Continual Deep Learning by Functional Regularisation of Memorable Past

Anonymous Authors[1]

Abstract

Continually learning new skills is important for intelligent systems, yet standard deep learning methods suffer from catastrophic forgetting of the past. Recent works address this with weight regularisation. Functional regularisation, although computationally expensive, is expected to perform better, but rarely does so in practice. In this paper, we fix this issue by using a new functional-regularisation approach that utilises a few *memorable past* examples crucial to avoid forgetting. By using a Gaussian Process formulation of deep networks, our approach enables training in weight-space while identifying both the memorable past and a functional prior. Our method achieves state-of-the-art performance on standard benchmarks and opens a new direction for life-long learning where regularisation and memory-based methods are naturally combined.

1. Introduction

The ability to quickly adapt to changing environments is an important quality of intelligent systems. For such quick adaptation, it is important to be able to identify, memorise, and recall useful past experiences when acquiring new ones. Unfortunately, standard deep-learning methods lack such qualities, and can quickly forget previously acquired skills when learning new ones (Kirkpatrick et al., 2017). Such catastrophic forgetting presents a big challenge for applications, such as robotics, where new tasks can appear during training, and data from previous tasks might be unavailable for retraining.

In recent years, many methods have been proposed to address catastrophic forgetting in deep neural networks (DNNs). One popular approach is to keep network weights close to the values obtained for the previous tasks/data (Kirkpatrick et al., 2017; Nguyen et al., 2018; Zenke et al., 2017; Ebrahimi et al., 2019; Serra et al., 2018). However, this may not always ensure the quality of predictions on previous tasks. Since the network outputs depend on the weights in a complex way, such *weight-regularisation* may not be effective. A better approach is to use *functional-regularisation*, where we directly regularise the network outputs (Benjamin et al., 2018), but this is costly because it requires derivatives of outputs at many input locations. Existing approaches reduce these costs by carefully selecting the locations, e.g. by using a *working memory* (Benjamin et al., 2018) or Gaussian-Process (GP) inducing points (Titsias et al., 2019), but currently they do not consistently outperform existing weight-regularisation methods.

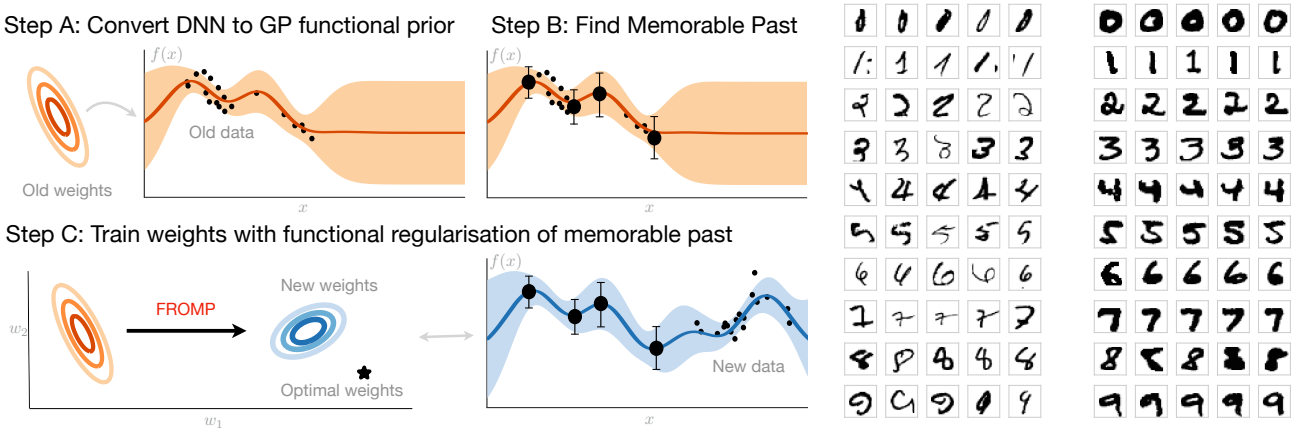
To address this issue, we propose a new functional-regularisation method called Functional Regularisation of Memorable Past (FROMP). Our key idea is to regularise the network outputs at a few *memorable past* examples that are crucial to avoid forgetting. We use a GP formulation of DNNs to obtain a weight-training method that exploits correlations among memorable examples in the function space (see Figure 1a). FROMP involves a slight modification of Adam and a minor increase in the computation cost. It achieves state-of-the-art performance on standard benchmarks, and is consistently better than both the existing weight-regularisation and functional-regularisation methods. Our work in this paper focuses on avoiding forgetting, but it also opens a new direction for life-long learning methods where regularisation methods are naturally combined with memory-based methods.

1.1. Related Works

Our goal is to consistently outperform weight-regularisation which can be inadequate and brittle for continual learning (see Figure 5 and Appendix G for an example). The proposed method performs better than existing weight-regularisation approaches and further addresses many issues with the existing functional-regularisation methods. Arguably the work most closely related to ours is the GP-based method of Titsias et al. (2019), but there are several key differences. First, our kernel uses *all* the network weights (they use just the last layer) which is important, especially in the early stages of learning when all the weights are changing. Second, our functional prior, although of a similar form as Titsias et al. (2019), regularises the mean to be close to the

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.



(a) FROMP for continual deep learning

(b) Most (left) vs least (right) memorable

Figure 1. (a) Our FROMP method consists of three main steps where we convert a DNN to GP using Khan et al. (2019), find memorable examples, and train weights with functional regularisation of those examples. (b) Memorable past on MNIST – they are difficult to classify and close to the boundary.

past mean, which is lacking in the regulariser of Titsias et al. (2019) (see the discussion after Equation 7). Third, our memorable past examples play a similar role as the inducing inputs, but are much cheaper to obtain (Titsias et al. (2019) requires solving a discrete optimisation problem), and have an intuitive interpretation (see Figure 1b). Due to these differences, our method outperforms the method of Titsias et al. (2019), which, unlike ours, performs worse than the weight-regularisation method of Swaroop et al. (2019). We also obtain state-of-the-art performance on a larger split CIFAR benchmark, a comparison which is missing in Titsias et al. (2019). Finally, our method is also different from Benjamin et al. (2018), which lacks a mechanism to automatically weight past memory and estimate uncertainty.

Our method is based on a set of memorable past examples. Many such memory-based approaches exist (Rebuffi et al., 2017; Shin et al., 2017; Aljundi et al., 2019; Chaudhry et al., 2019; Lopez-Paz & Ranzato, 2017; Chaudhry et al., 2018). Compared to these approaches, an advantage of our method is that the memory is obtained within the functional-regularisation framework and does not require solving a separate optimisation problem. The computation is also straightforward, simply requiring a forward-pass through the network followed by sorting (see Section 3.2).

2. Continual Learning with Weight/Functional Regularisation

In deep learning, we minimise loss functions to estimate network weights. For example, in supervised multi-class classification problems, we are given a dataset $\mathcal{D}$ of N input-output pairs with outputs $\mathbf{y}_i$, a one-hot encoded vector of K classes, and inputs $\mathbf{x}_i$, a vector of length D. Our goal is to minimise a loss which takes the following form: $N\bar{\ell}(\mathbf{w}) + \delta R(\mathbf{w})$, where $\bar{\ell}(\mathbf{w}) := \frac{1}{N}\sum_{i=1}^{N} \ell(\mathbf{y}_i, \mathbf{f}_w(\mathbf{x}_i))$ with deep neural network $\mathbf{f}_w(\mathbf{x}) \in \mathbb{R}^K$ and its weights $\mathbf{w} \in \mathbb{R}^P$, $\ell(\mathbf{y}, \mathbf{f})$ denotes a differentiable loss function (e.g., cross entropy) between an output $\mathbf{y}$ and the network output $\mathbf{f}$, $R(\mathbf{w})$ is a regularisation function (usually an L_2-regulariser $R(\mathbf{w}) := \mathbf{w}^\top\mathbf{w}$), and $\delta > 0$ controls the regularisation strength. Standard deep-learning approaches rely on an unbiased stochastic gradient of the loss $\bar{\ell}$. This usually requires access to all the data examples for all classes throughout training (Bottou, 2010). It is this unbiased, minibatch setting where deep-learning excels and achieves state-of-the-art performance.

In reality, we do not always have access to all the data at once, and it is not possible to obtain unbiased stochastic gradients. New classes may appear during training and old classes may never be seen again. For such settings, vanilla mini-batch stochastic-gradient methods lead to catastrophic forgetting of past information (Kirkpatrick et al., 2017). Our goal in this paper is to design methods that can avoid, or minimise, such catastrophic forgetting. We focus on a particular setting where the classification task is divided into several tasks, e.g., a task may consist of a classification problem over a subset of classes. We assume that the tasks arrive sequentially one after the other. Once the learning is over, we may never see that task again. Such continual-learning settings have been considered in other works (Kirkpatrick et al., 2017; Nguyen et al., 2018; Zenke et al., 2017) with the goal to avoid forgetting of previous tasks.

Recent methods have proposed weight-regularisation as a way to combat catastrophic forgetting. The main idea is to find the important weights for past tasks, and keep

new weights close to them. For example, when training on the task t while given weights $\mathbf{w}_{t-1}$ trained on the past tasks, we can minimise the following loss: $N\bar{\ell}_t(\mathbf{w}) + \delta(\mathbf{w} - \mathbf{w}_{t-1})^\top \mathbf{F}_{t-1}(\mathbf{w} - \mathbf{w}_{t-1})$, where $\bar{\ell}_t(\mathbf{w})$ is the loss defined over all data examples from task t and $\mathbf{F}_{t-1}$ is a preconditioning matrix that favours the weights relevant to the past tasks more than the rest. The Elastic-Weight Consolidation (EWC) method (Kirkpatrick et al., 2017) and Ritter et al. (2018), for example, use the Fisher information matrix as the pre-conditioner, while variational continual learning (VCL) (Nguyen et al., 2018) employs the precision matrix of the variational approximation. To reduce complexity, usually a diagonal (or block diagonal) matrix is used. Such weight-space methods reduce forgetting but do not produce satisfactory results.

The challenge in using weight-regularisation lies in the fact that the exact values of the weights do not really matter due to parametric symmetries (Benjamin et al., 2018; Bishop, 2006). Making current weights closer to the previous ones may not always ensure that the predictions on the past tasks also remain unchanged. Since the network outputs depend on the weights in a complex way, it is difficult to ensure the effectiveness of *weight-regularisation*. A better approach is to directly regularise the outputs, because what matters is the network output, not the values of the weights. For example, we can use an L_2-regulariser over the function values on data examples from past tasks (e.g., see (Benjamin et al., 2018)) :

$$\min_{w} N\bar{\ell}_t(\mathbf{w}) + \delta \sum_{s=1}^{t-1} (\mathbf{f}_{t,s} - \mathbf{f}_{t-1,s})^\top (\mathbf{f}_{t,s} - \mathbf{f}_{t-1,s}), \quad (1)$$

where $\mathbf{f}_{t,s}$ and $\mathbf{f}_{t-1,s}$ are vectors of function values $f_w(\mathbf{x}_i)$ and $f_{w_{t-1}}(\mathbf{x}_i)$ respectively for all $i \in \mathcal{D}_s$ with $\mathcal{D}_s$ being the dataset for the task s. Rather than making the weights $\mathbf{w}$ similar to $\mathbf{w}_{t-1}$, such *functional-regularisation* approaches directly force the function values to be similar. Because of this, we expect them to perform better. This is also expected for a Bayesian approach, as posterior approximations in the function-space might be better than those in the weight-space.

Unfortunately, functional-regularisation is computationally infeasible because it requires us to store all past data and compute function values over them. This computational issue is typically solved by using a subset of inputs. Benjamin et al. (2018) employ a *working memory* (Lopez-Paz & Ranzato, 2017; Rebuffi et al., 2017) while Titsias et al. (2019) use the inducing point method based on a Gaussian process framework. As discussed earlier, such approaches do not consistently perform better than existing weight-regularisation methods. This could be due to the methods they use to build memory or enforce functional regularisation. Our goal in this paper is to design a functional-regularisation method that is consistently better than weight-regularisation. We build upon the method of Khan et al. (2019) to convert deep networks into Gaussian processes, described next.

3. Functional-Regularisation of Memorable Past (FROMP)

3.1. From Deep Networks to Functional Priors

Khan et al. (2019) propose an approach called DNN2GP to convert deep networks to Gaussian processes (GPs). In step A, we employ such GPs as functional priors to regularise the next task. The DNN2GP approach is very similar to the standard weight-space to function-space conversion for linear basis-function models (Rasmussen & Williams, 2006). For example, consider a linear regression model on a scalar output $y_i = f_w(\mathbf{x}_i) + \epsilon_i$ with a function output $f_w(\mathbf{x}_i) := \boldsymbol{\phi}(\mathbf{x}_i)^\top \mathbf{w}$ using a feature map $\boldsymbol{\phi}(\mathbf{x})$. Assume Gaussian noise $\mathcal{N}(\epsilon_i|0, \Lambda^{-1})$ and a Gaussian prior $\mathcal{N}(\mathbf{w}|0, \delta^{-1}\mathbf{I}_P)$ where $\mathbf{I}_P$ is the identity matrix of size $P \times P$. It can then be shown that the posterior distribution of this linear model, denoted by $\mathcal{N}(\mathbf{w}|\mathbf{w}_{\text{lin}}, \boldsymbol{\Sigma}_{\text{lin}})$, induces a GP posterior on function $f_w(\mathbf{x})$ whose mean and covariance functions are given as follows (see Appendix A.1 or Chapter 2 in Rasmussen & Williams (2006)):

$$m_{\text{lin}}(\mathbf{x}) := f_{w_{\text{lin}}}(\mathbf{x}), \quad \kappa_{\text{lin}}(\mathbf{x}, \mathbf{x}') := \boldsymbol{\phi}(\mathbf{x})^\top \boldsymbol{\Sigma}_{\text{lin}} \boldsymbol{\phi}(\mathbf{x}'), \quad (2)$$

where $\mathbf{w}_{\text{lin}}$ is simply the Maximum A Posteriori (MAP) estimate of the linear model, and

$$\boldsymbol{\Sigma}_{\text{lin}}^{-1} := \sum_i \boldsymbol{\phi}(\mathbf{x}_i) \Lambda \boldsymbol{\phi}(\mathbf{x}_i)^\top + \delta \mathbf{I}_P. \quad (3)$$

DNN2GP computes a similar GP posterior but for a *neural network* whose posterior is approximated by a Gaussian. Specifically, given a local minimum $\mathbf{w}_*$ of the loss $N\bar{\ell}(\mathbf{w}) + \frac{1}{2}\delta \mathbf{w}^\top \mathbf{w}$ for a scalar output $f_w(\mathbf{x})$, we can construct a Gaussian posterior approximation. Following Khan et al. (2019), we employ a variant of the Laplace approximation with mean $\boldsymbol{\mu}_* = \mathbf{w}_*$ and covariance

$$\boldsymbol{\Sigma}_*^{-1} = \sum_{i=1}^{N} \mathbf{J}_{w_*}(\mathbf{x}_i)^\top \Lambda_{w_*}(\mathbf{x}_i, \mathrm{y}_i) \mathbf{J}_{w_*}(\mathbf{x}_i) + \delta \mathbf{I}_P, \quad (4)$$

where $\Lambda_{w_*}(\mathbf{x}, \mathrm{y}) := \nabla^2_{\text{ff}} \ell(\mathrm{y}, \mathrm{f})$ is the scalar Hessian of the loss function, and $\mathbf{J}_{w_*}(\mathbf{x}) := \nabla_w \mathrm{f}_w(\mathbf{x})^\top$ is the $1 \times P$ Jacobian; all quantities evaluated at $\mathbf{w} = \mathbf{w}_*$. Essentially, this variant uses a Gauss-Newton approximation for the covariance instead of the Hessian. Comparing Equations 3 and 4, we can interpret $\boldsymbol{\Sigma}_*$ as the covariance of a linear model with a feature map $\mathbf{J}_{w_*}(\mathbf{x})^\top$ and noise precision $\Lambda_{w_*}(\mathbf{x}, y)$. Using this similarity, Khan et al. (2019) derive a GP posterior approximation for neural networks. They show this for a generic loss function (see App. B2 in their paper), e.g., for a regression loss, the

mean and covariance functions of the GP posterior take the following form:

$$m_{w_*}(\mathbf{x}) := f_{w_*}(\mathbf{x}), \quad \kappa_{w_*}(\mathbf{x}, \mathbf{x}') := \mathbf{J}_{w_*}(\mathbf{x}) \boldsymbol{\Sigma}_* \mathbf{J}_{w_*}(\mathbf{x}')^\top. \quad (5)$$

This is equivalent to Equation 2 if we set the feature map $\boldsymbol{\phi}(\mathbf{x}) \equiv \mathbf{J}_{w_*}(\mathbf{x})^\top$. A similar equation holds for other loss functions such as those used for binary and multiclass classification; see Appendix A.2 for details. We denote such GP posteriors by $\mathcal{GP}\left(m_{w_*}(\mathbf{x}), \kappa_{w_*}(\mathbf{x}, \mathbf{x}')\right)$, and use them as a *functional prior* to regularise the next task.

The above result holds at a minimiser $\mathbf{w}_*$, but can be extended to a sequence of weights obtained during optimisation (Khan et al., 2019). For example, for Gaussian variational approximations $q(\mathbf{w})$, we can obtain GP posteriors by replacing $\mathbf{w}_*$ by $\mathbf{w} \sim q(\mathbf{w})$ in Equation 5. We denote such GPs by $\mathcal{GP}\left(m_w(\mathbf{x}), \kappa_w(\mathbf{x}, \mathbf{x}')\right)$. The result also applies to variants of Newton's method, RMSprop, and Adam (see Appendix A.3). The GP posteriors obtained are related to the Neural Tangent Kernel (NTK) (Jacot et al., 2018). The prior distribution to obtain the posterior in Equation 5 corresponds to the NTK at finite width. A slightly different kernel is obtained when a variational approximation is used. Unlike the method of Titsias et al. (2019), the kernel above uses *all* the network weights, and uses the Jacobians instead of using the functions themselves.

3.2. Identifying Memorable past

To reduce the computation cost of functional regularisation, we identify a few memorable past examples. To do so, we exploit a property of linear models. Consider a linear model where different noise precision Λ_i is assigned to each pair $\{\mathbf{x}_i, y_i\}$. For MAP estimation, the examples with high value of Λ_i contribute more, as is clear from the objective: $\mathbf{w}_{\text{MAP}} = \arg\max_w \sum_{i=1}^N \Lambda_i (y_i - \boldsymbol{\phi}(\mathbf{x}_i)^\top \mathbf{w})^2 + \delta\, \mathbf{w}^\top \mathbf{w}$. The noise precision Λ_i can therefore be interpreted as the relevance of the data example i. Such relevant examples are crucial to ensure that the solution stays at $\mathbf{w}_{\text{MAP}}$ or close to it. These ideas are widely used in the theory of leverage-score sampling (Alaoui & Mahoney, 2015; Ma et al., 2015) to identify the most *influential* examples. Computation using such methods is infeasible since they require inverting a large matrix. Titsias et al. (2019) use an approximation by inverting smaller matrices, but they require solving a discrete optimisation problem to select examples. We propose a method which is not only cheap and effective, but also yields intuitive results.

We use the linear model corresponding to the GP posterior from Section 3.1. The linear model assigns different noise precision to each data example. See Equations 3 and 4 where the quantity $\Lambda_{w_*}(\mathbf{x}_i, y_i)$ plays the same role as the noise precision Λ. Therefore, $\Lambda_{w_*}(\mathbf{x}_i, y_i)$ can be used as a relevance measure, and a simple approach to pick influential examples is to sort it $\forall i$ and pick the top few examples. We refer to such a set of examples as the *memorable past* examples. An example is shown in Figure 1b where our approach picks many examples that are difficult to classify. The memorable past can be intuitively thought of as *examples close to the decision boundary*. An advantage of using this approach is that $\Lambda_{w_*}(\mathbf{x}_i, y_i)$ is extremely cheap to compute. It is simply the second derivative of the loss, which can be obtained with a forward pass to get $\ell(y_i, \hat{y}_i)$, followed by double differentiation with respect to $\hat{y}_i$. After training on every task, we select a few memorable examples in $\mathcal{D}_t$, denoting the set of such examples by $\mathcal{M}_t$.

3.3. Training in weight-space with a functional prior

We will now describe the final step for weight-training with functional-regularisation. We use the Bayesian formulation of continual learning and replace the prior distribution in weight space by a *functional prior*. Given a loss of the form $N\bar{\ell}_t(\mathbf{w}) + R(\mathbf{w})$, a Bayesian formulation in weight-space employs a regulariser that uses the previous posterior, i.e., $R(\mathbf{w}) \equiv -\log p(\mathbf{w}|\mathcal{D}_{1:t-1})$. Computing the exact posterior, or a tempered version of it, would in theory avoid catastrophic forgetting, but that is expensive and we must use approximations. For example, Nguyen et al. (2018) use the variational approximation from the previous task $p(\mathbf{w}|\mathcal{D}_{1:t-1}) \approx q_{t-1}(\mathbf{w}) = \mathcal{N}(\mathbf{w}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ as the weight regulariser. Our goal is to replace the weight regulariser by a functional regulariser obtained by using the GP posteriors described in Section 3.1.

We use functional regularisation defined over memorable examples. Denote by $\mathbf{f}$ the vector of function values defined at all memorable past $\mathcal{M}_s$ in all tasks $s < t$. Denoting a sample from $q(\mathbf{w})$ by $\mathbf{w}$, we can obtain a GP posterior over $\mathbf{f}$ by using Equation 5. We denote it by $\tilde{q}_w(\mathbf{f}) = \mathcal{N}(\mathbf{f}|\mathbf{m}_t(\mathbf{w}), \mathbf{K}_t(\mathbf{w}))$, where $\mathbf{m}_t(\mathbf{w})$ and $\mathbf{K}_t(\mathbf{w})$ respectively denote the mean vector and kernel matrix obtained by evaluating $\mathcal{GP}\left(m_{w_t}(\mathbf{x}), \kappa_{w_t}(\mathbf{x}, \mathbf{x}')\right)$ at the memorable past examples. Similarly, denoting a sample from $q_{t-1}(\mathbf{w})$ by $\mathbf{w}_{t-1}$, we can obtain another GP posterior, which we call the *functional prior*, denoted by $\tilde{q}_{w_{t-1}}(\mathbf{f}) = \mathcal{N}(\mathbf{f}|\mathbf{m}_{t-1}, \mathbf{K}_{t-1})$. Using these two GPs, we can replace the weight regulariser used by Nguyen et al. (2018) by a *functional regulariser* equal to the expectation of the functional prior:

$$\min_{q(w)} \mathbb{E}_{q(w)}\left[(N/\tau)\bar{\ell}_t(\mathbf{w}) + \log q(\mathbf{w})\right] - \underbrace{\mathbb{E}_{q(w)}\left[\log q_{t-1}(\mathbf{w})\right]}_{\approx \mathbb{E}_{\tilde{q}_w(\mathbf{f})}\left[\log \tilde{q}_{w_{t-1}}(\mathbf{f})\right]}, \quad (6)$$

where the last term is the weight regulariser, and $\tau > 0$ is a tempering parameter. Fortunately, the functional regulariser

has a closed-form expression:

$$\mathbb{E}_{\tilde{q}_w(\mathbf{f})}\left[\log \tilde{q}_{w_{t-1}}(\mathbf{f})\right] = -\tfrac{1}{2}\Big[\mathrm{Tr}(\mathbf{K}_{t-1}^{-1}\mathbf{K}_t(\mathbf{w})) + (\mathbf{m}_t(\mathbf{w}) - \mathbf{m}_{t-1})^\top \mathbf{K}_{t-1}^{-1}(\mathbf{m}_t(\mathbf{w}) - \mathbf{m}_{t-1})\Big] + \text{const.} \tag{7}$$

This term depends on $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ through the sample $\mathbf{w} \sim q(\mathbf{w})$. The regulariser is an approximation for reasons discussed in Appendix D. Our regulariser has a similar form[1] to Titsias et al. (2019), but our regulariser forces the mean $\mathbf{m}_t(\mathbf{w})$ to be close to $\mathbf{m}_{t-1}$, which is missing in their regulariser.

Optimising $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ in Equation 6 with this functional prior can be very expensive for large networks. We make five approximations to reduce the cost, discussed in detail in Appendix B. First, for the functional prior, we use the mean of $q_{t-1}(\mathbf{w})$, instead of a sample $\mathbf{w}_{t-1}$, which corresponds to using the GP posterior of Equation 5. Second, for Equation 7, we ignore the derivative with respect to $\mathbf{K}_t(\mathbf{w})$ and only use $\mathbf{m}_t(\mathbf{w})$, which assumes that the Jacobians do not change significantly. Third, instead of using the full $\mathbf{K}_{t-1}$, we factorise it across tasks, i.e., let it be a block-diagonal matrix with $\mathbf{K}_{t-1,s}, \forall s$ as the diagonal. This makes the cost of inversion linear in the number of tasks. Fourth, following Khan et al. (2019), we propose to use a deterministic optimiser for Equation 6, which corresponds to setting $\mathbf{w} = \boldsymbol{\mu}$. Finally, we use a diagonal $\boldsymbol{\Sigma}$, which corresponds to a mean-field approximation, reducing the cost of inversion. As shown in Appendix B, the resulting algorithm finds the minimum $\mathbf{w}$ of the following:

$$N\bar{\ell}_t(\mathbf{w}) + \tfrac{1}{2}\tau \sum_{s=1}^{t-1} \left[\mathbf{m}_{t,s}(\mathbf{w}) - \mathbf{m}_{t-1,s}\right]^\top \mathbf{K}_{t-1,s}^{-1} \left[\mathbf{m}_{t,s}(\mathbf{w}) - \mathbf{m}_{t-1,s}\right], \tag{8}$$

where $\mathbf{m}_{t,s}$ is the sub-vector of $\mathbf{m}_t$ corresponding to the task s. The above is an computationally-cheap approximation of Equation 6 and forces the network to produce similar outputs at memorable past examples. The objective is an improved version of Equation 1 (Benjamin et al., 2018). For regression, the mean $\mathbf{m}_{t,s}$ in Equation 8 is equal to the vector $\mathbf{f}_{t,s}$ used in Equation 1. Our functional regulariser additionally includes a kernel matrix $\mathbf{K}_{t-1,s}$ to take care of the uncertainty and weighting of past tasks' memorable examples. Due to a full kernel matrix, the functional regulariser exploits the correlations between memorable past examples. This is in contrast with a weight-space approach, where modelling correlations is infeasible since $\boldsymbol{\Sigma}$ is extremely large. In our case, training is cheap since the objective in Equation 8 can be optimised using Adam. *Our approach therefore provides a cheap weight-space training method while exploiting correlations in function-space.* Due to these properties, we expect our method to perform better. We can expect further improvements by relaxing these assumptions (see Appendix B), e.g., we can use a full kernel matrix, use a variational approximation, or employ a block-diagonal covariance matrix. We leave such comparisons as future work since they require sophisticated implementation to scale.

[1] Their regulariser is $\mathbb{E}_{q(u_{t-1})}[\log p_w(\mathbf{u}_{t-1})] = -\tfrac{1}{2}\{\mathrm{Tr}[\mathbf{K}(\mathbf{w})^{-1}\boldsymbol{\Sigma}_{t-1}] + \boldsymbol{\mu}_{t-1}^\top \mathbf{K}(\mathbf{w})^{-1}\boldsymbol{\mu}_{t-1}\}$, where $p_w(\mathbf{u}_{t-1}) = \mathcal{N}(0, K(\mathbf{w}))$ with the kernel evaluated at inducing inputs $\mathbf{u}_{t-1}$ and $q(\mathbf{u}_{t-1}) = \mathcal{N}(\boldsymbol{\mu}_{t-1}, \boldsymbol{\Sigma}_{t-1})$.

Algorithm 1 FROMP for binary classification on task t given $q_{t-1}(\mathbf{w}) := \mathcal{N}(\boldsymbol{\mu}_{t-1}, \mathrm{diag}(\mathbf{v}_{t-1}))$, and memorable pasts $\mathcal{M}_{1:t-1}$. Additional computations on top of Adam are highlighted in red.

Function `FROMP` $(\mathcal{D}_t, \boldsymbol{\mu}_{t-1}, \mathbf{v}_{t-1}, \mathcal{M}_{1:t-1})$ **:**
  Get $\mathbf{m}_{t-1,s}, \mathbf{K}_{t-1,s}^{-1}, \forall$ tasks $s < t$ (Equation 10)
  Initialise $\mathbf{w} \leftarrow \boldsymbol{\mu}_{t-1}$
  while *not converged* **do**
    Randomly sample $\{\mathbf{x}_i, y_i\} \in \mathcal{D}_t$
    $\mathbf{g} \leftarrow N\,\nabla_w \ell(y_i, f_w(\mathbf{x}_i))$
    $\mathbf{g}_f \leftarrow$ `g_FR` $(\mathbf{w}, \mathbf{m}_{t-1}, \mathbf{K}_{t-1}^{-1}, \mathcal{M}_{1:t-1})$
    Adam update with gradient $\mathbf{g} + \tau \mathbf{g}_f$
  end
  $\boldsymbol{\mu}_t \leftarrow \mathbf{w}$ and compute $\mathbf{v}_t$ (Equation 9)
  $\mathcal{M}_t \leftarrow$ `memorable_past` $(\mathcal{D}_t, \mathbf{w})$
  return $\boldsymbol{\mu}_t, \mathbf{v}_t, \mathcal{M}_t$

Function `g_FR` $(\mathbf{w}_t, \mathbf{m}_{t-1}, \mathbf{K}_{t-1}^{-1}, \mathcal{M}_{1:t-1})$ **:**
  Initialise $\mathbf{g}_f \leftarrow \mathbf{0}$
  for *task* $s = 1, 2, \ldots, t-1$ **do**
    Compute $\mathbf{m}_{t,s}$ (Equation 10)
    $\mathbf{h}_i \leftarrow \Lambda_{w_t}(\mathbf{x}_i)\,\mathbf{J}_{w_t}(\mathbf{x}_i)^\top, \forall \mathbf{x}_i \in \mathcal{M}_s$
    Form matrix $\mathbf{H}$ with $\mathbf{h}_i$ as columns
    $\mathbf{g}_f \leftarrow \mathbf{g}_f + \mathbf{H}\mathbf{K}_{t-1,s}^{-1}(\mathbf{m}_{t,s} - \mathbf{m}_{t-1,s})$
  end
  return $\mathbf{g}_f$

Function `memorable_past` $(\mathcal{D}_t, \mathbf{w}_t)$ **:**
  Calculate $\Lambda_{w_t}(\mathbf{x}_i),\ \forall \mathbf{x}_i \in \mathcal{D}_t$.
  return M examples with highest $\Lambda_{w_t}(\mathbf{x}_i)$.

3.4. The final algorithm and computational complexity

The resulting algorithm, FROMP, is shown in Algorithm 1 for binary classification (extension to multiclass classification is in Appendix C). For binary classification, we assume a sigmoid $\sigma(f_w(\mathbf{x}))$ function and cross-entropy loss. As shown in Appendix A.2, the Jacobian (of size $1 \times P$) and noise precision (a scalar) are as follows: $\mathbf{J}_w(\mathbf{x}) = \nabla_w f_w(\mathbf{x})^\top$ and $\Lambda_w(\mathbf{x}) = \sigma(f_w(\mathbf{x}))\,[1 - \sigma(f_w(\mathbf{x}))]$. To compute the mean and kernel, we need the diagonal of the covariance, which we denote by $\mathbf{v}$. This can be obtained using Equation 4 but with the sum over $\mathcal{D}_{1:t}$. The update below com-

putes this recursively:

$$\frac{\mathbf{1}}{\mathbf{v}_t} = \left[\frac{\mathbf{1}}{\mathbf{v}_{t-1}} + \sum_{i\in\mathcal{D}_t} \operatorname{diag}\left(\mathbf{J}_{w_t}(\mathbf{x}_i)^\top \Lambda_{w_t}(\mathbf{x}_i)\mathbf{J}_{w_t}(\mathbf{x}_i)\right)\right], \quad (9)$$

where '/' denote element-wise division and diag($\mathbf{A}$) is the diagonal of $\mathbf{A}$. Using this in an expression similar to Equation 5, we can compute the mean and kernel matrix (see Appendix A.2 for details):

$$\begin{aligned}\mathbf{m}_{t,s}[i] &= \sigma\left(f_{w_t}(\mathbf{x}_i)\right),\\ \mathbf{K}_{t,s}[i,j] &= \Lambda_{w_t}(\mathbf{x}_i)\left[\mathbf{J}_{w_t}(\mathbf{x}_i)\operatorname{Diag}(\mathbf{v}_t)\mathbf{J}_{w_t}(\mathbf{x}_j)^\top\right]\Lambda_{w_t}(\mathbf{x}_j),\end{aligned} \quad (10)$$

over all memorable examples $\mathbf{x}_i, \mathbf{x}_j$, where Diag($\mathbf{a}$) denotes a diagonal matrix with $\mathbf{a}$ as the diagonal. Using these expressions, we can write the gradient of Equation 8 at $\mathbf{w}_t$. Since we ignore the gradient with respect to $\mathbf{K}_t$, we get a simple expression with an additional term added to the gradient of the loss: $N\nabla_{w_t}\bar{\ell}_t(\mathbf{w}_t) + \tau\sum_{s=1}^{t-1}(\nabla_{w_t}\mathbf{m}_{t,s})\mathbf{K}_{t-1,s}^{-1}(\mathbf{m}_{t,s} - \mathbf{m}_{t-1,s})$ where $\nabla_{w_t}\mathbf{m}_{t,s}[i] = \nabla_{w_t}\left[\sigma\left(f_{w_t}(\mathbf{x}_i)\right)\right] = \Lambda_{w_t}(\mathbf{x}_i)\mathbf{J}_{w_t}(\mathbf{x}_i)^\top$. This term is computed in subroutine `g_FR` in Algorithm 1.

The additional computations on top of Adam are highlighted in red in Algorithm 1. Every iteration requires functional gradients (in `g_FR`) whose cost is dominated by the computation of $\mathbf{J}_w(\mathbf{x}_i)$ at all $\mathbf{x}_i \in \mathcal{M}_s, \forall s < t$. Assuming the size of the memorable past is M per task, this adds an additional $O(MPt)$ computation, where P is the number of parameters and t is the task number. This increases only linearly with the size of the memorable past. We need two additional computations but they are required only *once per task*. First, inversion of $\mathbf{K}_s, \forall s < t$, which has cost $O(M^3t)$. This is linear in number of tasks and is feasible when M is not too large. Second, computation of $\mathbf{v}_t$ in Equation 9 requires a full pass through the dataset $\mathcal{D}_t$, with cost $O(NP)$ where N is the dataset size. This cost can be reduced by estimating $\mathbf{v}_t$ using a minibatch of data (as is common for EWC (Kirkpatrick et al., 2017)).

4. Experiments

To identify the benefits of the functional prior (step A) and memorable past (step B), we compare FROMP to three variants: (1) FROMP-L_2 where we replace the kernel in Equation 5 by the identity matrix, similar to Equation 1, (2) FRO*R*P where memorable examples selected randomly ("R" stands for random), (3) FRO*R*P-L_2 which is same as FRO*R*P, but the kernel in Equation 5 is replace by the identity matrix. We present comparisons on four benchmarks: a toy dataset, permuted MNIST, Split MNIST, and Split CIFAR (a split version of CIFAR-10 & CIFAR-100). Results for the toy dataset are summarised in Figure 4 and Appendix G, where we also visually show the brittleness of weight-space methods. In all experiments, we use the Adam optimiser (Kingma & Ba, 2015). Details on hyperparameter settings are in Appendix F.

4.1. Permuted and Split MNIST

Permuted MNIST consists of a series of tasks, with each applying a fixed permutation of pixels to the entire MNIST dataset. Similarly to previous work (Kirkpatrick et al., 2017; Zenke et al., 2017; Nguyen et al., 2018; Titsias et al., 2019), we use a fully connected single-head network with two hidden layers, each consisting of 100 hidden units. We train for 10 tasks. The number of memorable examples is set in the range 10–200. We also test on the Split MNIST benchmark (Zenke et al., 2017), which consists of five binary classification tasks built from MNIST: 0/1, 2/3, 4/5, 6/7, and 8/9. Following the settings of previous work, we use a fully connected multi-head network with two hidden layers, each with 256 hidden units. We select 40 memorable points per task. We also run FROMP on a smaller network architecture (Swaroop et al., 2019), obtaining $(99.2 \pm 0.1)\%$ (see Appendix F.2).

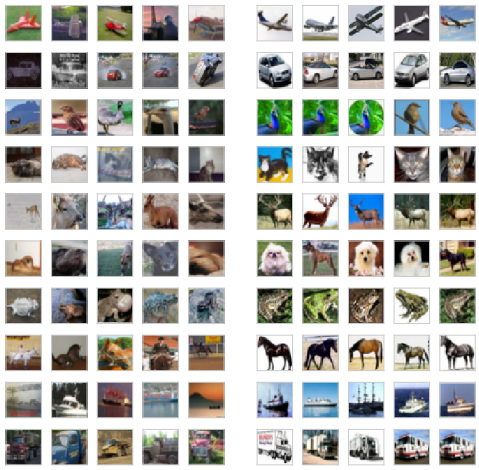
The final average accuracy is shown in Figure 2a where FROMP achieves better performance than weight-regularisation methods (EWC, VCL, SI) as well as a function-regularisation method called FRCL recently proposed by Titsias et al. (2019). FROMP also improves over FRORP-L_2 and FROMP-L_2, demonstrating the effectiveness of the kernel. The improvement compared to FRORP is not significant. However, as shown in Figure 3c, we do see an improvement when the number of memorable examples are small (compare FRORP vs FROMP). We believe this is because a random memorable past already achieves a performance close to the highest achievable performance, and we see no further improvement by choosing the examples carefully. For few memorable examples, we do see an advantage of FROMP over FRORP. Finally, Figure 1b shows the most and least memorable examples chosen by sorting $\Lambda_w(\mathbf{x}, y)$. The most memorable examples appear to be more difficult to classify than the least memorable examples, which suggests that they may lie closer to the decision boundary.

4.2. Split CIFAR

Split CIFAR is a more difficult benchmark than the MNIST benchmarks, and consists of 6 tasks. The first task is the full CIFAR-10 dataset, followed by 5 tasks, each consisting of 10 consecutive classes from CIFAR-100. We use the same model architecture as Zenke et al. (2017): a multi-head CNN with 4 convolutional layers, then 2 dense layers with dropout. The number of memorable examples is set in the range 10–200, and we again run each method 5 times. We compare to two additional baselines. The first baseline consists of networks trained on each task *separately*. Such

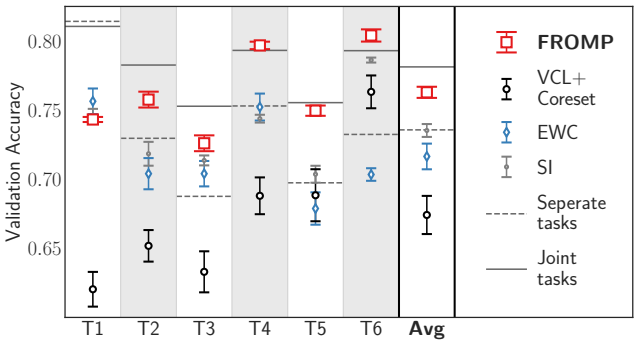
Method	Permuted	Split
DLP (Smola et al., 2003)	82%	61.2%
EWC (Kirkpatrick et al., 2017)	84%	63.1%
SI (Zenke et al., 2017)	86%	98.9%
Improved VCL (Swaroop et al., 2019)	93 ± 1%	98.4 ± 0.4%
+ random Coreset	**94.6** ± 0.3%	98.2 ± 0.4%
FRCL-RND (Titsias et al., 2019)	94.2 ± 0.1%	97.1 ± 0.7%
FRCL-TR (Titsias et al., 2019)	94.3 ± 0.2%	97.8 ± 0.7%
FRORP-L_2	87.9 ± 0.7%	98.5 ± 0.2%
FROMP-L_2	94.6 ± 0.1%	98.7 ± 0.1%
FRORP	94.6 ± 0.1%	**99.0** ± 0.1%
FROMP	**94.9** ± 0.1%	**99.0** ± 0.1%

(a) MNIST comparisons: for Permuted, we use 200 examples as memorable/coreset/inducing points. For Split, we use 40.

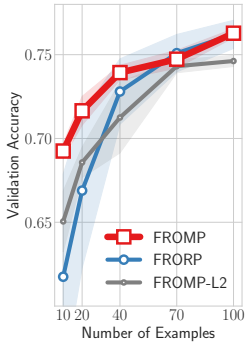


(b) Most (left) vs least (right) memorable

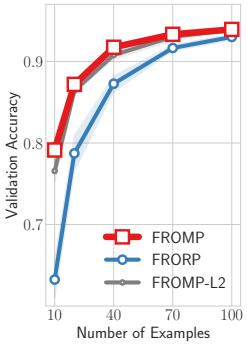
Figure 2. (a) On MNIST, FROMP obtains better accuracy than weight-regularisation (EWC, SI, VCL) and functional-regularisation (FRCL). Note that FRCL does not outperform 'Improved VCL + random coreset' while FROMP does. Errors are reported over 5 runs.



(a) Split CIFAR: Individual task accuracy and their average



(b) Split CIFAR



(c) Permuted MNIST

Figure 3. Fig. (a) shows that FROMP outperforms weight-regularisation methods (see Appendix F.3 for numerical values). 'Tx' means Task x. Figs. (b) and (c) show average accuracy with respect to the number of memorable examples. A careful selection of memorable examples in FROMP gives better results than random examples in FRORP, especially when the memory size is small. For CIFAR, the kernel in FROMP improves performance over FROMP-L2, which does not use a kernel.

training cannot profit from forward/backward transfer from other tasks, and sets a lower limit for our performance as we want to do better than this limit. The second baseline is where we train all tasks *jointly*, which would yield perhaps the best results and we would like to match this upper limit.

The results are summarised in Figure 3a, where we see that FROMP is close to the upper limit and outperforms other methods. The weight-regularisation methods EWC and SI do not perform well on the later tasks while VCL forgets earlier tasks. Poor performance of VCL is most likely due to the difficulty of using Bayes By Backprop (Blundell et al., 2015) on CNNs[2] (Osawa et al., 2019; Shridhar et al., 2019).

[2]Previous results by Nguyen et al. (2018) and Swaroop et al. (2019) were obtained using multi-layer perceptrons.

FROMP performs consistently better across all tasks (except the first task where it is close to the best). It also improves over the lower limit ('separate tasks') by a large margin. In fact, on tasks 4-6, FROMP matches the performance to the network trained jointly on all tasks, which implies that it completely avoids catastrophic forgetting on them. The overall performance is also the best (see the 'Avg' column).

Figure 3b shows the performance with respect to the number of memorable past examples. Similar to Figure 3c, carefully selecting memorable examples and using the kernel improves the performance, especially when the number of memorable examples is small. For example, with 10 such memorable examples, a careful selection increases the average accuracy from 62% to 69% (see FROMP vs FRORP). The kernel increases the accuracy from 65% to 69% (see

FROMP vs FROMP-L_2). Figure 2b shows a few images with most and least memorable past examples where we again see that the most memorable might be more difficult to classify.

Finally, we analyse the forward and backward transfer obtained by FROMP. Forward transfer means the accuracy on the *current* tasks increases as number of past tasks increases, while backward transfer means the accuracy on the *previous* tasks increases as more tasks are observed. As discussed in Appendix E, we find that, for split CIFAR, FROMP's forward transfer is much better than VCL and EWC, while its backward transfer is comparable to EWC. We define a forward transfer metric as the average improvement in accuracy on a new task over a model trained *only* on that task (see Appendix E for an expression). A higher value is better and quantifies the performance gain by observing past tasks. FROMP achieves 6.1 ± 0.7, a much higher value compared to 0.17 ± 0.9 obtained with EWC and 1.8 ± 3.1 with VCL+coresets. For backward transfer, we used the BWT metric defined in Lopez-Paz & Ranzato (2017) which roughly captures the difference in accuracy obtained when a task is first trained and its accuracy after the final task. Again, higher is better and quantifies the gain obtained with the future tasks. Here, FROMP has a score of -2.6 ± 0.9, which is comparable to EWC's score of -2.3 ± 1.4 but better than VCL+coresets which obtains -9.2 ± 1.8.

5. Discussion

We propose FROMP, a functional-regularisation approach for continual learning while avoiding catastrophic forgetting. FROMP uses a Gaussian Process formulation of neural networks to convert weight-space distributions into function-space. With this formulation, we proposed ways to identify relevant *memorable past* examples, and functionally regularise the training of neural network weights. FROMP achieves state-of-the-art performance across benchmarks. This paper presents the first steps in a new direction and there are several future investigations of interest. Would using VI instead of a Laplace approximation result in better accuracy? What are some ways to choose memorable examples? Is there a common principle behind them? Do we obtain improvements when we relax some assumptions? And what kind of improvements? Will this approach work at very large scales? These are some of the questions that we hope to investigate in the future. Our hope is that our method opens a new direction for principled methods to combine regularisation and memory-based methods.

References

Alaoui, A. and Mahoney, M. W. Fast randomized kernel ridge regression with statistical guarantees. In *Advances in Neural Information Processing Systems*, pp. 775–783, 2015.

Aljundi, R., Lin, M., Goujaud, B., and Bengio, Y. Gradient based sample selection for online continual learning. *ArXiv*, abs/1903.08671, 2019.

Benjamin, A. S., Rolnick, D., and Kording, K. Measuring and regularizing networks in function space. *arXiv preprint arXiv:1805.08289*, 2018.

Bishop, C. M. *Pattern recognition and machine learning*. Springer, 2006.

Blundell, C., Cornebise, J., Kavukcuoglu, K., and Wierstra, D. Weight uncertainty in neural networks. *ICML*, 2015.

Bottou, L. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pp. 177–186. Springer, 2010.

Chaudhry, A., Ranzato, M., Rohrbach, M., and Elhoseiny, M. Efficient lifelong learning with A-GEM. *ArXiv*, abs/1812.00420, 2018.

Chaudhry, A., Rohrbach, M., Elhoseiny, M., Ajanthan, T., Dokania, P. K., Torr, P. H. S., and Ranzato, M. On tiny episodic memories in continual learning. *ArXiv*, abs/1902.10486, 2019.

Ebrahimi, S., Elhoseiny, M., and Trevor Darrell, M. R. Uncertainty-guided continual learning with Bayesian neural networks. *ArXiv*, abs/1906.02425, 2019.

Jacot, A., Gabriel, F., and Hongler, C. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in neural information processing systems*, pp. 8571–8580, 2018.

Khan, M., Nielsen, D., Tangkaratt, V., Lin, W., Gal, Y., and Srivastava, A. Fast and scalable Bayesian deep learning by weight-perturbation in Adam. In *ICML*, pp. 2611–2620. PMLR. org, 2018.

Khan, M. E., Immer, A., Abedi, E., and Korzepa, M. Approximate inference turns deep networks into Gaussian processes. *Advances in neural information processing systems*, 2019.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *ICLR*, 2015.

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. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

Lopez-Paz, D. and Ranzato, M. Gradient episodic memory for continual learning. In *Advances in Neural Information Processing Systems*, pp. 6467–6476, 2017.

Ma, P., Mahoney, M. W., and Yu, B. A statistical perspective on algorithmic leveraging. *Journal of Machine Learning Research*, 16(27):861–911, 2015. URL http://jmlr.org/papers/v16/ma15a.html.

Nguyen, C. V., Li, Y., Bui, T. D., and Turner, R. E. Variational continual learning. *ICLR*, 2018.

Osawa, K., Swaroop, S., Khan, M. E. E., Jain, A., Eschenhagen, R., Turner, R. E., and Yokota, R. Practical deep learning with Bayesian principles. *Advances in neural information processing systems*, 2019.

Rasmussen, C. and Williams, C. *Gaussian Processes for Machine Learning*. Adaptive Computation and Machine Learning. MIT Press, Cambridge, MA, USA, January 2006.

Rebuffi, S.-A., Kolesnikov, A., Sperl, G., and Lampert, C. H. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 2001–2010, 2017.

Ritter, H., Botev, A., and Barber, D. Online structured laplace approximations for overcoming catastrophic forgetting. In *Advances in Neural Information Processing Systems*, pp. 3738–3748, 2018.

Serra, J., Suris, D., Miron, M., and Karatzoglou, A. Overcoming catastrophic forgetting with hard attention to the task. In *ICML*, pp. 4548–4557. PMLR. org, 2018.

Shin, H., Lee, J. K., Kim, J., and Kim, J. Continual learning with deep generative replay. In *Advances in neural information processing systems*, 2017.

Shridhar, K., Laumann, F., and Liwicki, M. A comprehensive guide to Bayesian convolutional neural network with variational inference. *ArXiv*, abs/1901.02731, 2019.

Smola, A. J., Vishwanathan, V., and Eskin, E. Laplace propagation. In *Advances in neural information processing systems*, pp. 441–448, 2003.

Swaroop, S., Nguyen, C. V., Bui, T. D., and Turner, R. E. Improving and understanding variational continual learning. *arXiv preprint arXiv:1905.02099*, 2019.

Titsias, M. K., Schwarz, J., Matthews, A. G. d. G., Pascanu, R., and Teh, Y. W. Functional regularisation for continual learning using Gaussian processes. *arXiv preprint arXiv:1901.11356*, 2019.

Train, K. E. *Discrete choice methods with simulation*. Cambridge university press, 2009.

Zenke, F., Poole, B., and Ganguli, S. Continual learning through synaptic intelligence. In *ICML*, pp. 3987–3995. JMLR. org, 2017.

A. Deep Networks to Functional Priors with DNN2GP

A.1. GP posteriors from the Minimiser of Linear Model

The posterior distribution of a linear model induces a GP posterior as shown by Rasmussen & Williams (2006). We discuss this in detail now for the following linear model discussed in Section 3.1:

$$y_i = f_w(\mathbf{x}_i) + \epsilon_i, \text{ where } f_w(\mathbf{x}_i) := \boldsymbol{\phi}(\mathbf{x}_i)^\top \mathbf{w}, \quad \epsilon_i \sim \mathcal{N}(\epsilon_i|0, \Lambda^{-1}), \quad \text{and } \mathbf{w} \sim \mathcal{N}(\mathbf{w}|0, \delta^{-1}\mathbf{I}_P) \tag{11}$$

with a feature map $\boldsymbol{\phi}(\mathbf{x})$. Rasmussen & Williams (2006) show that the predictive distribution for a test input $\mathbf{x}$ takes the following form (see Equation 2.11 in their book):

$$p(f(\mathbf{x})|\mathbf{x}, \mathcal{D}) = \mathcal{N}(f(\mathbf{x}) \,|\, \Lambda \boldsymbol{\phi}(\mathbf{x})^\top \mathbf{A}^{-1} \boldsymbol{\Phi}\mathbf{y}, \;\; \boldsymbol{\phi}(\mathbf{x})^\top \mathbf{A}^{-1} \boldsymbol{\phi}(\mathbf{x})), \quad \text{where } \mathbf{A} := \sum_i \boldsymbol{\phi}(\mathbf{x}_i)\, \Lambda\, \boldsymbol{\phi}(\mathbf{x}_i)^\top + \delta \mathbf{I}_P. \tag{12}$$

where $\mathcal{D}$ is set of training points $\{y_i, \mathbf{x}_i\}$ for i, and $\boldsymbol{\Phi}$ is a matrix with $\boldsymbol{\phi}(\mathbf{x}_i)$ as columns.

Rasmussen & Williams (2006) derive the above predictive distribution by using the weight-space posterior $\mathcal{N}(\mathbf{w}|\mathbf{w}_{\text{lin}}, \boldsymbol{\Sigma}_{\text{lin}})$ with the mean and covariance defined as below:

$$\mathbf{w}_{\text{lin}} := \Lambda \mathbf{A}^{-1} \boldsymbol{\Phi}\mathbf{y}, \quad \boldsymbol{\Sigma}_{\text{lin}} := \mathbf{A}^{-1}. \tag{13}$$

The mean $\mathbf{w}_{lin}$ is also the minimiser of the least-squares loss and $\mathbf{A}$ is the hessian at that solution.

Rasmussen & Williams (2006) show that the predictive distribution in Equation 12 corresponds to a GP posterior with the following mean and covariance functions:

$$m_{\text{lin}}(\mathbf{x}) = \Lambda \boldsymbol{\phi}(\mathbf{x})^\top \mathbf{A}^{-1} \boldsymbol{\Phi}\mathbf{y} = \boldsymbol{\phi}(\mathbf{x})^\top \mathbf{w}_{\text{lin}} = f_{w_{\text{lin}}}(\mathbf{x}), \tag{14}$$

$$\kappa_{\text{lin}}(\mathbf{x}, \mathbf{x}') := \boldsymbol{\phi}(\mathbf{x})^\top \boldsymbol{\Sigma}_{\text{lin}}\, \boldsymbol{\phi}(\mathbf{x}'), \tag{15}$$

This is the result shown in Equation 2 in Section 3.1. We can also write the predictive distribution of the observation $y = f(\mathbf{x}) + \epsilon$ where $\epsilon \sim \mathcal{N}(0, \Lambda^{-1})$ as follows:

$$p(y|\mathbf{x}, \mathcal{D}) = \mathcal{N}(y \,|\, \underbrace{f_{w_{\text{lin}}}(\mathbf{x})}_{m_{\text{lin}}(\mathbf{x})}, \; \underbrace{\boldsymbol{\phi}(\mathbf{x})^\top \boldsymbol{\Sigma}_{\text{lin}} \boldsymbol{\phi}(\mathbf{x})}_{\kappa_{\text{lin}}(\mathbf{x},\mathbf{x})} + \Lambda^{-1}), \quad \text{where } \boldsymbol{\Sigma}_{\text{lin}}^{-1} := \sum_i \boldsymbol{\phi}(\mathbf{x}_i)\, \Lambda\, \boldsymbol{\phi}(\mathbf{x}_i)^\top + \delta \mathbf{I}_P. \tag{16}$$

We will make use of Equations 14 to 16 to write the mean and covariance function of the posterior approximation for neural networks, as shown in the next section.

A.2. GP Posteriors from the Minimiser of Neural Networks

Khan et al. (2019) derive GP predictive distributions for the minimisers of a variety of loss functions in Appendix B of their paper. We describe these below along with the resulting GP posteriors. Throughout, we denote a minimiser of the loss by $\mathbf{w}_*$.

A regression loss: For a regression loss function $\ell(y, f) := \frac{1}{2}\Lambda(y - f)^2$, they derive the following expression for the predictive distribution for the observations y (see Equation 44, Appendix B.2 in their paper):

$$\hat{p}(y|\mathbf{x}, \mathcal{D}) := \mathcal{N}(y \,|\, f_{w_*}(\mathbf{x}), \;\; \mathbf{J}_{w_*}(\mathbf{x}) \boldsymbol{\Sigma}_* \mathbf{J}_{w_*}(\mathbf{x})^\top + \Lambda^{-1}), \quad \text{where } \boldsymbol{\Sigma}_*^{-1} := \sum_i \mathbf{J}_{w_*}(\mathbf{x}_i)^\top\, \Lambda\, \mathbf{J}_{w_*}(\mathbf{x}_i) + \delta \mathbf{I}_P. \tag{17}$$

We use $\hat{p}(y|\mathbf{x}, \mathcal{D})$ since this predictive distribution is not exact and is obtained using a type of Laplace approximation. Comparing this to Equation 16, we can write the mean and covariance functions in a similar fashion as Equations 14 and 15:

$$m_{w_*}(\mathbf{x}) := f_{w_*}(\mathbf{x}), \quad \kappa_{w_*}(\mathbf{x}, \mathbf{x}') := \mathbf{J}_{w_*}(\mathbf{x})\, \boldsymbol{\Sigma}_*\, \mathbf{J}_{w_*}(\mathbf{x}')^\top. \tag{18}$$

This is the result shown in Equation 5 in Section 3.1.

A binary classification loss: A similar expression is available for binary classification with $y \in \{0, 1\}$, considering the loss $\ell(y, f) := -y \log \sigma(f) - (1-y)\log(1-\sigma(f)) = -yf + \log(1+e^f)$ where $\sigma(f) := 1/(1+e^{-f})$ is the sigmoid function. See Equation 48, Appendix B.2 in Khan et al. (2019). The predictive distribution is given as follows:

$$\hat{p}(y|\mathbf{x}, \mathcal{D}) := \mathcal{N}(y \,|\, \sigma(f_{w_*}(\mathbf{x})),\ \ \Lambda_{w_*}(\mathbf{x})\, \mathbf{J}_{w_*}(\mathbf{x})\, \boldsymbol{\Sigma}_*\, \mathbf{J}_{w_*}(\mathbf{x})^\top \Lambda_{w_*}(\mathbf{x}) + \Lambda_{w_*}(\mathbf{x})), \tag{19}$$

$$\text{where } \boldsymbol{\Sigma}_*^{-1} := \sum_i \mathbf{J}_{w_*}(\mathbf{x}_i)^\top\, \Lambda_{w_*}(\mathbf{x}_i)\, \mathbf{J}_{w_*}(\mathbf{x}_i) + \delta \mathbf{I}_P. \tag{20}$$

where $\boldsymbol{\Lambda}_{w_*}(\mathbf{x}) := \sigma\left(f_{w_*}(\mathbf{x})\right)\left[1 - \sigma\left(f_{w_*}(\mathbf{x})\right)\right]$. The predictive distribution does not respect the fact that y is binary and treats it like a Gaussian. This makes it comparable to Equation 16. Comparing the two, we can conclude that the above corresponds to the predictive posterior distribution of a GP regression model with $y = f(\mathbf{x}) + \epsilon$ where $\epsilon \sim \mathcal{N}(0, \Lambda_{w_*}(\mathbf{x}))$ with the mean and covariance function as shown below:

$$m_{w_*}(\mathbf{x}) := \sigma(f_{w_*}(\mathbf{x})), \quad \kappa_{w_*}(\mathbf{x}, \mathbf{x}') := \Lambda_{w_*}(\mathbf{x})\, \mathbf{J}_{w_*}(\mathbf{x})\, \boldsymbol{\Sigma}_*\, \mathbf{J}_{w_*}(\mathbf{x}')^\top \Lambda_{w_*}(\mathbf{x}). \tag{21}$$

This is the result used in Equation 10 in Section 3.4 for binary classification. A difference here is that the mean function is passed through the sigmoid function and the covariance function has $\Lambda_{w_*}(\mathbf{x})$ multiplied on the both sides. These changes appear because of the nonlinearity in the loss function introduced due to the sigmoid link function.

A multiclass classification loss: The above result straightforwardly extends to the multiclass classification case by using multinomial-logit likelihood (or softmax function). For this the loss can be written as follows:

$$\ell(\mathbf{y}, \mathbf{f}) = -\mathbf{y}^\top \mathcal{S}(\mathbf{f}) + \log\left(1 + \sum_{k=1}^{K-1} e^{f_k}\right), \text{ where } k\text{'th element of } \mathcal{S}(\mathbf{f}) \text{ is given by } \frac{e^{f_j}}{1 + \sum_{k=1}^{K-1} e^{f_k}} \tag{22}$$

where the number of categories is equal to K, $\mathbf{y}$ is a one-hot-encoding vector of size $K-1$, $\mathbf{f}$ is $K-1$ length output of the neural network, and $\mathcal{S}(\mathbf{f})$ is the softmax operation which maps a $K-1$ length real vector to a $K-1$ dimensional vector with entries in the open interval $(0, 1)$. The encoding in $K-1$ length vectors ignores the last category which then ensures identifiability (Train, 2009). In a similar fashion to the binary case, the predictive distribution of the $K-1$ length output $\mathbf{y}$ for an input $\mathbf{x}$ can be written as follows:

$$\hat{p}(\mathbf{y}|\mathbf{x}, \mathcal{D}) := \mathcal{N}(\mathbf{y} \,|\, \mathcal{S}(\mathbf{f}_{w_*}(\mathbf{x})),\ \ \boldsymbol{\Lambda}_{w_*}(\mathbf{x})\, \mathbf{J}_{w_*}(\mathbf{x})\, \boldsymbol{\Sigma}_*\, \mathbf{J}_{w_*}(\mathbf{x})^\top \boldsymbol{\Lambda}_{w_*}(\mathbf{x})^\top + \boldsymbol{\Lambda}_{w_*}(\mathbf{x})),$$

$$\text{where } \boldsymbol{\Sigma}_*^{-1} := \sum_i \mathbf{J}_{w_*}(\mathbf{x}_i)^\top\, \boldsymbol{\Lambda}_{w_*}(\mathbf{x}_i)\, \mathbf{J}_{w_*}(\mathbf{x}_i) + \delta \mathbf{I}_P. \tag{23}$$

where $\boldsymbol{\Lambda}_{w_*}(\mathbf{x}) := \mathcal{S}\left(\mathbf{f}_{w_*}(\mathbf{x})\right)\left[1 - \mathcal{S}\left(\mathbf{f}_{w_*}(\mathbf{x})\right)\right]^\top$ is a $(K-1) \times (K-1)$ matrix and $\mathbf{J}_{w_*}(\mathbf{x})$ is the $(K-1) \times P$ Jacobian matrix. The mean function in this case is a $K-1$ length matrix and the covariance function is a square matrix of size $K-1$. Their expressions are shown below:

$$\mathbf{m}_{w_*}(\mathbf{x}) := \mathcal{S}(\mathbf{f}_{w_*}(\mathbf{x})), \quad \boldsymbol{K}_{w_*}(\mathbf{x}, \mathbf{x}') := \boldsymbol{\Lambda}_{w_*}(\mathbf{x})\, \mathbf{J}_{w_*}(\mathbf{x})\, \boldsymbol{\Sigma}_*\, \mathbf{J}_{w_*}(\mathbf{x}')^\top \boldsymbol{\Lambda}_{w_*}(\mathbf{x}'). \tag{24}$$

General case: The results above hold for a generic loss function derived from a generalised linear model (GLM) with an invertible function $\mathbf{h}(\mathbf{f})$, e.g., $\ell(\mathbf{y}, \mathbf{f}) := -\log p(\mathbf{y}|\mathbf{h}(\mathbf{f}))$. For example, for a Bernoulli distribution, the link function $h(f)$ is equal to σ. In the GLM literature, $\mathbf{h}^{-1}$ is known as the link function. Given such a loss, the only quantity that changes in the above calculations is $\boldsymbol{\Lambda}_{w_*}(\mathbf{x}, \mathbf{y}) := \nabla^2_{ff}\ell(\mathbf{y}, \mathbf{f})$, which is the second derivative of the loss with respect to $\mathbf{f}$, and might depend both on $\mathbf{x}$ and $\mathbf{y}$.

A.3. GP Posterior from the Iterations of a Neural-Network Optimiser

The results of the previous section hold only at a minimiser $\mathbf{w}_*$. Khan et al. (2019) generalise this to iterations of optimisers. They did this for a variational inference algorithm and also for its deterministic version that resembles RMSprop. We now describe these two versions. We will only consider binary classification using the setup described in the previous section. The results can be easily generalised to multiclass classification.

GP posterior from iterations of a variational inference algorithm: Given a Gaussian variational approximation $q_j(\mathbf{w}) := \mathcal{N}(\mathbf{w}|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)$ at iteration j, Khan et al. (2019) used a natural-gradient variational inference algorithm called the

variational-online Newton (VON) method (Khan et al., 2018). Given a $q_j(\mathbf{w})$, the algorithm proceeds by first sampling $\mathbf{w}_j \sim q_j(\mathbf{w})$, and then updating the variational distribution. Surprisingly, the procedure used to derive a GP predictive distribution for the minimiser generalises to this update too. An expression for the predictive distribution is given below:

$$\hat{p}_{j+1}(y|\mathbf{x}, \mathcal{D}) := \mathcal{N}(y \mid \sigma(f_{w_j}(\mathbf{x})),\ \Lambda_{w_j}(\mathbf{x})\, \mathbf{J}_{w_j}(\mathbf{x})\, \mathbf{\Sigma}_j\, \mathbf{J}_{w_j}(\mathbf{x})^\top \Lambda_{w_j}(\mathbf{x}) + \Lambda_{w_j}(\mathbf{x})^{-1}), \tag{25}$$

$$\text{where } \mathbf{\Sigma}_{j+1}^{-1} := (1-\beta_j)\mathbf{\Sigma}_j^{-1} + \beta_j \left[\sum_i \mathbf{J}_{w_j}(\mathbf{x}_i)^\top \Lambda_{w_j}(\mathbf{x}_i)\, \mathbf{J}_{w_j}(\mathbf{x}_i) + \delta \mathbf{I}_P \right], \tag{26}$$

$$\boldsymbol{\mu}_{j+1} := \boldsymbol{\mu}_j - \beta_j \mathbf{\Sigma}_{j+1} \left[N \nabla_w \bar{\ell}(\mathbf{w}_j) + \delta \boldsymbol{\mu}_j \right]. \tag{27}$$

where $\bar{\ell}(\mathbf{w}) := \frac{1}{N}\sum_{i=1}^N \ell(y_i, f_w(\mathbf{x}_i))$. The predictive distribution takes the same form as before, but now the covariance and mean are updated according to the VON updates. The VON updates are essential to ensure the validity of the GP posterior, however, as Khan et al. (2019) discuss, the RMSprop/Adam have similar update which enable us to apply the above results even when running such algorithms. We describe this next.

GP posterior from iterations of RMSprop/Adam: Khan et al. (2019) propose a deterministic version of the above update where $\mathbf{w}_j$ is not sampled from $q_j(\mathbf{w})$ rather is set to be equal to $\boldsymbol{\mu}_j$, i.e., $\mathbf{w}_j = \boldsymbol{\mu}_j$. This gives rise to the following update:

$$\mathbf{\Sigma}_{j+1}^{-1} \leftarrow (1-\beta_j)\mathbf{\Sigma}_j^{-1} + \beta_j \left[\sum_i \mathbf{J}_{w_j}(\mathbf{x}_i)^\top \Lambda_{w_j}(\mathbf{x}_i)\, \mathbf{J}_{w_j}(\mathbf{x}_i) + \delta \mathbf{I}_P \right], \tag{28}$$

$$\mathbf{w}_{j+1} \leftarrow \mathbf{w}_j - \beta_j \mathbf{\Sigma}_{j+1} \left[N \nabla_w \bar{\ell}(\mathbf{w}_j) + \delta \mathbf{w}_j \right], \tag{29}$$

with the variational approximation defined as $q_j(\mathbf{w}) := \mathcal{N}(\mathbf{w}|\mathbf{w}_j, \mathbf{\Sigma}_j)$. The form of the predictive distribution remains the same as Equation 25.

As discussed in Khan et al. (2018), the above algorithm can be made similar to RMSprop by using a diagonal covariance. By reparameterising the diagonal of $\mathbf{\Sigma}^{-1}$ as $\mathbf{s} + \delta\mathbf{1}$ where $\mathbf{s}$ is an unknown vector, we can rewrite the updates to update $\boldsymbol{\mu}$ and $\mathbf{s}$. This can then be written in a form similar to RMSprop as shown below:

$$\mathbf{s}_{j+1} \leftarrow (1-\beta_j)\mathbf{s}_j + \beta_j \Big[\sum_i \Lambda_{w_j}(\mathbf{x}_i) \left[\mathbf{J}_{w_j}(\mathbf{x}_i) \circ \mathbf{J}_{w_j}(\mathbf{x}_i) \right]^\top \Big] \tag{30}$$

$$\mathbf{w}_{j+1} \leftarrow \mathbf{w}_j - \beta_t \frac{1}{\mathbf{s}_{j+1} + \delta\mathbf{1}} \circ \left[N \nabla_w \bar{\ell}(\mathbf{w}_t) + \delta \mathbf{w}_j \right], \tag{31}$$

where $\circ$ defines element-wise product of two vectors, and the diagonal of $\mathbf{\Sigma}_{j+1}^{-1}$ is equal to $(\mathbf{s}_{j+1} + \delta\mathbf{1})$. This algorithm differs from RMSprop in two ways. First, the scale vector $\mathbf{s}_j$ is updated using the sum of the square of the Jacobians instead of the square of the mini-batch gradients. Second, there is no square-root in the preconditioner for the gradient in the second line. This algorithm is the diagonal version of the Online Generalised Gauss-Newton (OGGN) algorithm discussed in Khan et al. (2019).

In practice, we ignore these two differences and employ the RMSprop/Adam update instead. As a consequence the variance estimates might not be very good during the iteration, even though the fixed-point of the algorithm is not changed (Khan et al., 2018). This is the price we pay for the convenience of using RMSprop/Adam. We correct the approximation after convergence of the algorithm by recomputing the diagonal of the covariance according to Equation 30. Denoting the converged solution by $\mathbf{w}_*$, we compute the diagonal $\mathbf{v}_*$ of the covariance $\mathbf{\Sigma}_*$ as shown below:

$$\mathbf{v}_* = \mathbf{1} / \Big[\delta\mathbf{1} + \sum_{i=1}^N \Lambda_{w_*}(\mathbf{x}_i) \left[\mathbf{J}_{w_*}(\mathbf{x}_i) \circ \mathbf{J}_{w_*}(\mathbf{x}_i) \right]^\top \Big], \tag{32}$$

B. Detailed Derivation of FROMP Algorithm

In this section, we provide further details on Section 3.3.

$$\mathcal{L}(q(\mathbf{w})) := \mathbb{E}_{q(w)} \left[\frac{N}{\tau} \bar{\ell}_t(\mathbf{w}) + \log q(\mathbf{w}) \right] - \mathbb{E}_{\tilde{q}_{w_t}(\mathbf{f})} \left[\log \tilde{q}_{w_{t-1}}(\mathbf{f}) \right], \text{ where } \mathbf{w}_t \sim q(\mathbf{w}) \text{ and } \mathbf{w}_{t-1} \sim q_{t-1}(\mathbf{w}) \tag{33}$$

Optimising this objective requires us to obtain the GP posterior $\tilde{q}_{w_t}(\mathbf{f})$. This can be easily done applying the DNN2GP result from Equation 25 to this loss function. The VON update for the objective above takes the following form:

$$\boldsymbol{\Sigma}^{-1} \leftarrow (1-\beta)\boldsymbol{\Sigma}^{-1} + \beta\left[\sum_i \mathbf{J}_{w_t}(\mathbf{x}_i)^\top \Lambda_{w_t}(\mathbf{x}_i)\,\mathbf{J}_{w_t}(\mathbf{x}_i) - \nabla_\Sigma \mathbb{E}_{\tilde{q}_{w_t}(\mathbf{f})}\left[\log \tilde{q}_{w_{t-1}}(\mathbf{f})\right]\right], \tag{34}$$

$$\boldsymbol{\mu} \leftarrow \boldsymbol{\mu} - \beta\boldsymbol{\Sigma}\left[\frac{N}{\tau}\nabla_w \bar{\ell}_t(\mathbf{w}_t) - \nabla_\mu \mathbb{E}_{\tilde{q}_{w_t}(\mathbf{f})}\left[\log \tilde{q}_{w_{t-1}}(\mathbf{f})\right]\right]. \tag{35}$$

where $\bar{\ell}_t(\mathbf{w}) := \frac{1}{N}\sum_{i\in\mathcal{D}_t} \ell(y_i, f_w(\mathbf{x}_i))$ and we have ignored the iteration subscript to simplify notation.

Using the $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ obtained with this iteration, we can define the following GP predictive posterior at a sample $\mathbf{w}_t \sim q(\mathbf{w})$:

$$\hat{p}_t(y|\mathbf{x},\mathcal{D}) := \mathcal{N}(y \mid \sigma(f_{w_t}(\mathbf{x})),\ \Lambda_{w_t}(\mathbf{x})\,\mathbf{J}_{w_t}(\mathbf{x})\,\boldsymbol{\Sigma}\,\mathbf{J}_{w_t}(\mathbf{x})^\top\Lambda_{w_t}(\mathbf{x}) + \Lambda_{w_t}(\mathbf{x})^{-1}), \tag{36}$$

Comparing this to Equation 25, we can write the mean and covariance function as follows:

$$m_{w_t}(\mathbf{x}) := \sigma(f_{w_t}(\mathbf{x})), \quad \kappa_{w_t}(\mathbf{x},\mathbf{x}') := \Lambda_{w_t}(\mathbf{x})\,\mathbf{J}_{w_t}(\mathbf{x})\,\boldsymbol{\Sigma}\,\mathbf{J}_{w_t}(\mathbf{x}')^\top\Lambda_{w_t}(\mathbf{x}). \tag{37}$$

The mean vector obtained by concatenating $m_{w_t}(\mathbf{x})$ at all $\mathbf{x} \in \mathcal{M}$ is denoted by $\mathbf{m}_t$. Similarly, the covariance matrix $\mathbf{K}_t$ is defined as the matrix with ij'th entry as $\kappa_{w_t}(\mathbf{x}_i, \mathbf{x}_j)$. The corresponding mean and covariance obtained from samples from $q_{t-1}(\mathbf{w})$ are denoted by $\mathbf{m}_{t-1}$ and $\mathbf{K}_{t-1}$.

Given these quantities, the functional regularisation term has an analytical expression given as follows:

$$\mathbb{E}_{\tilde{q}_{w_t}(\mathbf{f})}\left[\log \tilde{q}_{w_{t-1}}(\mathbf{f})\right] = -\tfrac{1}{2}\left[\mathrm{Tr}(\mathbf{K}_{t-1}^{-1}\mathbf{K}_t) + (\mathbf{m}_t - \mathbf{m}_{t-1})^\top \mathbf{K}_{t-1}^{-1}(\mathbf{m}_t - \mathbf{m}_{t-1})\right] + c, \tag{38}$$

Our goal is to obtain the derivative of this term with respect to $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$. Both $\mathbf{m}_t$ and $\mathbf{K}_t$ are functions of $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ through the sample $\mathbf{w}_t = \boldsymbol{\mu} + \boldsymbol{\Sigma}^{1/2}\boldsymbol{\epsilon}$ where $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I})$. Therefore, we can compute these derivative using the chain rule.

We note that the resulting algorithm is costly for large problems, and propose five approximations to reduce the computation cost, as described below.

Approximation 1: Instead of sampling $\mathbf{w}_{t-1}$, we set $\mathbf{w}_{t-1} = \boldsymbol{\mu}_{t-1}$ which is the mean of the posterior approximation $q_{t-1}(\mathbf{w})$ until task $t-1$. Therefore, we replace $\mathbb{E}_{\tilde{q}_{w_t}(\mathbf{f})}\left[\log \tilde{q}_{w_{t-1}}(\mathbf{f})\right]$ by $\mathbb{E}_{\tilde{q}_{w_t}(\mathbf{f})}\left[\log \tilde{q}_{\boldsymbol{\mu}_{t-1}}(\mathbf{f})\right]$. This affects the mean $\mathbf{m}_{t-1}$ and $\mathbf{K}_{t-1}$ in Equation 38.

Approximation 2: When computing the derivation of the functional regulariser, we will ignore the derivative with respect to $\mathbf{K}_t$ and only consider $\mathbf{m}_t$. Therefore, the derivatives needed for the update in Equations 34 and 35 can be approximated as follows:

$$\nabla_\mu \mathbb{E}_{\tilde{q}_{w_t}(\mathbf{f})}\left[\log \tilde{q}_{w_{t-1}}(\mathbf{f})\right] \approx -\left[\nabla_\mu \mathbf{m}_t\right]\mathbf{K}_{t-1}^{-1}(\mathbf{m}_t - \mathbf{m}_{t-1}), \tag{39}$$

$$\nabla_\Sigma \mathbb{E}_{\tilde{q}_{w_t}(\mathbf{f})}\left[\log \tilde{q}_{w_{t-1}}(\mathbf{f})\right] \approx -\left[\nabla_\Sigma \mathbf{m}_t\right]\mathbf{K}_{t-1}^{-1}(\mathbf{m}_t - \mathbf{m}_{t-1}), \tag{40}$$

This avoids having to calculate complex derivatives (e.g., derivatives of Jacobians).

Approximation 3: Instead of using the full $\mathbf{K}_{t-1}$, we factorise it across tasks, i.e., we approximate it by a block-diagonal matrix containing the kernel matrix $\mathbf{K}_{t-1,s}$ for all past tasks s as the diagonal. This makes the cost of inversion linear in the number of tasks.

Approximation 4: Similarly to Equations 28 and 29, we use a deterministic version of the VON update by setting $\mathbf{w}_t = \boldsymbol{\mu}$, which corresponds to setting the random noise ϵ to zero in $\mathbf{w}_t = \boldsymbol{\mu} + \boldsymbol{\Sigma}^{1/2}\boldsymbol{\epsilon}$. This approximation simplifies the gradient computation in Equations 39 and 40, since now the gradient with respect to $\boldsymbol{\Sigma}$ is zero. For example, in the binary classification case, $m_\mu(\mathbf{x}) := \sigma(f_\mu(\mathbf{x}))$, which does not depend on $\boldsymbol{\Sigma}$. The gradient of $\mathbf{m}_t$ with respect to $\boldsymbol{\mu}$ is given as follows using the chain rule (here $\mathbf{m}_{t,s}$ is the sub-vector of $\mathbf{m}_t$ corresponding to the task s).

$$\nabla_\mu \mathbf{m}_{t,s}[i] = \nabla_\mu\left[\sigma\left(f_\mu(\mathbf{x}_i)\right)\right] = \Lambda_\mu(\mathbf{x}_i)\,\mathbf{J}_\mu(\mathbf{x}_i)^\top, \text{ where } \mathbf{x}_i \in \mathcal{M}_s, \tag{41}$$

and where the second equality holds for canonical link functions. With these simplifications, we can write the VON update as follows:

$$\boldsymbol{\Sigma}^{-1} \leftarrow (1-\beta)\boldsymbol{\Sigma}^{-1} + \beta\left[\sum_i \mathbf{J}_\mu(\mathbf{x}_i)^\top \Lambda_\mu(\mathbf{x}_i)\,\mathbf{J}_\mu(\mathbf{x}_i)\right], \tag{42}$$

$$\boldsymbol{\mu} \leftarrow \boldsymbol{\mu} - \beta\boldsymbol{\Sigma}\left[\frac{N}{\tau}\nabla_\mu \bar{\ell}_t(\boldsymbol{\mu}) + \sum_{s=1}^{t-1}\left[\nabla_\mu \mathbf{m}_{t,s}\right]\mathbf{K}_{t-1,s}^{-1}(\mathbf{m}_{t,s}-\mathbf{m}_{t-1,s})\right]. \tag{43}$$

Approximation 5: Similarly to Equations 30 and 31, our final approximation is to use a diagonal covariance $\boldsymbol{\Sigma}$ and replace the above update by an RMSprop-like update where we denote $\boldsymbol{\mu}$ by $\mathbf{w}$:

$$\mathbf{s}^{-1} \leftarrow (1-\beta)\mathbf{s} + \beta\left[\sum_i \Lambda_w(\mathbf{x}_i)\left[\mathbf{J}_w(\mathbf{x}_i)\circ\mathbf{J}_w(\mathbf{x}_i)\right]^\top\right], \tag{44}$$

$$\mathbf{w} \leftarrow \mathbf{w} - \beta\frac{1}{\mathbf{s}+\delta\mathbf{1}}\circ\left[\frac{N}{\tau}\nabla_w \bar{\ell}_t(\mathbf{w}) + \sum_{s=1}^{t-1}\left[\nabla_w \mathbf{m}_{t,s}\right]\mathbf{K}_{t-1,s}^{-1}(\mathbf{m}_{t,s}-\mathbf{m}_{t-1,s})\right], \tag{45}$$

where we have added a regulariser δ to $\mathbf{s}$ in the second line to avoid dividing by zero. Previously (Khan et al., 2018), this regulariser was the prior precision. Ideally, when using a functional prior, we would replace this by another term. However, this term was ignored by making Approximation 4, and we use δ instead. The final Gaussian approximation is obtained with the mean equal to $\mathbf{w}$ and covariance is equal to a diagonal matrix with $1/(\mathbf{s}+\delta\mathbf{1})$ as its diagonal.

It is easy to see that the solutions found by this algorithm is the fixed point of this objective:

$$\min_w N\bar{\ell}_t(\mathbf{w}) + \tau\sum_{s=1}^{t-1}(\mathbf{m}_{t,s}-\mathbf{m}_{t-1,s})^\top\mathbf{K}_{t-1,s}^{-1}(\mathbf{m}_{t,s}-\mathbf{m}_{t-1,s}), \tag{46}$$

Ultimately, this is an approximation of the objective given in Equation 33, and is computationally cheaper to optimise.

We follow the recommendations of Khan et al. (2019) and use RMSprop/Adam instead of Equations 28 and 29. This algorithm still optimises the objective given in Equation 46, but the estimate of the covariance is not accurate. We correct the approximation after convergence of the algorithm by recomputing the diagonal of the covariance according to Equation 44. Denoting the converged solution by $\mathbf{w}_*$, we compute the diagonal $\mathbf{v}_*$ of the covariance $\boldsymbol{\Sigma}_*$ as shown below:

$$\mathbf{v}_* = \mathbf{1}/\left[\delta\mathbf{1} + \sum_{i=1}^{N}\Lambda_{w_*}(\mathbf{x}_i)\left[\mathbf{J}_{w_*}(\mathbf{x}_i)\circ\mathbf{J}_{w_*}(\mathbf{x}_i)\right]^\top\right], \tag{47}$$

C. Multiclass setting

When there are more than two classes per task, we need to use multiclass versions of the equations presented so far. We still make the same approximations as described in Appendix B.

Reducing Complexity in the Multiclass setting: We could use the full multiclass version of the GP predictive (Equation 23), but this is expensive. To keep computational complexity low, we employ an individual GP over each of the K classes seen in a previous task, and treat the GPs as independent.

We have K separate GPs. Let $\mathrm{y}^{(k)}$ be the k-th item of $\mathbf{y}$. Then the predictive distribution over each $\mathrm{y}^{(k)}$ for an input $\mathbf{x}$ is:

$$\hat{p}(\mathrm{y}^{(k)}|\mathbf{x},\mathcal{D}) := \mathcal{N}(\mathrm{y}^{(k)} \mid \mathcal{S}(\mathbf{f}_{w_*}(\mathbf{x}))^{(k)},\ \boldsymbol{\Lambda}_{w_*}(\mathbf{x})^{(k)}\,\mathbf{J}_{w_*}(\mathbf{x})\,\boldsymbol{\Sigma}_*\,\mathbf{J}_{w_*}(\mathbf{x})^\top\boldsymbol{\Lambda}_{w_*}(\mathbf{x})^{(k)\top} + \Lambda_{w_*}(\mathbf{x})^{(k,k)}), \tag{48}$$

where $\mathcal{S}(\mathbf{f}_{w_*}(\mathbf{x}))^{(k)}$ is the k-th output of the softmax function, $\boldsymbol{\Lambda}_{w_*}(\mathbf{x})^{(k)}$ is the k-th row of the Hessian matrix and $\Lambda_{w_*}(\mathbf{x})^{(k,k)}$ is the k,k-th element of the Hessian matrix. The Jacobians $\mathbf{J}_{w_*}(\mathbf{x})$ are now of size $K\times P$. Note that we have allowed $\mathcal{S}$ and $\Lambda_{w_*}(\mathbf{x})$ to be of size K instead of $K-1$. This is because we are treating the K GPs separately.

The kernel matrix $\mathbf{K}_{t-1}$ is now a block diagonal matrix for each previous task's classes. This allows us to only compute inverses of each block diagonal (size $M\times M$), repeated for each class in each past task ($K(t-1)$ times), where M is the

number of memorable past examples in each task. This changes computational complexity to be linear in the number of classes per task, K, compared to Section 3.4 (which has analysis for binary classification for each task).

When choosing a memorable past (the subset of points to regularise function values over) for the logistic regression case, we can simply sort the $\Lambda_{w_*}(\mathbf{x}_i)$'s for all $\{\mathbf{x}_i\} \in \mathcal{D}_t$ and pick the largest, as explained in Section 3.2. In the multiclass case, these are now $K \times K$ matrices $\mathbf{\Lambda}_{w_*}(\mathbf{x}_i)$. We instead sort by $\mathrm{Tr}(\mathbf{\Lambda}_{w_*}(\mathbf{x}_i))$ to select the memorable past examples.

FROMP for multiclass classification: The solutions found by the multiclass algorithm is the fixed point of this objective (compare with Equation 46):

$$\min_{w} N\bar{\ell}_t(\mathbf{w}) + \tau \sum_{s=1}^{t-1} \sum_{k \in C_s} (\mathbf{m}_{t,s,k} - \mathbf{m}_{t-1,s,k})^\top \mathbf{K}_{t-1,s,k}^{-1} (\mathbf{m}_{t,s,k} - \mathbf{m}_{t-1,s,k}), \tag{49}$$

where we define C_s as the set of classes k seen in previous task s, $\mathbf{m}_{t,s,k}$ is the vector of $m_{w_t}(\mathbf{x})$ for class k evaluated at the memorable points $\{\mathbf{x}_i\} \in \mathcal{M}_s$, $\mathbf{m}_{t-1,s,k}$ is the vector of $m_{w_{t-1}}(\mathbf{x})$ for class k, and $\mathbf{K}_{t-1,s,k}$ is the kernel matrix from the previous task just for class k, always evaluated over just the memorable points from previous task s. By decomposing the last term over individual outputs and over the memorable past from each task, we have reduced the computational complexity per update.

D. Functional prior approximation

We discuss why replacing weight space integral by a function space integral, as done below, results in an approximation:

$$\mathbb{E}_{q(w)}[\log q_{t-1}(\mathbf{w})] \approx \mathbb{E}_{\tilde{q}_{w_t}(\mathbf{f})}\left[\log \tilde{q}_{w_{t-1}}(\mathbf{f})\right],$$

A change of variable in many cases results in an equality, e.g., for $\mathbf{f} = \mathbf{X}\mathbf{w}$ with a matrix $\mathbf{X}$ and given any function $h(\mathbf{f})$, we can express the weight space integral as the function space integral:

$$\int h(\mathbf{X}\mathbf{w})\mathcal{N}(\mathbf{w}|\boldsymbol{\mu}, \boldsymbol{\Sigma})d\mathbf{w} = \int h(\mathbf{f})\mathcal{N}(\mathbf{f}|\mathbf{X}\boldsymbol{\mu}, \mathbf{X}\boldsymbol{\Sigma}\mathbf{X}^\top)d\mathbf{f}. \tag{50}$$

Unfortunately, $\log q_{t-1}(\mathbf{w})$ can not always be written as a function of $\mathbf{f} := \mathbf{J}_{w_t}\mathbf{w}$. Therefore, the change of variable does not result in an equality. For our purpose, as long as the approximations provide a reasonable surrogate for optimisation, the approximation is not expected to cause issues.

E. Further details on continual learning metrics reported

We report a backward transfer metric and a forward transfer metric on Split CIFAR (higher is better for both). The backward transfer metric is exactly as defined in Lopez-Paz & Ranzato (2017). The forward transfer metric is a measure of how well the method uses previously seen knowledge to improve classification accuracy on newly seen tasks. Let there be a total of T tasks. Let $R_{i,j}$ be the classification accuracy of the model on task t_j after training on task t_i. Let R_i^{ind} be the classification accuracy of an independent model trained only on task i. Then,

$$\text{Backward Transfer, BWT} = \frac{1}{T-1}\sum_{i=1}^{T-1} R_{T,i} - R_{i,i},$$

$$\text{Forward Transfer, FWT} = \frac{1}{T-1}\sum_{i=2}^{T} R_{i,i} - R_i^{\text{ind}}.$$

FROMP achieves 6.1 ± 0.7, a much higher value compared to 0.17 ± 0.9 obtained with EWC and 1.8 ± 3.1 with VCL+coresets. For backward transfer, we used the BWT metric defined in (Lopez-Paz & Ranzato, 2017) which roughly captures the difference in accuracy obtained when a task is first trained and its accuracy after the final task. Again, higher is better and quantifies the gain obtained with the future tasks. Here, FROMP has a score of -2.6 ± 0.9, which is comparable to EWC's score of -2.3 ± 1.4 but better than VCL+coresets which obtains -9.2 ± 1.8.

Table 1. Summary of metrics on Split CIFAR. FROMP outperforms the baselines EWC and VCL+coresets. All methods are run five times, with mean and standard deviation reported.

Method	Final average accuracy	Forward transfer	Backward transfer
EWC	$71.6 \pm 0.9\%$	0.17 ± 0.9	$\mathbf{-2.3} \pm 1.4$
VCL+coresets	$67.4 \pm 1.4\%$	1.8 ± 3.1	-9.2 ± 1.8
FROMP	$\mathbf{76.2} \pm 0.4\%$	$\mathbf{6.1} \pm 0.7$	$\mathbf{-2.6} \pm 0.9$

F. Further details on image experiments

F.1. Permuted MNIST

We use the Adam optimiser (Kingma & Ba, 2015) with Adam learning rate set to 0.001 and parameter $\beta_1 = 0.99$, and also employ gradient clipping. The minibatch size is 128, and we learn each task for 10 epochs. We use $\tau = N$ for all algorithms: FROMP, FRORP, FROMP-L_2 and FRORP-L_2. We use a fully connected single-head network with two hidden layers, each consisting of 100 hidden units with ReLU activation functions. We report performance after 10 tasks.

F.2. Split MNIST

We use the Adam optimiser (Kingma & Ba, 2015) with Adam learning rate set to 0.0001 and parameter $\beta_1 = 0.99$, and also employ gradient clipping. The minibatch size is 128, and we learn each task for 15 epochs. We find good settings of τ to be $\tau = N$ for FROMP and FRORP, and $\tau = 0.1N$ for FROMP-L_2 and FRORP-L_2. We use a fully connected multi-head network with two hidden layers, each with 256 hidden units and ReLU activation functions.

Smaller network architecture from Swaroop et al. (2019). Swaroop et al. (2019) use a smaller network than the network we use for the results in Figure 2a. They train VCL on a single-hidden layer network with 100 hidden units (and ReLU activation functions). To ensure faithful comparison, we reran FROMP (with 40 memorable points per task) on this smaller network, obtaining a mean and standard deviation over 5 runs of $(99.2 \pm 0.1)\%$. This is an improvement from Figure 2a, which uses a larger network. We believe this is due to the pruning effect described in Swaroop et al. (2019).

Sensitivity to the value of τ. We tested FROMP and FROMP-L_2 with different values of the hyperparameter τ. We found that τ can change by an order of magnitude without significantly affecting final average accuracy. Larger changes in τ led to greater than 0.1% loss in accuracy.

F.3. Split CIFAR

We use the Adam optimiser (Kingma & Ba, 2015) with Adam learning rate set to 0.001 and parameter $\beta_1 = 0.99$, and also employ gradient clipping. The minibatch size is 256, and we learn each task for 80 epochs. We find good settings of τ to be $\tau = 10N$ for FROMP and FRORP, and $\tau = 20N$ for FROMP-L_2 and FRORP-L_2.

Numerical results on Split CIFAR. We run all methods 5 times and report the mean and standard error. For baselines, we train from scratch on each task and jointly on all tasks achieving $(73.6 \pm 0.4)\%$ and $(78.1 \pm 0.3)\%$, respectively. The final average validation accuracy of FROMP is $(76.2 \pm 0.4)\%$, FROMP-L_2 is $(74.6 \pm 0.4)\%$, SI is $(73.5 \pm 0.5)\%$ (result from Zenke et al. (2017)), EWC is $(71.6 \pm 0.9)\%$, VCL + random coreset is $(67.4 \pm 1.4)\%$.

F.4. Fewer memorable past examples

When we have fewer memorable past examples (for Figures 3b and 3c), we increase τ to compensate for the fewer datapoints. For example, for permuted MNIST, when we have 40 memorable past examples per task (instead of 200), we use $\tau = (200/40)N = 5N$ (instead of $\tau = N$ for 200 memorable past points).

G. Toy data experiments

In this section, we use a 2D binary classification toy dataset with a small multi-layer perceptron to (i) demonstrate the brittleness and inconsistent behaviour of weight-regularisation, (ii) test FROMP's performance on different toy datasets of varying difficulty. As shown in Figure 4 in Appendix G, we find that weight-regularisation methods like VCL (+coresets)

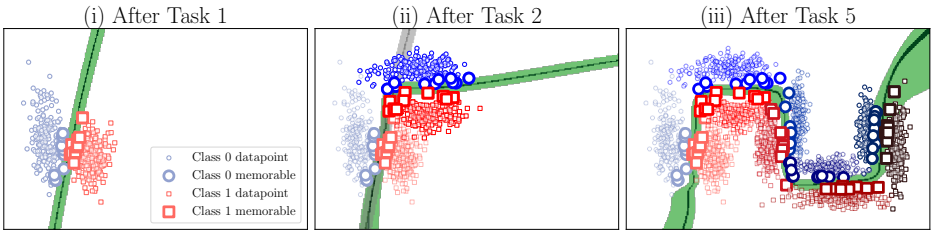


Figure 4. This figure demonstrates our approach on a toy dataset. Figure (i) shows the result of training on the first task where memorable past examples are shown with big markers. These points usually are the ones that support the decision boundary. Figure (ii) shows the result after training on the second task where we see that the new network outputs are forced to give the same prediction on memorable past examples as the previous network. The new decision boundary classifies both task 1 and 2 well. Figure (iii) shows the result after training on five tasks, along with the memorable-past of each task. With our method, the performance over past tasks is maintained.

perform much worse than functional-regularisation, with lower accuracy, higher variance over random seeds, and visually bad decision boundaries.

The toy dataset we use is shown in Figure 4, along with how FROMP does well. In Appendix G.1, we show weight-space regularisation's inconsistent behaviour on this dataset, with results and visualisations. In Appendix G.2, we show that FROMP performs consistently across many variations of the dataset. Finally, hyperparameters for our experiments are presented in Appendix G.3. For all these experiments, we use a 2-hidden layer single-head MLP with 20 hidden units in each layer.

G.1. Weight-space regularisation's inconsistent behaviour

Table 2. Train accuracy of FROMP, VCL (no coresets), VCL+coresets and batch-trained Adam (an upper bound on performance) on a toy 2D binary classification dataset, with mean and standard deviations over 5 runs for VCL and batch Adam, and 10 runs for FROMP. 'VCL' is without coresets. VCL-RP and FRORP have the same (random) coreset selections. VCL-MP is provided with 'ideal' coreset points as chosen by an independent run of FROMP. VCL (no coreset) does very poorly, forgetting previous tasks. VCL+coresets is brittle with high standard deviations, while FROMP is stable.

FROMP	FRORP	VCL-RP	VCL-MP	VCL	Batch Adam
$99.6 \pm 0.2\%$	$98.5 \pm 0.6\%$	$92 \pm 10\%$	$85 \pm 14\%$	$68 \pm 8\%$	$99.70 \pm 0.03\%$

Table 2 summarises the performance (measured by train accuracy) of FROMP and VCL+coresets on a toy dataset similar to that in Figure 4. FROMP is very consistent, while VCL (with coresets) is extremely brittle: it can perform well sometimes (1 run out of 5), but usually does not (4 runs out of 5). This is regardless of the coreset points chosen for VCL. Note that coresets are chosen independently of training in VCL. Without coresets, VCL forgets many past tasks, with very low performance.

For VCL-MP, the coreset is chosen as the memorable past from an independent run of FROMP, with datapoints all on the task boundary. This selection of coreset is intuitively better than a random coreset selection. The results we show here are not specific to coreset selection. Any coreset selection (whether random or otherwise) all show the same inconsistency when VCL is trained with them. We provide visualisations of the brittleness of VCL in Figure 5.

G.2. Dataset variations

Figures 6 to 10 visualise the different dataset variations presented in Table 3. We pick the middle performing FROMP run (out of 5) and batch Adam run to show.

G.3. VCL and FROMP hyperparameter settings for toy datasets

FROMP. We optimised the number of epochs, Adam learning rate, and batch size. We optimised by running different hyperparameter settings for 5 runs on the toy dataset in Figure 4, and picking the settings with largest mean train accuracy. We found the best settings were: number of epochs=50, batch size=20, learning rate=0.01. The hyperparameters were then fixed across all toy data experimental runs, including across dataset variations (number of epochs was appropriately scaled

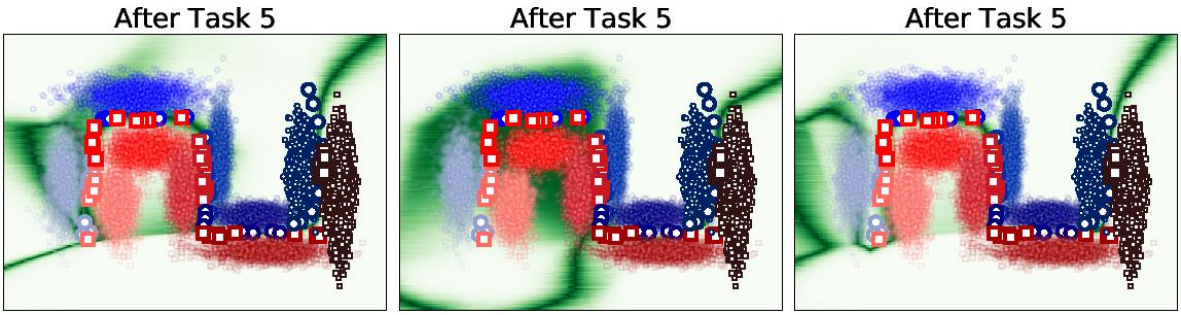


Figure 5. Three runs of VCL-MP on toy 2D data. These are the middle performing 3 runs out of 5 runs with different random seeds. VCL's inconsistent behaviour is clear.

Table 3. Train accuracy of FROMP and batch-trained Adam (upper bound on performance) on variations of a toy 2D binary classification dataset, with mean and standard deviations over 10 runs (3 runs for Adam). FROMP performs well across variations. VCL (with coresets) performs significantly worse even on the original dataset (92 ± 10%). See Appendix G.2 for further experiments and for visualisations.

Dataset variation	**FROMP**	**Batch Adam**
Original dataset	99.6 ± 0.2%	99.7 ± 0.0%
10x less data (400 per task)	99.9 ± 0.0%	99.7 ± 0.2%
10x more data (40000 per task)	96.9 ± 3.0%	99.7 ± 0.0%
Introduced 6th task	97.8 ± 3.3%	99.6 ± 0.1%
Increased std dev of each class distribution	96.0 ± 2.4%	96.9 ± 0.4%
2 tasks have overlapping data	90.1 ± 0.8%	91.1 ± 0.3%

by 10 if dataset size was scaled by 10).

VCL+coresets. We optimised the number of epochs, the number of coreset epochs (because VCL+coresets trains on non-coreset data first, then on coreset data just before test-time: see Nguyen et al. (2018)), learning rate (we use Adam to optimise the means and standard deviations of each parameter), batch size, and prior variance. We optimised by running various settings for 5 runs and picking the settings with largest mean train accuracy. We found the best settings were: number of epochs=200, number of coreset epochs=200, a standard normal prior (variance=1), batch size=40, learning rate=0.01. VCL is slow to run (an order of magnitude longer) compared to the other methods (FROMP and batch Adam).

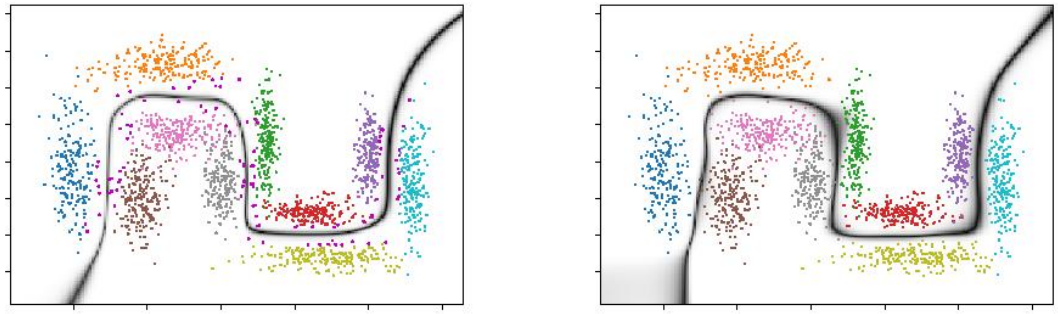


Figure 6. FROMP (middle performing of 5 runs) and batch Adam on a dataset 10x smaller (400 points per task).

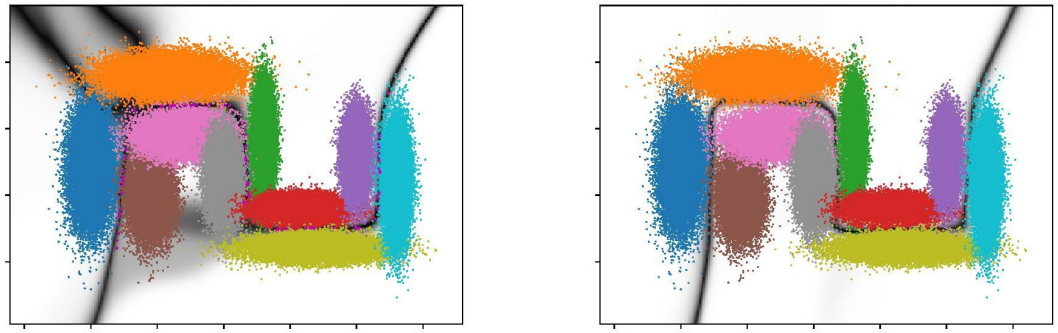


Figure 7. FROMP (middle performing of 5 runs), left, and batch Adam, right, on a dataset 10x larger (40,000 points per task).

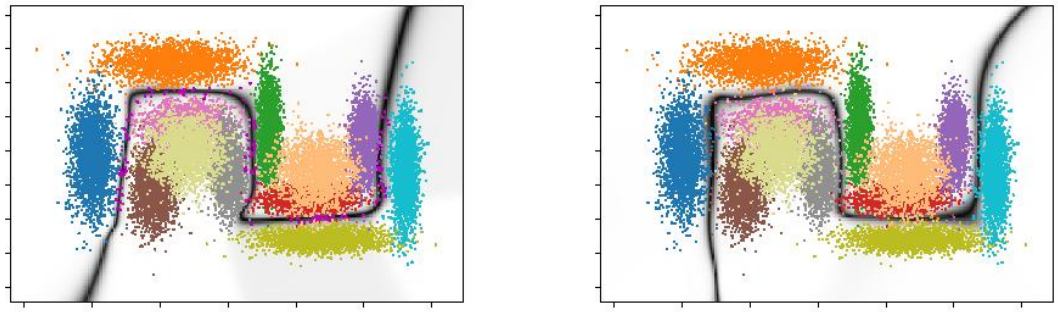


Figure 8. FROMP (middle performing of 5 runs), left, and batch Adam, right, on a dataset with a new, easy, 6th task.

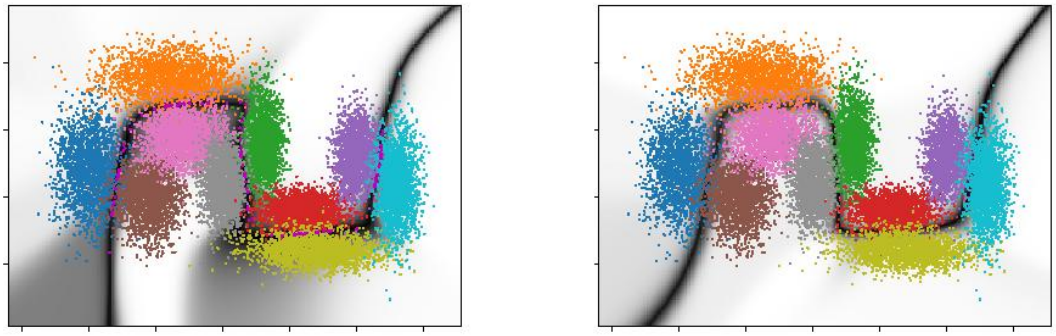


Figure 9. FROMP (middle performing of 5 runs), left, and batch Adam, right, on a dataset with increased standard deviations of each class' points, making classification tougher.

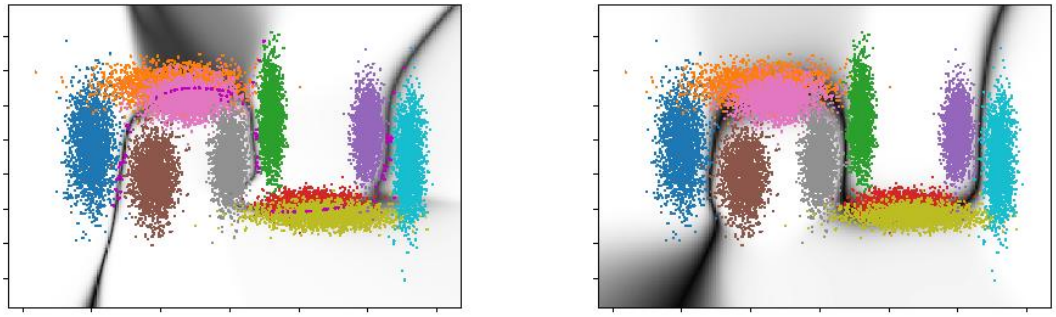


Figure 10. FROMP (middle performing of 5 runs), left, and batch Adam, right, on a dataset with 2 tasks having overlapping data, which is not separable.