. Differentiating with respect to $\boldsymbol{X}^{-1}$, and $\boldsymbol{Y}^{-1}$ and setting the result to zero, we get

$$0 = \frac{\partial D_{\mathrm{KL}}(q||p)}{\partial \boldsymbol{X}^{-1}} = \frac{1}{2}\left[-m\boldsymbol{X} + \mathrm{Tr}(\boldsymbol{B}\boldsymbol{Y}^{-1})\boldsymbol{A} + \mathrm{Tr}(\boldsymbol{D}\boldsymbol{Y}^{-1})\boldsymbol{C}\right] \tag{42}$$

$$0 = \frac{\partial D_{\mathrm{KL}}(q||p)}{\partial \boldsymbol{Y}^{-1}} = \frac{1}{2}\left[-n\boldsymbol{Y} + \mathrm{Tr}(\boldsymbol{A}\boldsymbol{X}^{-1})\boldsymbol{B} + \mathrm{Tr}(\boldsymbol{C}\boldsymbol{X}^{-1})\boldsymbol{D}\right]. \tag{43}$$

Rearranging these equations, we find the self-consistency equations:

$$\boldsymbol{X} = \frac{1}{m}\left[\mathrm{Tr}(\boldsymbol{B}\boldsymbol{Y}^{-1})\boldsymbol{A} + \mathrm{Tr}(\boldsymbol{D}\boldsymbol{Y}^{-1})\boldsymbol{C}\right] \tag{44}$$

$$\boldsymbol{Y} = \frac{1}{n}\left[\mathrm{Tr}(\boldsymbol{A}\boldsymbol{X}^{-1})\boldsymbol{B} + \mathrm{Tr}(\boldsymbol{D}\boldsymbol{X}^{-1})\boldsymbol{D}\right]. \tag{45}$$

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 44 and Equation 45. However, we can find $\boldsymbol{X}$ and $\boldsymbol{Y}$ numerically by iteratively applying the following update rules:

$$\boldsymbol{X}_{k+1} = (1-\beta)\boldsymbol{X}_k + \frac{\beta}{m}\left(\mathrm{Tr}(\boldsymbol{B}\boldsymbol{Y}_k^{-1})\boldsymbol{A} + \mathrm{Tr}(\boldsymbol{D}\boldsymbol{Y}_k^{-1})\boldsymbol{C}\right) \tag{46}$$

$$\boldsymbol{Y}_{k+1} = (1-\beta)\boldsymbol{Y}_k + \frac{\beta}{n}\left(\mathrm{Tr}(\boldsymbol{A}\boldsymbol{X}_k^{-1})\boldsymbol{C} + \mathrm{Tr}(\boldsymbol{C}\boldsymbol{X}_k^{-1})\boldsymbol{D}\right) \tag{47}$$

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 5, top row), the KL divergence from the true sum (Figure 5, middle row), and the scale-optimized KL divergence from the true sum (Figure 5, bottom row). Here we define the scale-optimized KL divergence as

$$\begin{aligned}
\mathrm{KL}_\lambda[\boldsymbol{\Lambda}_1||\boldsymbol{\Lambda}_2] &= \min_\lambda \mathrm{KL}[\lambda\boldsymbol{\Lambda}_1||\boldsymbol{\Lambda}_2] && (48)\\
&= \frac{1}{2}\left(\log\frac{|\boldsymbol{\Lambda}_1|}{|\boldsymbol{\Lambda}_2|} + d\log\frac{\mathrm{Tr}(\boldsymbol{\Lambda}_1^{-1}\boldsymbol{\Lambda}_2)}{d}\right), && (49)
\end{aligned}$$

where d is the dimensionality of the precision matrices $\boldsymbol{\Lambda}_1$ and $\boldsymbol{\Lambda}_2$ and we take $\mathrm{KL}[\boldsymbol{\Lambda}_1, \boldsymbol{\Lambda}_2] = D_{\mathrm{KL}}(\mathcal{N}(\boldsymbol{0}, \boldsymbol{\Lambda}_1^{-1})||\mathcal{N}(\boldsymbol{0}, \boldsymbol{\Lambda}_2^{-1}))$. 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

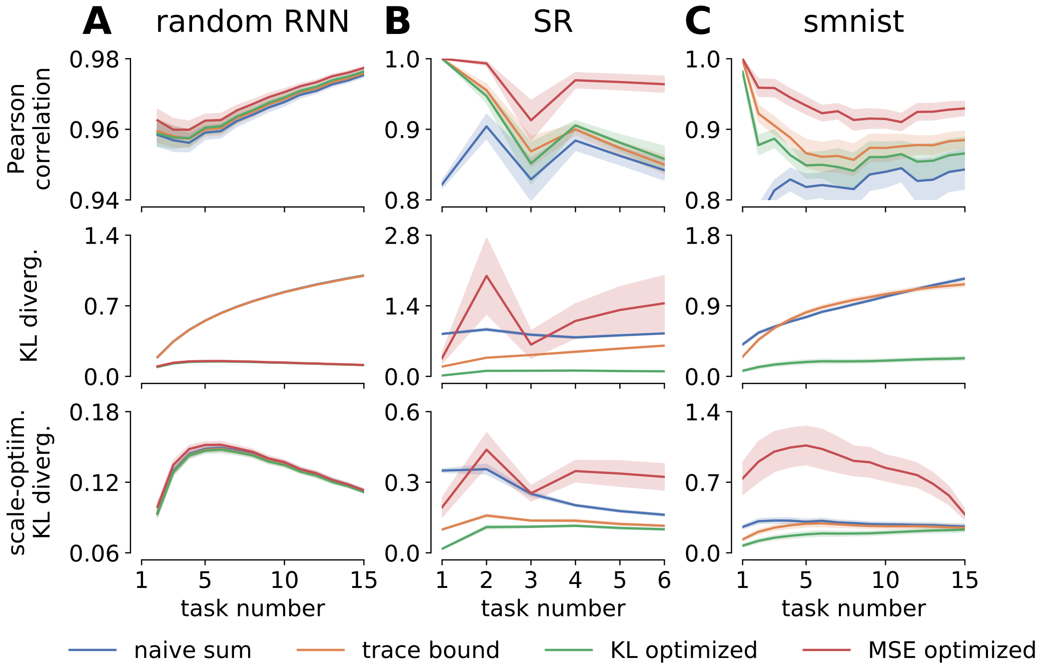

Figure 5: **Comparison of different Kronecker approximations to consecutive sums of two Kronecker products.** **(A)** Comparison of approximations for Fisher matrices computed from random RNNs with dynamics as described in Section 3.1. **(B)** As in (A) for the Fisher matrices from the stimulus-response tasks with 50 hidden units. **(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.

to have a slightly better correlation and slightly worse scaled KL (Figure 5, 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 5, green).

D KFAC approximation to the Fisher matrix in recurrent neural networks

Recall from Section 3.1 that the dynamics of the RNN are given by the equations:

$$\boldsymbol{h}_t = \boldsymbol{A}\boldsymbol{r}_{t-1} + \boldsymbol{B}\boldsymbol{x}_t + \boldsymbol{\xi}_t = \boldsymbol{W}\boldsymbol{z}_t + \boldsymbol{\xi}_t \tag{50}$$

$$\boldsymbol{r}_t = \phi(\boldsymbol{h}_t) \tag{51}$$

$$\boldsymbol{y}_t \sim p(\boldsymbol{y}_t|\boldsymbol{C}\boldsymbol{r}_t) \tag{52}$$

where $\boldsymbol{z}_t = (\boldsymbol{r}_{t-1}^\top, \boldsymbol{x}_t^\top)^\top$ and $\boldsymbol{W} = (\boldsymbol{A}^\top, \boldsymbol{B}^\top)^\top$. The nonlinearity $\phi(\boldsymbol{h})$ is applied to the hidden activations $\boldsymbol{h}$ element-wise. 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

$$\ell(\boldsymbol{W}, \boldsymbol{C}) = \sum_{t=1}^{T} \log p(\boldsymbol{y}_t|\boldsymbol{C}\boldsymbol{r}_t), \tag{53}$$

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 $\mathrm{vec}(\boldsymbol{W})$ as :

$$\overline{\boldsymbol{w}} = \sum_{t=1}^{T} \overline{\boldsymbol{h}}_t \frac{\partial \boldsymbol{h}_t}{\partial \mathrm{vec}(\boldsymbol{W})} = \sum_{t=1}^{T} \overline{\boldsymbol{h}}_t \boldsymbol{z}_t^\top = \sum_{t=1}^{T} \boldsymbol{z}_t \otimes \overline{\boldsymbol{h}}_t \tag{54}$$

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:

$$\begin{aligned} \boldsymbol{F_W} &= \mathbb{E}_{\{(\boldsymbol{\xi},\boldsymbol{x},\boldsymbol{y})\}\sim\mathcal{M}} \left[\overline{\boldsymbol{w}}\,\overline{\boldsymbol{w}}^\top\right] &(55)\\ &= \mathbb{E}\left[\left(\sum_{t=1}^{T} \boldsymbol{z}_t \otimes \overline{\boldsymbol{h}}_t\right)\left(\sum_{s=1}^{T} \boldsymbol{z}_s \otimes \overline{\boldsymbol{h}}_s\right)^\top\right] &(56)\\ &= \sum_{t=1}^{T}\sum_{s=1}^{T} \mathbb{E}\left[\left(\boldsymbol{z}_t \boldsymbol{z}_s^\top\right) \otimes \left(\overline{\boldsymbol{h}}_t \overline{\boldsymbol{h}}_s^\top\right)\right]. &(57) \end{aligned}$$

Here the expectations are taken with respect to the model distribution. Unfortunately, computing $\boldsymbol{F_W}$ 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 $\overline{\boldsymbol{h}}_t$:

$$\boldsymbol{F_W} \approx \sum_{t=1}^{T}\sum_{s=1}^{T} \mathbb{E}\left[\boldsymbol{z}_t \boldsymbol{z}_s^\top\right] \otimes \mathbb{E}_{\{(\boldsymbol{\xi},\boldsymbol{x},\boldsymbol{y})\}\sim\mathcal{M}} \left[\overline{\boldsymbol{h}}_t \overline{\boldsymbol{h}}_s^\top\right]. \tag{58}$$

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:

$$\boldsymbol{F_W} \approx \sum_{\tau=-T}^{T} (T - |\tau|)\mathcal{A}_\tau \otimes \mathcal{G}_\tau. \tag{59}$$

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:

$$\boldsymbol{F_W} \approx \mathbb{E}\left[T\right] \mathbb{E}\left[\boldsymbol{z}\boldsymbol{z}^\top\right] \otimes \mathbb{E}\left[\overline{\boldsymbol{h}}\,\overline{\boldsymbol{h}}^\top\right], \tag{60}$$

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:

$$\boldsymbol{F_C} \approx \mathbb{E}\left[T\right] \mathbb{E}\left[\boldsymbol{r}\boldsymbol{r}^\top\right] \otimes \mathbb{E}\left[\overline{\boldsymbol{y}}\,\overline{\boldsymbol{y}}^\top\right]. \tag{61}$$

The quality of these assumptions and comparisons with the 'approximate Fisher matrices' used in OWM and DOWM are discussed in Appendix E.

E 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

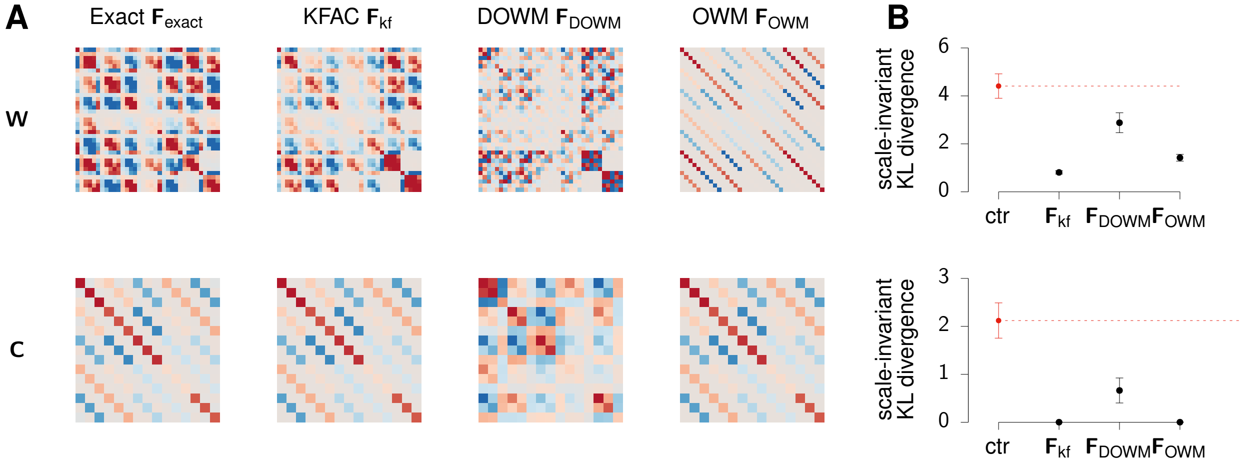

Figure 6: **Comparison of projection matrices.** In a Bayesian framework, we can formalize what is meant by directions 'important for previous tasks' as those that are strongly constrained by the prior $p(\boldsymbol{\theta}|\mathcal{D}_{1:k-1})$. To see how this compares with OWM and DOWM, we considered 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)** $\boldsymbol{F}_{\text{exact}}$ (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)** Scale-invariant KL-divergence (Equation 48) 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). **(C–D)** Same as (A–B) but for the parameter $\boldsymbol{C}$.

approximations to the prior Fisher matrix. These approaches are typically motivated as a way to retrict 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 Section 3.1, 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:

$$\boldsymbol{P}_z^{1:k} = \boldsymbol{Z}_{1:k}(\boldsymbol{Z}_{1:k}\boldsymbol{Z}_{1:k}^\top + \alpha\boldsymbol{I})^{-1}{\boldsymbol{Z}_{1:k}}^\top \tag{62}$$

$$\approx k\alpha\left(\mathbb{E}\left[\boldsymbol{z}\boldsymbol{z}^\top\right] + \alpha\boldsymbol{I}\right)^{-1} \tag{63}$$

$$\boldsymbol{P}_{wz}^{1:k} = \boldsymbol{W}\boldsymbol{Z}_{1:k}(\boldsymbol{W}\boldsymbol{Z}_{1:k}\boldsymbol{Z}_{1:k}^\top\boldsymbol{W}^\top + \alpha\boldsymbol{I})^{-1}(\boldsymbol{W}\boldsymbol{Z}_{1:k})^\top \tag{64}$$

$$\approx k\alpha\left(\boldsymbol{W}\mathbb{E}\left[\boldsymbol{z}\boldsymbol{z}^\top\right]\boldsymbol{W}^\top + \alpha\boldsymbol{I}\right)^{-1}, \tag{65}$$

which are used to derive update rules for $\boldsymbol{W}$ as:

$$\text{vec}(\Delta\boldsymbol{W}) \propto \left(\boldsymbol{P}_z^{1:k} \otimes \boldsymbol{P}_{wz}^{1:k}\right)\overline{\boldsymbol{w}} \tag{66}$$

$$\propto \left(\mathbb{E}\left[\boldsymbol{z}\boldsymbol{z}^\top\right] + \alpha\boldsymbol{I}\right)^{-1} \otimes \left(\boldsymbol{W}\mathbb{E}\left[\boldsymbol{z}\boldsymbol{z}^\top\right]\boldsymbol{W}^\top + \alpha\boldsymbol{I}\right)^{-1}\overline{\boldsymbol{w}} \tag{67}$$

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., $\boldsymbol{P}_{wz} = \boldsymbol{I}$).

We recall that the NCL update rule on task $k+1$ is given by

$$\text{vec}(\Delta\boldsymbol{W}) \propto \left(\mathbb{E}\left[\boldsymbol{z}\boldsymbol{z}^\top\right] + \alpha\boldsymbol{I}\right)^{-1} \otimes \left(\mathbb{E}\left[\overline{\boldsymbol{h}}\,\overline{\boldsymbol{h}}^\top\right] + \alpha\boldsymbol{I}\right)^{-1}\overline{\boldsymbol{w}} + (\text{vec}(\boldsymbol{W}_k) - \text{vec}(\boldsymbol{W})). \tag{68}$$

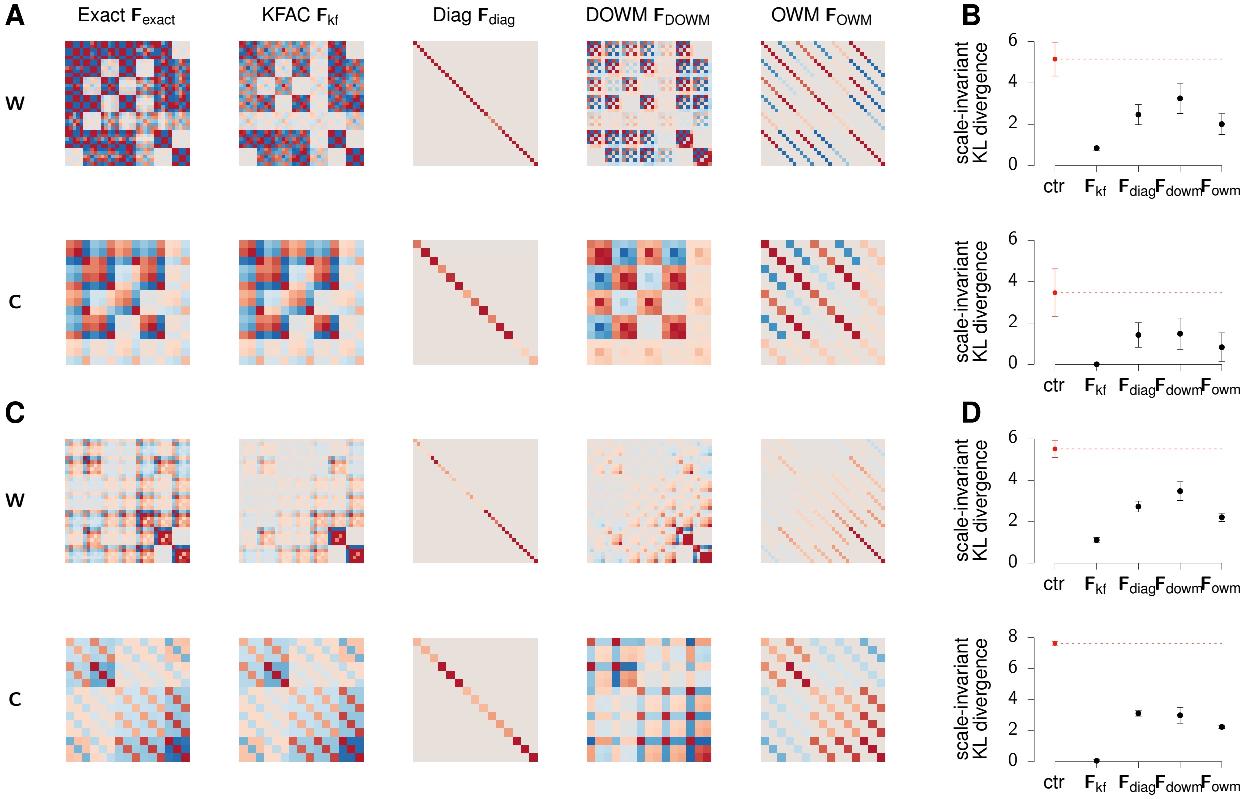

Figure 7: **Comparison of Fisher Approximations in a Linear RNN with rotated Gaussian and categorical likelihoods. (A)** Exact and approximations to the Fisher information matrix of the recurrent and input weight matrix $\boldsymbol{W}$ (left) and the linear readout $\boldsymbol{C}$ (bottom) of a linear 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 $\boldsymbol{F} \in \{\boldsymbol{F}_{\text{kf}}, \boldsymbol{F}_{\text{diag}}, \boldsymbol{F}_{\text{DOWM}}, \boldsymbol{F}_{\text{OWM}}\}$. **(C-D)** As in (A-B), now for a categorical noise model $p(\boldsymbol{y}|\boldsymbol{Cr}) = \text{Cat}\left(\text{softmax}(\boldsymbol{Cr})\right)$.

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 $\boldsymbol{P}_z$. 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 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 6). Here we found that OWM and DOWM provided reasonable approximations to the true Fisher matrix with both Gaussian (Figure 6) and categorical (Figure 7) 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 A. Here it is also worth noting that while we use an optimal sum of Kronecker factors to update the prior precision after each task in NCL (Appendix C), 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 in Appendix C suggest that it is a fairly reasonable approximation up to a scale factor which can be absorbed into the learning rate.

F Natural gradient descent and the Fisher Information Matrix

When optimizing a model with stochastic gradient descent, the parameters $\boldsymbol{\theta}$ are generally changed in the direction of steepest gradient of the loss function $\mathcal{L}$:

$$\boldsymbol{g} = \nabla_{\boldsymbol{\theta}} \mathcal{L}. \tag{69}$$

This gives rise to a learning rule

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \gamma \boldsymbol{g} \tag{70}$$

where γ is a learning rate which is usually set to a small constant or updated according to some learning rate schedule. However, we note that the parameter change itself has units of $[\boldsymbol{\theta}]^{-1}$ which suggests that such a naïve optimization procedure might be pathological under some circumstances. Consider instead the more general definition of the normalized gradient $\hat{\boldsymbol{g}}$:

$$\hat{\boldsymbol{g}} = \lim_{\epsilon \to 0} \frac{1}{Z(\epsilon)} \text{argmin}_{\delta} \mathcal{L}(\boldsymbol{\theta} + \delta) \qquad\qquad d(\boldsymbol{\theta}, \boldsymbol{\theta} + \delta) \leq \epsilon. \tag{71}$$

Here, $\hat{\boldsymbol{g}}$ is the direction in state space which minimizes $\mathcal{L}$ given a step of size ϵ according to some distance metric $d(\cdot, \cdot)$. Canonical gradient descent is in this case recovered when $d(\cdot, \cdot)$ is Euclidean distance in parameter space

$$d(\boldsymbol{\theta}, \boldsymbol{\theta}') = ||\boldsymbol{\theta} - \boldsymbol{\theta}'||_2^2. \tag{72}$$

We now formulate $\mathcal{L}(\boldsymbol{\theta})$ as depending on a statistical model $p(\mathcal{D}|\boldsymbol{\theta})$ such that $\mathcal{L}(\boldsymbol{\theta}) = \mathcal{L}(p(\mathcal{D}|\boldsymbol{\theta}))$. This allows us to define the direction of steepest gradient in terms of the change in probability distributions

$$d(\boldsymbol{\theta}, \boldsymbol{\theta}') = \text{KL}\left[p(\mathcal{D}|\boldsymbol{\theta}')||p(\mathcal{D}|\boldsymbol{\theta})\right]. \tag{73}$$

It can be shown that the direction of steepest decent for small step sizes is in this case given by (Kunstner et al., 2019; Amari, 1998)

$$\boldsymbol{g} \propto \boldsymbol{F}^{-1} \nabla \mathcal{L}(\boldsymbol{\theta}), \tag{74}$$

where $\boldsymbol{F}$ is the Fisher information matrix

$$\boldsymbol{F}(\boldsymbol{\theta}) = \mathbb{E}_{p(\mathcal{D}|\boldsymbol{\theta})} \left[\nabla \log p(\mathcal{D}|\boldsymbol{\theta}) \nabla \log p(\mathcal{D}|\boldsymbol{\theta})^T\right]. \tag{75}$$

We thus get an update rule of the form

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \gamma \boldsymbol{F}^{-1} \nabla_{\boldsymbol{\theta}} \mathcal{L}, \tag{76}$$

which has units of $[\boldsymbol{\theta}]$ and corresponds to a step in the direction of parameter space that maximizes the decrease in $\mathcal{L}$ for an infinitesimal change in $p(\mathcal{D}|\boldsymbol{\theta})$ as measured using KL divergences. It has been shown in a large body of previous work that such natural gradient descent leads to improved performance (Bernacchia et al., 2018; Osawa et al., 2019; Amari, 1998), and the main bottleneck to its implementation is usually the increased cost of computing $\boldsymbol{F}$ or a suitable approximation to this quantity.

We note that this optimization method is very similar to that derived for NCL in Section 2.2 and Appendix A except that NCL uses the approximate Fisher for *previous* tasks instead of the Fisher information matrix of the current loss. This is important since (i) it mitigates the need for computing a fairly expensive Fisher matrix at every update step, and (ii) it ensures that parameters are updated in directions that preserve the performance on previous tasks.

G Task details

Here we provide additional implementation details for the stimulus-response tasks and the SMNIST tasks.

Stimulus-response tasks Detailed descriptions of the stimulus-response tasks used in this work can be found in the appendix of Yang et al. (2019).

Here we provide a brief overview of the six tasks. 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 $\theta_{out} = 2\pi - \theta_{in}$ 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 $\theta_{out} = 2\pi - \theta_{in}$ 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 is 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.

H Further results

H.1 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 2.3):

$$\mathcal{L}_k^{(\lambda)}(\boldsymbol{\theta}) = \log p(\mathcal{D}_k|\boldsymbol{\theta}) - \lambda(\boldsymbol{\theta} - \boldsymbol{\mu}_{k-1})^\top \boldsymbol{\Lambda}_{k-1}(\boldsymbol{\theta} - \boldsymbol{\mu}_{k-1}). \tag{77}$$

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 8A), 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 8B). A similar pattern was observed for the stimulus-response task set with 50 units and the SMNIST task set (Section H.3).

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 preconditioner $\boldsymbol{\Lambda}_{k-1}^{-1}$ as λ increases (Section 2.3). 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:

$$m^{(i)} \leftarrow \beta_1 m^{(i-1)} + (1-\beta_1)\nabla_\theta \mathcal{L}^{(\lambda_m)} \tag{78}$$

$$v^{(i)} \leftarrow \beta_2 v^{(i-1)} + (1-\beta_2)\left(\nabla_\theta \mathcal{L}^{(\lambda_v)}\right)^2 \tag{79}$$

where $\mathcal{L}^{(\lambda)}$ is defined in Equation 77 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:

$$\hat{m}^{(i)} \leftarrow m^{(i)}/(1-\beta_1^i) \tag{80}$$

$$\hat{v}^{(i)} \leftarrow v^{(i)}/(1-\beta_2^i) \tag{81}$$

$$\theta^{(i)} \leftarrow \theta^{(i-1)} + \gamma \hat{m}^{(i)}/(\sqrt{\hat{v}^{(i)}} + \epsilon), \tag{82}$$

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 8C), and (ii) "reverse decoupled", where we fix $\lambda_v = 1$ and vary λ_m (Figure 8D). 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 8C). "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 8D). 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 $\boldsymbol{\Lambda}_{k-1}$ at the end of training on task k. We found that the Adam preconditioner increasingly resembled $\boldsymbol{\Lambda}_{k-1}$, the preconditioner used by NCL, as λ increased (Figure 9).

In summary, our results suggest that preconditioning with $\boldsymbol{\Lambda}_{k-1}$ 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.

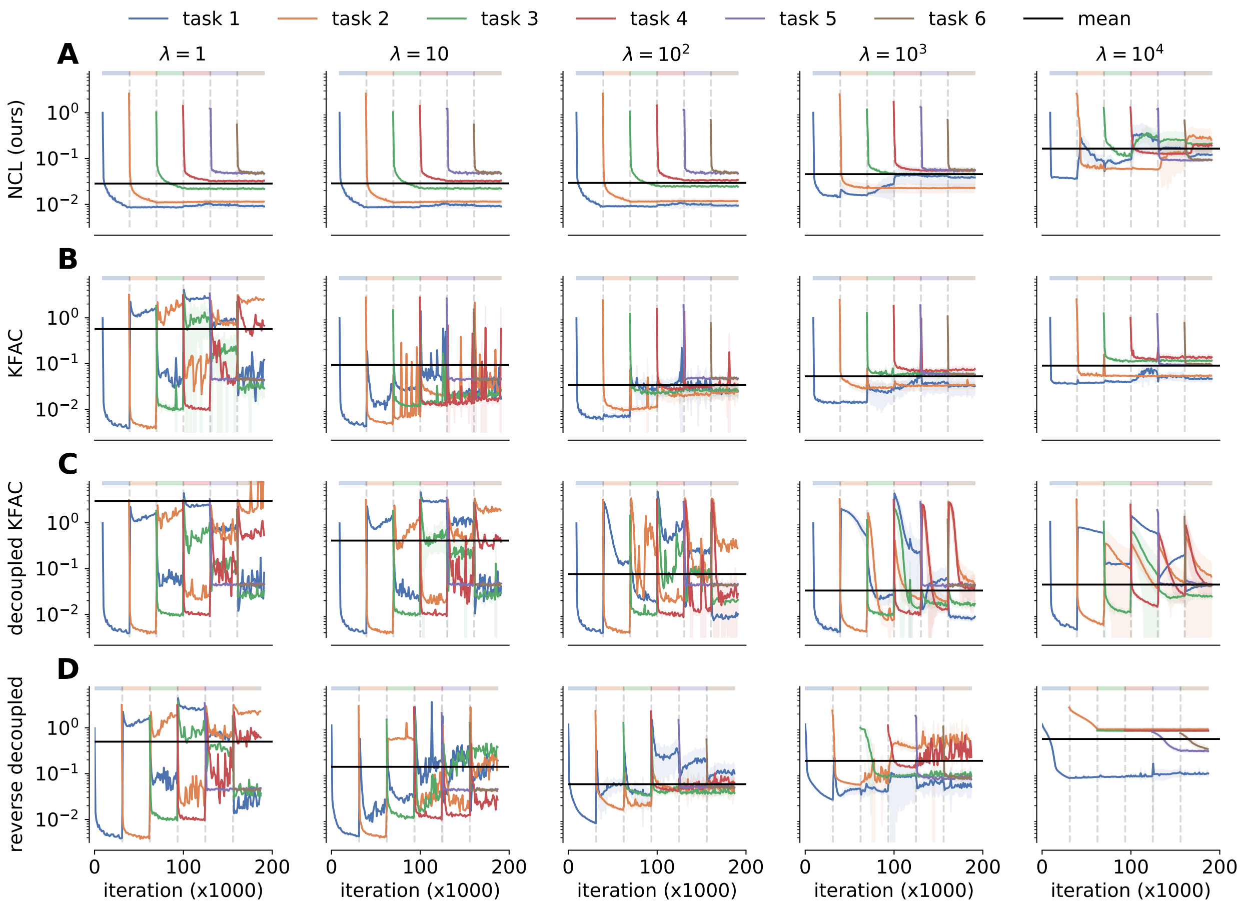

Figure 8: **Continual learning on SR tasks for different λ . (A)** Evolution of the loss during training for each of the six stimulus-response tasks for NCL with different values of λ. The performance of NCL is generally robust across different choices of λ until it starts overfitting too heavily on 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 $\lambda = 1$ is used for the gradient term and different values of λ are used for the preconditioner. Interestingly, this is sufficient to overcome the catastrophic forgetting observed for KFAC with $\lambda = 1$. 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 (B), now with 'reverse decoupled Adam' where $\lambda = 1$ is used for the preconditioner and different values of λ are used for the gradient term. For higher values of λ, this performs worse than both KFAC and decoupled KFAC.

H.2 Performance with low capacity networks

In this section, we provide further details of the comparisons between different continual learning methods on the stimulus-response tasks analyzed in Section 3.2, now for smaller networks with 50 recurrent units. Here we found that NCL outperformed both the alternative projection based methods and KFAC with Adam (Figure 10). Results in Figure 10 are reported for networks with optimized hyperparameters for the projection based methods (Section H.3). In particular, we note that the value of α used when inverting the approximate Fisher matrix is quite large for DOWM which reduces the difference in learning rates between directions that are estimated to be important for previous tasks and those that are unimportant (note that the gradient preconditioners are approximately proportional to the identity matrix for large α). When instead using a small value of α simply for numerical stability, DOWM greatly overfits on the first task and largely fails to learn subsequent tasks.

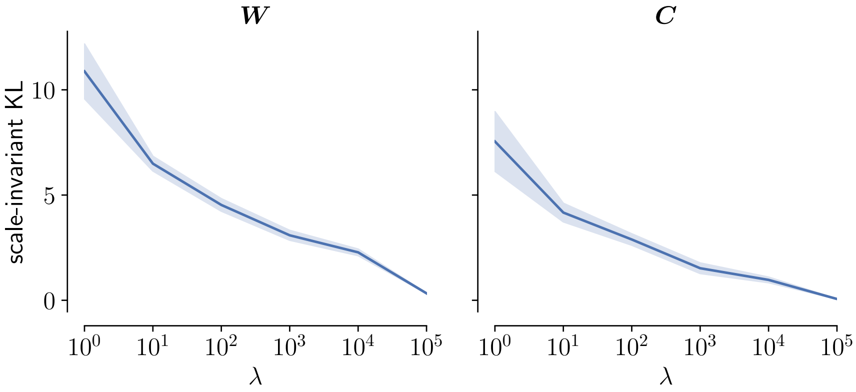

Figure 9: **Similarity of the Adam preconditioner and diagonal Fisher matrix.** Scale-invariant KL divergence (Equation 48) between the diagonal of $\boldsymbol{\Lambda}_{k-1}$ and the preconditioner used by Adam ($\sqrt{\boldsymbol{v}}$; Kingma and Ba, 2014) at the end of training on task k. Results 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).

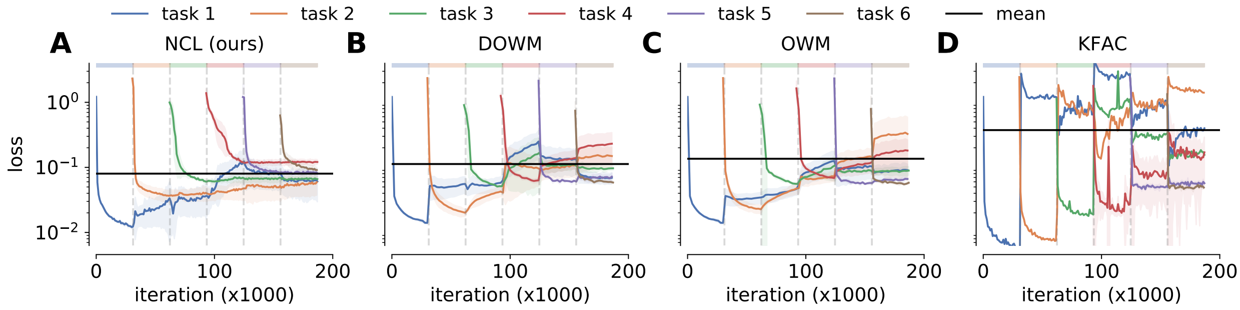

Figure 10: **Continual learning on stimulus-response tasks with a 50-unit RNN.** Evolution of the loss during training for each of the six stimulus-response tasks for NCL (A), DOWM (B), OWM (C), and KFAC with $\lambda = 1$ (D). Solid black lines indicate the mean loss over all tasks at the end of training.

H.3 Hyperparameter optimizations

Here we provide the results of our hyperparameter optimizations for each task set. Note that we optimized over the parameter α used to invert the approximate Fisher matrices in the projection based methods (NCL, OWM and DOWM), and that we optimized over the parameter λ used to scale the importance of the prior for weight regularization with 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 make up for its projection into too small a subspace (Appendix E). 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 will be interesting to combine NCL with methods such as Tikhonov dampening (Tikhonov, 1943) in future work to automatically adjust α and make NCL a hyperparameter-free method.

We generally report results in the main text and appendix using the optimal hyperparameter settings for each

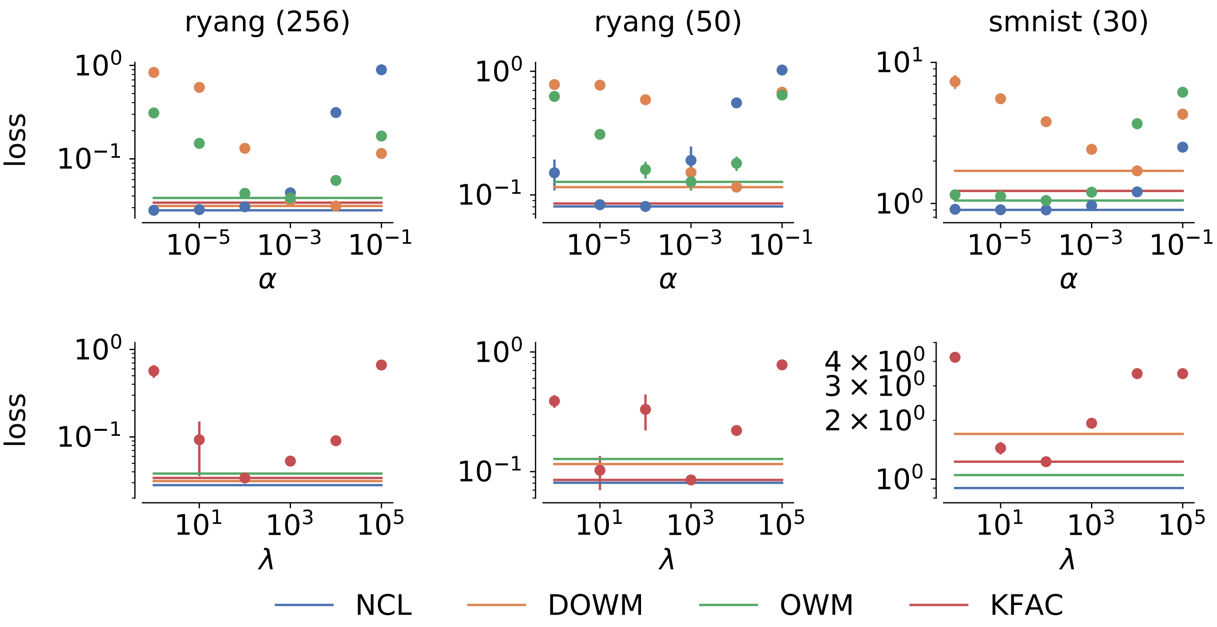

Figure 11: **Hyperparameter optimization.** **(A)** Comparison of the average loss across tasks on the stimulus-response task set with a 256-unit RNN as a function of α for the projection based methods (top panel) and as a fuction of λ for KFAC (bottom panel). Circles and error bars indicate mean and s.e.m. across 5 random seeds. Horizontal lines indicate the optimal value for each method. **(B)** As in (A), now for the stimulus-response task set with 50 units. **(C)** As in (A), now for the SMNIST task set.

method unless otherwise noted. However, $\alpha = 10^{-5}$ was used for both NCL and Laplace-DOWM in Figure 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.

H.4 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 3.4 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 12). However, DOWM also learned representations that were less well-separated after the first 1-2 classification tasks (Figure 12, bottom) than those learned by NCL. This is consistent with our results in Section 3.3 where DOWM exhibited high performance on the first task even after learning all 15 tasks, but performed less well on later tasks (Figure 12). These results may be explained by the observation that DOWM tends to overestimate the number of dimensions that are important for learned tasks (Section 3.3) 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 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.

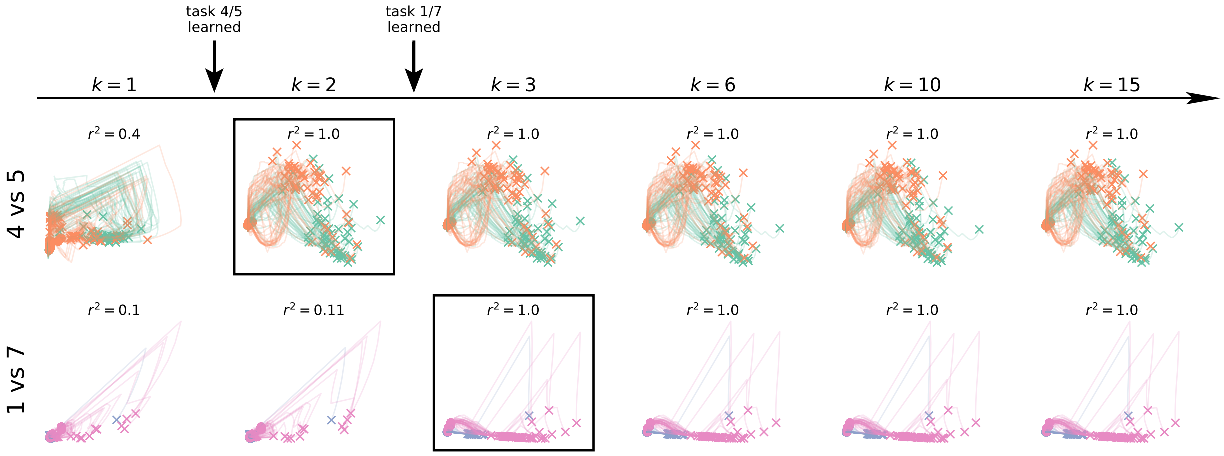

Figure 12: **Latent dynamics during SMNIST.** We considered two example tasks, 4 vs 5 (top) and 1 vs 7 (bottom). For each task, we simulated the response of a network trained by DOWM to 100 digits drawn from that task distribution at different times during learning. We then fitted a factor analysis model 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).

I NCL for variational continual learning

Online variational inference In variational continual learning (Nguyen et al., 2017), the posterior $p(\boldsymbol{\theta}|\mathcal{D}_k, \boldsymbol{\phi}_{k-1})$ is approximated with a Gaussian variational distribution $q(\boldsymbol{\theta}_k|\boldsymbol{\phi}_k) = \mathcal{N}(\boldsymbol{\theta}_k; \boldsymbol{\mu}, \boldsymbol{\Sigma}_k)$, where $\boldsymbol{\phi}_k = (\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$. We then treat $\boldsymbol{\mu}_k$ and $\boldsymbol{\Sigma}_k$ as variational parameters and minimize the KL-divergence between $q(\boldsymbol{\theta}_k|\boldsymbol{\phi}_k)$ and the approximate posterior $p(\boldsymbol{\theta}|\mathcal{D}_k, \boldsymbol{\phi}_k) \propto q(\boldsymbol{\theta}|\boldsymbol{\mu}_{k-1}, \boldsymbol{\Sigma}_{k-1})p(\mathcal{D}_k|\boldsymbol{\theta})$:

$$\mathrm{KL}\left(q(\boldsymbol{\theta}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)||\frac{1}{Z_k}q(\boldsymbol{\theta}|\boldsymbol{\mu}_{k-1}, \boldsymbol{\Sigma}_{k-1})p(\mathcal{D}_k|\boldsymbol{\theta})\right) \tag{83}$$

with respect to $\boldsymbol{\mu}_k$ and $\boldsymbol{\Sigma}_k$. This is equivalent to maximizing the evidence lower-bound (ELBO):

$$\mathcal{L}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) = \mathbb{E}_{\boldsymbol{q(\theta|\mu_k,\Sigma_k)}}\left[\log p(\mathcal{D}_k|\boldsymbol{\theta})\right] - \mathrm{KL}\left(q(\boldsymbol{\theta}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)||q(\boldsymbol{\theta}|\boldsymbol{\mu}_{k-1}, \boldsymbol{\Sigma}_{k-1})\right), \tag{84}$$

to the data log likelihood

$$\log p(\mathcal{D}_k|\boldsymbol{\phi}_{k-1}) = \log \int p(\mathcal{D}_k|\boldsymbol{\theta})q(\boldsymbol{\theta}|\boldsymbol{\phi}_{k-1})d\boldsymbol{\theta} \geq \mathcal{L} \tag{85}$$

with $q(\boldsymbol{\theta}|\boldsymbol{\phi}_{k-1})$ as the 'prior' for task k.

Maximizing $\mathcal{L}$ requires the computation of both the first likelihood term and the second KL term in Equation 84. While the second term can be computed analytically, the first term is intractable for general likelihoods $p(\mathcal{D}_k|\boldsymbol{\theta})$. To address this, Nguyen et al. (2017) estimate this likelihood term using Monte Carlo sampling:

$$\mathbb{E}_{q(\boldsymbol{\theta}|\boldsymbol{\phi}_k)}\left[\log p(\mathcal{D}_k|\boldsymbol{\theta})\right] \approx \frac{1}{K}\sum_i \log p(\mathcal{D}_k|\boldsymbol{\theta}_i), \tag{86}$$

where $\{\boldsymbol{\theta}_i\}_{i=1}^M \sim q(\boldsymbol{\theta}|\boldsymbol{\phi}_k)$ are drawn from the variational posterior via the reparameterization trick. This allows direct optimization of the variational parameters $\boldsymbol{\mu}_k$ and $\boldsymbol{\Sigma}_k$. To make the method scale to large models with potentially millions of parameters, Nguyen et al. (2017) also make a mean-field approximation to the posterior

$$q(\boldsymbol{\theta}|\boldsymbol{\phi}_k) = \mathcal{N}(\boldsymbol{\theta}; \boldsymbol{\mu}_k, \mathrm{diag}(\boldsymbol{\sigma}_k)). \tag{87}$$

Natural variational continual learning We now propose an alternative approach to maximizing $\mathcal{L}$ with respect to $\boldsymbol{\phi}_k = (\boldsymbol{\mu}_k, \boldsymbol{\Lambda}_k)$ within the NCL framework, where $\boldsymbol{\Lambda}_k = \boldsymbol{\Sigma}_k^{-1}$ is the precision matrix of q at step k. We again solve a trust-region subproblem to find the optimal parameter updates for $\boldsymbol{\mu}_k$ and $\boldsymbol{\Lambda}_k$:

$$\boldsymbol{\Delta}_{\boldsymbol{\mu}_k}, \boldsymbol{\Delta}_{\boldsymbol{\Lambda}_k} = \underset{\boldsymbol{\Delta}_{\boldsymbol{\mu}_k}, \boldsymbol{\Delta}_{\boldsymbol{\Lambda}_k}}{\arg\min}\ \mathcal{L}(\boldsymbol{\mu}_k, \boldsymbol{\Lambda}_k) + \nabla_{\boldsymbol{\mu}_k}\mathcal{L}^\top \boldsymbol{\Delta}_{\boldsymbol{\mu}_k} + \nabla_{\boldsymbol{\Lambda}_k}\mathcal{L}^\top \boldsymbol{\Delta}_{\boldsymbol{\Lambda}_k} \tag{88}$$

$$\text{such that} \quad \mathcal{C}(\boldsymbol{\Delta}_{\boldsymbol{\mu}_k}, \boldsymbol{\Delta}_{\boldsymbol{\Lambda}_k}) \leq r^2, \tag{89}$$

where

$$\mathcal{C}(\boldsymbol{\Delta}_{\boldsymbol{\mu}_k}, \boldsymbol{\Delta}_{\boldsymbol{\Lambda}_k}) = \frac{1}{2}\boldsymbol{\Delta}_{\boldsymbol{\mu}_k}^\top \boldsymbol{\Lambda}_{k-1}\boldsymbol{\Delta}_{\boldsymbol{\mu}_k} + \frac{1}{4}\mathrm{vec}(\boldsymbol{\Delta}_{\boldsymbol{\Lambda}_k})^\top(\boldsymbol{\Lambda}_k^{-1} \otimes \boldsymbol{\Lambda}_k^{-1})\mathrm{vec}(\boldsymbol{\Delta}_{\boldsymbol{\Lambda}_k}). \tag{90}$$

The solution to this optimization problem is given by:

$$\boldsymbol{\Delta}_{\boldsymbol{\mu}_k} = \boldsymbol{\Lambda}_{k-1}^{-1}\nabla_{\boldsymbol{\mu}_k}\mathcal{L} \tag{91}$$

$$\boldsymbol{\Delta}_{\boldsymbol{\Lambda}_k} = 2\boldsymbol{\Lambda}_k \nabla_{\boldsymbol{\Lambda}_k}\mathcal{L}\boldsymbol{\Lambda}_k. \tag{92}$$

To compute $\nabla_{\boldsymbol{\mu}_k}\mathcal{L}$ and $\nabla_{\boldsymbol{\Lambda}_k}\mathcal{L}$, we make use of the following identities (Opper and Archambeau, 2009):

$$\nabla_{\boldsymbol{\mu}}\ \mathbb{E}_{\boldsymbol{\theta}\sim\mathcal{N}(\boldsymbol{\theta};\boldsymbol{\mu},\boldsymbol{\Sigma})}\left[f(\boldsymbol{\theta})\right] = \mathbb{E}_{\boldsymbol{\theta}\sim\mathcal{N}(\boldsymbol{\theta};\boldsymbol{\mu},\boldsymbol{\Sigma})}\left[\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta})\right] \tag{93}$$

$$\nabla_{\boldsymbol{\Sigma}}\ \mathbb{E}_{\boldsymbol{\theta}\sim\mathcal{N}(\boldsymbol{\theta};\boldsymbol{\mu},\boldsymbol{\Sigma})}\left[f(\boldsymbol{\theta})\right] = \frac{1}{2}\mathbb{E}_{\boldsymbol{\theta}\sim\mathcal{N}(\boldsymbol{\theta};\boldsymbol{\mu},\boldsymbol{\Sigma})}\left[\nabla_{\boldsymbol{\theta}}^2 f(\boldsymbol{\theta})\right]. \tag{94}$$

Applying these identities to compute the gradients of $\mathcal{L}$ (Equation 84), we find

$$\nabla_{\boldsymbol{\mu}_k}\mathcal{L} = \mathbb{E}_{\boldsymbol{\theta}\sim q(\boldsymbol{\theta}|\boldsymbol{\phi}_k)}\left[\nabla_{\boldsymbol{\theta}} \log p(\mathcal{D}_k|\boldsymbol{\theta}) - \boldsymbol{\Lambda}_{k-1}(\boldsymbol{\theta}_k - \boldsymbol{\mu}_{k-1})\right] \tag{95}$$

$$\nabla_{\boldsymbol{\Sigma}_k}\mathcal{L} = \frac{1}{2}\mathbb{E}_{\boldsymbol{\theta}\sim q(\boldsymbol{\theta}|\boldsymbol{\phi}_k)}\left[\nabla_{\boldsymbol{\theta}}^2 \log p(\mathcal{D}_k|\boldsymbol{\theta}) - \boldsymbol{\Lambda}_{k-1} + \boldsymbol{\Lambda}_k\right]. \tag{96}$$

Using the fact that $d\boldsymbol{\Lambda}_k = -\boldsymbol{\Lambda}_k^{-1} d\boldsymbol{\Sigma}_k \boldsymbol{\Lambda}_k^{-1}$, we have

$$\nabla_{\boldsymbol{\Lambda}_k}\mathcal{L} = -\boldsymbol{\Lambda}_k^{-1}\nabla_{\boldsymbol{\Sigma}_k}\mathcal{L}\boldsymbol{\Lambda}_k^{-1} \tag{97}$$

$$= -\frac{1}{2}\boldsymbol{\Lambda}_k^{-1}\mathbb{E}_{\boldsymbol{\theta}\sim q(\boldsymbol{\theta}|\boldsymbol{\phi}_k)}\left[\nabla_{\boldsymbol{\theta}}^2 \log p(\mathcal{D}_k|\boldsymbol{\theta}) - \boldsymbol{\Lambda}_{k-1} + \boldsymbol{\Lambda}_k\right]\boldsymbol{\Lambda}_k^{-1}. \tag{98}$$

This suggests that we can compute $\boldsymbol{\Delta}_{\boldsymbol{\mu}_k}$ and $\boldsymbol{\Delta}_{\boldsymbol{\Lambda}_k}$ as:

$$\boldsymbol{\Delta}_{\boldsymbol{\mu}_k} = \boldsymbol{\Lambda}_{k-1}^{-1}\mathbb{E}_{\boldsymbol{\theta}\sim q(\boldsymbol{\theta}|\boldsymbol{\phi}_k)}\left[\nabla_{\boldsymbol{\theta}} \log p(\mathcal{D}_k|\boldsymbol{\theta}_k)\right] - (\boldsymbol{\mu}_k - \boldsymbol{\mu}_{k-1}) \tag{99}$$

$$\boldsymbol{\Delta}_{\boldsymbol{\Lambda}_k} = \boldsymbol{\Lambda}_{k-1} - \boldsymbol{\Lambda}_k - \mathbb{E}_{\boldsymbol{\theta}\sim q(\boldsymbol{\theta}|\boldsymbol{\phi}_k)}\left[\nabla_{\boldsymbol{\theta}}^2 \log p(\mathcal{D}_k|\boldsymbol{\theta})\right]. \tag{100}$$

This gives the following update rule at learning iteration i during task k:

$$\boldsymbol{\mu}_k^{(i+1)} = (1-\beta)\boldsymbol{\mu}_k^{(i)} + \beta\left[\boldsymbol{\mu}_{k-1} + \boldsymbol{\Lambda}_{k-1}^{-1}\mathbb{E}_{\boldsymbol{\theta}\sim q(\boldsymbol{\theta}|\boldsymbol{\phi}_k^{(i)})}\left[\nabla_{\boldsymbol{\theta}} \log p(\mathcal{D}_k|\boldsymbol{\theta})\right]\right] \tag{101}$$

$$\boldsymbol{\Lambda}_k^{(i+1)} = (1-\beta)\boldsymbol{\Lambda}_k^{(i)} + \beta\left[\boldsymbol{\Lambda}_{k-1} - \mathbb{E}_{\boldsymbol{\theta}\sim q(\boldsymbol{\theta}|\boldsymbol{\phi}_k^{(i)})}\left[\nabla_{\boldsymbol{\theta}}^2 \log p(\mathcal{D}_k|\boldsymbol{\theta})\right]\right], \tag{102}$$

Note that this update rule is equivalent to preconditioning the gradients $\nabla_{\boldsymbol{\mu}_k}\mathcal{L}$ and $\nabla_{\boldsymbol{\Lambda}_k}\mathcal{L}$ with $\boldsymbol{\Lambda}_{k-1}^{-1}$ and $\boldsymbol{\Lambda}_k \otimes \boldsymbol{\Lambda}_k$ respectively.

As for the online Laplace approximation (Section 2.1), one of the main difficulties of implementing the update rule described in Equation 101 and Equation 102 is that it is impractical to compute and store the Hessian of the negative log-likelihood for large models. Furthermore, we need $\boldsymbol{\Lambda}_k^{-1}$ to remain PSD which is not

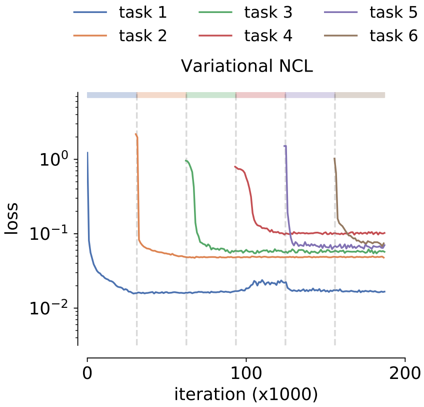

Figure 13: **Variational NCL applied to the stimulus-response task.** Evolution of the loss during training for each of the six stimulus-response tasks using variational NCL.

guaranteed as the Hessian is not necessarily PSD. In practice we therefore again approximate the Hessian with the Fisher-information matrix:

$$H_k = -\mathbb{E}\left[\nabla_\theta^2 \log p(\boldsymbol{\theta})\right] \approx F_k = \mathbb{E}_{\hat{\mathcal{D}}_k \sim p(\mathcal{D}_k|\boldsymbol{\theta})}\left[\nabla_\theta \log p(\hat{\mathcal{D}}_k|\boldsymbol{\theta}) \nabla_\theta \log p(\hat{\mathcal{D}}_k|\boldsymbol{\theta})^\top\right]. \tag{103}$$

As in Section 2.2 we use a Kronecker factored approximation to the FIM for computational tractability. With these approximations, we arrive at the learning rule:

$$\boldsymbol{\mu}_k^{(i+1)} = (1-\beta)\boldsymbol{\mu}_k^{(i)} + \beta\left[\boldsymbol{\mu}_{k-1} + \boldsymbol{\Lambda}_{k-1}^{-1}\mathbb{E}_{\boldsymbol{\theta}\sim q(\boldsymbol{\theta}|\boldsymbol{\phi}_k^{(i)})}\left[\nabla_{\boldsymbol{\theta}} \log p(\mathcal{D}_k|\boldsymbol{\theta})\right]\right], \tag{104}$$

$$\boldsymbol{\Lambda}_k^{(i+1)} = (1-\beta)\boldsymbol{\Lambda}_k^{(i)} + \beta\left[\boldsymbol{\Lambda}_{k-1} + \mathbb{E}_{\boldsymbol{\theta}\sim q(\boldsymbol{\theta}|\boldsymbol{\phi}_k^{(i)})}\left[F_k(\boldsymbol{\theta})\right]\right]. \tag{105}$$

These update rules define the ‘natural variational continual learning’ (NVCL) algorithm which is the variational equivalent of the Laplace algorithm derived in Section 2.2 and used in Section 3.

Experiments To understand how Equations 104-105 encourage continual learning, we note that the first two terms of Equation 104 urge the new parameters $\boldsymbol{\mu}_k$ to stay close to $\boldsymbol{\mu}_{k-1}$. The third term of Equation 104 improves the average performance of the learner on task k by moving $\boldsymbol{\mu}_k$ along $\boldsymbol{\Lambda}_{k-1}^{-1}p(\mathcal{D}_k|\boldsymbol{\theta})$. This is a valid search direction because $\boldsymbol{\Lambda}_{k-1}^{-1} = \boldsymbol{\Sigma}_k$ is the covariance of $q(\boldsymbol{\theta}|\boldsymbol{\phi}_{k-1})$ and is thus positive semi-definite (PSD). Importantly, the preconditioner $\boldsymbol{\Lambda}_{k-1}^{-1}$ ensures that $\boldsymbol{\mu}_k$ changes primarily along “flat” directions of $q(\boldsymbol{\theta}|\boldsymbol{\phi}_{k-1})$. This in turn encourages $q(\boldsymbol{\theta}|\boldsymbol{\phi}_k)$ to stay close to $q(\boldsymbol{\theta}|\boldsymbol{\phi}_{k-1})$ in the KL sense. In Equation 105, the first two terms again encourage $\boldsymbol{\Lambda}_k$ to remain close to $\boldsymbol{\Lambda}_{k-1}$. The third term in Equation 105 updates the precision matrix of the approximate posterior with the average Fisher matrix for task k. This encourages the curvature of the approximate posterior to be similar to that of the loss landscape of task k, and thus (at least locally) parameters that have similar performance on the task will have similar probabilities under the approximate posterior.

To test the natural VCL algorithm, we applied it to the stimulus-response task set considered in Section 3.2 using an RNN with 256 units. Similar to the Laplace version of NCL, we found that NVCL was capable of solving all six tasks without forgetting (Figure 13). While this can be seen as a proof-of-principle that our natural VCL algorithm works, we leave more extensive comparisons between the variational and Laplace algorithms for future work.

Related work Previous studies have proposed the use of variants of natural gradient descent to optimize the variational continual learning objective (Tseran et al., 2018; Osawa et al., 2019). The key differences between the method proposed in this section and previous methods are two-fold: (i) we precondition the gradient updates on task k with $\boldsymbol{\Lambda}_{k-1}^{-1}$ as opposed to $\boldsymbol{\Lambda}_k^{-1}$ as is done in prior work, and (ii) we estimate the Fisher matrix on each task by drawing samples from the model distribution as opposed to the empirical distribution as is the case in Tseran et al. (2018); Osawa et al. (2019). It has previously been argued that drawing from the model distribution instead of using the 'empirical' Fisher matrix is important to retain the desirable properties of natural gradient descent (Kunstner et al., 2019).

J Details of toy example in schematic

In Figure 1A, we consider two regression tasks with losses defined as:

$$\ell_1(\boldsymbol{\theta}) = \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}_1)^T \boldsymbol{Q}_1 (\boldsymbol{\theta} - \boldsymbol{\theta}_1) \tag{106}$$

$$\ell_2(\boldsymbol{\theta}) = \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}_2)^T \boldsymbol{Q}_2 (\boldsymbol{\theta} - \boldsymbol{\theta}_2), \tag{107}$$

where $\boldsymbol{\theta}_1 = (3, -6)^\top$, $\boldsymbol{\theta}_2 = (3, 6)^\top$,

$$\boldsymbol{Q}_1 = \boldsymbol{R}(\phi_1) \begin{bmatrix} 1 & 0 \\ 0 & \zeta \end{bmatrix} \boldsymbol{R}(\phi_1)^T, \tag{108}$$

$$\boldsymbol{Q}_2 = \boldsymbol{R}(\phi_2) \begin{bmatrix} 2 & 0 \\ 0 & \zeta \end{bmatrix} \boldsymbol{R}(\phi_2)^T, \tag{109}$$

$$\boldsymbol{R}(\phi) = \begin{bmatrix} \cos(\phi) & -\sin(\phi) \\ \sin(\phi) & \cos(\phi) \end{bmatrix}, \tag{110}$$

and $\zeta = 5.5$. We train on task 1 first and find the optimal $\boldsymbol{\theta} = \boldsymbol{\theta}_1$. 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 $\boldsymbol{\theta}$). Now we proceed to train on task 2 by maximizing the posterior (see Equation 4):

$$\mathcal{L}_2(\boldsymbol{\theta}) = \ell_2(\boldsymbol{\theta}) + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}_1)^T \boldsymbol{Q}_1 (\boldsymbol{\theta} - \boldsymbol{\theta}_1) \tag{111}$$

$$= \ell_2(\boldsymbol{\theta}) + \ell_1(\boldsymbol{\theta}) \tag{112}$$

The gradient of $\mathcal{L}_2(\boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$ is given by:

$$\nabla_{\boldsymbol{\theta}} \mathcal{L} = \boldsymbol{Q}_1(\boldsymbol{\theta} - \boldsymbol{\theta}_1) + \boldsymbol{Q}_2(\boldsymbol{\theta} - \boldsymbol{\theta}_2). \tag{113}$$

We can optimize $\ell(\boldsymbol{\theta})$ using the following four methods:

$$\text{Laplace:} \quad \Delta\boldsymbol{\theta} \propto \boldsymbol{Q}_1(\boldsymbol{\theta} - \boldsymbol{\theta}_1) + \boldsymbol{Q}_2(\boldsymbol{\theta} - \boldsymbol{\theta}_2) \tag{114}$$

$$\text{NCL:} \quad \Delta\boldsymbol{\theta} \propto (\boldsymbol{\theta} - \boldsymbol{\theta}_1) + \boldsymbol{Q}_1^{-1}\boldsymbol{Q}_2(\boldsymbol{\theta} - \boldsymbol{\theta}_2) \tag{115}$$

$$\text{GD:} \quad \Delta\boldsymbol{\theta} \propto \boldsymbol{Q}_2(\boldsymbol{\theta} - \boldsymbol{\theta}_2) \tag{116}$$

$$\text{Projected:} \quad \Delta\boldsymbol{\theta} \propto \boldsymbol{Q}_1^{-1}\boldsymbol{Q}_2(\boldsymbol{\theta} - \boldsymbol{\theta}_2), \tag{117}$$

where γ is the learning rate and $\boldsymbol{Q}_1 + \boldsymbol{Q}_2$ is the Hessian of $\mathcal{L}(\boldsymbol{w})$. Note that in 'GD' and 'projected' we optimize on task 2 only rather than on the Laplace posterior.

In Figure 1B, we consider a slight modification to ℓ_2 such that the loss is no longer convex:

$$\ell_2(\boldsymbol{w}) = \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}_2)^T \boldsymbol{Q}_2 (\boldsymbol{\theta} - \boldsymbol{\theta}_2) + a - a \exp\left(-\frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{v})^T \boldsymbol{Q}_v (\boldsymbol{\theta} - \boldsymbol{v})\right), \tag{118}$$

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 $G := a \exp\left(-\frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{v})^T \boldsymbol{Q}_v (\boldsymbol{\theta} - \boldsymbol{v})\right)$, we thus

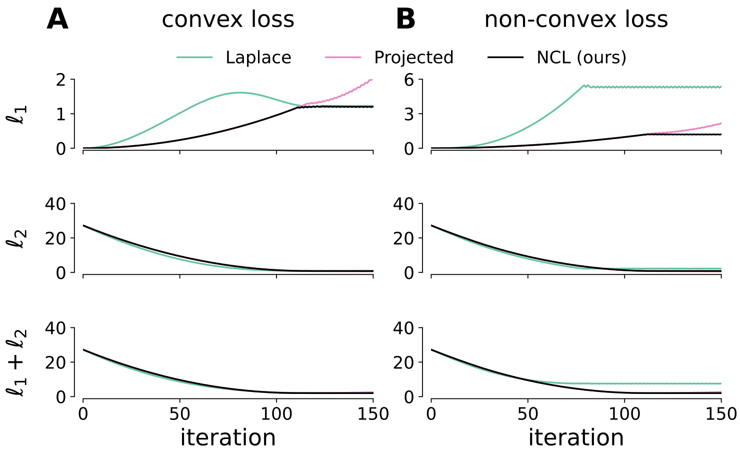

Figure 14: **Losses on toy optimization problem. (A)** Loss as a function of optimization step on task 1 (top), task 2 (middle) and the combined loss (bottom) on the convex toy continual learning problem for different optimization methods. **(B)** As in (A), now for the non-convex problem.

have the following updates when learning task 2:

$$\text{Laplace:} \quad \boldsymbol{\Delta\theta} \propto \boldsymbol{Q}_1(\boldsymbol{\theta} - \boldsymbol{\theta}_1) + \boldsymbol{Q}_2(\boldsymbol{\theta} - \boldsymbol{\theta}_2) + \boldsymbol{Q}_v(\boldsymbol{\theta} - \boldsymbol{v})G \tag{119}$$

$$\text{NCL:} \quad \boldsymbol{\Delta\theta} \propto (\boldsymbol{\theta} - \boldsymbol{\theta}_1) + \boldsymbol{Q}_1^{-1}\boldsymbol{Q}_2(\boldsymbol{\theta} - \boldsymbol{\theta}_2) + \boldsymbol{Q}_1^{-1}\boldsymbol{Q}_v(\boldsymbol{\theta} - \boldsymbol{v})G \tag{120}$$

$$\text{GD:} \quad \boldsymbol{\Delta\theta} \propto \boldsymbol{Q}_2(\boldsymbol{\theta} - \boldsymbol{\theta}_2) + \boldsymbol{Q}_v(\boldsymbol{\theta} - \boldsymbol{v})G \tag{121}$$

$$\text{Projected:} \quad \boldsymbol{\Delta\theta} \propto \boldsymbol{Q}_1^{-1}\boldsymbol{Q}_2(\boldsymbol{\theta} - \boldsymbol{\theta}_2) + \boldsymbol{Q}_1^{-1}\boldsymbol{Q}_v(\boldsymbol{\theta} - \boldsymbol{v})G. \tag{122}$$

In this non-convex case, the different methods can converge to different local minima (c.f. Figure 1B).

The losses on both tasks as well as the combined loss as a function of optimization step are illustrated in Figure 14 for the convex and non-convex settings.

References

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

Bernacchia, A., Lengyel, M., and Hennequin, G. (2018). Exact natural gradient in deep linear networks and its application to the nonlinear case. *Advances in Neural Information Processing Systems*, 31:5941–5950.

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

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

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

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

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.

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

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

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*.

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

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

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.

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.