



On Sequential Bayesian Inference for Continual Learning

Samuel Kessler ^{1,*} , Adam Cobb ², Tim G. J. Rudner ³, Stefan Zohren ¹ and Stephen J. Roberts ¹ 

¹ Department of Engineering Science, University of Oxford, Oxford OX2 6ED, UK; zohren@robots.ox.ac.uk (S.Z.); sjrob@robots.ox.ac.uk (S.J.R.)

² SRI International, Arlington, VA 22209, USA; adam.cobb@sri.com

³ Department of Computer Science, University of Oxford, Oxford OX1 3QG, UK; tim.rudner@cs.ox.ac.uk

* Correspondence: skessler@robots.ox.ac.uk

Abstract: Sequential Bayesian inference can be used for *continual learning* to prevent catastrophic forgetting of past tasks and provide an informative prior when learning new tasks. We revisit sequential Bayesian inference and assess whether using the previous task's posterior as a prior for a new task can prevent catastrophic forgetting in Bayesian neural networks. Our first contribution is to perform sequential Bayesian inference using Hamiltonian Monte Carlo. We propagate the posterior as a prior for new tasks by approximating the posterior via fitting a density estimator on Hamiltonian Monte Carlo samples. We find that this approach fails to prevent catastrophic forgetting, demonstrating the difficulty in performing sequential Bayesian inference in neural networks. From there, we study simple analytical examples of sequential Bayesian inference and CL and highlight the issue of model misspecification, which can lead to sub-optimal continual learning performance despite exact inference. Furthermore, we discuss how task data imbalances can cause forgetting. From these limitations, we argue that we need probabilistic models of the continual learning generative process rather than relying on sequential Bayesian inference over Bayesian neural network weights. Our final contribution is to propose a simple baseline called *Prototypical Bayesian Continual Learning*, which is competitive with the best performing Bayesian continual learning methods on class incremental continual learning computer vision benchmarks.

Keywords: continual learning; lifelong learning; sequential Bayesian inference; Bayesian deep learning; Bayesian neural networks



Citation: Kessler, S.; Cobb, A.; Rudner, T.G.J.; Zohren, S.; Roberts, S.J. On Sequential Bayesian Inference for Continual Learning. *Entropy* **2023**, *25*, 884. <https://doi.org/10.3390/e25060884>

Academic Editors: Irad E. Ben-Gal and Amichai Painsky

Received: 1 May 2023
Revised: 24 May 2023
Accepted: 28 May 2023
Published: 31 May 2023



Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

The goal of continual learning (CL) is to find a predictor that learns to solve a sequence of new tasks without losing the ability to solve previously learned tasks. One key challenge of CL with neural networks (NNs) is that model parameters from previously learned tasks are “overwritten” during gradient-based learning of new tasks, which leads to *catastrophic forgetting* of previously learned abilities [1,2]. One approach to CL hinges on using recursive applications of Bayes' Theorem, using the weight posterior in a Bayesian neural network (BNN) as the prior for a new task [3]. However, obtaining a full posterior over NN weights is computationally demanding and we often need to resort to approximations, such as the Laplace method [4] or variational inference [5,6] to obtain a neural network weight posterior.

When performing Bayesian CL, sequential Bayesian inference is performed with an approximate BNN posterior, not the true posterior [7–12]. If we consider the performance of sequential Bayesian inference with a variational approximation over a BNN weight posterior, then we barely observe an improvement over simply learning new tasks with stochastic gradient descent (SGD). We will develop this statement further in Section 2.2. Therefore, if we had access to the true BNN weight posterior, would this be enough to prevent forgetting by sequential Bayesian inference?

Our contributions in this paper are to revisit Bayesian CL. (1) Experimentally, we perform sequential Bayesian inference using the true Bayesian NN weight posterior. We do this by using the gold standard of Bayesian inference methods, Hamiltonian Monte Carlo (HMC) [13]. We use density estimation over HMC samples and use this approximate posterior density as a prior for the next task within the HMC sampling process. Surprisingly, our HMC method for CL yields no noticeable benefits over an approximate inference method (VCL Nguyen et al. [9]) despite using samples from the true posterior. (2) As a result, we consider a simple analytical example and highlight that exact inference with a misspecified model can still cause forgetting. (3) We show mathematically that under certain assumptions, task data imbalances will cause forgetting in Bayesian NNs. (4) We propose a new probabilistic model for CL and show that by explicitly modeling the generative process of the data, we can achieve good performance, avoiding the need to rely on recursive Bayesian inference over NN weights to prevent forgetting. Our proposed model, *Prototypical Bayesian Continual Learning* (ProtoCL), is conceptually simple, scalable, and competitive with state-of-the-art Bayesian CL methods in the class-incremental learning setting.

2. Background

2.1. The Continual Learning Problem

Continual learning (CL) is a learning setting whereby a model must learn to make predictions over a set of tasks sequentially while maintaining performance across all previously learned tasks. In CL, the model is sequentially shown T tasks, denoted \mathcal{T}_t for $t = 1, \dots, T$. Each task, \mathcal{T}_t , is comprised of a dataset $\mathcal{D}_t = \{(x_i, y_i)\}_{i=1}^{N_t}$, which a model needs to learn to make predictions with. More generally, tasks are denoted by distinct tuples comprised of the conditional and marginal data distributions, $\{p_t(y|\mathbf{x}), p_t(\mathbf{x})\}$. After task \mathcal{T}_t , the model will lose access to the training dataset but its performance will be continually evaluated on all tasks \mathcal{T}_i for $i \leq t$. We decompose predictors as $g = h \circ f$ such that $\hat{y} = g(x)$. We define f as an embedding function mapping $f: \mathcal{X} \rightarrow \mathcal{Z}$ and h as a head mapping to outputs $h: \mathcal{Z} \rightarrow \mathcal{Y}$. Some continual learning methods use a separate head per task $\{h_i\}_{i=1}^T$, these methods are called multi-headed while those that use one head are called single-headed.

2.2. Bayesian Continual Learning

We consider a setting in which task data arrives sequentially at timesteps, $t = 1, 2, \dots, T$. At the first timestep, $t = 1$, that is, for task \mathcal{T}_1 , the model receives the first dataset \mathcal{D}_1 and learns the conditional distribution $p(y_i|x_i, \theta)$ for all $(x_i, y_i) \in \mathcal{D}_1$ (i indexes a datapoint in \mathcal{D}_1). We denote the parameters θ as having a prior distribution $p(\theta)$ for \mathcal{T}_1 . The posterior predictive distribution for a test point $x_1^* \in \mathcal{D}_1$ is hence:

$$p(y_1^*|x_1^*, \mathcal{D}_1) = \int p(y_1^*|x_1^*, \theta)p(\theta|\mathcal{D}_1)d\theta. \quad (1)$$

We note that computing this posterior predictive distribution requires $p(\theta|\mathcal{D}_1)$. For $t = 2$, a CL model is required to fit $p(y_i|x_i, \theta)$ for $(x_i, y_i) \in \mathcal{D}_1 \cup \mathcal{D}_2$. The posterior predictive distribution for a new test point $x_2^* \in \mathcal{D}_1 \cup \mathcal{D}_2$ point is:

$$p(y_2^*|x_2^*, \mathcal{D}_1, \mathcal{D}_2) = \int p(y_2^*|x_2^*, \theta)p(\theta|\mathcal{D}_1, \mathcal{D}_2)d\theta. \quad (2)$$

The posterior must thus be updated to reflect this new conditional distribution. We can use repeated application of Bayes' rule to calculate the posterior distributions $p(\theta|\mathcal{D}_1, \dots, \mathcal{D}_T)$ as:

$$p(\theta|\mathcal{D}_1, \dots, \mathcal{D}_{T-1}, \mathcal{D}_T) = \frac{p(\mathcal{D}_T|\theta)p(\theta|\mathcal{D}_1, \dots, \mathcal{D}_{T-1})}{p(\mathcal{D}_T|\mathcal{D}_1, \dots, \mathcal{D}_{T-1})}. \quad (3)$$

In the CL setting, we lose access to previous training datasets; however, using repeated applications of Bayes' rule Equation (3) allows us to sequentially incorporate information

from past tasks in the parameters θ . At $t = 1$, we have access to \mathcal{D}_1 and the posterior over parameters is:

$$\log p(\theta|\mathcal{D}_1) = \log p(\mathcal{D}_1|\theta) + \log p(\theta) - \log p(\mathcal{D}_1). \quad (4)$$

At $t = 2$, we require $p(\theta|\mathcal{D}_1, \mathcal{D}_2)$ to calculate the posterior predictive distribution in Equation (2). However, we have lost access to \mathcal{D}_1 . According to Bayes' rule, the posterior may be written as:

$$\log p(\theta|\mathcal{D}_1, \mathcal{D}_2) = \log p(\mathcal{D}_2|\theta) + \log p(\theta|\mathcal{D}_1) - \log p(\mathcal{D}_2|\mathcal{D}_1), \quad (5)$$

where we used the conditional independence of \mathcal{D}_2 and \mathcal{D}_1 given θ . We note that the likelihood $p(\mathcal{D}_2|\theta)$ is only dependent upon the current task dataset, \mathcal{D}_2 , and that the prior $p(\theta|\mathcal{D}_1)$ encodes parameter knowledge from the previous task. Hence, we can use the posterior evaluated at t as a prior for learning a new task at $t + 1$. From Equation (3), we require that our model with parameters θ is a sufficient statistic of \mathcal{D}_1 , i.e., $p(\mathcal{D}_2|\theta, \mathcal{D}_1) = p(\mathcal{D}_2|\theta)$, making the likelihood conditionally independent of \mathcal{D}_1 given θ . This observation motivates the use of high-capacity predictors, such as Bayesian neural networks, that are flexible enough to learn from \mathcal{D}_1 .

Continual Learning Example: Split-MNIST

For the MNIST dataset [14], we know that if we were to train a BNN we would achieve good performance by inferring the posterior $p(\theta|\mathcal{D})$ and integrating out the posterior to infer the posterior predictive over a test point Equation (1). Therefore, if we were to split the dataset MNIST into 5 two-class classification tasks, then we should be able to recursively recover the multi-task posterior $p(\theta|\mathcal{D}) = p(\theta|\mathcal{D}_1 \dots, \mathcal{D}_5)$ using Equation (3). This problem is called Split-MNIST [15], where the first task involves the classification of the digits $\{0, 1\}$, the second task classification of the digits $\{2, 3\}$, and so on.

We can define three different CL settings [16–18]. When we allow the CL agent to make predictions with a task identifier τ the scenario is referred to as *task-incremental*. The identifier τ could be used to select different heads Section 2.1, for instance. This scenario is not compatible with sequential Bayesian inference outlined in Equation (3) since no task identifier is required for making predictions. *Domain-incremental* learning is another scenario that does not have access to τ during evaluation and requires the CL agent to perform classification to the same output space for each task; for example, for Split-MNIST the output space is $\{0, 1\}$ for all tasks, so this amounts to classifying between even and odd digits. Domain incremental learning is compatible with sequential Bayesian inference with a Bernoulli likelihood. The third scenario is *class-incremental* learning which also does not have access to τ but the agent needs to classify each example to its corresponding class. For Split-MNIST, for example, the output space is $\{0, \dots, 9\}$ for each task. Class-incremental learning is compatible with sequential Bayesian inference with a categorical likelihood.

2.3. Variational Continual Learning

Variational CL (VCL; Nguyen et al. [9]) simplifies the Bayesian inference problem in Equation (3) into a sequence of approximate Bayesian updates on the distribution over random neural network weights θ . To do so, VCL uses the variational posterior from previous tasks as a prior for new tasks. In this way, learning to solve the first task entails finding a variational distribution $q_1(\theta|\mathcal{D}_1)$ that maximizes a corresponding variational objective. For the subsequent task, the prior is chosen to be $q_1(\theta|\mathcal{D}_1)$, and the goal becomes to learn a variational distribution $q_2(\theta|\mathcal{D}_2)$ that maximizes a corresponding variational objective under this prior. Denoting the recursive posterior inferred from multiple datasets by $q_t(\theta|\mathcal{D}_{1:t})$, we can express the variational CL objective for the t -th task as:

$$\mathcal{L}(\theta, \mathcal{D}_t) = \mathbb{D}_{\text{KL}}[q_t(\theta)||q_{t-1}(\theta|\mathcal{D}_{1:t-1})] - \mathbb{E}_{q_t}[\log p(\mathcal{D}_t|\theta)]. \quad (6)$$

When applying VCL to the problem of Split-MNIST Figure 1, we can see that single-headed VCL barely performs better than SGD when remembering past tasks. Multi-headed VCL performs better, despite not being a requirement from sequential Bayesian inference Equation (3). Therefore, why does single-head VCL not improve over SGD if we can recursively build up an approximate posterior using Equation (3)? We hypothesize that it could be due to using a variational approximation of the posterior and so we are not actually strictly performing the Bayesian CL process described in Section 2.2. We test this hypothesis in the next section by propagating the true BNN posterior to verify whether we can recursively obtain the true multi-task posterior and so improve on single-head VCL and prevent catastrophic forgetting.

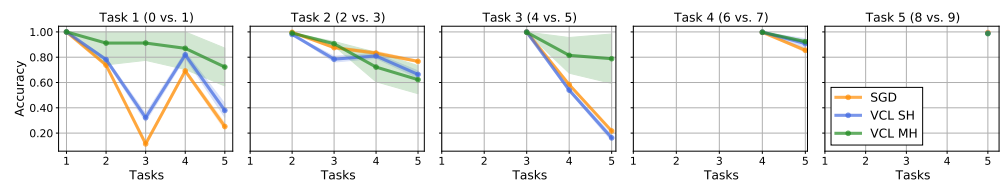


Figure 1. Accuracy on Split-MNIST for various CL methods with a two-layer BNN, all accuracies are an average and standard deviation over 10 runs with different random seeds. We compare an NN trained with SGD (single-headed) with VCL. We consider single-headed (SH) and multi-head (MH) VCL variants.

3. Bayesian Continual Learning with Hamiltonian Monte Carlo

To perform inference over BNN weights we use the HMC algorithm [13]. We then use these samples and learn a density estimator that can be used as a prior for a new task (we considered Sequential Monte Carlo, but it is unable to scale to the dimensions required for the NNs we consider [19]. HMC on the other hand has recently been successfully scaled to relatively small BNNs of the size considered in this paper [20] and ResNet models but at large computational cost [21]). HMC is considered the gold standard in approximate inference and is guaranteed to asymptotically produce samples from the true posterior (in the NeurIPS 2021 Bayesian Deep Learning Competition (https://izmailovpavel.github.io/neurips_bdl_competition), the goal was to find an approximate inference method that is as “close” as possible to the posterior samples from HMC). We use posterior samples of θ from HMC and then fit a density estimator over these samples, to use as a prior for a new task. This allows us to use a multi-modal posterior distribution over θ rather than a diagonal Gaussian variational posterior such as in VCL. More concretely, to propagate the posterior $p(\theta|\mathcal{D}_1)$ we use a density estimator, defined $\hat{p}(\theta|\mathcal{D}_1)$, to fit a probability density on HMC samples as a posterior. For the next task \mathcal{T}_2 we can use $\hat{p}(\theta|\mathcal{D}_1)$ as a prior for a new HMC sampling chain and so on (see Figure 2). The density estimator priors need to satisfy two key conditions for use within HMC sampling. Firstly, that they are a probability density function. Secondly, that they are differentiable with respect to the input samples.

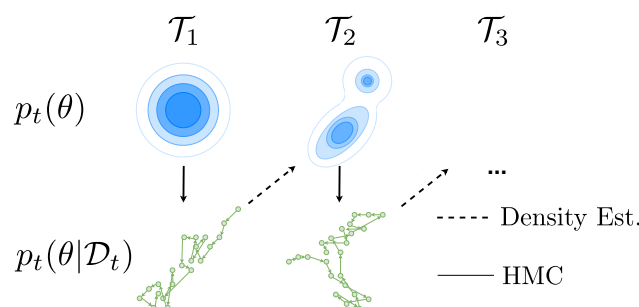


Figure 2. Illustration of the posterior propagation process; priors in blue are in the top row and posterior samples on the bottom row. This is a two-step process where we first perform HMC with an isotropic Gaussian prior for \mathcal{T}_1 then perform density estimation on the HMC samples from the posterior to obtain $\hat{p}_1(\theta|\mathcal{D}_1)$. This posterior can then be used as a prior for the new task \mathcal{T}_2 and so on.

We use a toy dataset (Figure 3) with two classes and inputs $x \in \mathbb{R}^2$ [22]. Each task is a binary classification problem where the decision boundary extends from left to right for each new task. We train a two-layer BNN, with a hidden state size of 10. We use Gaussian Mixture Models (GMM) as a density estimator for approximating the posterior with HMC samples. We also tried Normalizing Flows which should be more flexible [23]; however, these did not work robustly for HMC sampling (RealNVP was very sensitive to the choice of random seed, the samples from the learned distribution did not give accurate predictions for the current task and led to numerical instabilities when used as a prior within HMC sampling). To the best of our knowledge, we are the first to incorporate flexible priors into the sampling methods such as HMC.

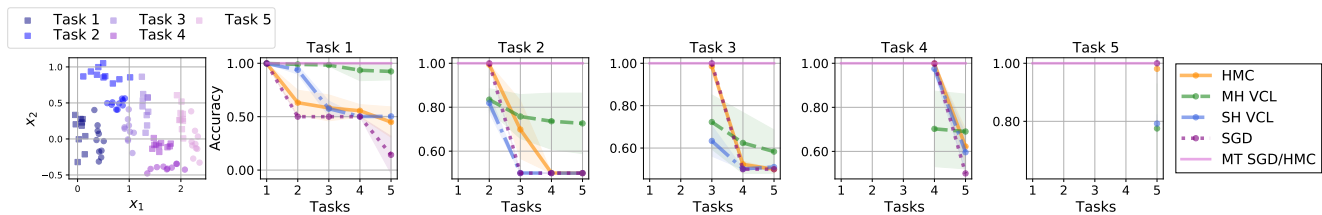


Figure 3. On the left is the toy dataset of 5 distinct 2-way classification tasks that involve classifying circles and squares [22]. Moreover, continual learning binary classification test accuracies over 10 seeds. The pink solid line is a multi-task (MT) baseline accuracy using SGD/HMC with the same model as for the CL experiments.

Training a BNN with HMC on the same multi-task dataset obtains a test accuracy of 1.0. Thus, the final posterior is suitable for continual learning under Equation (3) and we should be able to recursively arrive at the multi-task posterior with our recursive inference method with HMC. The results from Figure 3 demonstrate that using HMC with an approximate multi-modal posterior fails to prevent forgetting and is less effective than using multi-head VCL. In fact, multi-head VCL clearly outperforms HMC, indicating that the source of the knowledge retention is not through the propagation of the posterior but through the task-specific heads. For \mathcal{T}_2 , we use $\hat{p}(\theta|\mathcal{D}_1)$ instead of $p(\theta|\mathcal{D}_1)$ as a prior and this will bias the HMC sampling for all subsequent tasks. In the next paragraph, we detail the measures taken to ensure that our HMC chains have converged so we are sampling from the true posterior. Moreover, we assess the fidelity of the GMM density estimator with respect to the HMC samples. We also repeated these experiments with another toy dataset of five binary classification tasks where we observe similar results Appendix A.

For HMC, we ensure that we are sampling from the posterior by assessing chain convergence and effective sample sizes (Figure A5). The effective sample size measures the autocorrelation in the chain. The effective sample sizes for the HMC chains for our BNNs are similar to the literature [20]. Moreover, we ensure that the GMM approximate posterior is multi-modal and has a more complex posterior in comparison to VCL, and that the GMM samples produce equivalent results to HMC samples for the current task (Figure A4). See Appendix B for details.

The 2-d benchmarks we consider in this section are from previous works and are domain-incremental continual learning problems. The domain incremental setting is also simpler [18] than the class-incremental setting and thus a good starting point when attempting to perform exact sequential Bayesian inference. Despite this, we are not able to perform sequential Bayesian inference in BNNs despite using HMC, which is considered the gold standard of Bayesian deep learning. HMC and density estimation with a GMM produces richer, more accurate, and multi-modal posteriors. Despite this, we are still not able to sequentially build up the multi-task posterior or obtain much better results than an isotropic Gaussian posterior such as single-head VCL. The weak point of this method is the density estimation, the GMM removes probability mass over areas of the BNN weight space posterior, which is important for the new task. This demonstrates just how difficult a task it is to model BNN weight posteriors. In the next section, we study a different analytical

example of sequential Bayesian inference and look at how model misspecification and task data imbalances can cause forgetting in Bayesian CL.

4. Bayesian Continual Learning and Model Misspecification

We now consider a simple analytical example where we can perform the sequential Bayesian inference Equation (3) in closed form using conjugacy. We consider a simple setting where data points arrive online, one after another.

Observations y_1, y_2, \dots, y_t arrive online, and each observation is generated by a hidden variable $\theta_1, \theta_2, \dots, \theta_t \sim p$ where p is a probability density function. At time t , we wish to infer the *filtering distribution* $p(\theta_t | y_1, y_2, \dots, y_t)$ [24] using sequential Bayesian inference, similarly to the *Kalman filter* [25]. The likelihood is $p(y_t | \theta_t) = \mathcal{N}(y_t; f(\cdot; \theta_t), \sigma^2)$ such that the mean is parameterized by a linear model $y_t = f(\cdot; \theta_t) + \epsilon$ where $\epsilon \sim \mathcal{N}(0, \sigma^2)$ and $f(\cdot; \theta_t) = \theta_t$. We consider a Gaussian prior over the mean parameters θ such that $p(\theta_0) = \mathcal{N}(\theta_0; 0, \sigma_0^2)$. Since the conjugate prior for the mean is also Gaussian, the prior and posterior are $\mathcal{N}(\theta_{t-1}; \hat{\theta}_{t-1}, \hat{\sigma}_{t-1}^2)$ and $\mathcal{N}(\theta_t; \hat{\theta}_t, \hat{\sigma}_t^2)$. By using sequential Bayesian inference we can have closed-form update equations for our posterior parameters:

$$\hat{\theta}_t = \hat{\sigma}_t^2 \left(\frac{y_t}{\sigma^2} + \frac{\hat{\theta}_{t-1}}{\hat{\sigma}_{t-1}^2} \right) = \hat{\sigma}_t^2 \left(\sum_{i=1}^t \frac{y_i}{\sigma^2} + \frac{\hat{\theta}_0}{\hat{\sigma}_0^2} \right), \quad \frac{1}{\hat{\sigma}_t^2} = \frac{1}{\sigma^2} + \frac{1}{\hat{\sigma}_{t-1}^2}. \quad (7)$$

From Equation (7), the posterior mean follows a Gaussian distribution, where the posterior mean is a sum of the online observation and the online prior. Therefore, the posterior mean is a weighted sum of the data and the final value of the posterior is not dependent on the order of the data. We consider the situation where there is a task change (this non-stationarity is referred to as a changepoint in the time-series literature, in Figure 4A at $t = 110$). Concretely, for task 1 the dataset is generated according to $\mathcal{N}(-1, \sigma^2)$, so we want the model to regress to this task. For task 2, the data is generated according to $\mathcal{N}(1, \sigma^2)$ and so we want our continual learning agent to regress well to this task too. As with all continual learning benchmarks, we require our model to retain performance on past tasks and perform equally well on both tasks at the end of training at $t = 220$. From Figure 4A, we can see that the linear model will regress to the first dataset well, as data is seen online and the linear model is updated online. However, as is seen in data from the second task, the linear model eventually tracks the global mean over both tasks Equation (7) rather than a mean for each task Figure 4A. This is even more pronounced when there is a task dataset imbalance Figure 4B.

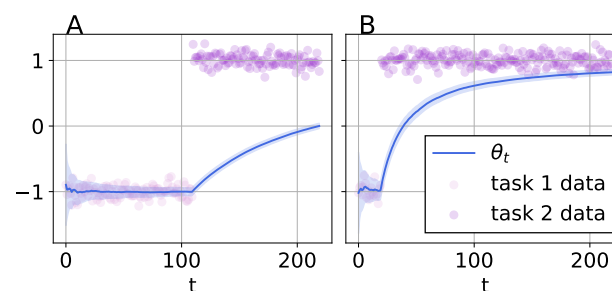


Figure 4. Posterior estimate of the filtering distribution Equation (7) for a linear model where data is divided into two tasks. We study two different scenarios with two tasks or changepoints, with a balanced and an imbalanced task dataset. In scenario **A**, we perform 110 sequential inference updates to the linear model with data from task 1, then another 110 sequential updates with data from task 2. In scenario **B**, the task datasets are imbalanced. We perform 20 sequential Bayesian updates to the linear model with data from task 1 and then 200 updates with data from task 2.

The model is clearly misspecified since a linear model cannot regress to both of these tasks simultaneously. A more suitable model would be a mixture model, which is

able to regress to both task datasets. *Despite performing exact inference, a misspecified model can forget*, Figure 4. Performance on the first task is reduced while learning the second task, this becomes even more pronounced with task dataset imbalances Figure 4B. In the case of HMC, we verified that our Bayesian neural network had a perfect performance on all tasks *beforehand*. In Figure 3, we had a well-specified model but struggled with exact sequential Bayesian inference, Equation (3). When learning with linear models online, we are performing exact inference; however, we have a misspecified model. It is important to disentangle model misspecification and exact inference and highlight that model misspecification is a caveat that has not been highlighted in the CL literature as far as we are aware. Furthermore, we can only ensure that our models are well specified if we have access to data from all tasks a priori. Therefore, in the scenario of *online continual learning* [26–28], we cannot know if our model will perform well on all past and future tasks without making assumptions on the task distributions.

5. Sequential Bayesian Inference and Imbalanced Task Data

Neural Networks are complex models with a broad hypothesis space and hence are a suitably well-specified model when tackling continual learning problems [29]. However, we struggle to fit the posterior samples from HMC to perform sequential Bayesian inference in Section 3.

We continue to use Bayesian filtering and assume a Bayesian NN where the posterior is Gaussian with a full covariance. By modeling the entire covariance, we enable modeling of how each individual weight varies with respect to all others. We do this by interpreting online learning in Bayesian NNs as filtering [30]. Our treatment is similar to Aitchison [31], who derives an optimizer by leveraging Bayesian filtering. We consider inference in the graphical model depicted in Figure 5. The aim is to infer the optimal BNN weights, θ_t^* at t given a single observation and the BNN weight prior. The previous BNN weights are used as a prior for inferring the posterior BNN parameters. We consider the online setting, where a single data point (x_t, y_t) is observed at a time.

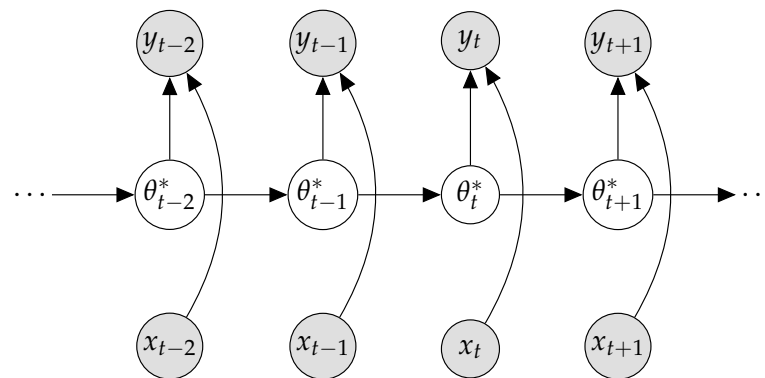


Figure 5. Graphical model for filtering. Grey and white nodes and latent variables are observed, respectively.

Instead of modeling the full covariance, we instead consider each parameter θ_i as a function of all the other parameters θ_{-it} . We also assume that the values of the weights are close to those of the previous timestep [32]. To obtain the updated equations for BNN parameters given a new observation and prior, we make two simplifying assumptions as follows.

Assumption 1. For a Bayesian neural network with output $f(x_t; \theta)$ and likelihood $\mathcal{L}(x_t, y_t; \theta)$, the derivative evaluated at θ_t is $\mathbf{z}_t = \partial \mathcal{L}(x_t, y_t; \theta) / \partial \theta |_{\theta = \theta_t}$ and the Hessian is \mathbf{H} . We assume a quadratic loss for a data point (x_t, y_t) of the form:

$$\mathcal{L}(x_t, y_t; \theta) = \mathcal{L}_t(\theta) = -\frac{1}{2} \theta^\top \mathbf{H} \theta + \mathbf{z}_t^\top \theta, \tag{8}$$

the result of a second-order Taylor expansion. The Hessian is assumed to be constant with respect to $(\mathbf{x}_t, \mathbf{y}_t)$ (but not with respect to $\boldsymbol{\theta}$).

To construct the dynamical equation for $\boldsymbol{\theta}$, consider the gradient for the i -th weight while all other parameters are set to their current estimate at the optimal value for the θ_{it}^* :

$$\theta_{it}^* = -\frac{1}{H_{ii}} \mathbf{H}_{-ii}^\top \boldsymbol{\theta}_{-it}, \tag{9}$$

since $z_{it} = 0$ at a mode. The equation above shows us that the dynamics of the optimal weight θ_{it}^* is dependent on all the other current values of the parameters $\boldsymbol{\theta}_{-it}$. The dynamics of $\boldsymbol{\theta}_{-it}$ are a complex stochastic process dependent on many different variables such as the dataset, model architecture, learning rate schedule, etc.

Assumption 2. *Since reasoning about the dynamics of $\boldsymbol{\theta}_{-it}$ is intractable, we assume that at the next timestep, the optimal weights are close to the previous timesteps with a discretized Ornstein–Uhlenbeck process for the weights $\boldsymbol{\theta}_{-it}$ with reversion speed $\vartheta \in \mathbb{R}_+$ and noise variance η_{-i}^2 :*

$$p(\boldsymbol{\theta}_{-i,t+1} | \boldsymbol{\theta}_{-i,t}) = \mathcal{N}((1 - \vartheta)\boldsymbol{\theta}_{-i,t}, \eta_{-i}^2), \tag{10}$$

this implies that the dynamics for the optimal weight are defined by

$$p(\theta_{i,t+1}^* | \theta_{i,t}^*) = \mathcal{N}((1 - \vartheta)\theta_{i,t}^*, \eta^2), \tag{11}$$

where $\eta^2 = \eta_{-i}^2 \mathbf{H}_{-ii}^\top \mathbf{H}_{-ii}$.

In simple terms, in Assumption 2, we assume a parsimonious model of the dynamics, and that the next value of $\boldsymbol{\theta}_{-i,t}$ is close to their previous value according to a Gaussian, similarly to Aitchison [31].

Lemma 1. *Under Assumptions 1 and 2 the dynamics and likelihood are Gaussian. Thus, we are able to infer the posterior distribution over the optimal weights using Bayesian updates and by linearizing the BNN the update equations for the posterior of the mean and variance of the BNN for a new data point are:*

$$\mu_{t,\text{post}} = \sigma_{t,\text{post}}^2 \left(\frac{\mu_{t,\text{prior}}}{\sigma_{t,\text{prior}}^2(\eta^2)} + \frac{\mathbf{y}_t}{\sigma^2} g(\mathbf{x}_t) \right) \quad \text{and} \quad \frac{1}{\sigma_{t,\text{post}}^2} = \frac{g(\mathbf{x}_t)^2}{\sigma^2} + \frac{1}{\sigma_{t,\text{prior}}^2(\eta^2)}, \tag{12}$$

where we drop the notation for the i -th parameter, the posterior is $\mathcal{N}(\theta_i^*; \mu_{t,\text{post}}, \sigma_{t,\text{post}}^2)$ and $g(\mathbf{x}_t) = \frac{\partial f(\mathbf{x}_t; \theta_{it}^*)}{\partial \theta_{it}^*}$ and $\sigma_{t,\text{prior}}^2$ is a function of η^2 .

See Appendix E for the derivation of Lemma 1. From Equation (12), we can notice that the posterior mean depends linearly on the prior and a data-dependent term and so will behave similarly to our previous example in Section 4. Under Assumption 1 and Assumption 2, if there is a data imbalance between tasks in Equation (12), then the data-dependent term will dominate the prior term if there is more data for the current task.

In Section 3, we showed that it is very difficult with current machine learning tools to perform sequential Bayesian inference for simple CL problems with small Bayesian NNs. When we disentangle Bayesian inference and model misspecification, we show showed that misspecified models can forget despite exact Bayesian inference. The only way to ensure that our model is well specified is to show that the multi-task posterior produces reasonable posterior predictive distributions $p(y|x, \mathcal{D}) = \int p(y|x, \mathcal{D}, \boldsymbol{\theta}) p(\boldsymbol{\theta} | \mathcal{D}) d\boldsymbol{\theta}$ for one’s application. Additionally, in this section, we have shown that if there is a task dataset size imbalance, then we can obtain forgetting under certain assumptions.

6. Related Work

There has been a recent resurgence in the field of CL [33] given the advent of deep learning. Methods that approximate sequential Bayesian inference Equation (3) have been seminal in CL's revival and have used a diagonal Laplace approximation [3,7]. The diagonal Laplace approximation has been enhanced by modeling covariances between neural network weights in the same layer [8]. Instead of the Laplace approximation, we can use a variational approximation for sequential Bayesian inference, named VCL [9,34]. The variational Gaussian variance of each Bayesian NN parameter can be used to pre-condition the learning rates of each weight and create a mask per task by using pruning [10]. Using richer priors has also been explored [11,35–38]. For example, one can learn a scaling of the Gaussian NN weight parameters for each task by learning a new variational adaptation parameter which can strengthen the contribution of a specific neuron [39]. The online Laplace approximation can be seen as a special case of VCL where the KL-divergence term Equation (6) is tempered and the temperature tends to 0 [12]. Gaussian processes have also been applied to CL problems leveraging inducing points to retain previous task functions [40,41].

Bayesian methods that regularize weights have not matched up to the performance of experience replay-based CL methods [42] in terms of accuracy on CL image classification benchmarks. Instead of regularizing high-dimensional weight spaces, regularizing task functions is a more direct approach to combat forgetting [43]. Bayesian NN weights can also be generated by a hypernetwork, where the hypernetwork needs only simple CL techniques to prevent forgetting [44]. In particular, one can leverage the duality between the Laplace approximation and Gaussian processes to develop a functional regularization approach to Bayesian CL [45] or using function-space variational inference [46,47].

In the next section, we propose a simple Bayesian continual learning baseline that models the data-generating continual learning process and performs exact sequential Bayesian inference in a low-dimensional embedding space. Previous work has explored modeling the data-generating process by inferring the joint distribution of inputs and targets $p(\mathbf{x}, \mathbf{y})$ and learning a generative model to replay data to prevent forgetting [48], and by learning a generative model per class and evaluating the likelihood of the inputs given each class $p(\mathbf{x}|\mathbf{y})$ [49].

7. Prototypical Bayesian Continual Learning

We have shown that sequential Bayes over NN parameters is very difficult (Section 3), and is only suitable for situations where the multi-task posterior is suitable for all tasks. We now show that a more fruitful approach is to model the full data-generating process of the CL problem and we propose a simple and scalable approach for doing so. In particular, we represent classes by prototypes [50,51] to prevent catastrophic forgetting. We refer to this framework as Prototypical Bayesian Continual Learning, or ProtoCL for short. This approach can be viewed as a probabilistic variant of iCarl [51], which creates embedding functions for different classes that are simply class means and predictions made by nearest neighbors. ProtoCL also bears similarities to the few-shot learning model Probabilistic Clustering for Online Classification [52], developed for few-shot image classification.

Model. ProtoCL models the generative CL process. We consider classes $j \in \{1, \dots, J\}$, generated from a categorical distribution with a Dirichlet prior:

$$y_{i,t} \sim \text{Cat}(p_{1:j}), \quad p_{1:j} \sim \text{Dir}(\alpha_t). \quad (13)$$

Images are embedded into an embedding space by an encoder, $z = f(x; w)$ with parameters w . The per class embeddings are Gaussian, and their mean has a prior which is also Gaussian:

$$z_{it}|y_{it} \sim \mathcal{N}(\bar{z}_{yt}, \Sigma_\epsilon), \quad \bar{z}_{yt} \sim \mathcal{N}(\boldsymbol{\mu}_{yt}, \Lambda_{yt}^{-1}). \quad (14)$$

See Figure 6 for an overview of the model. To alleviate forgetting in CL, ProtoCL uses a coreset of past task data to continue to embed past classes distinctly as prototypes. The posterior distribution over class probabilities $\{p_j\}_{j=1}^J$ and class embeddings $\{\bar{z}_j\}_{j=1}^J$ is denoted in short hand as $p(\theta)$ with parameters $\eta_t = \{\alpha_t, \mu_{1:J,t}, \Lambda_{1:J,t}^{-1}\}$. ProtoCL models each class prototype but does not use task-specific NN parameters or modules such as multi-head VCL. By modeling a probabilistic model over an embedding space, this allows us to use powerful embedding functions $f(\cdot; w)$ without having to parameterize them probabilistically, and so this approach is more scalable than VCL, for instance.

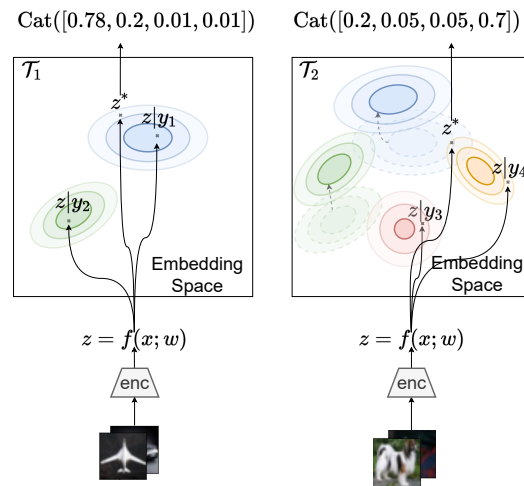


Figure 6. Overview of ProtoCL.

Inference. As the Dirichlet prior is conjugate with the categorical distribution and likewise, the Gaussian over prototypes with a Gaussian prior over the prototype mean, we can calculate posteriors in closed form and update the parameters η_t as new data is observed without using gradient-based updates. We optimize the model by maximizing the posterior predictive distribution and use a softmax over class probabilities to perform predictions. We perform gradient-based learning of the NN embedding function $f(\cdot; w)$ and update the parameters η_t at each iteration of gradient descent as well, see Algorithm 1.

Algorithm 1 ProtoCL continual learning

- 1: **Input:** task datasets $\mathcal{T}_{1:T}$, initialize embedding function: $f(\cdot; w)$, coreset: $\mathcal{M} = \emptyset$.
 - 2: **for** \mathcal{T}_1 to \mathcal{T}_T **do**
 - 3: **for** each batch in $\mathcal{T}_i \cup \mathcal{M}$ **do**
 - 4: Optimize $f(\cdot; w)$ by maximizing the posterior predictive $p(\mathbf{z}, y)$ Equation (18)
 - 5: Obtain posterior over θ by updating η , Equations (15)–(17).
 - 6: **end for**
 - 7: Add random subset from \mathcal{T}_i to \mathcal{M} .
 - 8: **end for**
-

Sequential updates. We can obtain our parameter updates for the Dirichlet posterior by Categorical-Dirichlet conjugacy:

$$\alpha_{t+1,j} = \alpha_{t,j} + \sum_{i=1}^{N_t} \mathbb{I}(y_t^i = j), \tag{15}$$

where N_t are the number of points seen during the update at timestep t . Moreover, due to Gaussian-Gaussian conjugacy, the posterior for the Gaussian prototypes is governed by:

$$\Lambda_{y_{t+1}} = \Lambda_{y_t} + N_y \Sigma_\epsilon^{-1} \quad (16)$$

$$\Lambda_{y_{t+1}} \boldsymbol{\mu}_{y_{t+1}} = N_y \Sigma_\epsilon^{-1} \bar{\mathbf{z}}_{y_t} + \Lambda_{y_t} \boldsymbol{\mu}_{y_t}, \forall y_t \in \mathcal{C}_t, \quad (17)$$

where N_y are the number of samples of class y and $\bar{\mathbf{z}}_{y_t} = (1/N_y) \sum_{i=1}^{N_y} z_{y_i}$, see Appendix D.2 for the detailed derivation.

Objective. We optimize the posterior predictive distribution of the prototypes and classes:

$$p(\mathbf{z}, y) = \int p(\mathbf{z}, y | \boldsymbol{\theta}_t; \boldsymbol{\eta}_t) p(\boldsymbol{\theta}_t; \boldsymbol{\eta}_t) d\boldsymbol{\theta}_t = p(y) \prod_{i=1}^{N_t} \mathcal{N}(\mathbf{z}_{it} | y_{it}; \boldsymbol{\mu}_{y_{i,t}}, \Sigma_\epsilon + \Lambda_{y_{i,t}}^{-1}). \quad (18)$$

where the $p(y) = \alpha_y / \sum_{j=1}^J \alpha_j$, see Appendix D.3 for the detailed derivation. This objective can then be optimized using gradient-based optimization for learning the prototype embedding function $\mathbf{z} = f(\mathbf{x}; \boldsymbol{w})$.

Predictions. To make a prediction for a test point \mathbf{x}^* , the class with the maximum (log)-posterior predictive is chosen, where the posterior predictive is:

$$p(y^* = j | \mathbf{x}^*, \mathbf{x}_{1:t}, y_{1:t}) = p(y^* = j | \mathbf{z}^*, \boldsymbol{\theta}_t) = \frac{p(y^* = j, \mathbf{z}^* | \boldsymbol{\theta}_t)}{\sum_i p(y = i, \mathbf{z}^* | \boldsymbol{\theta}_t)}, \quad (19)$$

see Appendix D.4 for further details.

Preventing forgetting. As we wish to retain the class prototypes, we make use of coresets: experience from previous tasks. At the end of learning a task \mathcal{T}_t , we retain a subset $\mathcal{M}_t \subset \mathcal{D}_t$ and augment each new task dataset to ensure that posterior parameters $\boldsymbol{\eta}_t$ and prototypes are able to retain previous task information.

Class-incremental learning. In this CL setting, we do not tell the CL agent which task it is being evaluated on with a task identifier τ . Therefore, we cannot use the task identifier to select a specific head to use for classifying a test point. Moreover, we require the CL agent to identify each class, $\{0, \dots, 9\}$ for Split-MNIST and Split-CIFAR10, and not just $\{0, 1\}$ as in domain-incremental learning. Class-incremental learning is more general, realistic, and harder a problem setting, and thus important to focus on rather than other settings, despite domain-incremental learning also being compatible with sequential Bayesian inference as described in Equation (3).

Implementation. For Split-MNIST and Split-FMNIST, the baselines and ProtoCL all use two-layer NNs with a hidden state size of 200. For Split-CIFAR10 and Split-CIFAR100, the baselines and ProtoCL use a four-layer convolution neural network with two fully connected layers of size 512 similarly to Pan et al. [22]. For ProtoCL and all baselines that rely on replay, we fix the size of the coreset to 200 points per task. For all ProtoCL models, we allow the prior Dirichlet parameters to be learned and set their initial value to 0.7 found by a random search over MNIST with ProtoCL. An important hyperparameter for ProtoCL is the embedding dimension of the Gaussian prototypes for Split-MNIST and Split-FMNIST, this was set to 128, while for the larger vision datasets, this was set to 32 found using grid-search.

Results. ProtoCL produces good results on CL benchmarks on par or better than S-FSVI [47], which is state-of-the-art among Bayesian CL methods while being a lot more efficient to train and without requiring expensive variational inference. ProtoCL can flexibly scale to larger CL vision benchmarks, producing better results than S-FSVI. The code to reproduce all experiments can be found here https://github.com/skezle/bayes_cl_posterior. All our experiments are in the more realistic class incremental learning setting, which is a harder setting than those reported in most CL papers, so the results in Table 1 are lower for certain baselines than in the respective papers. We use 200 data points per task, see Figure A6 for a sensitivity analysis of the performance over the Split-MNIST benchmark as a function of core size for ProtoCL. In Table 2 we show how ProtoCL is able to scale to larger and more challenging CL vision benchmarks. ProtoCL demonstrates competitive

performance versus the baselines we consider while at the same time requiring a fraction of the computational cost in terms of training times benchmarked on the same GPU.

Table 1. Mean accuracies across all tasks over CL vision benchmarks for *class incremental learning* [17]. All results are averages and standard errors over 10 seeds. * Uses the predictive entropy to make a decision about which head for class incremental learning.

Method	Coreset	Split-MNIST	Split-FMNIST
VCL [9]	✗	33.01 ± 0.08	32.77 ± 1.25
+ coreset	✓	52.98 ± 18.56	61.12 ± 16.96
HIBNN * [11]	✗	85.50 ± 3.20	43.70 ± 20.21
FROMP [22]	✓	84.40 ± 0.00	68.54 ± 0.00
S-FSVI [47]	✓	92.94 ± 0.17	80.55 ± 0.41
ProtoCL (ours)	✓	93.73 ± 1.05	82.73 ± 1.70

Table 2. Mean accuracies across all tasks over CL vision benchmarks for *class incremental learning* [17]. All results are averages and standard errors over 10 seeds. Training times have been benchmarked using an Nvidia RTX3090 GPU.

Method	Training Time (s) (↓)	Split CIFAR-10 (Acc) (↑)
FROMP [22]	1425 ± 28	48.92 ± 10.86
S-FSVI [47]	44434 ± 91	50.85 ± 3.87
ProtoCL (ours)	384 ± 6	55.81 ± 2.10
Split CIFAR-100 (Acc) (↑)		
S-FSVI [47]	37355 ± 1135	20.04 ± 2.37
ProtoCL (ours)	1425 ± 28	23.96 ± 1.34

The stated aim of ProtoCL is not to provide a novel state-of-the-art method for CL, but rather to propose a simple baseline that takes an alternative route than weight-space sequential Bayesian inference. We can achieve strong results that mitigate forgetting, namely by modeling the generative CL process and using sequential Bayesian inference over a few parameters in the class prototype embedding space. We argue that modeling the generative CL process is a fruitful direction for further research rather than attempting sequential Bayesian inference over the weights of a BNN. ProtoCL scales to 10 tasks of Split-CIFAR100, which, to the best of our knowledge, is the highest number of tasks and classes that have been considered by previous Bayesian continual learning methods.

8. Discussion and Conclusions

In this paper, we revisited the use of sequential Bayesian inference for CL. We can use sequential Bayes to recursively build up the multi-task posterior Equation (3). Previous methods have relied on approximate inference and see little benefit over SGD. We test the hypothesis of whether this poor performance is due to the approximate inference scheme by using HMC in two simple CL problems. HMC asymptotically samples from the true posterior, and we use a density estimator over HMC samples to use as a prior for a new task within the HMC sampling process. This density is multi-modal and accurate with respect to the current task but is not able to improve over using an approximate posterior. This demonstrates just how challenging it is to work with BNN weight posteriors. The source of error comes from the density estimation step. We then look at an analytical example of sequential Bayesian inference where we perform exact inference; however, due to model misspecification, we observe forgetting. The only way to ensure a well-specified model is to assess the multi-task performance over all tasks a priori. This might not be possible in online CL settings. We then model an analytical example over Bayesian NNs and, under certain assumptions, show that if there are task data imbalances then this will cause forgetting. Because of these results, we argue against performing weight space sequential Bayesian

inference and instead model the generative CL problem. We introduce a simple baseline called ProtoCL. ProtoCL does not require complex variational optimization and achieves competitive results to the state-of-the-art in the realistic setting of class incremental learning.

This conclusion should not be a surprise since the latest Bayesian CL papers have all relied on multi-head architectures or inducing points/coresets to prevent forgetting, rather than better weight-space inference schemes. Our observations are in line with recent theory from [53], which states that optimal CL requires perfect memory. Although the results were shown with deterministic NNs the same results follow for BNN with a single set of parameters. Future research directions include enabling coresets of task data to efficiently and accurately approximate the posterior of a BNN to remember previous tasks.

Author Contributions: S.K. lead the research including conceptualization, performing the experiments and writing the paper. S.J.R. helped with conceptualization. A.C. helped with the development of the ideas and the implementation of HMC with a density estimator as a prior. T.G.J.R. ran the S-FSVI baselines for the class incremental continual learning experiments. T.G.J.R., A.C. and S.J.R. helped to write the paper. Funding acquisition, S.Z. and S.J.R. All authors have read and agreed to the published version of the manuscript.

Funding: S.K. acknowledges funding from the Oxford-Man Institute of Quantitative Finance. T.G.J.R. acknowledges funding from the Rhodes Trust, Qualcomm, and the Engineering and Physical Sciences Research Council (EPSRC). This material is based upon work supported by the United States Air Force and DARPA under Contract No. FA8750-20-C-0002. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Air Force and DARPA.

Institutional Review Board Statement: Not applicable.

Data Availability Statement: All data is publically available, code to reproduce all experiments can be found here https://github.com/skezle/bayes_cl_posterior.

Acknowledgments: We would like to thank Sebastian Farquhar, Laurence Aitchison, Jeremias Knoblauch, and Chris Holmes for discussions. We would also like to thank Philip Ball for his help with writing the paper.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

Abbreviations

The following abbreviations are used in this manuscript:

CL	Continual Learning
NN	Neural Network
BNN	Bayesian Neural Network
HMC	Hamiltonian Monte Carlo
VCL	Variational Continual Learning
SGD	Stochastic Gradient Descent
SH	Single Head
MH	Multi-head
GMM	Gaussian Mixture Model
ProtoCL	Prototypical Bayesian Continual Learning

Appendix A. The Toy Gaussians Dataset

See Figure A1 for a visualization of the toy Gaussians dataset, which we use as a simple CL problem. This is used for evaluating our method for propagating the true posterior by using HMC for posterior inference and then using a density estimator on HMC samples as a prior for a new task. We construct 5, 2-way classification problems for CL. Each 2-way task involves classifying adjacent circles and squares Figure A1. With a 2 layer network with 10 neurons we obtain a test accuracy of 1.0 for the multi-task learning of all 5 tasks

together. Hence, according to Equation (3) a BNN with the same size should be able to learn all 5 binary classification tasks continually by sequentially building up the posterior.

Appendix B. HMC Implementation Details

We set the prior for \mathcal{T}_1 , to $p_1(\theta) = \mathcal{N}(0, \tau^{-1}\mathbb{I})$ with $\tau = 10$. We burn-in the HMC chain for 1000 steps and sample for 10,000 more steps and run 20 different chains to obtain samples from our posterior, which we then pass to our density estimator. We use a step size of 0.001 and trajectory length of $L = 20$, see Appendix C for further implementation details of the density estimation procedure. For the GMM, we optimize for the number of components by using a holdout set of HMC samples.

Appendix C. Density Estimation Diagnostics

We provide plots to show that the HMC chains indeed sample from the posterior have converged in Figures A3 and A5. We run 20 HMC sampling chains and randomly select one chain to plot for each seed (of 10). We run HMC over 10 seeds and aggregate the results Figures 3 and A1. The posteriors $p(\theta|\mathcal{D}_1), \dots$ are approximated with a GMM and used as a prior for the second task and so forth.

We provide empirical evidence to show that the density estimators have fit to HMC samples of the posterior in Figures A2 and A4, where we show the number of components of the GMM density estimator, which we use as a prior for a new task, are all multi-modal posteriors. We show the BNN accuracy when sampling BNN weights from our GMM all recover the accuracy of the converged HMC samples. The effective sample size (ESS) of the 20 chains is a measure of how correlated the samples are (higher is better). The reported ESS values for our experiments are in line with previous work which uses HMC for BNN inference [20].

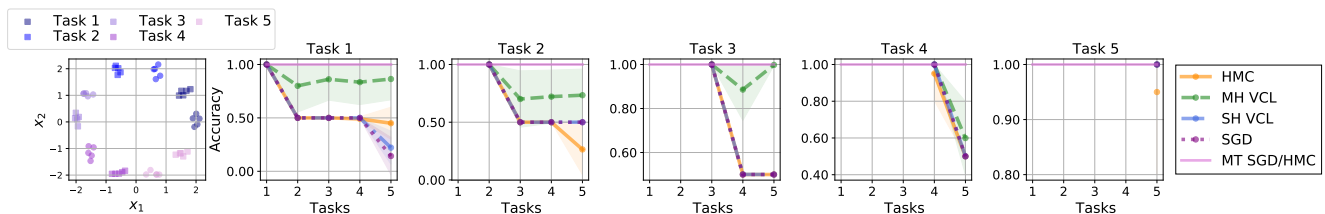


Figure A1. Continual learning binary classification accuracies from the toy Gaussian dataset similar to [44] using 10 random seeds. The pink solid line is a multi-task (MT) baseline test accuracy using SGD/HMC.

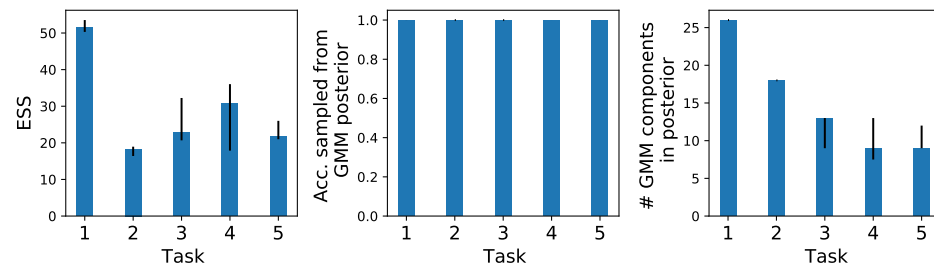


Figure A2. Diagnostics from using a GMM prior fit to samples of the posterior generated from HMC, all results are for 10 random seeds. **Left**, effective sample sizes (ESS) of the resulting HMC chains of the posterior, all are greater than those reported in other works using HMC for BNNs [20]. **Middle**, the accuracy of the BNN when using samples from the GMM density estimator instead of the samples from HMC. **Right**, The optimal number of components of each GMM posterior fitted with a holdout set of HMC samples by maximizing the likelihood.

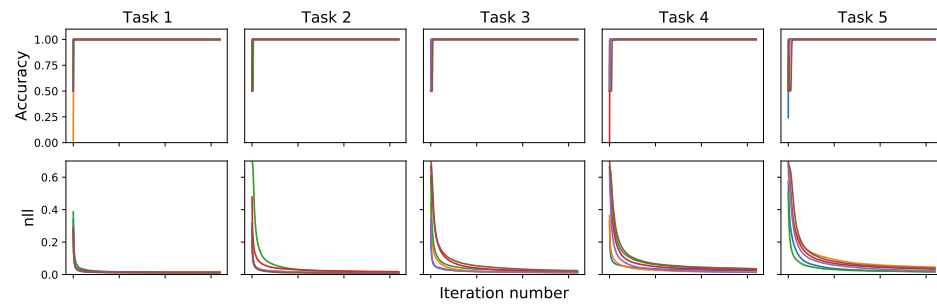


Figure A3. Convergence plots from a one randomly sampled HMC chain (of 20) for each task over 10 different runs (seeds) for 5 tasks from the toy Gaussian dataset similar to Henning et al. [44] (visualized in Figure A1). We use a GMM density estimator as the prior conditioned on the previous task data.

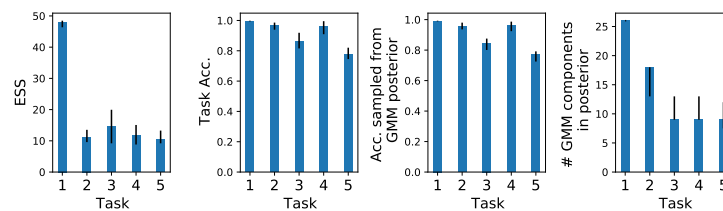


Figure A4. Diagnostics from using a GMM to fit samples of the posterior HMC samples, all results are for 10 random seeds on the toy dataset from Pan et al. [22] (and visualized in Figure 3). **Left**, effective sample sizes (ESS) of the resulting HMC chains of the posterior, all are greater than those reported in other works using HMC for BNNs [20]. **Middle left**, the current task accuracy from HMC sampling. **Middle right**, the accuracy of the BNN when using samples from the GMM density estimator instead of the converged HMC samples. **Right**, The optimal number of components of each GMM posterior fitted with a holdout set of HMC samples by maximizing the likelihood.

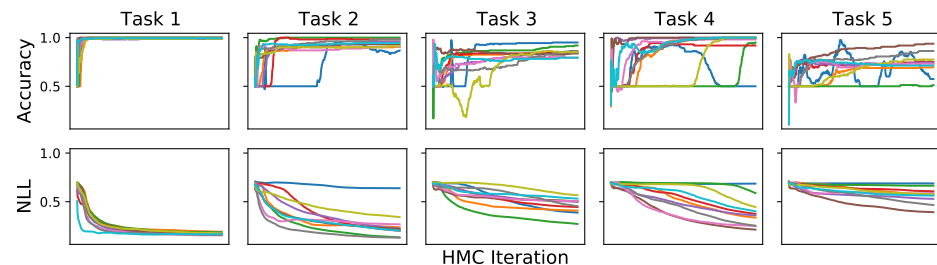


Figure A5. Convergence plots from a randomly sampled HMC chain (of 20) for each task over 10 different seeds for 5 tasks from the toy dataset from [22] (see Figure 3 for a visualization of the data). We use a GMM density estimator as a prior.

Appendix D. Prototypical Bayesian Continual Learning

ProtoCL models the generative process of CL where new tasks are comprised of new classes $j \in \{1, \dots, J\}$ of a total of J and can be modeled by using a categorical distribution with a Dirichlet prior:

$$y_{i,t} \sim \text{Cat}(p_{1:j}), \quad p_{1:j} \sim \text{Dir}(\alpha_t). \tag{A1}$$

We learn a joint embedding space for our data with a NN, $z = f(x; w)$ with parameters w . The embedding space for each class is Gaussian whose mean has a prior which is also Gaussian:

$$z_{it}|y_{it} \sim \mathcal{N}(\bar{z}_{yt}, \Sigma_\epsilon), \quad \bar{z}_{yt} \sim \mathcal{N}(\mu_{yt}, \Lambda_{yt}^{-1}). \tag{A2}$$

By ensuring that we have an embedding per class and using a memory of past data, we ensure that the embedding does not drift. The posterior parameters are $\eta_t = \{\alpha_t, \boldsymbol{\mu}_{1:J,t}, \Lambda_{1:J,t}^{-1}\}$.

Appendix D.1. Inference

As the Dirichlet prior is conjugate with the categorical distribution and so is the Gaussian distribution with a Gaussian prior over the mean of the embedding, then we can calculate posteriors in closed form and update our parameters as we see new data online without using gradient-based updates. We perform gradient-based learning of the NN embedding function $f(\cdot; \boldsymbol{w})$ with parameters \boldsymbol{w} . We optimize the model by maximizing the log-predictive posterior of the data and use the softmax over class probabilities to perform predictions. The posterior over class probabilities $\{p_j\}_{j=1}^J$ and class embeddings $\{\bar{\boldsymbol{z}}_{y_j}\}_{j=1}^J$ is denoted as $p(\theta)$ for short hand and has parameters are $\eta_t = \{\alpha_t, \boldsymbol{\mu}_{1:J,t}, \Lambda_{1:J,t}^{-1}\}$ are updated in closed form at each iteration of gradient descent.

Appendix D.2. Sequential Updates

We can obtain our posterior:

$$p(\boldsymbol{\theta}_t | \mathcal{D}_t) \propto p(\mathcal{D}_t | \boldsymbol{\theta}_t) p(\boldsymbol{\theta}_t) \tag{A3}$$

$$= \prod_{i=1}^{N_t} p(\boldsymbol{z}_t^i | \boldsymbol{y}_t^i; \bar{\boldsymbol{z}}_{\boldsymbol{y}_t}, \Sigma_{\epsilon, \boldsymbol{y}_t}) p(\boldsymbol{y}_t^i | p_{1:J}) p(p_{1:J}; \alpha_t) p(\bar{\boldsymbol{z}}_{\boldsymbol{y}_t}; \boldsymbol{\mu}_{\boldsymbol{y}_t,t}, \Lambda_{\boldsymbol{y}_t,t}^{-1}) \tag{A4}$$

$$= \mathcal{N}(\boldsymbol{\mu}_{t+1}, \Sigma_{t+1}) \text{Dir}(\alpha_{t+1}), \tag{A5}$$

where N_t is the number of data points seen during update t . Concentrating on the Categorical-Dirichlet conjugacy:

$$\text{Dir}(\alpha_{t+1}) \propto p(p_{1:J}; \alpha_t) \prod_{i=1}^{N_t} p(\boldsymbol{y}_t^i; p_{1:J}) \tag{A6}$$

$$\propto \prod_{j=1}^J p_j^{\alpha_j - 1} \prod_{i=1}^{N_t} \prod_{j=1}^J p_j^{\mathbb{I}(\boldsymbol{y}_t^i = j)} \tag{A7}$$

$$= \prod_{j=1}^J p_j^{\alpha_j - 1 + \sum_{i=1}^{N_t} \mathbb{I}(\boldsymbol{y}_t^i = j)}. \tag{A8}$$

Thus:

$$\alpha_{t+1,j} = \alpha_{t,j} + \sum_{i=1}^{N_t} \mathbb{I}(\boldsymbol{y}_t^i = j). \tag{A9}$$

Moreover, due to Gaussian-Gaussian conjugacy, then the posterior for the Gaussian prototype of the embedding for each class is:

$$\mathcal{N}(\boldsymbol{\mu}_{t+1}, \Lambda_{t+1}) \propto \prod_{i=1}^{N_t} \mathcal{N}(\boldsymbol{z}_t^i | \boldsymbol{y}_t^i; \bar{\boldsymbol{z}}_{\boldsymbol{y}_t}, \Sigma_{\epsilon}) \mathcal{N}(\bar{\boldsymbol{z}}_{\boldsymbol{y}_t}; \boldsymbol{\mu}_{\boldsymbol{y}_t,t}, \Lambda_{\boldsymbol{y}_t,t}^{-1}) \tag{A10}$$

$$= \prod_{\boldsymbol{y}_t \in \{1, \dots, J\}} \mathcal{N}(\boldsymbol{z}_{\boldsymbol{y}_t} | \boldsymbol{y}_t; \bar{\boldsymbol{z}}_{\boldsymbol{y}_t}, \frac{1}{N_{\boldsymbol{y}_t}} \Sigma_{\epsilon}) \mathcal{N}(\bar{\boldsymbol{z}}_{\boldsymbol{y}_t}; \boldsymbol{\mu}_{\boldsymbol{y}_t,t}, \Lambda_{\boldsymbol{y}_t,t}^{-1}) \tag{A11}$$

$$= \prod_{\boldsymbol{y}_t \in \{1, \dots, J\}} \mathcal{N}(\bar{\boldsymbol{z}}_{\boldsymbol{y}_t}; \boldsymbol{\mu}_{t+1}, \Lambda_{\boldsymbol{y}_t,t}^{-1}), \tag{A12}$$

where N_{y_t} is the number of points of class y_t from the set of all classes $C = \{1, \dots, J\}$. The update equations for the mean and variance of the posterior are:

$$\Lambda_{y_{t+1}} = \Lambda_{y_t} + N_{y_t} \Sigma_\epsilon^{-1}, \quad \forall y_t \in C_t \tag{A13}$$

$$\Lambda_{y_{t+1}} \boldsymbol{\mu}_{y_{t+1}} = N_{y_t} \Sigma_\epsilon^{-1} \bar{\mathbf{z}}_{y_t} + \Lambda_{y_t} \boldsymbol{\mu}_{y_t}, \quad \forall y_t \in C_t. \tag{A14}$$

Appendix D.3. ProtoCL Objective

The posterior predictive distribution we want to optimize is:

$$p(\mathbf{z}, y) = \int p(\mathbf{z}, y | \boldsymbol{\theta}; \eta) p(\boldsymbol{\theta}; \eta) d\boldsymbol{\theta}, \tag{A15}$$

where $p(\boldsymbol{\theta})$ denotes the distributions over class probabilities $\{p_j\}_{j=1}^J$ and mean embeddings $\{\bar{\mathbf{z}}_y\}_{y=1}^J$,

$$p(\mathbf{z}, y) = \int \prod_{i=1}^{N_t} p(\mathbf{z}_{it} | y_{it}; \bar{\mathbf{z}}_{y_t}, \Sigma_\epsilon) p(y_{it} | p_{1:J}) p(p_{1:J}; \alpha_t) p(\bar{\mathbf{z}}_{y_t}; \boldsymbol{\mu}_{y_t, t}, \Lambda_{y_t, t}^{-1}) dp_{1:J} d\bar{\mathbf{z}}_{y_t} \tag{A16}$$

$$= \int \prod_{i=1}^{N_t} p(\mathbf{z}_{it} | y_{it}; \mathbf{z}_{y_t}, \Sigma_\epsilon) p(\bar{\mathbf{z}}_{y_t}; \boldsymbol{\mu}_{y_t, t}, \Lambda_{y_t, t}^{-1}) d\bar{\mathbf{z}}_{y_t} \underbrace{\int \prod_{i=1}^{N_t} p(y_{it} | p_{1:J}) p(p_{1:J}; \alpha_t) dp_{1:J}}_{\Pi_i p(y_i) = p(y)} \tag{A17}$$

$$= p(y) \prod_{i=1}^{N_t} Z_i^{-1} \int \mathcal{N}(\bar{\mathbf{z}}_{y_t}; \mathbf{c}, C) d\bar{\mathbf{z}}_{y_t} \tag{A18}$$

$$= p(y) \prod_{i=1}^{N_t} \mathcal{N}(\mathbf{z}_{it} | y_{it}; \boldsymbol{\mu}_{y_t, t}, \Sigma_\epsilon + \Lambda_{y_t, t}^{-1}). \tag{A19}$$

where in Equation (A18) we use §8.1.8 in [54]. The term $p(y)$ is:

$$p(y) = \int p(y | p_{1:J}) p(p_{1:J}; \alpha_t) dp_{1:J} \tag{A20}$$

$$= \int p_y \frac{\Gamma(\sum_{j=1}^J \alpha_j)}{\prod_{j=1}^J \Gamma(\alpha_j)} \prod_{j=1}^J p_j^{\alpha_j - 1} dp_{1:J} \tag{A21}$$

$$= \frac{\Gamma(\sum_{j=1}^J \alpha_j)}{\prod_{j=1}^J \Gamma(\alpha_j)} \int \prod_{j=1}^J p_j^{\mathbb{I}(y=j) + \alpha_j - 1} dp_{1:J} \tag{A22}$$

$$= \frac{\Gamma(\sum_{j=1}^J \alpha_j)}{\prod_{j=1}^J \Gamma(\alpha_j)} \frac{\prod_{j=1}^J \Gamma(\mathbb{I}(y=j) + \alpha_j)}{\Gamma(1 + \sum_{j=1}^J \alpha_j)} \tag{A23}$$

$$= \frac{\Gamma(\sum_{j=1}^J \alpha_j)}{\prod_{j=1}^J \Gamma(\alpha_j)} \frac{\prod_{j=1}^J \Gamma(\mathbb{I}(y=j) + \alpha_j)}{\sum_{j=1}^J \alpha_j \Gamma(\sum_{j=1}^J \alpha_j)} \tag{A24}$$

$$= \frac{\prod_{j=1, j \neq y}^J \Gamma(\alpha_j)}{\prod_{j=1}^J \Gamma(\alpha_j)} \frac{\Gamma(1 + \alpha_y)}{\sum_{j=1}^J \alpha_j} \tag{A25}$$

$$= \frac{\prod_{j=1, j \neq y}^J \Gamma(\alpha_j)}{\prod_{j=1}^J \Gamma(\alpha_j)} \frac{\alpha_y \Gamma(\alpha_y)}{\sum_{j=1}^J \alpha_j} \tag{A26}$$

$$= \frac{\alpha_y}{\sum_{j=1}^J \alpha_j}, \tag{A27}$$

where we use the identity $\Gamma(n + 1) = n\Gamma(n)$.

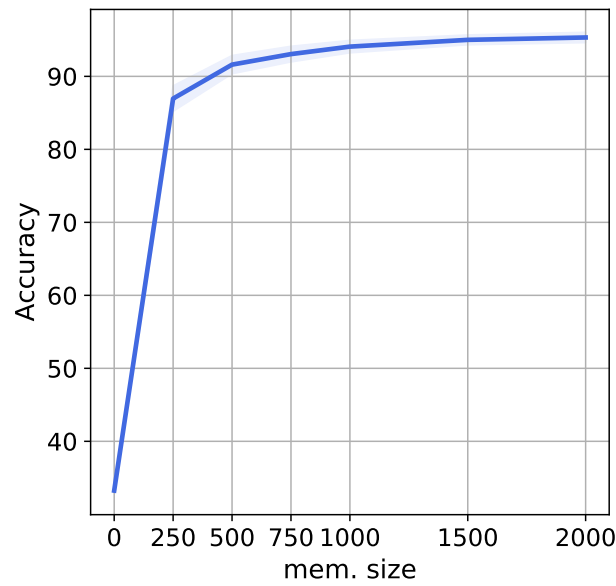


Figure A6. Split-MNIST average test accuracy over five tasks for different memory sizes. On the x-axis, we show the size of the entire memory buffer shared by all five tasks. Accuracies are over a mean and standard deviation over five different runs with different random seeds.

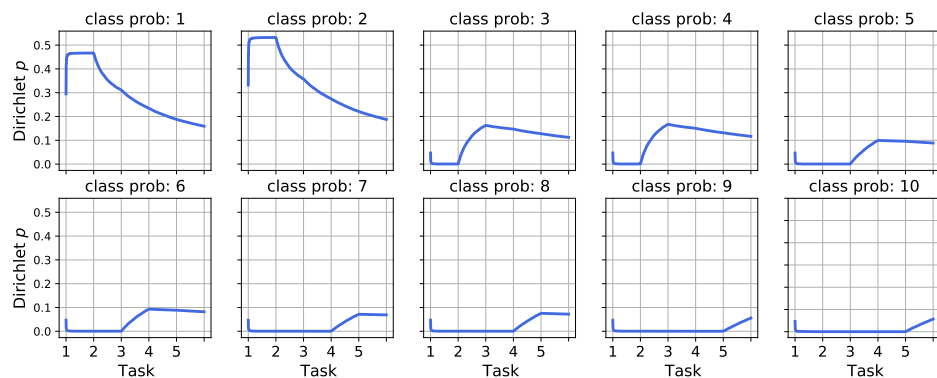


Figure A7. The evolution of the Dirichlet parameters α_t for each class in Split-MNIST tasks for ProtoCL. All α_j are shown over 10 seeds with ± 1 standard error. By the end of training, all classes are roughly equally likely, as we have trained on equal amounts of all classes.

Appendix D.4. Predictions

To make a prediction for a test point \mathbf{x}^* :

$$p(y^* = j | \mathbf{x}^*, \mathbf{x}_{1:t}, y_{1:t}) = p(y^* = j | \mathbf{z}^*, \boldsymbol{\theta}_t) \tag{A28}$$

$$= \frac{p(\mathbf{z}^* | y^* = j, \boldsymbol{\theta}_t) p(y^* = j | \boldsymbol{\theta}_t)}{\sum_i p(\mathbf{z}^* | y^* = i, \boldsymbol{\theta}_t) p(y^* = i | \boldsymbol{\theta}_t)} \tag{A29}$$

$$= \frac{p(y^* = j, \mathbf{z}^* | \boldsymbol{\theta}_t)}{\sum_i p(y = i, \mathbf{z}^* | \boldsymbol{\theta}_t)}, \tag{A30}$$

where $\boldsymbol{\theta}_t$ are sufficient statistics for $(\mathbf{x}_{1:t}, y_{1:t})$.

Preventing forgetting. As we wish to retain the task-specific prototypes, at the end of learning a task \mathcal{T}_t we take a small subset of the data as a memory to ensure that posterior parameters and prototypes do not drift, see Algorithm 1.

Appendix D.5. Experimental Setup

The prototype variance, Σ_c is set to a diagonal matrix with the variances of each prototype set to 0.05. The prototype prior precisions, Λ_{y_t} , are also diagonals and initialized randomly and exponentiated to ensure a positive semi-definite covariance for the sequential updates. The parameters $\alpha_j \forall j$ are set to 0.78, which was found by random search over the validation set on MNIST. We also allow α_j to be learned in the gradient update step in addition to the sequential update step (lines 4 and 5 Algorithm 1), see Figure A7 to see the evolution of the α_j or all classes j over the course of learning Split-MNIST.

For the Split-MNIST and Split-FMNIST benchmarks, we use an NN with two layers of size 200 and trained for 50 epochs with an Adam optimizer. We perform a grid-search over learning rates, dropout rates, and weight decay coefficients. The embedding dimension is set to 128. For the Split-CIFAR10 and Split-CIFAR100 benchmarks, we use the same network as Pan et al. [22], which consists of four convolution layers and two linear layers. We train the networks for 80 epochs for each task with the Adam optimizer with a learning rate of 1×10^{-3} . The embedding dimension is set to 32. All experiments are run on a single GPU NVIDIA RTX 3090.

Appendix E. Sequential Bayesian Estimation as Bayesian Neural Network Optimization

We shall consider inference in the graphical model depicted in Figure A8. The aim is to infer the optimal BNN weights, θ_t^* at time t given observations and the previous BNN weights. We assume a Gaussian posterior over weights with full covariance; hence, we model interactions between all weights. We shall consider the online setting where we see one data point (x_t, y_t) at a time and we will make no assumption as to whether the data comes from the same task or different tasks over the course of learning.

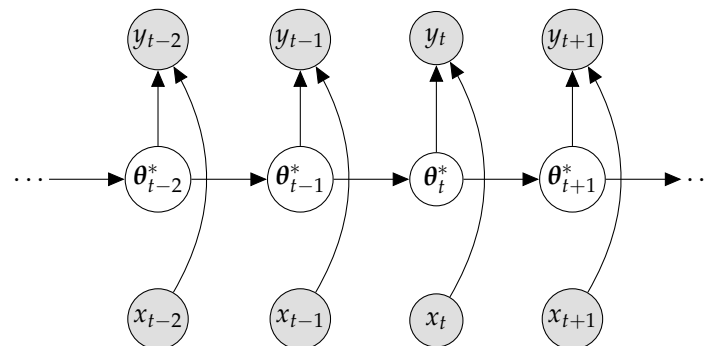


Figure A8. Graphical model of under which we perform inference in Section 5. Grey nodes are observed and white are latent variables.

We set up the problem of sequential Bayesian inference as a filtering problem and we leverage the work of Aitchison [31], which casts NN optimization as Bayesian sequential inference. We make the reasonable assumption that the distribution over weights is a Gaussian with full covariance. Since reasoning about the full covariance matrix of a BNN is intractable, we instead consider the i -th parameter and reason about the dynamics of the optimal estimates θ_{it}^* as a function of all the other parameters θ_{-it} . Each weight is functionally dependent on all others. If we had access to the full covariance of the parameters, then we could reason about the unknown optimal weights θ_{it}^* given the values of all the other weights θ_{-it} . However, since we do not have access to the full covariance, another approach is to reason about the dynamics of θ_{it}^* given the dynamics of θ_{-it} and assume that the values of the weights are close to those of the previous timestep [32] and so we cast the problem as a dynamical system.

Consider a quadratic loss of the form:

$$\mathcal{L}(x_t, y_t; \boldsymbol{\theta}) = \mathcal{L}_t(\boldsymbol{\theta}) = -\frac{1}{2} \boldsymbol{\theta}^\top \mathbf{H} \boldsymbol{\theta} + \mathbf{z}_t^\top \boldsymbol{\theta}, \tag{A31}$$

which we can arrive at by simple Taylor expansion, where \mathbf{H} is the Hessian which is assumed to be constant across data points but not across the parameters $\boldsymbol{\theta}$. If the BNN output takes the form $f(x_t; \boldsymbol{\theta})$, then the derivative evaluated at $\boldsymbol{\theta}_t$ is $\mathbf{z}_t = \frac{\partial \mathcal{L}(x_t, y_t; \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_t}$. To construct the dynamical equations for our weights, consider the gradient for a single datapoint:

$$\frac{\partial \mathcal{L}_t(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = -\mathbf{H} \boldsymbol{\theta} + \mathbf{z}_t. \tag{A32}$$

If we consider the gradient for the i -th weight while all other parameters are set to their current estimate:

$$\frac{\partial \mathcal{L}(\theta_i, \boldsymbol{\theta}_{-i})}{\partial \theta_i} = -H_{ii} \theta_{it} - \mathbf{H}_{-ii}^\top \boldsymbol{\theta}_{-it} + z_{ti}. \tag{A33}$$

When the gradient is set to zero we recover the optimal value for θ_{it} , denoted as θ_{it}^* :

$$\theta_{it}^* = -\frac{1}{H_{ii}} \mathbf{H}_{-ii}^\top \boldsymbol{\theta}_{-it}. \tag{A34}$$

since $z_{ti} = 0$ at the modes. The equation above shows us that the dynamics of the optimal weight θ_{it}^* is dependent on all the other current values of the parameters $\boldsymbol{\theta}_{-it}$. That is, the dynamics of θ_{it}^* are governed by the dynamics of the weights $\boldsymbol{\theta}_{-it}$. The dynamics of $\boldsymbol{\theta}_{-it}$ are a complex stochastic process dependent on many different variables. Since reasoning about the dynamics is intractable, we instead assume a discretized Ornstein–Uhlenbeck process for the weights $\boldsymbol{\theta}_{-it}$ with reversion speed $\vartheta \in \mathbb{R}_+$ and noise variance η_{-i}^2 :

$$p(\boldsymbol{\theta}_{-i,t+1} | \boldsymbol{\theta}_{-i,t}) = \mathcal{N}((1 - \vartheta) \boldsymbol{\theta}_{-i,t}, \eta_{-i}^2), \tag{A35}$$

this implies that the dynamics for the optimal weight are defined by

$$p(\theta_{i,t+1}^* | \theta_{i,t}^*) = \mathcal{N}((1 - \vartheta) \theta_{i,t}^*, \eta^2), \tag{A36}$$

where $\eta^2 = \eta_{-i}^2 \mathbf{H}_{-ii}^\top \mathbf{H}_{-ii}$. This same assumption is made in Aitchison [31]. This assumes a parsimonious model of the dynamics. Together with our likelihood:

$$p(y_t | x_t; \boldsymbol{\theta}_t^*) = \mathcal{N}(y_t; f(x_t; \boldsymbol{\theta}_t^*), \sigma^2) \tag{A37}$$

where $f(\cdot, \boldsymbol{\theta})$ is a neural network prediction with weights $\boldsymbol{\theta}$, we can now define a linear dynamical system for the optimal weight θ_i^* by linearizing the Bayesian NN [32] and by using the transition dynamics in Equation (A36). Thus, we are able to infer the posterior distribution over the optimal weights using Kalman filter-like updates [25]. As the dynamics and likelihood are Gaussian, then the prior and posterior are also Gaussian, for ease of notation we drop the index i such that $\theta_{it}^* = \theta_t^*$:

$$p(\theta_t^* | (x, y)_{t-1}, \dots, (x, y)_1) = \mathcal{N}(\mu_{t,\text{prior}}, \sigma_{t,\text{prior}}^2) \tag{A38}$$

$$p(\theta_t^* | (x, y)_t, \dots, (x, y)_1) = \mathcal{N}(\mu_{t,\text{post}}, \sigma_{t,\text{post}}^2) \tag{A39}$$

By using the transition dynamics and the prior we can obtain closed-form updates:

$$p(\theta_t^* | (x, y)_{t-1}, \dots, (x, y)_1) = \int p(\theta_t^* | \theta_{t-1}^*) p(\theta_{t-1}^* | (x, y)_{t-1}, \dots, (x, y)_1) d\theta_{t-1}^* \quad (\text{A40})$$

$$\mathcal{N}(\theta_t^*; \mu_{t,\text{prior}}, \sigma_{t,\text{prior}}^2) = \int \mathcal{N}(\theta_t^*; (1 - \vartheta)\theta_{t-1}^*, \eta^2) \mathcal{N}(\theta_{t-1}^*; \mu_{t-1,\text{post}}, \sigma_{t-1,\text{post}}^2) d\theta_{t-1}^*. \quad (\text{A41})$$

Integrating out θ_{t-1}^* we can obtain updates for the prior for the next timestep as follows:

$$\mu_{t,\text{prior}} = (1 - \vartheta)\mu_{t-1,\text{post}} \quad (\text{A42})$$

$$\sigma_{t,\text{prior}}^2 = \eta^2 + (1 - \vartheta)^{-2}\sigma_{t-1,\text{post}}^2. \quad (\text{A43})$$

The updates for obtaining our posterior parameters: $\mu_{t,\text{post}}$ and $\sigma_{t,\text{post}}^2$, comes from applying Bayes' theorem:

$$\log \mathcal{N}(\theta_t^*; \mu_{t,\text{post}}, \sigma_{t,\text{post}}^2) \propto \log \mathcal{N}(y_t; f(\mathbf{x}_t; \theta_t^*), \sigma^2) + \log \mathcal{N}(\theta_t^*; \mu_{t,\text{prior}}, \sigma_{t,\text{prior}}^2), \quad (\text{A44})$$

by linearizing our Bayesian NN such that $f(\mathbf{x}_t, \theta_0) \approx f(\mathbf{x}_t, \theta_0) + \frac{\partial f(\mathbf{x}_t; \theta_t^*)}{\partial \theta_t^*} (\theta_t^* - \theta_0)$ and by substituting into Equation (A44) we obtain our update equation for the posterior of the mean of our BNN parameters:

$$-\frac{1}{2\sigma_{t,\text{post}}^2} (\theta_t^* - \mu_{t,\text{post}})^2 = -\frac{1}{2\sigma^2} (y - g(\mathbf{x}_t)\theta_t^*)^2 - \frac{1}{2\sigma_{t,\text{prior}}^2} (\theta_t^* - \mu_{t,\text{prior}})^2 \quad (\text{A45})$$

$$\mu_{t,\text{post}} = \sigma_{t,\text{post}}^2 \left(\frac{\mu_{t,\text{prior}}}{\sigma_{t,\text{prior}}^2} + \frac{y}{\sigma^2} g(\mathbf{x}_t) \right), \quad (\text{A46})$$

where $g(\mathbf{x}_t) = \frac{\partial f(\mathbf{x}_t; \theta_t^*)}{\partial \theta_t^*}$, and the update equation for the variance of the Gaussian posterior is:

$$\frac{1}{\sigma_{t,\text{post}}^2} = \frac{g(\mathbf{x}_t)^2}{\sigma^2} + \frac{1}{\sigma_{t,\text{prior}}^2}. \quad (\text{A47})$$

From our updated equations, Equation (A46) and Equation (A47), we can notice that the posterior mean depends linearly on the prior and an additional data dependent term. These equations are similar to the filtering example in Section 4. Therefore, under certain assumptions above, a BNN should behave similarly. If there exists a task data imbalance, then the data term will dominate the prior term in Equation (A46) and could lead to forgetting of previous tasks.

References

- McCloskey, M.; Cohen, N.J. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of Learning and Motivation*; Elsevier: Amsterdam, The Netherlands, 1989; Volume 24, pp. 109–165.
- French, R.M. Catastrophic forgetting in connectionist networks. *Trends Cogn. Sci.* **1999**, *3*, 128–135. [[CrossRef](#)]
- Kirkpatrick, J.; Pascanu, R.; Rabinowitz, N.; Veness, J.; Desjardins, G.; Rusu, A.A.; Milan, K.; Quan, J.; Ramalho, T.; Grabska-Barwinska, A.; et al. Overcoming catastrophic forgetting in neural networks. *Proc. Natl. Acad. Sci. USA* **2017**, *114*, 3521–3526.
- MacKay, D.J. A practical Bayesian framework for backpropagation networks. *Neural Comput.* **1992**, *4*, 448–472. [[CrossRef](#)]
- Graves, A. Practical variational inference for neural networks. *Adv. Neural Inf. Process. Syst.* **2011**, *24*, 1–9.
- Blundell, C.; Cornebise, J.; Kavukcuoglu, K.; Wierstra, D. Weight uncertainty in neural network. In Proceedings of the International Conference on Machine Learning, PMLR, Lille, France, 6–11 July 2015; pp. 1613–1622.
- Schwarz, J.; Czarnecki, W.; Luketina, J.; Grabska-Barwinska, A.; Teh, Y.W.; Pascanu, R.; Hadsell, R. Progress & compress: A scalable framework for continual learning. In Proceedings of the International Conference on Machine Learning, PMLR, Stockholm, Sweden, 10–15 July 2018; pp. 4528–4537.
- Ritter, H.; Botev, A.; Barber, D. Online structured laplace approximations for overcoming catastrophic forgetting. *Adv. Neural Inf. Process. Syst.* **2018**, *31*, 1–11.
- Nguyen, C.V.; Li, Y.; Bui, T.D.; Turner, R.E. Variational Continual Learning. In Proceedings of the International Conference on Learning Representations, Vancouver, BC, Canada, 30 April–3 May 2018.

10. Ebrahimi, S.; Elhoseiny, M.; Darrell, T.; Rohrbach, M. Uncertainty-Guided Continual Learning in Bayesian Neural Networks. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops, Long Beach, CA, USA, 16–17 June 2019; pp. 75–78.
11. Kessler, S.; Nguyen, V.; Zohren, S.; Roberts, S.J. Hierarchical indian buffet neural networks for bayesian continual learning. In Proceedings of the Uncertainty in Artificial Intelligence, PMLR, Online, 27–30 July 2021; pp. 749–759.
12. Loo, N.; Swaroop, S.; Turner, R.E. Generalized Variational Continual Learning. In Proceedings of the International Conference on Learning Representations, Addis Ababa, Ethiopia, 26–30 April 2020.
13. Neal, R.M. MCMC using Hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo*; Chapman and Hall: New York, NY, USA, 2011; pp. 113–162.
14. LeCun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. Gradient-based learning applied to document recognition. *Proc. IEEE* **1998**, *86*, 2278–2324. [[CrossRef](#)]
15. Zenke, F.; Poole, B.; Ganguli, S. Continual learning through synaptic intelligence. In Proceedings of the International Conference on Machine Learning, PMLR, Sydney, Australia, 6–11 August 2017; pp. 3987–3995.
16. Hsu, Y.C.; Liu, Y.C.; Ramasamy, A.; Kira, Z. Re-evaluating continual learning scenarios: A categorization and case for strong baselines. *arXiv* **2018**, arXiv:1810.12488.
17. Van de Ven, G.M.; Toliás, A.S. Three scenarios for continual learning. *arXiv* **2019**, arXiv:1904.07734.
18. van de Ven, G.M.; Tuytelaars, T.; Toliás, A.S. Three types of incremental learning. *Nat. Mach. Intell.* **2022**, *4*, 1185–1197. [[CrossRef](#)]
19. Chopin, N.; Papaspiliopoulos, O. *An Introduction to Sequential Monte Carlo*; Springer: Cham, Switzerland, 2020; Volume 4.
20. Cobb, A.D.; Jalaian, B. Scaling Hamiltonian Monte Carlo Inference for Bayesian Neural Networks with Symmetric Splitting. In Proceedings of the Thirty-Seventh Conference on Uncertainty in Artificial Intelligence, Online, 27–30 July 2021; pp. 675–685.
21. Izmailov, P.; Vikram, S.; Hoffman, M.D.; Wilson, A.G.G. What are Bayesian neural network posteriors really like? In Proceedings of the International Conference on Machine Learning, PMLR, Online, 18–24 July 2021; pp. 4629–4640.
22. Pan, P.; Swaroop, S.; Immer, A.; Eschenhagen, R.; Turner, R.; Khan, M.E.E. Continual deep learning by functional regularisation of memorable past. *Adv. Neural Inf. Process. Syst.* **2020**, *33*, 4453–4464.
23. Dinh, L.; Sohl-Dickstein, J.; Bengio, S. Density estimation using real NVP. *arXiv* **2016**, arXiv:1605.08803.
24. Doucet, A.; De Freitas, N.; Gordon, N. An introduction to sequential Monte Carlo methods. In *Sequential Monte Carlo Methods in Practice*; Springer: Berlin/Heidelberg, Germany, 2001; pp. 3–14.
25. Kalman, R.E. A new approach to linear filtering and prediction problems. *J. Basic Eng. Mar.* **1960**, *82*, 35–45. [[CrossRef](#)]
26. Aljundi, R.; Lin, M.; Goujaud, B.; Bengio, Y. Gradient based sample selection for online continual learning. *Adv. Neural Inf. Process. Syst.* **2019**, *32*, 1–10.
27. Aljundi, R.; Kelchtermans, K.; Tuytelaars, T. Task-free continual learning. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Long Beach, CA, USA, 15–20 June 2019; pp. 11254–11263.
28. De Lange, M.; Aljundi, R.; Masana, M.; Parisot, S.; Jia, X.; Leonardis, A.; Slabaugh, G.; Tuytelaars, T. A continual learning survey: Defying forgetting in classification tasks. *arXiv* **2019**, arXiv:1909.08383.
29. Wilson, A.G.; Izmailov, P. Bayesian deep learning and a probabilistic perspective of generalization. *Adv. Neural Inf. Process. Syst.* **2020**, *33*, 4697–4708.
30. Ciftcioglu, Ö.; Türkcan, E. *Adaptive Training of Feedforward Neural Networks by Kalman Filtering*; Netherlands Energy Research Foundation ECN: Petten, The Netherlands, 1995.
31. Aitchison, L. Bayesian filtering unifies adaptive and non-adaptive neural network optimization methods. *Adv. Neural Inf. Process. Syst.* **2020**, *33*, 18173–18182.
32. Jacot, A.; Gabriel, F.; Hongler, C. Neural tangent kernel: Convergence and generalization in neural networks. *Adv. Neural Inf. Process. Syst.* **2018**, *31*, 1–10.
33. Thrun, S.; Mitchell, T.M. Lifelong robot learning. *Robot. Auton. Syst.* **1995**, *15*, 25–46. [[CrossRef](#)]
34. Zeno, C.; Golan, I.; Hoffer, E.; Soudry, D. Task agnostic continual learning using online variational bayes. *arXiv* **2018**, arXiv:1803.10123.
35. Ahn, H.; Cha, S.; Lee, D.; Moon, T. Uncertainty-based continual learning with adaptive regularization. *Adv. Neural Inf. Process. Syst.* **2019**, *32*, 1–11.
36. Farquhar, S.; Osborne, M.A.; Gal, Y. Radial bayesian neural networks: Beyond discrete support in large-scale bayesian deep learning. In Proceedings of the International Conference on Artificial Intelligence and Statistics, PMLR, Online, 26–28 August 2020; pp. 1352–1362.
37. Mehta, N.; Liang, K.; Verma, V.K.; Carin, L. Continual learning using a Bayesian nonparametric dictionary of weight factors. In Proceedings of the International Conference on Artificial Intelligence and Statistics. PMLR, Online, 13–15 April 2021; pp. 100–108.
38. Kumar, A.; Chatterjee, S.; Rai, P. Bayesian structural adaptation for continual learning. In Proceedings of the International Conference on Machine Learning, PMLR, Online, 18–24 July 2021; pp. 5850–5860.
39. Adel, T.; Zhao, H.; Turner, R.E. Continual learning with adaptive weights (claw). *arXiv* **2019**, arXiv:1911.09514.
40. Titsias, M.K.; Schwarz, J.; Matthews, A.G.d.G.; Pascanu, R.; Teh, Y.W. Functional Regularisation for Continual Learning with Gaussian Processes. In Proceedings of the ICLR, Addis Ababa, Ethiopia, 26–30 April 2020.
41. Kapoor, S.; Karaletsos, T.; Bui, T.D. Variational auto-regressive Gaussian processes for continual learning. In Proceedings of the International Conference on Machine Learning, PMLR, Online, 18–24 July 2021; pp. 5290–5300.

42. Buzzega, P.; Boschini, M.; Porrello, A.; Abati, D.; Calderara, S. Dark experience for general continual learning: A strong, simple baseline. *Adv. Neural Inf. Process. Syst.* **2020**, *33*, 15920–15930.
43. Benjamin, A.; Rolnick, D.; Kording, K. Measuring and regularizing networks in function space. In Proceedings of the International Conference on Learning Representations, Vancouver, BC, Canada, 30 April–3 May 2018.
44. Henning, C.; Cervera, M.; D’Angelo, F.; Von Oswald, J.; Traber, R.; Ehret, B.; Kobayashi, S.; Grewe, B.F.; Sacramento, J. Posterior meta-replay for continual learning. *Adv. Neural Inf. Process. Syst.* **2021**, *34*, 14135–14149.
45. Swaroop, S.; Nguyen, C.V.; Bui, T.D.; Turner, R.E. Improving and understanding variational continual learning. *arXiv* **2019**, arXiv:1905.02099.
46. Rudner, T.G.J.; Chen, Z.; Teh, Y.W.; Gal, Y. Tractable Function-Space Variational Inference in Bayesian Neural Networks. In Proceedings of the 36th Conference on Neural Information Processing Systems (NeurIPS 2022), New Orleans, LA, USA, 28 November–9 December 2022.
47. Rudner, T.G.J.; Smith, F.B.; Feng, Q.; Teh, Y.W.; Gal, Y. Continual Learning via Sequential Function-Space Variational Inference. In Proceedings of the 38th International Conference on Machine Learning, PMLR, Virtual, 18–24 July 2022.
48. Lavda, F.; Ramapuram, J.; Gregorova, M.; Kalousis, A. Continual classification learning using generative models. *arXiv* **2018**, arXiv:1810.10612.
49. van de Ven, G.M.; Li, Z.; Tolia, A.S. Class-incremental learning with generative classifiers. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Nashville, TN, USA, 20–25 June 2021; pp. 3611–3620.
50. Snell, J.; Swersky, K.; Zemel, R. Prototypical networks for few-shot learning. *Adv. Neural Inf. Process. Syst.* **2017**, *30*, 1–11.
51. Rebuffi, S.A.; Kolesnikov, A.; Sperl, G.; Lampert, C.H. ICARL: Incremental classifier and representation learning. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017; pp. 2001–2010.
52. Harrison, J.; Sharma, A.; Finn, C.; Pavone, M. Continuous meta-learning without tasks. *Adv. Neural Inf. Process. Syst.* **2020**, *33*, 17571–17581.
53. Knoblauch, J.; Husain, H.; Diethe, T. Optimal continual learning has perfect memory and is NP-hard. In Proceedings of the International Conference on Machine Learning, PMLR, Virtual, 13–18 July 2020; pp. 5327–5337.
54. Petersen, K.B.; Pedersen, M.S. *The Matrix Cookbook*; Technical University of Denmark: Lyngby, Denmark, 2008; Volume 7, p. 510.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.