Efficient Weight-Space Laplace–Gaussian Filtering and Smoothing for Sequential Deep Learning

Joanna Sliwa[1], **Frank Schneider**[1], **Nathanael Bosch**[1], **Agustinus Kristiadi**[2], **Philipp Hennig**[1]

[1] Tübingen AI Center, University of Tübingen, Tübingen, Germany
[2] Vector Institute, Toronto, Canada

Abstract

Efficiently learning a sequence of related tasks, such as in continual learning, poses a significant challenge for neural nets due to the delicate trade-off between catastrophic forgetting and loss of plasticity. We address this challenge with a grounded framework for sequentially learning related tasks based on Bayesian inference. Specifically, we treat the model's parameters as a nonlinear Gaussian state-space model and perform efficient inference using Gaussian filtering and smoothing. This general formalism subsumes existing continual learning approaches, while also offering a clearer conceptual understanding of its components. Leveraging Laplace approximations during filtering, we construct Gaussian posterior measures on the weight space of a neural network for each task. We use it as an efficient regularizer by exploiting the structure of the generalized Gauss-Newton matrix (GGN) to construct diagonal plus low-rank approximations. The dynamics model allows targeted control of the learning process and the incorporation of domain-specific knowledge, such as modeling the type of shift between tasks. Additionally, using Bayesian approximate smoothing can enhance the performance of task-specific models without needing to re-access any data.

1 Introduction

As deep learning continues to advance, problems remain where retaining previously learned knowledge is essential. An example is *continual learning*, where a model encounters new tasks but needs to preserve knowledge on all previously learned tasks. Despite the model's capacity to learn all tasks *simultaneously*, learning new tasks *sequentially* often leads to partial "forgetting". This divergence between a sequentially & simultaneously learned model is referred to as *catastrophic forgetting* [1]. A common approach to address this trade-off constrains the network to encourage changes that maintain the performance on previous tasks, e.g. via weight-space regularization. However, doing so evidently requires a meaningful and structured quantification of uncertainty—knowing what is still relevant, and which parts can be flexible. Otherwise, models either cannot retain the performance on previous tasks, or they become too stiff, unable to adapt to new tasks, also known as *loss of plasticity* [2]. A number of recent works try to address this issue [3–5].

Building on these, here, we treat the weight space of the deep network as the state space of a nonlinear Gaussian model. This allows us to apply established concepts from Bayesian filtering and smoothing to sequentially learn tasks with neural nets. The main advantage of this approach is that it maps the complex problem of continual learning to a well-understood formalism in Bayesian inference, yielding both efficient computational methods and a clear conceptual understanding of the components of the model, for instance, by providing clear ways to include domain knowledge about the sequence of tasks. Existing approaches in continual learning, such as Elastic Weight

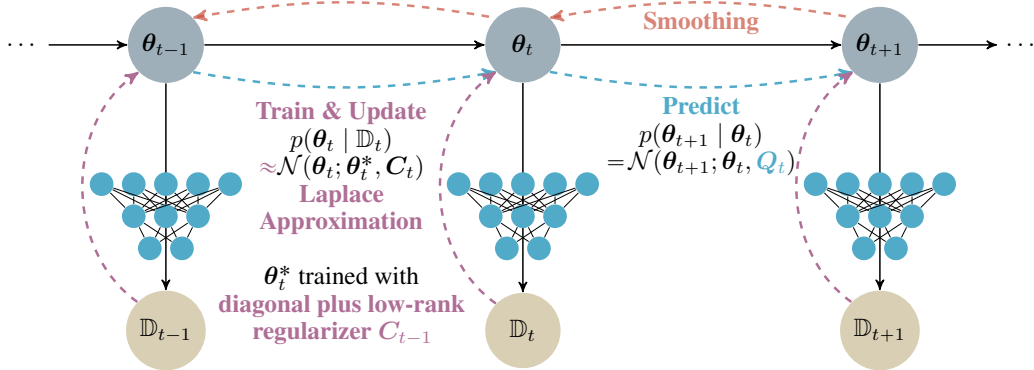---

Correspondence to: Joanna Sliwa <joanna.sliwa@uni-tuebingen.de>.

Figure 1: **An efficient weight-space Laplace–Gaussian filter (- -➤ , - -➤) and smoother (◄ - -) for sequential deep learning.** We treat the model's parameters as a nonlinear Gaussian state-space model and perform efficient inference using low-rank Laplace–Gaussian filtering and smoothing. During the update step (- -➤) we train the neural network on the current task using the parameter covariance as a regularizer, then approximate the posterior distribution with a low-rank Laplace approximation. The predict step (- -➤) adds noise to the model parameters, where the noise covariance Q can be used to model the type of shift between tasks. Smoothing (◄ - -) then allows for training task-specific model parameters θ_t that are informed by *all* tasks, without additional training.

Consolidation (EWC) [6] or Online Structured Laplace Approximations (OSLA) [3] are then special cases of our general formalism for specific choices, e.g. of the employed curvature approximation or dynamics model.

Contributions: We propose a grounded framework for sequentially training neural nets on related tasks, as in continual, transfer, or online learning, based on a Laplace–Gaussian filter and smoother:

1. *We construct an algorithmically efficient and scalable way to deal with required curvature approximations.* As part of the filter (Section 3), we employ a weight-space regularizer utilizing the Laplace approximation [e.g. 7, 8] with the generalized Gauss-Newton (GGN) matrix [9]. By exploiting the low-rank structure of the GGN, we construct *diagonal plus low-rank approximations* of the precision matrices via truncated SVDs. All required operations of both the filter and the smoother can then be computed efficiently. We provide an open-source JAX implementation of this efficient diagonal plus low-rank Laplace–Gaussian filter.[1] We showcase the competitiveness of our approximation on standard continual learning tasks (Section 5.1).

2. *Changes between tasks can be described with established concepts from Bayesian inference.* A Bayesian filter can incorporate domain knowledge about relationships between the tasks. Specifically, we study how structured uncertainties can be integrated via the process noise matrix Q. This matrix models (possible) stochastic drifts between tasks. We demonstrate that via this matrix, one can integrate prior domain knowledge about how model parameters are likely to change between related tasks (Section 5.2). For example, it is straightforward to include information indicating that primarily the model's upper (or lower) layers should change between tasks.

3. *Previous tasks can be efficiently informed by knowledge of subsequent tasks.* We examine the benefits of Bayesian smoothing and sequentially train *task-specific models* that are informed by *all* tasks, without requiring any renewed access to the data, and we demonstrate that smoothing can significantly boost the performance (Section 5.3). Such an approach may be beneficial in the low-data regime and extends our framework to fields outside of continual learning.

2   Background

Notation: We consider supervised learning with a dataset $\mathbb{D} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i) | i = 1, \ldots, N\}$ containing training inputs $\boldsymbol{x}_i$ and outputs $\boldsymbol{y}_i$. The objective is to find parameters $\boldsymbol{\theta} \in \mathbb{R}^D$ of a deep network $f_{\boldsymbol{\theta}}$ that minimize a given loss, i.e. $\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{y}; f_{\boldsymbol{\theta}}(\boldsymbol{x})) = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}, \mathbb{D})$. The first

---

[1]A snippet available at https://github.com/a90952852/lr-lgf

and second derivatives of the loss with respect to the parameters are represented by the gradient $\boldsymbol{g}(\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{y}; f_{\boldsymbol{\theta}}(\boldsymbol{x})) \in \mathbb{R}^D$ and the Hessian $\boldsymbol{H}(\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}}^2 \mathcal{L}(\boldsymbol{y}; f_{\boldsymbol{\theta}}(\boldsymbol{x})) \in \mathbb{R}^{D \times D}$.

Continual learning: *Continual learning* still lacks a unified definition (see Appendix A). For our purposes, we describe it as sequentially learning a series of tasks while maintaining good performance on *all* tasks simultaneously. Specifically, for a sequence of tasks $t \in \{t_1, \ldots, t_T\}$, each described by a dataset $\mathbb{D}_t$, we prioritize the *average* performance across all tasks so far, measured by the average loss (or accuracy) $\frac{1}{T} \sum_t \mathcal{L}(\boldsymbol{\theta}, \mathbb{D}_t)$. Crucially, tasks are experienced sequentially, without access to data from previous tasks. To achieve the required balance between model flexibility & rigidity, we adopt a regularization-based approach (see Section 4), wherein we train the model on task t using a non-isotropic ℓ_2 regularizer. This regularizer represents prior knowledge and thus constrains from previous tasks [see 3, 6]. It effectively encodes which weights should remain unchanged to maintain performance on previous tasks and which can be adjusted to perform well on the new task. To implement this, we utilize the Laplace approximation of the Bayesian posterior over the model's parameters such that the posterior of the previous task becomes the prior for the current one.

Laplace approximations: The Bayesian posterior over the model's parameters, $p(\boldsymbol{\theta} \mid \mathbb{D})$, describes the belief over the specific values of each parameter and thus reflects (un)certainty about each parameter's value and identifies which parameters still offer flexibility to learn new tasks. The *Laplace approximation* [e.g. 7, 8] provides a local Gaussian approximation to this typically intractable posterior. It arises from a second-order Taylor expansion of the loss around the maximum a posteriori (MAP) estimate of the parameters, i.e. the trained $\boldsymbol{\theta}^*$, as $\mathcal{L}(\boldsymbol{\theta}, \mathbb{D}) \approx \mathcal{L}(\boldsymbol{\theta}^*, \mathbb{D}) + \frac{1}{2} (\boldsymbol{\theta} - \boldsymbol{\theta}^*)^\top \boldsymbol{H}(\boldsymbol{\theta}^*) (\boldsymbol{\theta} - \boldsymbol{\theta}^*)$, yielding a Gaussian distribution $p(\boldsymbol{\theta} \mid \mathbb{D}) \approx \mathcal{N}(\boldsymbol{\theta}; \boldsymbol{\theta}^*, \boldsymbol{H}^{-1}(\boldsymbol{\theta}^*))$ called the Laplace approximation. Using this approximation as a weight-space regularizer results in the regularized loss for task t as $\mathcal{L}^{\text{reg}}(\boldsymbol{\theta}, \mathbb{D}_t) = \mathcal{L}(\boldsymbol{\theta}, \mathbb{D}_t) + \frac{\lambda}{2} (\boldsymbol{\theta} - \boldsymbol{\theta}_{t-1}^*)^\top \boldsymbol{H}_{t-1}(\boldsymbol{\theta}_{t-1}^*) (\boldsymbol{\theta} - \boldsymbol{\theta}_{t-1}^*)$, with λ the regularization strength. Intuitively this means that for task t, we prefer solutions close to the trained (MAP) parameters $\boldsymbol{\theta}_{t-1}^*$ of the previous task. We allow more flexibility in parameters with low loss curvature while aiming to preserve those whose change would significantly increase the loss on previous tasks.

Generalized Gauss-Newton: Since the Hessian is the second derivative of a composition of two functions, $\mathcal{L}$ and f, we can rewrite it with $\boldsymbol{J} \in \mathbb{R}^{D \times C}$ and $\hat{\boldsymbol{H}} \in \mathbb{R}^{C \times C}$ as

$$\boldsymbol{H}(\boldsymbol{\theta}) = \frac{\partial^2 \mathcal{L}}{\partial \boldsymbol{\theta}^2} = \frac{\partial f}{\partial \boldsymbol{\theta}} \frac{\partial^2 \mathcal{L}}{\partial f^2} \frac{\partial f}{\partial \boldsymbol{\theta}}^\top + \frac{\partial^2 f}{\partial \boldsymbol{\theta}^2} \frac{\partial \mathcal{L}}{\partial f} := \boldsymbol{J} \hat{\boldsymbol{H}} \boldsymbol{J}^\top + \frac{\partial^2 f}{\partial \boldsymbol{\theta}^2} \frac{\partial \mathcal{L}}{\partial f} \tag{1}$$

where C denotes the dimension of the neural network output. The generalized Gauss-Newton (GGN) matrix is defined as the first term of this expression, $\boldsymbol{J} \hat{\boldsymbol{H}} \boldsymbol{J}^\top$ [9]. Since typically $C \ll D$, the GGN is low-rank, can be stored in $O(DC)$, and it is guaranteed to be positive semi-definite. In addition, the GGN can be computed using mini-batches, i.e. $\boldsymbol{J} \hat{\boldsymbol{H}} \boldsymbol{J}^\top = \sum_{b=1}^B \boldsymbol{J}_t^{(b)} \hat{\boldsymbol{H}}_t^{(b)} (\boldsymbol{J}_t^{(b)})^\top$, where $\boldsymbol{J}_t^{(b)} \in \mathbb{R}^{D \times C}$ is the Jacobian of the neural network with respect to its parameters and $\hat{\boldsymbol{H}}_t^{(b)} \in \mathbb{R}^{C \times C}$ is the Hessian of the loss with respect to the neural network outputs, for the b-th mini-batch.

3 A Bayesian Inference Framework for Sequential Learning

We consider a sequence of tasks $t = 1, 2, \ldots, T$ with corresponding datasets $\mathbb{D}_t$. The goal is to compute a posterior distributions over neural net's parameters, given the data of all prior tasks, i.e. $p(\boldsymbol{\theta}_t \mid \mathbb{D}_{1:t})$. We might additionally be interested in the posterior distribution for task t given *all* available datasets, i.e. $p(\boldsymbol{\theta}_t \mid \mathbb{D}_{1:T})$. In the following we formulate these distributions as the *filtering* and *smoothing* distributions in a suitable Gaussian state-space model and develop an approximate inference algorithm to efficiently approximate these distributions.

We formulate sequential training as a *Bayesian state estimation* problem, where the parameters of the network are treated as the state of a state-space model of the form

$$\text{Transition model:} \qquad p(\boldsymbol{\theta}_{t+1} \mid \boldsymbol{\theta}_t) = \mathcal{N}(\boldsymbol{\theta}_{t+1}; \boldsymbol{\theta}_t, \boldsymbol{Q}), \tag{2a}$$

$$\text{Likelihood:} \qquad p(\mathbb{D}_t \mid \boldsymbol{\theta}_t) \propto \exp\left(-\frac{1}{\lambda} \mathcal{L}(\boldsymbol{\theta}_t, \mathbb{D}_t)\right). \tag{2b}$$

The un-normalized likelihood $p(\mathbb{D}_t \mid \boldsymbol{\theta}_t)$ encodes the supervised learning task on the dataset $\mathbb{D}_t$, defined by the loss function $\mathcal{L}$, scaled by a factor $1/\lambda \in \mathbb{R}_+$ which controls the strength of regularization.

The prior transition density $p(\boldsymbol{\theta}_{t+1} \mid \boldsymbol{\theta}_t)$ describes a prior belief over the change of weights from task t to task $t+1$, with diagonal Gaussian noise covariance $\boldsymbol{Q} \in \mathbb{R}^{D \times D}$. Then, computing the posterior over the weights given the data up to task t, that is $p(\boldsymbol{\theta}_t \mid \mathbb{D}_{1:t})$, is known as Bayesian filtering [10].

In state-space models, the posterior distribution $p(\boldsymbol{\theta}_t \mid \mathbb{D}_{1:t})$ can be computed recursively using the so-called general Bayesian filtering equations [10]:

$$\text{Predict step:} \quad p(\boldsymbol{\theta}_t \mid \mathbb{D}_{1:t-1}) = \int p(\boldsymbol{\theta}_t \mid \boldsymbol{\theta}_{t-1}) p(\boldsymbol{\theta}_{t-1} \mid \mathbb{D}_{1:t-1}) \, \mathrm{d}\boldsymbol{\theta}_{t-1}, \quad (3)$$

$$\text{Update step:} \quad p(\boldsymbol{\theta}_t \mid \mathbb{D}_{1:t}) \propto p(\mathbb{D}_t \mid \boldsymbol{\theta}_t) p(\boldsymbol{\theta}_t \mid \mathbb{D}_{1:t-1}). \quad (4)$$

These equations demonstrate the value of Bayesian filtering and smoothing for sequential learning as they describe an exact, recursive procedure to learn from a sequence of datasets. However, the exact Bayesian predict & update steps are intractable for all but the simplest state-space models. We will show how to efficiently approximate them with a low-rank Laplace–Gaussian filtering algorithm.

3.1 Laplace–Gaussian Filtering

The Laplace–Gaussian filter (LGF) [11] approximates the posterior with Gaussian distributions $p(\boldsymbol{\theta}_t \mid \mathbb{D}_{1:t}) \approx \mathcal{N}(\boldsymbol{\theta}_t; \boldsymbol{m}_t, \boldsymbol{C}_t)$, with mean $\boldsymbol{m}_t$ and covariance $\boldsymbol{C}_t$. This approach is commonly known as *Gaussian filtering*, and includes many well-known algorithms such as the extended Kalman filter or the unscented Kalman filter [10, 12, 13]. The predict an update steps of the LGF are as follows.

Predict step: Since we assume $p(\boldsymbol{\theta}_{t-1} \mid \mathbb{D}_{1:t-1})$ to be Gaussian with mean $\boldsymbol{m}_{t-1}$ and covariance $\boldsymbol{C}_{t-1}$, and since $p(\boldsymbol{\theta}_t \mid \boldsymbol{\theta}_{t-1})$ is Gaussian as given in Eq. (2a), the exact predictive distribution as in Eq. (3) is also Gaussian, with mean $\boldsymbol{m}_t^- = \boldsymbol{m}_{t-1}$ and covariance $\boldsymbol{C}_t^- = \boldsymbol{C}_{t-1} + \boldsymbol{Q}$. This is exactly equivalent to the predict step of a standard Kalman filter [14, 15].

Update step: The exact update of Eq. (4) is intractable as the likelihood model is not only non-linear, but also non-Gaussian and un-normalized. Therefore, we Laplace-approximate the filtering distribution with a Gaussian distribution $p(\boldsymbol{\theta}_t \mid \mathbb{D}_{1:t}) \approx \mathcal{N}(\boldsymbol{\theta}_t; \boldsymbol{m}_t, \boldsymbol{C}_t)$, where the mean $\boldsymbol{m}_t$ is chosen to be the mode of the filtering distribution, and where $\boldsymbol{C}_t$ is the inverse Hessian. This is known as Laplace–Gaussian filtering [11]. More precisely, we define the regularized loss function $\mathcal{L}_t^{\text{reg}}(\boldsymbol{\theta}_t)$ by taking the negative log of the un-normalized posterior and discarding constant terms, as

$$\mathcal{L}_t^{\text{reg}}(\boldsymbol{\theta}_t) := \mathcal{L}(\boldsymbol{\theta}_t, \mathbb{D}_t) + \tfrac{\lambda}{2} \left(\boldsymbol{\theta}_t - \boldsymbol{m}_t^-\right)^\top \left(\boldsymbol{C}_t^-\right)^{-1} \left(\boldsymbol{\theta}_t - \boldsymbol{m}_t^-\right) \propto -\log p(\boldsymbol{\theta}_t \mid \mathbb{D}_{1:t}). \quad (5)$$

Then, the mean and covariance of the Laplace-approximated filtering distribution are given by $\boldsymbol{m}_t = \arg\min_{\boldsymbol{\theta}} \mathcal{L}_t^{\text{reg}}(\boldsymbol{\theta})$ and $\boldsymbol{C}_t = \left(\nabla^2 \mathcal{L}_t^{\text{reg}}(\boldsymbol{m}_t)\right)^{-1}$. The first term is computed via optimization and the second term can be further decomposed into the loss Hessian and the prior covariance $\boldsymbol{C}_t^-$, as

$$\boldsymbol{C}_t = \left(\nabla^2 \mathcal{L}(\boldsymbol{\theta}, \mathbb{D}_t)\big|_{\boldsymbol{\theta}=\boldsymbol{m}_t} + \left(\boldsymbol{C}_t^-\right)^{-1}\right)^{-1}, \quad (6)$$

which follows from the linearity of the Hessian operator. In summary, the update step essentially consists of training the network on the new task using a regularized loss function which ensures that the network does not forget the previous tasks, and then updating the covariance of the filtering distribution based on the curvature of the un-regularized loss function and the prior covariance.

However, this does not scale to actual deep learning tasks yet, as the dense covariance matrices and the Hessian are of prohibitive size. We resolve these issues next using low-rank approximations.

3.2 Efficient Low-Rank Laplace–Gaussian Filtering with the GGN

The main bottleneck of the previous algorithm in the context of deep learning lies in computing and storing the dense covariance matrices $\boldsymbol{C}_t, \boldsymbol{C}_t^- \in \mathbb{R}^{D \times D}$ and the exact Hessian. We resolve both issues together by formulating the algorithm to track only diagonal plus low-rank matrices and by using the GGN as a low-rank Hessian approximation, i.e. $\boldsymbol{H}_t \approx \sum_{b=1}^B \boldsymbol{J}_t^{(b)} \hat{\boldsymbol{H}}_t^{(b)} (\boldsymbol{J}_t^{(b)})^\top$.

More precisely, we track diagonal plus low-rank approximations of the *precision* matrices, i.e. the inverse covariance matrices $\boldsymbol{P}_t = \boldsymbol{C}_t^{-1}$, of the form $\boldsymbol{P}_t = \boldsymbol{D}_t + \boldsymbol{U}_t \boldsymbol{\Sigma}_t \boldsymbol{U}_t^\top$,

where $D_t \in \mathbb{R}^{D \times D}$ is diagonal, $U_t \in \mathbb{R}^{D \times k}$ is a tall, and $\Sigma_t \in \mathbb{R}^{k \times k}$ a dense matrix, with rank $k \ll D$. This resolves the storage and computational issues related to the covariance matrices' size: Storing the diagonal and low-rank matrices requires only $D + Dk + k^2 \ll D^2$ parameters, and the cost of matrix-vector products with the precision matrix is reduced from $\mathcal{O}(D^2)$ to $\mathcal{O}(Dk)$.

What remains is to demonstrate how to preserve the diagonal plus low-rank structure in the predict and update steps.

Diagonal plus low-rank predictive precision: Given a diagonal plus low-rank precision matrix $P_{t-1} = D_{t-1} + U_{t-1} \Sigma_{t-1} U_{t-1}^\top$ and a diagonal process noise covariance Q, the predicted precision matrix is also diagonal plus low-rank, with parameters $\left(D_t^-, U_t^-, \Sigma_t^-\right)$ given by

$$D_t^- = \left(Q + D_{t-1}^{-1}\right)^{-1}, \qquad U_t^- = \left(Q + D_{t-1}^{-1}\right)^{-1} D_{t-1}^{-1} U_{t-1}, \qquad \text{and} \tag{7a}$$

$$\Sigma_t^- = \left(\Sigma_{t-1}^{-1} + U_{t-1}^\top D_{t-1}^{-1} U_{t-1} - U_{t-1}^\top D_{t-1}^{-\top} \left(Q_{t-1} + D_{t-1}^{-1}\right)^{-1} D_{t-1}^{-1} U_{t-1}\right)^{-1} \tag{7b}$$

This follows from applying the Woodbury matrix identity twice; full derivation in Appendix C.2.

Diagonal plus low-rank filtering precision: The filtering precision matrix P_t is a sum of the predicted precision matrix and the Hessian of the loss function for task t (Eq. (6)). By approximating the full Hessian with the GGN matrix, we can write the filtering precision matrix as

$$P_t = D_t^- + U_t^- \Sigma_t^- U_t^{-\top} + \sum_{b=1}^{B} J_t^{(b)} \hat{H}_t^{(b)} \left(J_t^{(b)}\right)^\top. \tag{8}$$

To see that this is again a diagonal plus low-rank matrix, but with increased rank, we denote matrix square-roots by $A^{1/2}$, with $A^{1/2}(A^{1/2})^\top = A$, and define the matrix $W_t \in \mathbb{R}^{D \times (k+BC)}$ as

$$W_t := \left[U_t^- \left(\Sigma_t^-\right)^{1/2} \quad J_t^{(1)} \left(\hat{H}_t^{(1)}\right)^{1/2} \quad \cdots \quad J_t^{(B)} \left(\hat{H}_t^{(B)}\right)^{1/2} \right]. \tag{9}$$

The filtering precision matrix can then be written as $P_t = D_t^- + W_t W_t^\top$, but with increased rank $k+BC$. To prevent the rank inflation, we compress the matrix W_t by performing a truncated singular value decomposition (SVD) of rank k to obtain $W_t \approx \tilde{U}_t \tilde{\Sigma}_t \tilde{V}_t^\top$, with $\tilde{U}_t \in \mathbb{R}^{D \times k}$, $\tilde{V}_t \in \mathbb{R}^{k \times D}$, and diagonal $\tilde{\Sigma}_t \in \mathbb{R}^{k \times k}$. Then, the filtering precision matrix is

$$P_t = D_t^- + \tilde{U}_t \tilde{\Sigma}_t \tilde{V}_t^\top \tilde{V}_t \tilde{\Sigma}_t \tilde{U}_t^\top = D_t^- + \tilde{U}_t \tilde{\Sigma}_t^2 \tilde{U}_t^\top . =: D_t + U_t \Sigma_t U_t^\top. \tag{10}$$

We obtain an efficient, diagonal plus low-rank approximation of the filtering precision matrix, enabling the Laplace–Gaussian filter from Section 3.1 to continual deep learning (see Algorithm 1).

Algorithm 1 Low-rank Laplace–Gaussian Filter (LR-LGF)

---

1 **Input:** Initial mean m_0, initial precision $P_0 = D_0 + U_0 \Sigma_0 U_0^\top$, process noise covariance Q, loss functions $\mathcal{L}_t$ for tasks $t = 1, 2, \ldots, T$, regularization strength λ, rank k.
2 **for** $t = 1, 2, \ldots, T$
3   **Predict**
4        $\triangleright$ Compute the predicted mean m_t^- and precision $P_t^- = D_t^- + U_t^- \Sigma_t^- U_t^{-\top}$ $\triangleleft$
5        $m_t^-, \ D_t^-, \ U_t^- \leftarrow m_{t-1}, \ \left(Q + D_{t-1}^{-1}\right)^{-1}, \ D_{t-1}^{-1} U_{t-1}$
6   **Update**
7        $\triangleright$ Train the neural network on task t using the regularized loss function $\mathcal{L}_t^{\text{reg}}$ $\triangleleft$
8        $\theta_t^* \leftarrow \arg\min_\theta \mathcal{L}_t^{\text{reg}}(\theta)$
9        $\triangleright$ Compute the GGN mini-batch-wise: $H_t \approx \sum_{b=1}^{B} J_t^{(b)} \hat{H}_t^{(b)} (J_t^{(b)})^\top$ $\triangleleft$
10       $J_t^{(b)}, \hat{H}_t^{(b)} \leftarrow \left(\frac{\partial f}{\partial \theta}\right)^{(b)}, \left(\frac{\partial^2 \mathcal{L}}{\partial f^2}\right)^{(b)} \qquad$ for batch $b = 1, \ldots, B$.
11       $\triangleright$ Perform a truncated SVD $\triangleleft$
12       $\tilde{U}_t, \tilde{\Sigma}_t, \tilde{V}_t^\top \leftarrow \text{tSVD}_k \left(\left[U_t^- \left(\Sigma_t^-\right)^{1/2} \quad J_t^{(1)} \left(\hat{H}_t^{(1)}\right)^{1/2} \quad \cdots \quad J_t^{(B)} \left(\hat{H}_t^{(B)}\right)^{1/2} \right] \right)$
13       $\triangleright$ Compute the filtering mean m_t and precision $P_t = D_t + U_t \Sigma_t U_t^\top$ $\triangleleft$
14       $m_t, \ D_t, \ U_t, \ \Sigma_t \leftarrow \theta_t^*, \ D_t^-, \ \tilde{U}_t, \ \tilde{\Sigma}_t^2$
15 **Output:** Filtering means $(m_t)_{t=1}^T$ and diagonal plus low-rank precisions $(D_t, U_t, \Sigma_t)_{t=1}^T$.

3.3 Task-Specific Models via Backwards Smoothing

Until now, our focus has been on computing and storing a single model trained sequentially on tasks. However, if the tasks differ, it may be beneficial to store *task-specific* models, that are informed by all datasets—still with the restriction that the datasets are only observed sequentially. For state-space models, this is known as *smoothing* [10, 16].

Since we consider Gaussian filtering distributions, and the transition model is linear and Gaussian (Eq. (2a)), the smoothing distribution is also Gaussian, i.e. $p(\boldsymbol{\theta}_t \mid \mathbb{D}_{1:T}) \approx \mathcal{N}(\boldsymbol{\theta}_t; \boldsymbol{m}_t^s, \boldsymbol{C}_t^s)$, and its mean and covariance can be computed recursively backwards in time: Starting with $\boldsymbol{m}_T^s = \boldsymbol{m}_T$ and $\boldsymbol{C}_T^s = \boldsymbol{C}_T$, the smoothing equations are given by [16]

$$\boldsymbol{m}_t^s = \boldsymbol{m}_t + \boldsymbol{G}_t \left(\boldsymbol{m}_{t+1}^s - \boldsymbol{m}_{t+1}^- \right), \quad \text{and} \quad \boldsymbol{C}_t^s = \boldsymbol{C}_t + \boldsymbol{G}_t \left(\boldsymbol{C}_{t+1}^s - \boldsymbol{C}_{t+1}^- \right) \boldsymbol{G}_t^\top, \quad (11)$$

where $\boldsymbol{G}_t = \boldsymbol{C}_t \left(\boldsymbol{C}_{t+1}^- \right)^{-1}$ is known as the *smoothing gain*. The smoothing equations can be formulated in terms of precision matrices, and it can be shown that if the filtering precision matrix is diagonal plus low-rank, then the smoothing precision matrix is also diagonal plus low-rank (full derivation in Appendix C.3). Equation (11) also shows why the diagonal plus low-rank structure is crucial for efficient computation of the smoothing means: If $\boldsymbol{G}_t \in \mathbb{R}^{D \times D}$ were dense, the matrix-vector product would have a prohibitive computational cost of $\mathcal{O}(D^2)$, but if we use $\boldsymbol{G}_t = \left(\boldsymbol{P}_t \right)^{-1} \boldsymbol{P}_{t+1}^-$ and implement the matrix vector product as a sequential product with two diagonal plus low-rank matrices, then the computational cost is reduced to $\mathcal{O}(Dk)$ and smoothing becomes feasible.

4 Related Work

Continual learning: There are several approaches to continual learning, which Wang et al. [17] categorized as either *regularization*-, *optimization*-, *representation*-, *architecture*-, or *replay*-based (Appendix A.1). In this paper, we use a (weight) regularization-based type of approach which tackle catastrophic forgetting in neural nets by constraining the model's weight space. They aim to identify weights that are important for the previous tasks and constrain their changes in the next tasks. A scalar hyperparameter λ, used on the regularizer, allows trading off performance on previous tasks vs. the ability to learn new tasks. *Elastic Weight Consolidation* (EWC) [6] takes inspiration from neuroscience and places a quadratic constraint on the model's parameters through a regularized loss. In each task, the regularizer consists of the sum of penalties of previous tasks where the importance of the weights is captured by the diagonal Fisher information matrix. In contrast, *Online Structured Laplace Approximations* (OSLA) [3] uses a block-diagonal K-FAC [18] Hessian approximation. Their regularizer consists of a single penalty which recursively updates with the most recent log likelihood scaled by λ. The authors observe improved performance over EWC, which they mostly attribute to a more expressive Hessian approximation also capturing parameter interactions within a layer. Other weight regularization-based methods try to find better ways to represent the importance measure [19], refine the penalty [20, 21], use a expansion-renormalization approach [22, 23], or target the network [24]. EWC and OSLA are closest to our approach since we update the precision of our penalty and scale the regularizer similarly. However, the other methods use different Hessian approximations and neither approach uses a Bayesian filter or smoother.

Gaussian Processes for CL: The state-space model that we consider in this work has a continuous-time representation as a Gaussian process with a Wiener process kernel. The GP maps the task-id to the neural network weights, that is $(t \mapsto \theta_t)$, but the actual model of interest which maps data inputs to outputs, i.e. $(x \mapsto y)$, is the neural network inside the observation model. In the related works [25, 26], the GP is used to learn the function of interest $(x \mapsto y)$. There is no fundamental distinction between tasks and thus there is no prior which encodes how continuous or distinct weights between those should be, which is precisely the role of the GP in our work. Titsias et al. [27] combines GPs and NNs and considers GPs with deep kernel functions, but here again the GP is an integral part of the function of interest as it outputs the quantity of interest y.

Bayesian filtering & smoothing: A well-established formalism for dealing with sequential data is Bayesian filtering & smoothing [10], and learning the weights of neural networks with an extended Kalman filter (EKF) has been proposed already in the '90s [28, 29]. But, the EKF and related methods [10, 12, 13] suffer from quadratic memory and cubic computational costs in the state dimension, which is prohibitive for modern deep learning tasks. Therefore, a number of approximations have been proposed. Diagonal EKF approximations have been proposed for both training and online
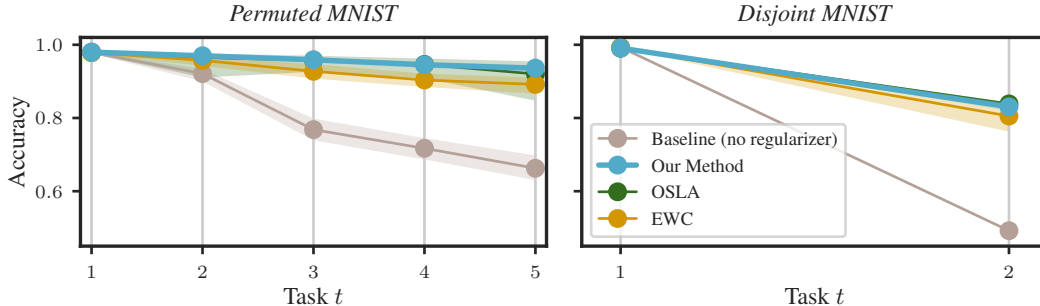
6

Figure 2: **Comparing our diagonal plus low-rank GGN approximation to other regularizers**. Mean performance of the current model on all previously encountered tasks (shaded areas are the min/max across 8 seeds). For *Permuted* and *Disjoint* MNIST we observe that our diagonal plus low-rank GGN approximation (—) leads to significantly lower rates of forgetting compared to no regularization (—). It also tends to be slightly better than EWC (—) and comparable to OSLA (—).

learning of neural networks, but as they ignore interactions between the weights their quality is often limited [4, 30]. Treating only the last layer of the network probabilistically also reduces the problem size to make the Kalman filter tractable, but also comes with a loss of expressiveness [31]. Recently, diagonal plus low-rank approximations have been proposed as a more expressive alternative and the resulting EKF variant, called LO-FI, has been shown to be effective for online learning from streaming data [5]. In contrast, we consider the continual learning problem where the data for each task $\mathbb{D}_t$ is observed in its entirety and we are concerned with the performance across past tasks. Therefore, instead of using the EKF we build on the Laplace–Gaussian filter [11], compute the MAP estimate in each filtering step via optimization, and compute the posterior with a diagonal plus low-rank Laplace approximation. This differs strongly from the EKF update and relates more closely to an *iterated* EKF (IEKF), which is known to be more accurate than the EKF [10, 32]. In addition, to improve performance on past tasks, we also propose the use of a smoother to compute task-specific models.

5 Experiments

We first demonstrate that our algorithmically efficient diagonal plus low-rank approximation of the GGN can be competitive with other weight-space regularizers proposed in the literature (Section 5.1). Next, we showcase the benefits of the filtering framework by studying how domain knowledge can be integrated via the dynamics model, specifically, the process noise matrix $\boldsymbol{Q}$ (Section 5.2). Finally, we examine the benefits of Bayesian smoothing and find that, among other things, it can boost the performance of task-specific models learned "earlier", without renewed access to data (Section 5.3).

5.1 Efficient Diagonal Plus Low-Rank Curvature Approximations

Setting: We compare our diagonal plus low-rank GGN approximation to both EWC and OSLA on two well-established In *Permuted* MNIST, each task $t \in \{1, \dots, 5\}$ consist of classifying the MNIST digits. However, each task uses a random (but for this task fixed) permutation of the image pixels. The *Disjoint* MNIST setting, consists of 2 tasks, where task $t = 1$ contains only images of labels $y \in \{0, 1, 2, 3, 4\}$, while task $t = 2$ has the remaining labels $y \in \{5, 6, 7, 8, 9\}$. For both continual learning problems, we train a small 2-layer MLP with 400 hidden units in each layer (see Appendix D.2 for full details) sequentially on all tasks. After each task, we record the current model's performance on all tasks encountered so far. For each problem and method, we tune the regularization strength λ and other hyperparameters independently on a grid, maximizing the average accuracy. We repeat each experiment with 8 different seeds and set the process noise matrix $\boldsymbol{Q}$ of our method to zero, describing a scenario where all sub-tasks essentially belong to the same general tasks, analogously to what EWC and OSLA implicitly do.

Results: Figure 2 compares EWC, OSLA, our method, and the baseline (no regularizer) on *Permuted* and *Disjoint* MNIST, with Table 2 summarizing the results after the last task. First, regularizing with a diagonal plus low-rank GGN approximation consistently leads to a significantly lower rate of
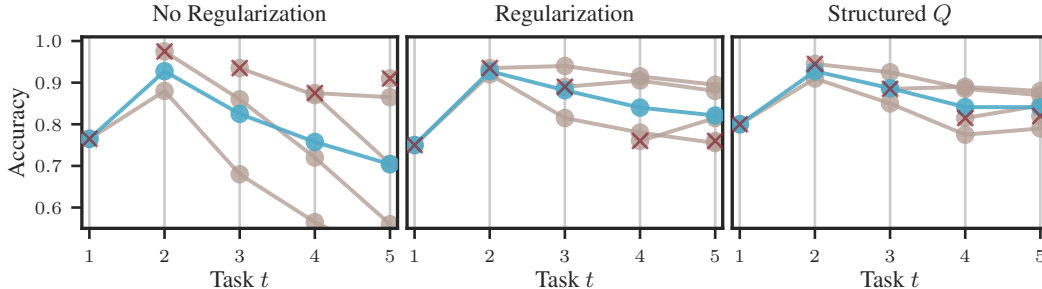
Figure 3: **The effect of Q on the average and current task's performance.** *(Left)* Without regularization, we see a significant drop in the average performance across all seen tasks (—), while the performance on the current task (✖) is strong. *(Center)* Adding regularization, helps boost the average performance across tasks, but to the detriment of the current task. However, older tasks (●) suffer much less from catastrophic forgetting. *(Right)* Additionally using a structured Q can boost the current task performance, while keeping the same average performance across tasks. See Figure 7 in Appendix D.3 for a summary of the same experimental results across 8 random seeds.

forgetting in both settings compared to no regularization. Our method's performance is slightly better than EWC and comparable to that of OSLA. This may indicate that the diagonal approximation of the Fisher used in EWC loses information. In contrast, information on the interactions between layers seems less crucial on these popular toy problems, since OSLA (which uses a block diagonal K-FAC approximation of the Hessian) and our method perform comparably. Overall, this indicates that our diagonal plus low-rank GGN approximation is competitive with other weight-space regularizers. Additionally, our approximation allows performing efficient filtering and smoothing operations on the model's parameters, which we will examine next.

5.2 Integrating Domain Knowledge via Q

We now examine the benefits of incorporating domain knowledge, i.e. how tasks are related, via the dynamics model, specifically, the process noise matrix Q. Roughly speaking, Q adds uncertainty to the next task's model parameters (see Eq. (2a)), describes how related the tasks are, and thus controls the model's flexibility across tasks. For $Q = 0$, we implicitly assume that all sub-tasks t belong to the same task, or that the optimal model parameters do not change between tasks. By using a structured Q we can incorporate how we believe the tasks and therefore the model parameters change, e.g. indicating that mostly the lower layers of the network change.

Setting: We move to a more realistic dataset, CAMELYON [33], which consists of images of either healthy or cancerous cells collected from different hospitals. To showcase the benefits of Q (and later the smoother in Section 5.3), we create a continual learning task with an *ordered* series of tasks that we term GRADUAL CAMELYON, by gradually change the brightness of the samples (all from hospital 0) between tasks. Task $t = 1$ has the darkest and the last task, $t = 5$, has the brightest pixels. We use a model with three convolutional layers followed by two dense layers (see Appendix D.2 for full experimental details), and we compare two types of transition noise: An isotropic, scalar-times-identity Q, and a structured Q with non-zero values only on the lowest layer (including biases), motivated by the knowledge that brightness changes likely mostly affect the lower convolutional layers.

Results: Figure 3 illustrates that while regularization helps boost the model's average performance across tasks this comes at the detriment of the current task's performance. By additionally using a non-zero transition noise Q, we can re-introduce some flexibility and prioritize performance on the current task t (see also Figure 7 *(right)*) while maintaining the same average accuracy on all tasks (Figure 7 *(center)*). We observe that for for the structured transition noise (Figure 3 (right)) we can better prioritize the current's task performance while keeping average accuracy high. See also Appendix D for results of our GGN approximation on other continual learning tasks that we created based on CAMELYON.

Since $Q > 0$ also indicates that task differs, in this case, it may be beneficial to store *task-specific models* instead of a single model for all tasks, which we will explore via smoothing next.
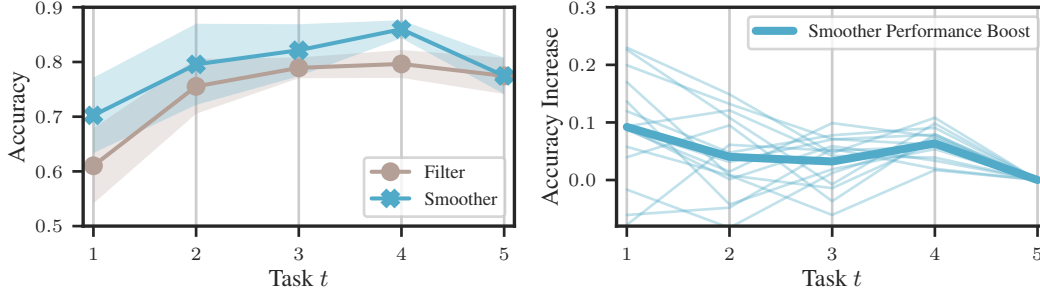
Figure 4: **Applying a smoother can significantly boost the performance on earlier tasks.** *(Left)* The performance on each individual tasks after filtering (—) or smoothing (—) up until task t. The shaded region shows one standard deviation across 15 random seeds. *(Right)* By incorporating information from all tasks, the smoother generally boosts performance without accessing any data. Thin lines show the accuracy increase on task t of smoothed vs. filtered model parameters on all 15 seeds individually. Although for a few seeds we observe a performance decrease on specific tasks, on average (thick line) we see a clear performance boost associated with the smoother.

5.3 Going Back in Time via Smoothing

We now investigate the benefits of applying a smoother to imbue previous tasks with the knowledge gained on later tasks. We return to the experimental setting of Section 5.2 and learn task-specific models $\boldsymbol{\theta}_t$, for each task $t = 1, \dots, 5$. As the filter computes posteriors $p(\boldsymbol{\theta}_t \mid \mathbb{D}_{1:t})$, the model for the last task $\boldsymbol{\theta}_5$ has incorporated information from all previous tasks. However, the model for the first task $\boldsymbol{\theta}_1$ is still only informed by the first task. Using a Bayesian smoother (Section 3.3), we can also update the first model with subsequent knowledge, *without any additional access to any data*. We simply update earlier model parameters based on the later model parameters.

This approach has promising applications in privacy-sensitive settings. By maintaining separate models for each task, the approach is particularly advantageous in scenarios where data and models cannot be shared. This is especially relevant for ICU patients or within electronic health records where hospitals often face restrictions on sharing patient data and may be required to store it only for a limited period [25]. On-device learning is another significant application [26, 34], useful in scenarios such as domestic robots or monitoring cameras.

Figure 4 illustrates the benefits of applying a smoother on GRADUAL CAMELYON. We use a similar setting to Section 5.2 but store all task-specific models and train with less data points per task. First, we observe that as more knowledge is incorporated via the filter, the performance increases (Figure 4 *(left)*). Thus, later models benefit from having been informed about earlier tasks. Then, we apply the smoother to pass this information backwards and inform earlier models about the knowledge gained on later tasks, and we observe an increased performance (Figure 4 *(left)*). In particular, we observe that while smoothing can sometimes slightly decrease the performance for particular seeds (Figure 4 *(right)*), on average it provides a significant performance boost, e.g. the accuracy on the first task increases from 61% to 70%. Employing a smoother may be particularly useful in settings where each task has only little data—and thus transferring knowledge between tasks is required—and the tasks are so different or challenging that a single model cannot accurately learn all tasks.

Bayesian smoothing also provides a grounded way to infer model parameters between two tasks, i.e. infer parameters for tasks for which we never explicitly observed any data. In our experiments (see Figure 8), we observe a performance increase of the smoothed model compared to the model obtain from a predict step in the filter. However, more work is needed to identify scenarios where this could be more beneficial than, e.g., using the model parameters of the most related tasks.

6 Conclusion

Limitations: Our diagonal plus low-rank GGN approximation stores matrices of size $1 \times D$, $D \times C$, $C \times C$, where C is the neural net's output dimension. This is beneficial if $C \ll D$, but imposes constraints if the number of classes C is large. In such cases, other curvature approximations

like K-FAC, block-diagonal or last-layer could be chosen, but these further reduce expressivity, and efficient filtering and smoothing is not straightforward. The method may be infeasible for models with millions of parameters. Furthermore, low-rank approximations with truncated SVDs can lead to an overestimated curvature, which can hurt performance. While incorporating task-specific knowledge into the dynamics model is desirable, it can be challenging to set the associated parameters well in practical applications. Finally, our experiments were designed to showcase specific classical scenarios which served as a proxy for real-world continual learning challenges and to highlight the strengths of our method.

Summary and discussion: We have presented an efficient low-rank Laplace–Gaussian filtering framework for sequential deep learning across multiple related tasks. Our approach treats the network's weights as states and the individual tasks as likelihood models in a Bayesian state-space model. To perform efficient approximate inference, we use a diagonal plus low-rank Gaussian approximation, together with a low-rank Laplace approximation via the GGN matrix to compute the filtering distributions, i.e. the posterior over the network's weights given all past and present tasks. We leverage this formalism to compute *task-specific* models via Bayesian smoothing, which incorporates knowledge from subsequent tasks into earlier models *without requiring renewed access to the data*, showing that this can enhance the performance on earlier tasks. Our methodology maps the complex problems of sequential or continual learning to well-understood Bayesian filtering and smoothing. This addresses two key challenges of continual learning from Verwimp et al. [34]: It raises computational efficiency, and conceptually clarifies the method's components, casting the relations between datasets through the observation and dynamics models of Markov Chains.

Acknowledgments

References

[1] Michael McCloskey and Neal J. Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In Gordon H. Bower, editor, *Psychology of Learning and Motivation*, volume 24, pages 109–165. Academic Press, 1989.

[2] Shibhansh Dohare, J. Fernando Hernandez-Garcia, Parash Rahman, A. Rupam Mahmood, and Richard S. Sutton. Maintaining Plasticity in Deep Continual Learning, 2023.

[3] Hippolyt Ritter, Aleksandar Botev, and David Barber. Online structured Laplace approximations for overcoming catastrophic forgetting. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 3742–3752, Red Hook, NY, USA, 2018. Curran Associates Inc.

[4] Peter G. Chang, Kevin Patrick Murphy, and Matt Jones. On diagonal approximations to the extended Kalman filter for online training of Bayesian neural networks. In *Continual Lifelong Learning Workshop at ACML 2022*, 2022. URL https://openreview.net/forum?id=asgeEt25kk.

[5] Peter G. Chang, Gerardo Durán-Martín, Alex Shestopaloff, Matt Jones, and Kevin Patrick Murphy. Low-rank extended Kalman filtering for online learning of neural networks from

streaming data. In Sarath Chandar, Razvan Pascanu, Hanie Sedghi, and Doina Precup, editors, *Proceedings of The 2nd Conference on Lifelong Learning Agents*, volume 232 of *Proceedings of Machine Learning Research*, pages 1025–1071. PMLR, 22–25 Aug 2023. URL https://proceedings.mlr.press/v232/chang23a.html.

[6] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13): 3521–3526, 2017.

[7] David J. C. MacKay. A practical Bayesian framework for backpropagation networks. *Neural Computation*, 4(3):448–472, 05 1992.

[8] Erik Daxberger, Agustinus Kristiadi, Alexander Immer, Runa Eschenhagen, Matthias Bauer, and Philipp Hennig. Laplace redux - effortless Bayesian deep learning. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=gDcaUj4Myhn.

[9] Nicol N. Schraudolph. Fast curvature matrix-vector products for second-order gradient descent. *Neural Computation*, 14(7):1723–1738, 2002. doi: 10.1162/08997660260028683.

[10] Simo Särkkä and Lennart Svensson. *Bayesian Filtering and Smoothing*. Institute of Mathematical Statistics Textbooks. Cambridge University Press, 2 edition, 2023.

[11] Shinsuke Koyama, Lucia Castellanos Pérez-Bolde, Cosma Rohilla Shalizi, and Robert E Kass. Approximate methods for state-space models. *Journal of the American Statistical Association*, 105(489):170–180, 2010.

[12] Andrew H Jazwinski. *Stochastic processes and filtering theory*. Courier Corporation, 2007.

[13] S. Julier, J. Uhlmann, and H.F. Durrant-Whyte. A new method for the nonlinear transformation of means and covariances in filters and estimators. *IEEE Transactions on Automatic Control*, 45(3):477–482, 2000. doi: 10.1109/9.847726.

[14] R. E. Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82(1):35–45, 03 1960. ISSN 0021-9223. doi: 10.1115/1.3662552. URL https://doi.org/10.1115/1.3662552.

[15] Rudolf E. Kalman and Richard S. Bucy. New results in linear filtering and prediction theory. *Journal of Basic Engineering*, 83:95–108, 1961. URL https://api.semanticscholar.org/CorpusID:8141345.

[16] H. E. Rauch, F. Tung, and C. T. Striebel. Maximum likelihood estimates of linear dynamic systems. *AIAA Journal*, 3(8):1445–1450, 1965. doi: 10.2514/3.3166. URL https://doi.org/10.2514/3.3166.

[17] Liyuan Wang, Xingxing Zhang, Hang Su, and Jun Zhu. A comprehensive survey of continual learning: Theory, method and application. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–20, 2024. doi: 10.1109/TPAMI.2024.3367329.

[18] James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, page 2408–2417. JMLR.org, 2015.

[19] Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 3987–3995. JMLR.org, 2017.

[20] Xialei Liu, Marc Masana, Luis Herranz, Joost Van de Weijer, Antonio M. López, and Andrew D. Bagdanov. Rotate your networks: Better weight consolidation and less catastrophic forgetting. In *2018 24th International Conference on Pattern Recognition (ICPR)*, pages 2262–2268, 2018. doi: 10.1109/ICPR.2018.8545895.

[21] D. Park, S. Hong, B. Han, and K. Lee. Continual learning by asymmetric loss approximation with single-side overestimation. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 3334–3343, Los Alamitos, CA, USA, nov 2019. IEEE Computer Society. doi: 10.1109/ICCV.2019.00343. URL https://doi.ieeecomputersociety.org/10.1109/ICCV.2019.00343.

[22] Sang-Woo Lee, Jin-Hwa Kim, Jaehyun Jun, Jung-Woo Ha, and Byoung-Tak Zhang. Overcoming catastrophic forgetting by incremental moment matching. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/f708f064faaf32a43e4d3c784e6af9ea-Paper.pdf.

[23] Jonathan Schwarz, Wojciech Czarnecki, Jelena Luketina, Agnieszka Grabska-Barwinska, Yee Whye Teh, Razvan Pascanu, and Raia Hadsell. Progress & compress: A scalable framework for continual learning. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4528–4537. PMLR, 10–15 Jul 2018.

[24] Cuong V. Nguyen, Yingzhen Li, Thang D. Bui, and Richard E. Turner. Variational continual learning. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=BkQqq0gRb.

[25] Pablo Moreno-Muñoz, Antonio Artés-Rodríguez, and Mauricio A. Álvarez. Continual multi-task gaussian processes, 2019.

[26] Sanyam Kapoor, Theofanis Karaletsos, and Thang D Bui. Variational auto-regressive gaussian processes for continual learning. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 5290–5300. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/kapoor21b.html.

[27] Michalis K. Titsias, Jonathan Schwarz, Alexander G. de G. Matthews, Razvan Pascanu, and Yee Whye Teh. Functional regularisation for continual learning with gaussian processes. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=HkxCzeHFDB.

[28] Sharad Singhal and Lance Wu. Training multilayer perceptrons with the extended Kalman algorithm. In D. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 1. Morgan-Kaufmann, 1988. URL https://proceedings.neurips.cc/paper_files/paper/1988/file/38b3eff8baf56627478ec76a704e9b52-Paper.pdf.

[29] Lee A Feldkamp, Danil V Prokhorov, Charles F Eagen, and Fumin Yuan. Enhanced multistream kalman filter training for recurrent networks. *Nonlinear modeling: advanced black-box techniques*, pages 29–53, 1998.

[30] G.V. Puskorius and L.A. Feldkamp. Decoupled extended Kalman filter training of feedforward layered networks. In *IJCNN-91-Seattle International Joint Conference on Neural Networks*, volume i, pages 771–777 vol.1, 1991. doi: 10.1109/IJCNN.1991.155276.

[31] Michalis Titsias, Alexandre Galashov, Amal Rannen-Triki, Razvan Pascanu, Yee Whye Teh, and Jorg Bornschein. Kalman filter for online classification of non-stationary data. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=ZzmKEpze8e.

[32] B. M. Bell and F. W. Cathey. The iterated Kalman filter update as a Gauss–Newton method. *IEEE Transaction on Automatic Control*, 38(2):294–297, 1993.

[33] Pang Wei Koh, Shiori Sagawa, Henrik Marklund, Sang Michael Xie, Marvin Zhang, Akshay Balsubramani, Weihua Hu, Michihiro Yasunaga, Richard Lanas Phillips, Irena Gao, Tony Lee, Etienne David, Ian Stavness, Wei Guo, Berton A. Earnshaw, Imran S. Haque, Sara Beery, Jure

Leskovec, Anshul Kundaje, Emma Pierson, Sergey Levine, Chelsea Finn, and Percy Liang. Wilds: A benchmark of in-the-wild distribution shifts, 2021.

[34] Eli Verwimp, Rahaf Aljundi, Shai Ben-David, Matthias Bethge, Andrea Cossu, Alexander Gepperth, Tyler L. Hayes, Eyke Hüllermeier, Christopher Kanan, Dhireesha Kudithipudi, Christoph H. Lampert, Martin Mundt, Razvan Pascanu, Adrian Popescu, Andreas S. Tolias, Joost van de Weijer, Bing Liu, Vincenzo Lomonaco, Tinne Tuytelaars, and Gido M van de Ven. Continual learning: Applications and the road forward. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=axBIMcGZn9.

[35] Michalis K. Titsias, Jonathan Schwarz, Alexander G. de G. Matthews, Razvan Pascanu, and Yee Whye Teh. Functional regularisation for continual learning with Gaussian processes. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=HkxCzeHFDB.

[36] Zhiyuan Chen and Bing Liu. *Lifelong Machine Learning*. Springer, 2018.

[37] Marcus Klasson, Hedvig Kjellstrom, and Cheng Zhang. Learn the time to learn: Replay scheduling in continual learning. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL https://openreview.net/forum?id=Q4aAITDgdP.

[38] David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy Lillicrap, and Gregory Wayne. Experience replay for continual learning. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/fa7cdfad1a5aaf8370ebeda47a1ff1c3-Paper.pdf.

[39] Seyed Iman Mirzadeh, Mehrdad Farajtabar, Dilan Gorur, Razvan Pascanu, and Hassan Ghasemzadeh. Linear mode connectivity in multitask and continual learning, 2021. URL https://arxiv.org/abs/2010.04495.

[40] Tameem Adel. Similarity-based adaptation for task-aware and task-free continual learning. *Journal of Artificial Intelligence Research*, 80:377–417, 06 2024. doi: 10.1613/jair.1.15693.

[41] Rahul Ramesh and Pratik Chaudhari. Model zoo: A growing "brain" that learns continually, 2022. URL https://arxiv.org/abs/2106.03027.

[42] Liyuan Wang, Xingxing Zhang, Qian Li, Jun Zhu, and Yi Zhong. Coscl: Cooperation of small continual learners is stronger than big one. In *Computer Vision – ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXVI*, page 254–271. Springer-Verlag, 2022. ISBN 978-3-031-19808-3. doi: 10.1007/978-3-031-19809-0_15.

[43] Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[44] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

[45] Joaquin Quionero-Candela, Masashi Sugiyama, Anton Schwaighofer, and Neil D. Lawrence. *Dataset Shift in Machine Learning*. The MIT Press, 2009. ISBN 0262170051.

Appendix

A Continual Learning

Kirkpatrick et al. [6] define continual learning as an ability to learn tasks sequentially and maintain knowledge for tasks from the past which are not experienced anymore. The aim is to have an agent that performs well across multiple tasks and can incorporate new information [3]. Our goal is to minimize the training loss summed over all tasks with the constraint that we can only access the loss of the current tasks [19]. Titsias et al. [35] mention that we should not need an extensive retraining on previous maintained data. Recent review on continual learning [17] summarizes it as observing tasks sequentially but behaving as if seeing them simultaneously. We need to obtain a balance between learning flexibility and memory stability and generalization within and between tasks. The book of lifelong learning [36] thoroughly defines continual learning and its connections to transfer, multi-task, online and reinforcement learning. The authors define that at a given point in time, a model has learned a sequence of previous tasks $t = t_1, t_2, \ldots, t_T$ with corresponding data $\mathbb{D} = \{\mathbb{D}_1, \mathbb{D}_2, \ldots, \mathbb{D}_T\}$. The tasks can be of the same or different type/domain. When the model encounters a new task t_{T+1} with data $\mathbb{D}_{T+1}$ it can leverage the past knowledge and learn the new task. Then it implements the new knowledge into the existing knowledge base. The main requirements are that the learning is in a continuous fashion, the knowledge is accumulated and the knowledge from the previous tasks can be used to learn new ones.

A.1 Additional Approaches to Continual Learning

In Section 4, we focus on regularization-based approaches to continual learning, as this is the approach used in the paper. For completeness, we also briefly summarize alternative approaches to continual learning below, following the taxonomy by Wang et al. [17].

Replay-based: Relay-based methods' key idea, mainly of experience replay, is to store a small set of training data points from the past tasks. Methods that use this approach are e.g. [37, 38]. One of the challenges is selecting representative samples and ensuring the memory buffer storage is efficient.

Optimization-based: These methods are based on changing the optimization design, e.g. by allowing gradient updates only in the orthogonal directions to previous tasks or finding only flat local minima.

Representation-based: These methods create and use representations for each tasks through self-supervised learning or pre-training.

Architecture-based: Researchers have also tried parameter allocation, model decomposition or modular networks. They either isolate some parameter subspace for a certain tasks, separate the model into task-sharing and task-specific components, or divide the network into modules dedicated to special tasks.

CL vs multi-task learning: Mirzadeh et al. [39] point out that when the model has access to all the data, it learns different solutions than when learning continuously.

Task similarity: The similarity of tasks in continual learning was studied in previous works [40–42]. Importantly, the authors state that one may benefit from the knowledge of task relatedness. In such a setting, we can employ an ensemble of small models that grows when competing (vs synergistic) tasks appear. When dissimilar tasks are learnt by one model, the tasks compete for the fixed capacity of the model. Additionally, evaluating the task similarity and initializing some components with the most similar past task' one may help with the training.

B Implementation Details

We implemented a class object has the following methods:

- `__INIT__` – creating a diagonal identity matrix and a zero low-rank term,
- `ADD_COMPUTE_INV_SUM` – inverting the precision according to the Woodbury identity, adding a diagonal matrix and final inverting

- `ADD_LOW-RANK` – adding the computed precision to the stored diagonal plus low-rank term (uses square roots), and deflating down to a memory limit (which the user provides as an optional argument, otherwise the deflation to C) via truncated SVD,

- `UPDATE_MP` – computing the vector matrix product of mean and the diagonal plus low rank precision,

C Filtering and Smoothing with Diagonal plus Low-Rank Matrices

C.1 Mathematical operations on diagonal plus low-rank matrices

In the following we describe how to perform mathematical operations on diagonal plus low-rank matrices $\boldsymbol{P} = \boldsymbol{D} + \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{U}^\top$, where $\boldsymbol{D} \in \mathbb{R}^{D \times D}$ is a diagonal matrix, $\boldsymbol{U} \in \mathbb{R}^{D \times k}$ is a tall matrix, and $\boldsymbol{\Sigma} \in \mathbb{R}^{k \times k}$ is a dense matrix, with $k \ll D$. And importantly, all of these operations should be such that they maintain the low-rank structure of the matrix such that we never need to store the full matrix in memory, and they should be computationally efficient and scale at most linearly in D.

- **Addition with a diagonal matrix:** Adding a diagonal matrix to a diagonal plus low-rank matrix results in a diagonal plus low-rank matrix:

$$\left(\boldsymbol{D} + \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{U}^\top\right) + \Lambda = \underbrace{\left(\boldsymbol{D} + \Lambda\right)}_{\boldsymbol{D}'} + \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{U}^\top \tag{12}$$

- **Matrix inversion:** The inverse of a diagonal plus low-rank matrix can be computed efficiently using the Woodbury matrix identity:

$$\left(\boldsymbol{D} + \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{U}^\top\right)^{-1} = \underbrace{\boldsymbol{D}^{-1}}_{\boldsymbol{D}'} + \underbrace{\boldsymbol{D}^{-1}\boldsymbol{U}}_{\boldsymbol{U}'}\underbrace{\left(-\left(\boldsymbol{\Sigma}^{-1} - \boldsymbol{U}^\top\boldsymbol{D}^{-1}\boldsymbol{U}\right)^{-1}\right)}_{\boldsymbol{\Sigma}'}\underbrace{\boldsymbol{U}^\top\boldsymbol{D}^{-1}}_{(\boldsymbol{U}')^\top} \tag{13}$$

- **Addition of two low-rank matrices:** Adding two diagonal plus low-rank matrices results in a diagonal plus low-rank matrix, but with increased rank $k' = k_1 + k_2$:

$$\left(\boldsymbol{D}_1 + \boldsymbol{U}_1\boldsymbol{\Sigma}_1\boldsymbol{U}_1^\top\right) + \left(\boldsymbol{D}_2 + \boldsymbol{U}_2\boldsymbol{\Sigma}_2\boldsymbol{U}_2^\top\right) = \left(\boldsymbol{D}_1 + \boldsymbol{D}_2\right) + \begin{bmatrix} \boldsymbol{U}_1 & \boldsymbol{U}_2 \end{bmatrix} \begin{bmatrix} \boldsymbol{\Sigma}_1 & 0 \\ 0 & \boldsymbol{\Sigma}_2 \end{bmatrix} \begin{bmatrix} \boldsymbol{U}_1^\top \\ \boldsymbol{U}_2^\top \end{bmatrix} \tag{14}$$

Alternatively, to keep the rank low we can perform a truncated singular value decomposition on the matrix square-root:

$$\tilde{\boldsymbol{U}}\tilde{\boldsymbol{\Sigma}}\tilde{\boldsymbol{V}}^\top = \text{tSVD}_k\left(\begin{bmatrix} \boldsymbol{U}_1\boldsymbol{\Sigma}_1^{1/2} & \boldsymbol{U}_2\boldsymbol{\Sigma}_2^{1/2} \end{bmatrix}\right) \tag{15}$$

and then approximate

$$\left(\boldsymbol{D}_1 + \boldsymbol{U}_1\boldsymbol{\Sigma}_1\boldsymbol{U}_1^\top\right) + \left(\boldsymbol{D}_2 + \boldsymbol{U}_2\boldsymbol{\Sigma}_2\boldsymbol{U}_2^\top\right) \approx \underbrace{\left(\boldsymbol{D}_1 + \boldsymbol{D}_2\right)}_{\boldsymbol{D}'} + \underbrace{\tilde{\boldsymbol{U}}}_{\boldsymbol{U}'}\underbrace{\tilde{\boldsymbol{\Sigma}}^2}_{\boldsymbol{\Sigma}'}\underbrace{\tilde{\boldsymbol{U}}^\top}_{(\boldsymbol{U}')^\top} \tag{16}$$

C.2 Low-Rank Kalman Predict Step in Information Form

Given a precision matrix P and a transition noise covariance Q, the predict step in a Gaussian filter computes the predictive precision as

$$\boldsymbol{P}^- = \left(\boldsymbol{P}^{-1} + \boldsymbol{Q}\right)^{-1}. \tag{17}$$

Now if the precision is a diagonal plus low-rank matrix $\boldsymbol{P} = \boldsymbol{D} + \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{U}^\top$, and the transition noise covariance $\boldsymbol{Q}$ is diagonal, we can show that the predictive precision is also diagonal plus low-rank. First, we apply the Woodbury matrix identity to the precision matrix:

$$\left(\boldsymbol{Q} + \boldsymbol{P}^{-1}\right)^{-1} = \left(\boldsymbol{Q} + \left(\boldsymbol{D} + \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{U}^\top\right)^{-1}\right)^{-1} \tag{18}$$

$$= \left(\boldsymbol{Q} + \boldsymbol{D}^{-1} - \boldsymbol{D}^{-1}\boldsymbol{U}\left(\boldsymbol{\Sigma}^{-1} + \boldsymbol{U}^\top\boldsymbol{D}^{-1}\boldsymbol{U}\right)^{-1}\boldsymbol{U}^\top\boldsymbol{D}^{-1}\right)^{-1}. \tag{19}$$

Defining $D' := Q + D^{-1}$, $U' := D^{-1}U$, and $\Sigma' := - \left(\Sigma^{-1} + U^\top D^{-1}U\right)^{-1}$, and applying the Woodbury matrix identity again, we get

$$\left(Q + P^{-1}\right)^{-1} = D'^{-1} - D'^{-1}U' \left(\Sigma'^{-1} + U'^\top D'^{-1}U'\right)^{-1} U'^\top D'^{-1} \tag{20}$$

This shows that the predictive precision is also diagonal plus low-rank:

$$P^- = D^- + U^-\Sigma^-(U^-)^\top, \tag{21}$$

with

$$D^- := D'^{-1} \tag{22}$$

$$= \left(Q + D^{-1}\right)^{-1} \tag{23}$$

$$U^- := D'^{-1}U' \tag{24}$$

$$= \left(Q + D^{-1}\right)^{-1} D^{-1}U, \tag{25}$$

$$\Sigma^- := - \left(\Sigma'^{-1} + U'^\top D'^{-1}U'\right)^{-1} \tag{26}$$

$$= \left(\Sigma^{-1} + U^\top D^{-1}U - U^\top D^{-\top} \left(Q + D^{-1}\right)^{-1} D^{-1}U\right)^{-1} \tag{27}$$

This, to perform the Kalman predict step in information form with a diagonal plus low-rank precision matrix, we compute and return the above quantities D^-, U^-, and Σ^-.

C.3 Low-Rank Kalman Smoother Step in Information Form

The standard Kalman smoother, or Rauch–Tung–Striebel smoother [16], computes Gaussian posterior distributions

$$p(\boldsymbol{\theta}_t \mid \mathbb{D}_{1:T}) = \mathcal{N}(\boldsymbol{\theta}_t; \boldsymbol{m}_t^s, \boldsymbol{C}_t^s) \tag{28}$$

by iterating the following backward recursion, starting with the filtering distribution $\boldsymbol{m}_T^s = \boldsymbol{m}_T$ and $\boldsymbol{C}_T^s = \boldsymbol{C}_T$:

$$\boldsymbol{G}_t = \boldsymbol{C}_t \left(\boldsymbol{C}_t^-\right)^{-1}, \tag{29}$$

$$\boldsymbol{m}_t^s = \boldsymbol{m}_t + \boldsymbol{G}_t \left(\boldsymbol{m}_{t+1}^s - \boldsymbol{m}_{t+1}^-\right), \tag{30}$$

$$\boldsymbol{C}_t^s = \boldsymbol{C}_t + \boldsymbol{G}_t \left(\boldsymbol{C}_{t+1}^s - \boldsymbol{C}_{t+1}^-\right) \boldsymbol{G}_t^\top. \tag{31}$$

Now let us formulate the smoother step in terms of diagonal plus low-rank precision matrices. Let the filtering precision at time t be $\boldsymbol{P}_t = \boldsymbol{D}_t + \boldsymbol{U}_t\boldsymbol{\Sigma}_t\boldsymbol{U}_t^\top$, the smoothing precision at time $t + 1$ be $\boldsymbol{P}_{t+1}^s = \boldsymbol{D}_{t+1}^s + \boldsymbol{U}_{t+1}^s\boldsymbol{\Sigma}_{t+1}^s(\boldsymbol{U}_{t+1}^s)^\top$, and let $\boldsymbol{Q}$ be diagonal. Recall that the predicted precision satisfies $\boldsymbol{P}_{t+1}^- = \left(\boldsymbol{P}_t^{-1} + \boldsymbol{Q}\right)^{-1}$. Then, the smoothing gain $\boldsymbol{G}_t$ is given by

$$\boldsymbol{G}_t = \boldsymbol{C}_t \left(\boldsymbol{C}_t^-\right)^{-1} = \boldsymbol{P}_t^{-1}\boldsymbol{P}_{t+1}^- = \boldsymbol{P}_t^{-1} \left(\boldsymbol{P}_t^{-1} + \boldsymbol{Q}\right)^{-1} = \left(\boldsymbol{I} + \boldsymbol{Q}\boldsymbol{P}_t\right)^{-1}. \tag{32}$$

Plugging in the diagonal plus low-rank form of the precision matrices, we get

$$\boldsymbol{G}_t = \big(\underbrace{\boldsymbol{I} + \boldsymbol{Q}\boldsymbol{D}_t}_{\boldsymbol{D}'} + \underbrace{\boldsymbol{Q}\boldsymbol{U}_t}_{\boldsymbol{U}'}\underbrace{\boldsymbol{\Sigma}_t}_{\boldsymbol{\Sigma}'}\underbrace{\boldsymbol{U}_t^\top}_{\boldsymbol{V}'^\top}\big)^{-1} \tag{33}$$

Applying the Woodbury matrix identity, we get

$$\boldsymbol{G}_t = \boldsymbol{D}'^{-1} - \boldsymbol{D}'^{-1}\boldsymbol{U}' \left(\boldsymbol{\Sigma}'^{-1} + \boldsymbol{V}'^\top \boldsymbol{D}'^{-1}\boldsymbol{U}'\right)^{-1} \boldsymbol{V}'^\top \boldsymbol{D}'^{-1}. \tag{34}$$

Therefore, the smoothing gain is diagonal plus low-rank $\boldsymbol{G}_t = \boldsymbol{D}_t^G + \boldsymbol{U}_t^G\boldsymbol{\Sigma}_t^G(\boldsymbol{V}_t^G)^\top$, with

$$\boldsymbol{D}_t^G := \boldsymbol{D}'^{-1} \tag{35}$$

$$= \left(\boldsymbol{I} + \boldsymbol{Q}\boldsymbol{D}_t\right)^{-1}, \tag{36}$$

$$\boldsymbol{U}_t^G := \boldsymbol{D}'^{-1}\boldsymbol{U}' \tag{37}$$

$$= \left(\boldsymbol{I} + \boldsymbol{Q}\boldsymbol{D}_t\right)^{-1} \boldsymbol{Q}\boldsymbol{U}_t, \tag{38}$$

$$\boldsymbol{\Sigma}_t^G := - \left(\boldsymbol{\Sigma}'^{-1} + \boldsymbol{V}'^\top \boldsymbol{D}'^{-1}\boldsymbol{U}'\right)^{-1} \tag{39}$$

$$= - \left(\boldsymbol{\Sigma}_t^{-1} + \boldsymbol{U}_t^\top \left(\boldsymbol{I} + \boldsymbol{Q}\boldsymbol{D}_t\right)^{-1} \boldsymbol{Q}\boldsymbol{U}_t\right)^{-1}, \tag{40}$$

$$\left(\boldsymbol{V}_t^G\right)^\top := \boldsymbol{V}'^\top \boldsymbol{D}'^{-1} \tag{41}$$

$$= \boldsymbol{U}_t^\top \left(\boldsymbol{I} + \boldsymbol{Q}\boldsymbol{D}_t\right)^{-1}. \tag{42}$$

This concludes the first part: Computing the smoothed mean as in Equation (30) can be done efficiently as $\boldsymbol{G}_t$ is diagonal plus low-rank.

The smoothing covariance/precision can again be approximated efficiently in a diagonal plus low-rank manner. We first re-write the smoothing covariance in terms of precisions, and re-order some terms to obtain an addition of two low-rank matrices:

$$(\boldsymbol{P}_t^s)^{-1} = (\boldsymbol{P}_t)^{-1} + \boldsymbol{G}_t \left((\boldsymbol{P}_{t+1}^s)^{-1} - (\boldsymbol{P}_{t+1}^-)^{-1} \right) \boldsymbol{G}_t^\top \tag{43}$$

$$= (\boldsymbol{P}_t)^{-1} + \boldsymbol{G}_t \left((\boldsymbol{P}_{t+1}^s)^{-1} - (\boldsymbol{P}_t)^{-1} - Q \right) \boldsymbol{G}_t^\top \tag{44}$$

$$= (\boldsymbol{P}_t)^{-1} - \boldsymbol{G}_t (\boldsymbol{P}_t)^{-1} \boldsymbol{G}_t^\top + \boldsymbol{G}_t \left((\boldsymbol{P}_{t+1}^s)^{-1} - Q \right) \boldsymbol{G}_t^\top \tag{45}$$

$$= (I - \boldsymbol{G}_t) (\boldsymbol{P}_t)^{-1} (I - \boldsymbol{G}_t)^\top + \boldsymbol{G}_t \left((\boldsymbol{P}_{t+1}^s)^{-1} - Q \right) \boldsymbol{G}_t^\top . \tag{46}$$

Then, since $\boldsymbol{P}_t$ and $\boldsymbol{P}_{t+1}^s$ are diagonal plus low-rank, their inverse is also diagonal plus low-rank (see Appendix C.1). Let $(\boldsymbol{P}_t)^{-1} = \boldsymbol{D}_t + \boldsymbol{U}_t \boldsymbol{\Sigma}_t \boldsymbol{U}_t^\top$ and $(\boldsymbol{P}_{t+1}^s)^{-1} = \boldsymbol{D}_{t+1}^s + \boldsymbol{U}_{t+1}^s \boldsymbol{\Sigma}_{t+1}^s (\boldsymbol{U}_{t+1}^s)^\top$. Then, the smoothing precision can be written as

$$(\boldsymbol{P}_t^s)^{-1} = (I - \boldsymbol{G}_t) \left(\boldsymbol{D}_t + \boldsymbol{U}_t \boldsymbol{\Sigma}_t \boldsymbol{U}_t^\top \right) (I - \boldsymbol{G}_t)^\top \tag{47}$$

$$+ \boldsymbol{G}_t \left(\boldsymbol{D}_{t+1}^s + \boldsymbol{U}_{t+1}^s \boldsymbol{\Sigma}_{t+1}^s (\boldsymbol{U}_{t+1}^s)^\top - Q \right) \boldsymbol{G}_t^\top . \tag{48}$$

Similarly to before in Appendix C.1, we can write the smoothing precision as a matrix product $(\boldsymbol{P}_t^s)^{-1} = \boldsymbol{W} \boldsymbol{W}^\top$, by defining $\boldsymbol{W}$ as

$$\boldsymbol{W} := \left[(I - \boldsymbol{G}_t) \boldsymbol{D}_t^{1/2} \quad (I - \boldsymbol{G}_t) \boldsymbol{U}_t \boldsymbol{\Sigma}_t^{1/2} \quad \boldsymbol{G}_t (\boldsymbol{D}_{t+1}^s - Q)^{1/2} \quad \boldsymbol{G}_t \boldsymbol{U}_{t+1}^s (\boldsymbol{\Sigma}_{t+1}^s)^{1/2} \right]. \tag{49}$$

Then, we can perform a truncated SVD on $\boldsymbol{W} \approx \boldsymbol{U} \boldsymbol{\Sigma} \boldsymbol{V}^\top$ to obtain a low-rank approximation of the smoothing covariance

$$(\boldsymbol{P}_t^s)^{-1} \approx \boldsymbol{U} \boldsymbol{\Sigma}^2 \boldsymbol{U}^\top . \tag{50}$$

A low-rank approximation of the precision follows again with the Woodbury matrix identity (see Appendix C.1).

D Experimental Details

D.1 Datasets

The datasets that we used are MNIST [43] and Camelyon [33]. Our implementation is in JAX [44].

In Section 5 we use *permuted* and *disjoint* MNIST. Next, we create three new settings within the CAMELYON dataset from the WILDS benchmark, a collection of datasets designed to address distribution shifts commonly encountered in real-world scenarios. These include domain generalization, where the objective is to generalize to unseen domains during training, and subpopulation shift, where class proportions differ. We adapted CAMELYON for a continual learning setup, expanding it with additional examples that are simple yet illustrative of dataset shifts, following as a guideline the work by Quionero-Candela et al. [45]. We believe they capture realistic scenarios e.g. brightness shifts due to different generations of machines, varying staining methods or imbalanced amount of data. As the GRADUAL CAMELYON was already described in section 5.2, here we only offer extended description and introduce the remaining two datasets.

GRADUAL CAMELYON: Apart from changing the brightness of the images, we apply normalization but with the same mean and standard deviation for normalizing each dataset. Since each task uses a different brightness shift, it is not negated by the shared normalization operation (in a real-world application this could happen if the machine's manufacturer has some default normalization, but each hospital might have an individual shift, e.g. due to individual machine's degradation, varying light setup, etc.). In Figure 5 we show some exemplary images from each task.

SHIFT CAMELYON: Shift from hospital 0 to hospital 1, where in $t = 0$ only samples from hospital 0 are present, in $t = 1$ there is an equal amount of samples from hospitals 0 and 1 and in the last task there are only samples from hospital 1,

IMBALANCED SHIFT CAMELYON: The same as Shift CAMELYON with the difference that the number of samples in two hospitals is unbalanced.
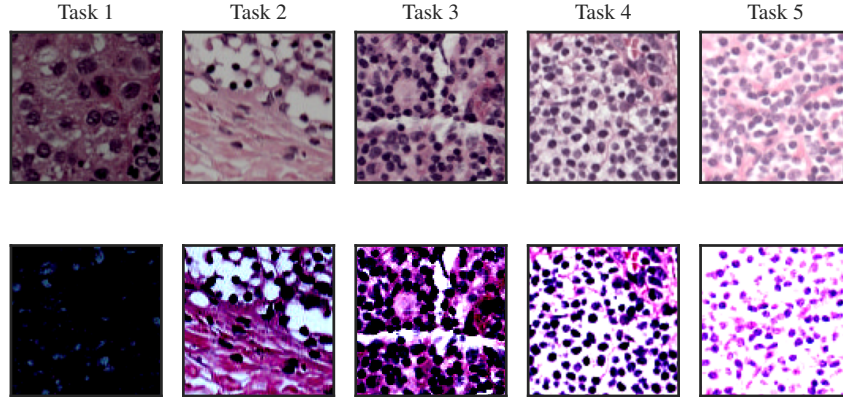
Figure 5: **Examples of the input data from** GRADUAL CAMELYON. (*Top*) To adjust the brightness, we apply a shift $x_t = x + \Delta_t$ (*bottom*) next, we normalize $(x_t - \mu_X)/\sigma_X$.
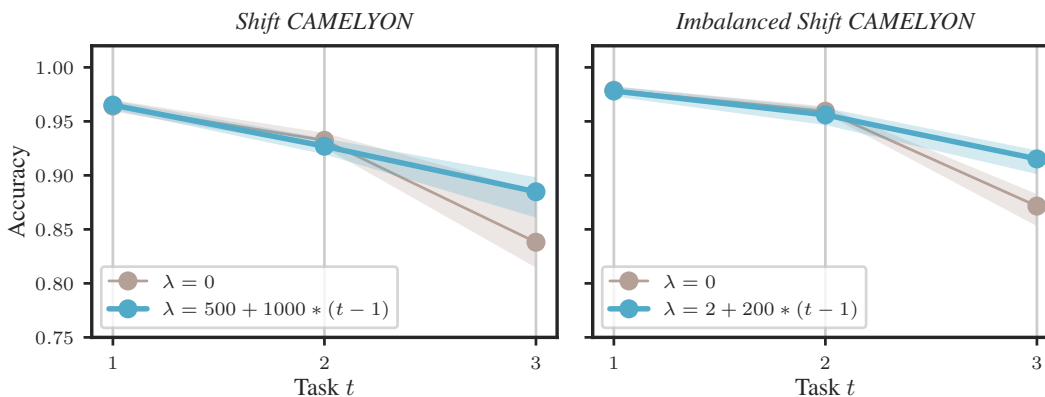

Figure 6: **Results on additional continual learning tasks derived from the Camelyon dataset.** We compare our diagonal plus low-rank GGN approximation (—) to no regularization (—). Note, that we have seen improved performance from employing a non-constant regularization strength λ, which instead scales with the task index t. The exact schedule for λ is described in the legend. Shaded areas indicate min/max performance across 8 random seeds.

D.2 Experimental Setup

Table 1 reports the values of all hyperparameters for each result presented in the paper. The number of seeds controlled all the variability i.e. initialization, data loaders, dataset splits, random operations etc. All of the experiments were run on local desktop with one NVIDIA GeForce RTX 2080 Ti with 11 GB memory.

D.3 Experimental Results

We provide additional experimental results in Table 2 and Figs. 6 and 7.

Table 1: **Full experimental setup of all presented results.** The hyperparameters were tuned via an independent search on a grid.

experiment	Section 5.1		Section 5.2, Appendix D.3				Section 5.3, Appendix D.3	
	Permuted MNIST	*Disjoint MNIST*	**Gradual Camelyon**	**Shift Camelyon**	**Imbalanced Shift Camelyon**	**Gradual Camelyon study of Q**	**Gradual Camelyon smoother**	**Gradual Camelyon inferred params**
layers	2		3 Conv 2 Dense					
units	400		32, (3x3) 16, (3x3) 4, (3x3) 8, 2					
epochs	10					5		
points	60 000	12 000	~31 000	~31 000	~53 000, ~31 000	2000		1000
λ	ours: 0.01 EWC: 100 OSLA: 1e8	ours: 1.5 EWC: 0.55 OSLA: 40	10000- 1000t		$5 + 1000*(t-1)$	$2 + 400*(t-1)$	$100000-10000*(t-1)$	$10000-1000*(t-1)$
seeds				8			15	5
Q			-			structured: 0.5 scalar: 0.5	1e10	100
batch size	128	128				32		
Hessian batch size	4	128				32		
learning rate initial λ					0.001-0.0001 0.0001			

19

Table 2: **Comparison of our diagonal low-rank GGN approximation to other regularization methods**. For *Permuted* and *Disjoint* MNIST, we compare the final *average* accuracy ($\pm$ one standard deviation as measured across 8 seeds) on all sub-tasks after sequentially learning the individual tasks using three different regularizers.

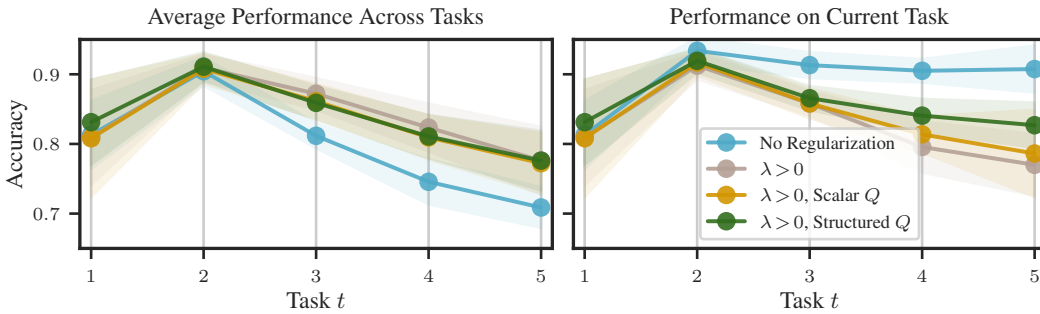|	*Permuted* MNIST	*Disjoint* MNIST
Our Method (LR-LGF, $Q\!=\!0$)	0.936 ± 0.005	0.830 ± 0.007
EWC	0.892 ± 0.015	0.805 ± 0.017
OSLA	0.920 ± 0.031	0.838 ± 0.003
Baseline (no regularizer)	0.663 ± 0.020	0.492 ± 0.001

Figure 7: **The effect of Q on the average and current task's performance (across seeds).** *(Left)* Looking at the average performance across currently observed tasks, we see that without regularization (—), performance significantly drops. With regularization (and possibly $Q\!>\!0$) (e.g. —, —, —), we can boost the average performance to roughly similar levels. *(Right)* Crucially, adding $Q\!>\!0$ allows us to boost the performance on the current task, while maintaining the same average performance. Adding a structured Q, that specifically targets the first convolutional layer (see Section 5.2), tends too perform slightly stronger. Shaded areas show $\pm$ one standard deviation across 8 random seeds.
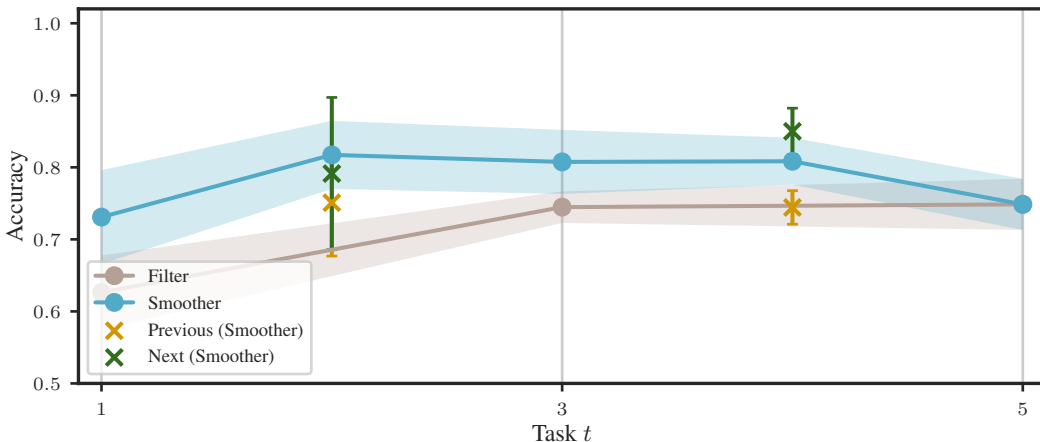

Figure 8: **Inferring model parameters "in-between" two observed tasks using Bayesian smoothing.** After filtering (—) and having seen tasks $t\!=\!1,3,5$, we apply the smoother (—). We also infer parameters for tasks $t\!=\!2,4$ which were not observed so far. We compare this to simply using the model parameter of the previous (✖), i.e. use $\boldsymbol{\theta}_1$ for task $t\!=\!2$, or next (✖), i.e. use $\boldsymbol{\theta}_3$ for task $t\!=\!2$, observed task. Shaded areas and error bars show $\pm$ one standard deviation across 8 random seeds.