CONTINUAL DEEP LEARNING BY FUNCTIONAL REGULARISATION OF MEMORABLE PAST

Anonymous authors
Paper under double-blind review

ABSTRACT

Continually learning new skills without forgetting old ones is an important quality for an intelligent system, yet most deep learning methods suffer from catastrophic forgetting of the past. Recent works have addressed this by regularising the network weights, but it is challenging to identify weights crucial to avoid forgetting. A better approach is to directly regularise the network outputs at past inputs, e.g., by using Gaussian processes (GPs), but this is usually computationally challenging. In this paper, we propose a scalable *functional-regularisation* approach where we regularise only over a few *memorable past* examples that are crucial to avoid forgetting. Our key idea is to use a GP formulation of deep networks, enabling us to both identify the memorable past and regularise over them. Our method achieves state-of-the-art performance on standard benchmarks and opens a new direction for life-long learning where regularisation methods are naturally combined with memory-based methods.

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 are not good at maintaining previously acquired skills, and can quickly forget them when learning new skills (Kirkpatrick et al., 2017). Such catastrophic forgetting presents a big challenge when deploying deep-learning methods for applications, such as robotics, where new tasks can appear during the training, and data from the previous tasks might be unavailable for retraining.

In recent years, many methods have been proposed to address catastrophic forgetting in deep learning. One of the most popular approaches is to regularise the network weights to keep them close to the weights obtained for the previous tasks (Kirkpatrick et al., 2017; Nguyen et al., 2018; Zenke et al., 2017). This is challenging due to the difficulty in identifying the weights that are relevant to past tasks. The exact values of the weights in fact do not matter directly, but rather the network output (Benjamin et al., 2018). Figuring out which weight affects the output is therefore usually difficult. Typically, the Fisher information matrix or covariance matrices over weights are used (Kirkpatrick et al., 2017; Nguyen et al., 2018), but they only partially address the issue.

A better approach is to directly regularise the network outputs, also referred to as *functional-regularisation*, that requires a memory of past examples (Benjamin et al., 2018; Lopez-Paz & Ranzato, 2017; Rebuffi et al., 2017). However, such methods still lack a mechanism to automatically weight more relevant past memory in the context of the new task, and also do not take uncertainty of the output into account. Methods based on Gaussian processes do this automatically (Titsias et al., 2019), but require optimisation over inducing points and specification of a good kernel, both of which are difficult tasks. In summary, existing methods fall short in building scalable functional-regularisation methods for continual learning.

In this paper, we propose a new functional-regularisation method where we regularise over a few *memorable past* examples (see Fig. 1). Our approach builds upon a recent method of Khan et al. (2019) that expresses Bayesian deep networks as Gaussian processes (GPs). We show that the GP formulation not only enables the identification of examples crucial to avoid forgetting, but also computes uncertainty over the network output to appropriately weight the past examples in the light of new ones. Motivated by the Gaussian process perspective, we propose a new loss function for

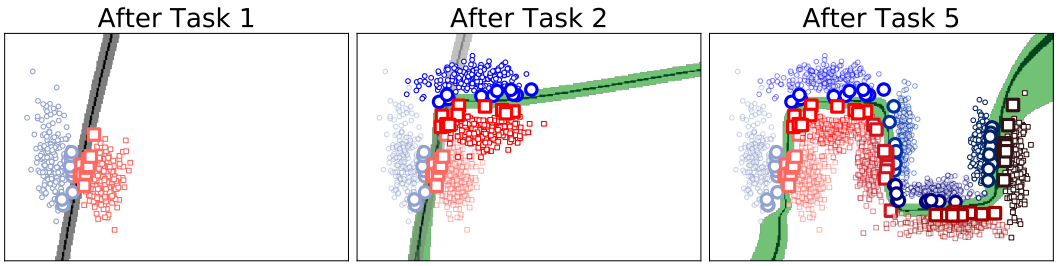


Figure 1: This figure illustrates our method. Leftmost figure shows the result of training on task 1. Examples corresponding to memorable-past, shown with big markers, are chosen using a GP formulation of the neural network. These points usually are the ones that support the decision boundary. Middle figure shows the result after task 2 where new network functions are regularised at memorable-past examples to give the same prediction as the previous ones. The resulting green decision boundary classifies both task 1 and 2 well. The rightmost figure shows the result along with memorable-past of each task where the performance over the past tasks is maintained.

function-regularisation with deep networks. A Laplace approximation of this objective enables a scalable training algorithm. Our work in this paper focuses on avoiding forgetting, but it opens a new direction for life-long learning methods where regularisation methods are naturally combined with memory-based methods.

Other related works. Broadly, existing work on continual learning can be split into three approaches: inference based, memory/rehearsal based, and model based. Inference based approaches have mostly focused on weight-regularisation, with some recent efforts on functional-regularisation. Our work falls in the latter category. Memory based approaches either maintain a memory of past data examples (Rebuffi et al., 2017) or train generative models on previous tasks to rehearse pseudo-inputs (Shin et al., 2017). An advantage of our method compared to previous ones is that building memory does not require solving an optimisation problem: the computation simply involves a forward-pass through the network followed by sorting (see Section 3.2).

Similarly to our work, there have also been some efforts in combining the different flavours of approaching continual learning, e.g., VCL plus coresets (Nguyen et al., 2018) and Gradient-Episodic Memory (Lopez-Paz & Ranzato, 2017; Chaudhry et al., 2018). Benjamin et al. (2018) have proposed a similar combination for functional regularisation. In these approaches, two separate methods are usually used for regularisation and memory-building. In contrast, in our approach, both of these are done within the same GP framework by using the method of Khan et al. (2019).

Finally, model based approaches change the model architecture during training (Rusu et al., 2016) and this can be combined with other approaches (Schwarz et al., 2018). It is possible to use similar features in our GP based framework, which is an interesting future direction to be pursued.

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 vector of K classes, and inputs $\mathbf{x}_i$, a vector of length D, and our goal is to minimise a loss which takes the following form: $\bar{\ell}(\mathbf{w}) + \delta R(\mathbf{w})$, 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 $\mathbf{w}$. $\ell(\mathbf{y}, \hat{\mathbf{y}})$ denotes a differentiable loss function between an output $\mathbf{y}$ and its prediction $\hat{\mathbf{y}}$, $R(\mathbf{w})$ 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 $\bar{\ell}$, which usually requires access to all of the data examples for all classes (Bottou, 2010). It is this unbiased, minibatch setting where deep-learning excels and achieves state-of-the-art performance on many benchmark datasets.

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 leads to catastrophic

forgetting of past information (Kirkpatrick et al., 2017). Our goal in this paper is to design methods that can avoid 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 previous works (Kirkpatrick et al., 2017; Nguyen et al., 2018; Zenke et al., 2017), and our goal is to avoid forgetting of old tasks in this setting.

Recent methods have proposed weight-regularisation as a way to combat catastrophic forgetting. The main idea is to keep the new network weights close to the old ones, e.g., when training a task t while given network weights $\mathbf{w}_{t-1}$ trained on task $t-1$, we can minimise the following loss: $\bar{\ell}_t(\mathbf{w}) + \delta(\mathbf{w}-\mathbf{w}_{t-1})^\top \mathbf{F}_t(\mathbf{w}-\mathbf{w}_{t-1})$, where $\bar{\ell}_t(\mathbf{w})$ is the loss defined over all data examples from task t and $\mathbf{F}_t$ is a preconditioning matrix that favors the weights relevant to the past tasks more than the rest. The Elastic-Weight Consolidation (EWC) method (Kirkpatrick et al., 2017), for example, uses the Fisher information matrix as the pre-conditioner, while Ritter et al. (2018) use the Hessian of the loss, and VCL (Nguyen et al., 2018) essentially employs the precision matrix of the variational approximation to do the same. 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). Since only the network outputs matter, an alternative approach is to directly regularise these. Benjamin et al. (2018) propose to use an L_2 regulariser over the function values on data examples from past tasks:

$$\min_w \bar{\ell}_t(\mathbf{w}) + \delta \sum_{s=1}^{t-1} \sum_{i \in \mathcal{M}_s} \|f_w(\mathbf{x}_i) - f_{w_{t-1}}(\mathbf{x}_i)\|_2^2, \tag{1}$$

where $\mathcal{M}_s$ is the set of small examples for task s stored in the *working memory* (Lopez-Paz & Ranzato, 2017; Rebuffi et al., 2017). One major issue with this approach is that the L_2 regulariser weights all the data points equally. In reality, some examples in the memory are more important than others to learn a given task. In addition, the uncertainty of the prediction is also ignored. As we will show, working with distributions over functions allows address these two issues.

Gaussian processes (GPs), for example, enable automatic reweighting of old tasks in light of a new one, which happens by using the posterior covariance. Unfortunately, both scalability of GPs as well as the necessity to save many instances of the past makes them impractical. A recent approach by Titsias et al. (2019) attempts to address these issue by employing sparse GP methods with inducing points and using a neural network feature map. However, the optimisation objective is only tractable when assuming independence across tasks. This approach heavily depends on a proper choice of inducing points and is limited by the choice of feature map, i.e. we cannot expect to achieve the performance of deep learning methods on all tasks.

Ideally, we would want to perform functional regularisation with a deep learning optimiser while borrowing ideas from the GP methods. Our work in this paper takes a step in this direction. Our proposal is to use a GP formulation of neural networks to perform functional regularisation that still allows training with standard deep-learning methods. We also show that the GP view helps in selecting an informative set of memory points.

3 Functional-Regularisation of Memorable Past (FROMP)

We will now describe our proposed method. First, we propose a method to use GP formulation of Khan et al. (2019) to compute GP-like posteriors over functions. Then, we propose a method to identify a set of memorable past examples for a task. Finally, we describe our functional-regularisation and discuss approximations used to build a scalable method.

3.1 From Deep Networks to Gaussian Process Posteriors over Functions

Khan et al. (2019) propose an approach to convert deep networks into Gaussian processes. Their main result (see Theorem 1 in their paper) states that, at a local minimiser $\mathbf{w}_*$ of $\bar{\ell}(\mathbf{w}) + \frac{\delta}{2}\mathbf{w}^\top\mathbf{w}$, a Laplace approximation of the posterior over $\mathbf{w}$ is equivalent to the posterior distribution of a linear

model. Specifically, they use the following scalable variant of the Laplace approximation:

$$p(\mathbf{w}|\mathcal{D}) \approx q(\mathbf{w}) := \mathcal{N}(\mathbf{w}|\boldsymbol{\mu}, \boldsymbol{\Sigma}), \text{ where } \boldsymbol{\mu} = \mathbf{w}_* \text{ and } \boldsymbol{\Sigma}^{-1} = \sum_{i=1}^{N} \mathbf{J}(\mathbf{x}_i)^\top \boldsymbol{\Lambda}_i \mathbf{J}(\mathbf{x}_i) + \delta \mathbf{I}_P, \tag{2}$$

where $\mathbf{J}(\mathbf{x}_i) := \nabla_w \mathbf{f}_w(\mathbf{x}_i)^\top$ is a $K \times P$ Jacobian matrix (P being the number of weights), and $\boldsymbol{\Lambda}_i := \nabla^2_{\hat{\mathbf{y}}\hat{\mathbf{y}}} \ell(\mathbf{y}_i, \hat{\mathbf{y}})$ is the $K \times K$ Hessian of the loss evaluated at $\hat{\mathbf{y}} = \mathbf{f}_w(\mathbf{x}_i)$, all evaluated at $\mathbf{w} = \mathbf{w}_*$. They show that $q(\mathbf{w})$ is equal to the posterior of the following linear model:

$$\tilde{\mathbf{y}}_i = \mathbf{J}(\mathbf{x}_i)\mathbf{w} + \boldsymbol{\epsilon}_i, \quad \text{with } \boldsymbol{\epsilon}_i \sim \mathcal{N}(0, \boldsymbol{\Lambda}_i^{-1}) \text{ and } \mathbf{w} \sim \mathcal{N}(0, \delta^{-1}\mathbf{I}_P), \tag{3}$$

where the observations are defined as $\tilde{\mathbf{y}}_i := \mathbf{J}(\mathbf{x}_i)\mathbf{w}_* - \boldsymbol{\Lambda}_i^{-1}\mathbf{r}_i$ with $\mathbf{r}_i := \nabla_{\hat{\mathbf{y}}} \ell(\mathbf{y}_i, \hat{\mathbf{y}})$ being the residuals. They also show that the predictive distribution of this linear model is equivalent to that of a GP regression model defined with a $K \times K$ neural tangent kernel (NTK) (Jacot et al., 2018):

$$\tilde{\mathbf{y}}_i = \mathbf{f}_{\text{GP}}(\mathbf{x}_i) + \boldsymbol{\epsilon}_i, \qquad \text{with } \mathbf{f}_{\text{GP}}(\mathbf{x}) \sim \mathcal{GP}\left(0, \delta^{-1}\mathbf{J}(\mathbf{x})\mathbf{J}(\mathbf{x}')^\top\right). \tag{4}$$

The above result does not directly find the posterior over the network outputs or a related function space, since $f_{GP}(\mathbf{x})$ is related to all layers. We derive a GP posterior approximation over the function output, by using a method similar to the one used in generalised linear models (Nelder & Wedderburn, 1972). For this method, we need to assume that the loss corresponds to a log probability distribution, i.e., $\ell(\mathbf{y}, \mathbf{f}_w(\mathbf{x})) := -\log p(\mathbf{y}|\mathbf{g}(\mathbf{f}_w(\mathbf{x})))$ where $\mathbf{g}(\cdot)$ is a *link function*. Below we demonstrate the method for a binary classification problem. Extensions to multi-class classification and other likelihoods can be obtained in a similar fashion (see Appendix A).

To model the output distribution according to a GP, we rewrite Eqs. (4) or (3) in terms of the true label $y_i \in \{0, 1\}$. Then, we can derive the mean and variance for the function $f(\mathbf{x}_i)$ which will yield an approximation to the network output. For a logistic likelihood, the link function is $\mathrm{p}_i := \sigma(f_{w_*}(\mathbf{x}_i))$ where σ is the sigmoid function and residuals are $\mathrm{r}_i = \mathrm{p}_i - \mathrm{y}_i$ and $\Lambda_i := \mathrm{p}_i(1 - \mathrm{p}_i)$. Because residuals directly contain y_i, we can rewrite Eq. (3) as a linear model,

$$\mathrm{y}_i = \mathrm{p}_i + \mathbf{J}(\mathbf{x}_i)(\mathbf{w} - \mathbf{w}_*) + \tau_i, \quad \text{with } \tau_i \sim \mathcal{N}(0, \Lambda_i) \text{ and } \mathbf{w} \sim \mathcal{N}(0, \delta^{-1}\mathbf{I}_P). \tag{5}$$

The Laplace approximation (2) is the posterior of this model after observing data $\mathcal{D}$. We can equivalently write the posterior predictive in a GP form (Rasmussen, 2003) with the following mean and covariance function:

$$m(\mathbf{x}_i) := \mathbb{E}_{q(w)}\left[f(\mathbf{x}_i)\right] = \mathrm{p}_i + \Lambda_i J(\mathbf{x}_i)(\boldsymbol{\mu} - \mathbf{w}_*) = \mathrm{p}_i, \tag{6}$$

$$k(\mathbf{x}_i, \mathbf{x}_j) := \mathbb{E}_{q(w)}\left[\left(f(\mathbf{x}_i) - m(\mathbf{x}_i)\right)\left(f(\mathbf{x}_j) - m(\mathbf{x}_j)\right)\right] \tag{7}$$

$$= \Lambda_i \mathbf{J}(\mathbf{x}_i)\mathbb{E}_{q(w)}\left[(\mathbf{w} - \mathbf{w}_*)(\mathbf{w} - \mathbf{w}_*)^\top\right]\mathbf{J}(\mathbf{x}_j)^\top \Lambda_j = \Lambda_i \mathbf{J}(\mathbf{x}_i)\boldsymbol{\Sigma}\mathbf{J}(\mathbf{x}_j)^\top \Lambda_j. \tag{8}$$

A GP defined with the above mean and covariance function can be viewed as an approximation to the posterior process over network outputs. Throughout the paper, we will denote the process obtained at a weight $\mathbf{w}_*$ by a distribution $q_{w_*}(\mathbf{f})$ where $\mathbf{f}$ is a vector of $f(\mathbf{x})$ evaluated at many inputs. We will use this GP posterior predictive for functional regularisation.

3.2 Memorable past

In the previous section, we derived a GP posterior approximation over the network outputs. Now, we propose a method to obtain a small set of examples that are crucial to avoid forgetting. We first note that the posterior GP mean in (4) corresponds to a kernel Ridge regression that requires the computation of $(\delta^{-1}\mathbf{K} + \boldsymbol{\Lambda}^{-1})^{-1}\tilde{\mathbf{y}}$ where $\boldsymbol{\Lambda}$ is a block-diagonal matrix containing all $\boldsymbol{\Lambda}_i$ and $\mathbf{K}$ is the NTK. The predictions therefore strongly depend on the eigenvalues of the preconditioning matrix. Selection of important data examples therefore boils down to selecting important columns of this matrix, such that the predictions remain unchanged. The theory of *leverage score sampling* (Alaoui & Mahoney, 2015; Bach, 2013) suggests picking the points proportional to the leverage score defined to be the diagonal of the following matrix: $\mathbf{K}(\mathbf{K} + \delta\boldsymbol{\Lambda}^{-1})^{-1}$. Typically, this matrix is difficult to compute and methods are employed to approximately obtain the leverage score (Alaoui & Mahoney, 2015).

Algorithm 1: Functional Regularisation of Memorable Past (FROMP)

Initialise $\mathcal{M} = \varnothing$
for each task t **do**
  for previous task $s = 1, 2, ..., t-1$ **do**
    Compute $\mathbf{m}_{s,\mathbf{w}_{t-1}}, \mathbf{K}^{-1}_{\mathbf{m}_{t-1},s}$, Eqs 6 and 8
  repeat
    $\mathbf{g}$ = stochastic $\nabla_{w_t} \ell(y_i, f_w(\mathbf{x}_i))$
    $\mathbf{g}$ += fr_grad$\Big(\mathbf{w}_t, \{\mathbf{m}_{s,\mathbf{w}_{t-1}}, \mathbf{K}^{-1}_{\mathbf{m}_{t-1},s}\}_{s=1}^{t-1}\Big)$
    $\mathbf{w}_t \leftarrow$ Adam update using $\mathbf{g}$
  until convergence
  Update $\mathbf{\Sigma}$, Eq 2
  $\mathcal{M}_t \leftarrow$ memorable_past $(\mathcal{D}_t, \mathbf{w}_t)$
  $\mathcal{M} = \mathcal{M} \cup \mathcal{M}_t$

def fr_grad$\Big(\mathbf{w}, \{\mathbf{m}_{s,\mathbf{w}_{t-1}}, \mathbf{K}^{-1}_{\mathbf{m}_{t-1},s}\}_{s=1}^{t-1}\Big)$
  $\mathbf{g} = 0$
  for task $s = 1, 2, ..., t-1$ **do**
    $\mathbf{g}$ +=
    $\nabla_{\mathbf{w}}(\mathbf{m}_{s,\mathbf{w}} - \mathbf{m}_{s,w_{t-1}})^\top \mathbf{K}^{-1}_{w_{t-1},s}(\mathbf{m}_{s,\mathbf{w}} - \mathbf{m}_{s,w_{t-1}})$
  return $\mathbf{g}$

def memorable_past $(\mathcal{D}_t, \mathbf{w}_t)$
  Calculate $\mathbf{\Lambda}$ (see Sec 3.2)
  $\mathcal{M}_t \leftarrow$ datapoints i from $\mathcal{D}_t$ with highest $\mathbf{\Lambda}_i$
  return $\mathcal{M}_t$

Figure 2: A pseudo-code for our FROMP algorithm. The additional computations above Adam involve computing $\{\mathbf{m}_{s,\mathbf{w}_{t-1}}, \mathbf{K}^{-1}_{\mathbf{m}_{t-1},s}\}_{s=1}^{t-1}$ just before training on a new task, and then adding their gradient with fr_grad(). We also choose memorable past examples after training on a task via $\mathbf{\Lambda}$, which only requires a single forward-pass.

For deep-learning applications too, exact computation of leverage score is very difficult. We instead propose a simple solution. Since the eigenvalues of the previous matrix heavily depend on $\delta\mathbf{\Lambda}_i^{-1}$, we can pick the data examples by simply sorting $\mathbf{\Lambda}_i$. For the inverse of $K + \delta\mathbf{\Lambda}^{-1}$ to have high eigenvalues, we should favor examples with smaller values of $\mathbf{\Lambda}_i^{-1}$, i.e., with the highest variance or the lowest precision. Therefore, we simply sort the $\mathbf{\Lambda}_i$ and pick the top M examples as the most relevant ones. This simple solution is very effective for our particular problem because $\mathbf{\Lambda}_i$ are noise variances for the data examples, obtained by using an already trained network. These are second derivatives of the loss for data examples, and so also reflect the sensitivity of the decision boundaries if a particular data point is perturbed. Therefore, they tend to reflect the relevance of data examples. An example is shown in Figure 1 where we clearly see that our solution picks the examples lying close to decision boundary. Computation of $\mathbf{\Lambda}_i$ requires us to run the forward pass to get the $\ell(\mathbf{y}_i, \hat{\mathbf{y}}_i)$ and then compute its second derivative with respect to $\hat{\mathbf{y}}_i$, both of which are cheap operations. Throughout the paper, examples chosen by this method are referred to as the *memorable past* examples, and denoted $\mathcal{M}_t$ for task t.

3.3 Functional-Regularisation

So far, we described the construction of the GP posterior predictive over the function space, as well as the construction of a set of memorable-past examples. We are now ready to describe our objective function where we employ these two to perform functional regularisation.

Suppose that we are given network weights $\mathbf{w}_{t-1}$ that are obtained by training over data examples from task $t-1$. Our goal then is to train a network with weights $\mathbf{w}$ such that its performance on memorable past $\mathcal{M}_{1:t-1}$ is unchanged. We denote the vector of function outputs over these examples by $\mathbf{a}_{1:t-1}$.A straightforward idea is to directly optimise the weights $\mathbf{w}$ such that the predictions using $q_w(\mathbf{f}_t)$ are good on current tasks while the predictive distribution $q_w(\mathbf{a}_{1:t-1})$ is close to $q_{w_{t-1}}(\mathbf{a}_{1:t-1})$. Since the number of tasks can be very large, we choose to regularise each task separately, i.e., we will match $q_w(\mathbf{a}_s)$ with $q_{w_{t-1}}(\mathbf{a}_s)$ separately for all tasks $s < t$ in line with Titsias et al. (2019). These choices give us the following objective function with trade-off parameter τ:

$$\min_w \quad \tau \mathbb{E}_{q_w(f_t)}\Big[\sum_{i \in \mathcal{D}_t} \ell(y_i, f(\mathbf{x}_i))\Big] + \sum_{s=1}^{t-1} \mathbb{D}_{KL}[q_w(\mathbf{a}_s) \,\|\, q_{w_{t-1}}(\mathbf{a}_s)]. \tag{9}$$

A major issue with this objective is that computing gradients wrt. the posterior predictive q_w will be computationally heavy as it involves gradients of the Jacobian and $\mathbf{\Lambda}_i$, which involve higher order derivatives. During training on a task, we have no distribution on parameters (cf. Laplace approximation) and can treat it as fixed. Therefore, taking the gradient wrt. parameter $\mathbf{w}$ of above

Table 1: 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. See Appendix B.2 for visualisations.

Dataset variation	**FROMP**	**Batch Adam**
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

objective will only involve differentiating the mean function $\mathbf{m}(\mathbf{x}_i)$ formed by the neural network and we have equivalence to the following objective:

$$\min_{w} \quad \tau \sum_{i \in \mathcal{D}_t} \ell(y_i, f_w(\mathbf{x}_i)) + \frac{1}{2} \sum_{s=1}^{t-1} (\mathbf{m}_{s,w} - \mathbf{m}_{s,w_{t-1}})^\top \mathbf{K}_{w_{t-1},s}^{-1} (\mathbf{m}_{s,w} - \mathbf{m}_{s,w_{t-1}}), \tag{10}$$

where $\mathbf{m}_{s,w_{t-1}}$ and $\mathbf{K}_{w_{t-1},s}$ are the vector and matrices containing corresponding the mean and covariance function of the network weights $\mathbf{w}_{t-1}$ evaluated at the memorable past for task s. Eqs 6 and 8 are used to calculate $\mathbf{m}_{s,w_{t-1}}$ and $\mathbf{K}_{w_{t-1},s}$ respectively. Essentially, this gives us something similar to Eq. 1 but instead of the L_2 distance a kernel is used to improve weighting of examples. The final FROMP algorithm is summarised in Algorithm 1. Please see App. A.3 for details on scaling to the multi-class setting. Note that it is possible to be stochastic in sampling a subset of previous tasks' memory during every update, ensuring the algorithm does not get slower as more tasks are trained.

4 Experiments

We run the proposed method FROMP on toy datasets, permuted and split MNIST (LeCun et al., 1998; Goodfellow et al., 2013), and a split version of CIFAR-10 and CIFAR-100 (Krizhevsky et al., 2009). We optimise the objective in Eq. (10) using Adam (Kingma & Ba, 2015) with parameter $\beta_1 = 0.99$ and further use gradient clipping to speed up training. To identify the individual benefits of the kernel in loss (10) as well as the selection of data points, we compare the following four methods: FROMP uses kernel and the proposed example selection, FRORP instead uses randomly chosen examples. Further, we replace the kernel by an identity matrix leading to functional L_2 regularisation and call the corresponding methods with and without our example selection technique FROMP-L_2 and FRORP-L_2.

4.1 Toy dataset

In this section, we test FROMP on many variations of the dataset. We want to test its performance when exposed to different datasets of varying difficulty. We use a 2-hidden layer MLP (with 20 hidden units in each layer) for all experiments. Appendix B.3 details hyperparameter selection. In Appendix B.1 we show the brittleness and inconsistent behaviour of weight-space regularisation methods on a toy 2D binary classification dataset like that in Figure 1. In contrast, FROMP performs extremely well across many variations of the dataset, showing consistently good results (see Table 1 and Appendix B.2 for visualisations).

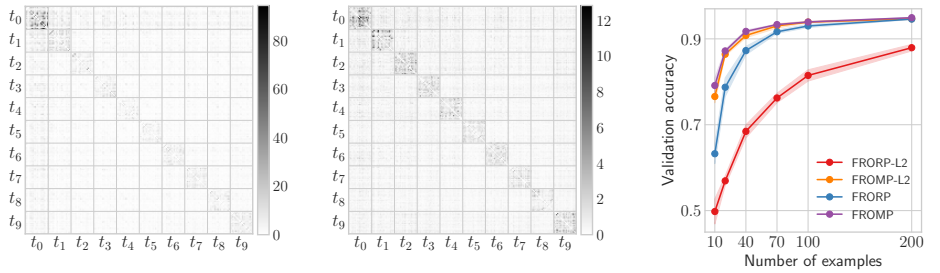
4.2 Permuted and Split MNIST

Permuted MNIST consists of a series of tasks where each task is a fixed permutation of pixels to the entire labelled MNIST dataset. Like in previous work (Nguyen et al., 2018; Kirkpatrick et al., 2017; Zenke et al., 2017; Titsias et al., 2019), we implement a fully connected single-head network with two hidden layers. Each hidden layer consists of 100 hidden units and ReLU activation functions. We set the learning rate to 0.001, batch size to 128, and learn each task for 10 epochs.

The Split MNIST experiment was introduced by Zenke et al. (2017) and consists of five binary classification tasks built from MNIST: 0/1, 2/3, 4/5, 6/7, and 8/9. We use a fully connected multi-head neural network with two hidden layers of each 256 hidden units and ReLU activation function.

Table 2: The average validation accuracy on Permuted-MNIST and Split-MNIST. "200p/t" denotes that 200 examples are selected for each task. We report mean and standard deviations over 5 runs, and use results from Nguyen et al. (2018) for baselines. FROMP is state-of-the-art with 200p/t. Additionally, as we reduce the number of points (see Figure 3), FROMP gracefully reduces accuracy, due to clever choice of memory past and the use of kernels in the functional regularisation.

Method	**Permuted MNIST**	**Split MNIST**
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 (200 p/t)	98.2% ± 0.4 (40 p/t)
FRCL-RND (Titsias et al., 2019)	94.2% ± 0.1 (200 p/t)	96.7% ± 1.0 (40 p/t)
FRCL-TR (Titsias et al., 2019)	94.3% ± 0.1 (200 p/t)	97.4% ± 0.6 (40 p/t)
FRORP-L_2	87.9% ± 0.7 (200 p/t)	98.5% ± 0.2 (40 p/t)
FROMP-L_2	94.6% ± 0.1 (200 p/t)	98.7% ± 0.1 (40 p/t)
FRORP	94.6% ± 0.1 (200 p/t)	**99.0**% ± 0.1 (40 p/t)
FROMP	**94.9**% ± 0.1 (200 p/t)	**99.0**% ± 0.1 (40 p/t)



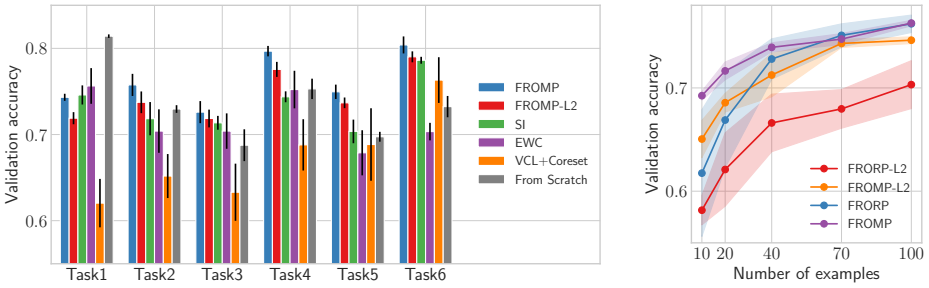
(a) random example kernel (b) *memorable past* kernel (c) accuracy vs. memory size

Figure 3: Permuted MNIST: added kernels across classes (with subtracted diagonal for visualisation purposes), and performance as a function of memory size. Memorable past examples lead to a more uniform kernel structure that prevents weighting previously overfit examples too highly, e.g. task one in the random selection exhibits strong correlation and low variance. As we reduce the number of examples in memory, FROMP gracefully reduces validation accuracy.

Following the settings of previous work, we select 40 inducing points per task. The learning rate is set to 0.0001, batch size to 128, and we learn each task for 15 epochs.

We report the final average accuracy for both benchmarks across all the tasks in Table 2 after tuning the hyperparameters of all algorithms. In particular, the proposed method achieves better performance than the weight-space methods EWC and VCL, as well as compared to the function-space method FRCL that is based on a GP formulation. Further, the benchmarks show superior performance of both the approach to select important examples (Sec. 3.2) and the functional regularisation using the kernel (Sec. 3.3). Memorable examples improve performance of the naive and efficient FRORP-L_2 method by more than 6% on permuted MNIST and by 0.2% on the split MNIST. Standard deviation is also reduced in both cases. FROMP does not profit much from memorable examples compared to FRORP, probably because the performance is already close to the maximum achievable. Furthermore, Fig. 3c shows that our selection method greatly reduces the number of memorable points required: the L_2 algorithm with random points requires nearly 100 points to match the performance when using 20 carefully selected points, and 200 points to match performance with 40, respectively. When combined with kernel-based functional regularisation, we obtain the best-performing method, particularly when memory size is small.

Figs. 3a and 3b show the summed kernels across all the classes in permuted MNIST for a random and memorable set of points. For visualisation purposes, the diagonal is suppressed. Note that random points lead to less uniform weighting in the kernel, making it even more important in functional



(a) final performance on individual tasks

(b) accuracy vs. memory size

Figure 4: Split CIFAR: performance for each task, and performance variation as function of memory size. We run all the proposed methods 5 times and report the mean and standard derivation. The left figure reports results for 200 memory examples per task. The final average validation accuracy of FROMP is 76.2% ± 0.2, FROMP-L_2 is 74.6% ± 0.3, SI is 73.5%, EWC is 71.6% ± 1.5, VCL + random coreset is 67.4% ± 2.4. FROMP outperforms all other methods. Additionally, as we reduce the memory size, FROMP still performs well, even with only 10 examples per task.

regularisation, leading to better performance (FRORP vs FROMP). All memorable points are important and the kernel is more uniform. In Fig. 3a, the kernel's weighting of task 1 is very different from other tasks, leading to different magnitudes in the functional regularisation among tasks and therefore to eventual forgetting. The kernel further tells us that the tasks are correlated, as expected.

4.3 Split CIFAR

We now test the proposed method on a more complex problem. Split CIFAR consists of 6 tasks. The first task is the full CIFAR-10 dataset, followed by 5 tasks, each corresponding to 10 consecutive classes from CIFAR-100. We follow the SI paper (Zenke et al., 2017) for our model architecture, using a multi-head CNN with 4 convolutional layers, followed by 2 dense layers with dropout. We use learning rate 0.0001 and batch size 256. All tasks are learned for 80 epochs, and hyperparameters tuned as before. In addition to continual learning baselines, we show the performance of networks trained from scratch on each task. These cannot profit from forward/backward transfer.

The experimental results in Figure 4a show that FROMP outperforms other methods by a notable margin. The weight-space methods employed as baselines either cannot learn later tasks to a high accuracy (EWC, SI), or forget previous tasks (VCL). Interestingly, over all tasks, FROMP also outperforms 'from scratch' training. Although we only focussed on preventing catastrophic forgetting, we find evidence of forward/backward transfer, a key requirement in continual learning.

In contrast to the rather simple MNIST benchmarks, both the benefit of selecting memorable points as well as using the kernel are clearly visible in Fig. 4b. If we only memorise few examples, the performance gap due to using the kernel is around 4%. The selection of memorable points according to our metric leads to an increase in performance of around 7%. Applying both kernel and memorable point selection increases the performance by up to 11%. Additionally, standard deviation is reduced when using memorable points or the kernel. It is clear that both parts of the proposed algorithm are vital in achieving state-of-the-art performance on this benchmark.

5 Discussion

We propose FROMP, a scalable function-regularisation approach for continual learning. FROMP uses a GP formulation of neural networks to select memorable past examples, regularising them using a kernel, and achieving state-of-the-art performance across benchmarks. This work enables a new way of combining regularisation methods and memory-based methods in continual learning. Future research could investigate other ways of selecting a memorable past (e.g. fixed memory size), more efficient ways of calculating kernel matrices, and consider the case where data does not arrive in tasks.

REFERENCES

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

Francis Bach. Sharp analysis of low-rank kernel matrix approximations. In *Conference on Learning Theory*, pp. 185–209, 2013.

Ari S Benjamin, David Rolnick, and Konrad Kording. Measuring and regularizing networks in function space. *arXiv preprint arXiv:1805.08289*, 2018.

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

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

Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient lifelong learning with a-gem. *ArXiv*, abs/1812.00420, 2018.

Sebastian Farquhar and Yarin Gal. A unifying bayesian view of continual learning. *ArXiv*, abs/1902.06494, 2019.

Ian J Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio. An empirical investigation of catastrophic forgetting in gradient-based neural networks. *arXiv preprint arXiv:1312.6211*, 2013.

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

Mohammad Emtiyaz Khan, Alexander Immer, Ehsan Abedi, and Maciej Korzepa. Approximate inference turns deep networks into gaussian processes. *NeurIPS*, 2019.

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

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

Yann LeCun, Corinna Cortes, and Christopher JC. Burges. The mnist database of handwritten digits. 1998.

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

John Ashworth Nelder and Robert WM Wedderburn. Generalized linear models. *Journal of the Royal Statistical Society: Series A (General)*, 135(3):370–384, 1972.

Cuong V Nguyen, Yingzhen Li, Thang D Bui, and Richard E Turner. Variational continual learning. *ICLR*, 2018.

Carl Edward Rasmussen. Gaussian processes in machine learning. In *Summer School on Machine Learning*, pp. 63–71. Springer, 2003.

Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert. icarl: Incremental classifier and representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 2001–2010, 2017.

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

Andrei A. Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *ArXiv*, abs/1606.04671, 2016.

Jonathan Schwarz, Jelena Luketina, Wojciech Marian Czarnecki, Agnieszka Grabska-Barwiska, Yee Whye Teh, Razvan Pascanu, and Raia Hadsell. Progress & compress: A scalable framework for continual learning. *ArXiv*, abs/1805.06370, 2018.

Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. In *NIPS*, 2017.

Alexander J Smola, Vishy Vishwanathan, and Eleazar Eskin. Laplace propagation. In *NIPS*, pp. 441–448, 2003.

Siddharth Swaroop, Cuong V Nguyen, Thang D Bui, and Richard E Turner. Improving and understanding variational continual learning. *arXiv preprint arXiv:1905.02099*, 2019.

Michalis K Titsias, Jonathan Schwarz, Alexander G de G Matthews, Razvan Pascanu, and Yee Whye Teh. Functional regularisation for continual learning using gaussian processes. *arXiv preprint arXiv:1901.11356*, 2019.

Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *ICML*, pp. 3987–3995. JMLR. org, 2017.

A CONVERTING THE GP OF KHAN ET AL. (2019) INTO OBSERVATION SPACE

A.1 NOTATION AND PROBLEM SETUP

We denote by $\mathcal{D} := \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^N$ a dataset of N examples with $\mathbf{x}_i \in \mathbb{R}^D$ and output $\mathbf{y}_i \in \mathbb{R}^K$. The complex parametric function, e.g. a neural network, is denoted by $\mathbf{f}(\mathbf{x}; \mathbf{w}) : \mathbb{R}^D \times \mathbb{R}^P \to \mathbb{R}^K$ with $P-$dimensional parameter vector $\mathbf{w}$. Finally, we have a twice differentiable loss function $\ell(\mathbf{y}, \mathbf{f}(\mathbf{x}; \mathbf{w}))$, e.g. squared loss or cross-entropy. We write $\tilde{\mathbf{y}}$ in place of $\mathbf{f}(\mathbf{x}; \mathbf{w})$.

To optimize above parametric model, gradient-based approaches are used. The relevant quantities

$$\nabla_{\mathbf{w}} \ell(\mathbf{y}_i, \tilde{\mathbf{y}}_i) = \boldsymbol{J}(\mathbf{x}_i; \mathbf{w})^\top \mathbf{r}_i, \tag{11}$$

$$\nabla^2_{\mathbf{w}} \ell(\mathbf{y}_i, \tilde{\mathbf{y}}_i) = \boldsymbol{J}(\mathbf{x}_i; \mathbf{w})^\top \boldsymbol{\Lambda}_i \boldsymbol{J}(\mathbf{x}_i; \mathbf{w}) + \left[\nabla^2_{\mathbf{w}} \mathbf{f}(\mathbf{x}_i; \mathbf{w})\right] \mathbf{r}_i \tag{12}$$

denote the gradient and Hessian of the loss wrt. the parameter. In particular, we have written the Jacobian $\boldsymbol{J}(\mathbf{x}_i; \mathbf{w}) := \nabla_{\mathbf{w}} \mathbf{f}(\mathbf{x}_i; \mathbf{w})$ and Hessian $\nabla^2_{\mathbf{w}} \mathbf{f}(\mathbf{x}_i; \mathbf{w})$ of the model. Further, $\mathbf{r}_i := \nabla_{\tilde{\mathbf{y}}} \ell(\mathbf{y}_i, \tilde{\mathbf{y}}_i)$ and $\boldsymbol{\Lambda}_i := \nabla^2_{\tilde{\mathbf{y}}} \ell(\mathbf{y}_i, \tilde{\mathbf{y}}_i)$ are gradient and Hessian of the loss.

A.2 CONVERSION TO OBSERVATION SPACE

The difference between Thm. 1 and 2 in (Khan et al., 2019) is that in the latter a sampled parameter vector $\mathbf{w}_s$ is used and in the first the mean $\mathbf{w}_*$. We will simply write $\mathbf{w}_*$ for either a sample or the mean and focus on the single-sample case. It is easy to extend to multiple samples. Both Theorems give rise to the following linear model with observations $\hat{\mathbf{y}}_i = \boldsymbol{J}(\mathbf{x}_i; \mathbf{w}_*)\mathbf{w}_* - \boldsymbol{\Lambda}_i^{-1} \mathbf{r}_i$:

$$\hat{\mathbf{y}}_i = \boldsymbol{J}(\mathbf{x}_i; \mathbf{w}_*)\mathbf{w} + \boldsymbol{\epsilon}_i \quad \text{with } \mathbf{w} \sim \mathcal{N}(0, \delta^{-1}\boldsymbol{I}) \text{ and } \boldsymbol{\epsilon}_i \sim \mathcal{N}(0, \boldsymbol{\Lambda}_i^{-1}). \tag{13}$$

Least-squares regression: recall $\mathbf{r}_i = \sigma^{-2}(\mathbf{f}(\mathbf{x}_i; \mathbf{w}_*) - \mathbf{y}_i)$ and $\boldsymbol{\Lambda}_i = \sigma^{-2}\boldsymbol{I}_K$. We can plug in and re-arrange this to

$$\mathbf{y}_i = \mathbf{f}(\mathbf{x}_i; \mathbf{w}_*) + \boldsymbol{J}(\mathbf{x}_i; \mathbf{w}_*)(\mathbf{w} - \mathbf{w}_*) + \boldsymbol{\epsilon}_i, \tag{14}$$

which is a linear model with the original models noise variance σ^2 and prior on $\mathbf{w}$. The posterior distribution of this linear model after observation the original data set $\mathcal{D}$ is equivalent to the one in Thms. 1 and 2 as we only did simple transformations. However, it is a model of the form $\prod_{i=1}^N p(\mathbf{y}_i|\mathbf{x}_i, \mathbf{w})p(\mathbf{w})$ in the observation space.

Logistic regression: we write $\mathrm{p}_i := \sigma(f(\mathbf{x}_i; \mathbf{w}_*))$. Recall that $\mathrm{r}_i = \mathrm{p}_i - \mathrm{y}_i$ and $\Lambda_i = \mathrm{p}_i(1 - \mathrm{p}_i)$. Writing $\varepsilon_i \sim \mathcal{N}(0, 1)$ for standard noise, the Theorems provide the linear model

$$\boldsymbol{J}(\mathbf{x}_i; \mathbf{w}_*)\mathbf{w}_* - \frac{\mathrm{p}_i - \mathrm{y}_i}{\Lambda_i} = \boldsymbol{J}(\mathbf{x}_i; \mathbf{w}_*)\mathbf{w} + \Lambda_i^{-\frac{1}{2}} \varepsilon_i, \tag{15}$$

which can be re-arranged and simplified with $\tau_i \sim \mathcal{N}(0, \Lambda_i)$:

$$\mathrm{y}_i = \mathrm{p}_i + \Lambda_i \boldsymbol{J}(\mathbf{x}_i; \mathbf{w}_*)(\mathbf{w} - \mathbf{w}_*) + \tau_i. \tag{16}$$

Softmax regression: we write $\mathbf{p}_i := \mathcal{S}(\mathbf{f}(\mathbf{x}_i; \mathbf{w}_*))$ and have targets $\mathbf{y}_i$ that are standard basis vectors, i.e. have a single 1 at some position and otherwise 0s. Then we have $\mathbf{r}_i = \mathbf{p}_i - \mathbf{y}_i$ and $\boldsymbol{\Lambda}_i = \mathrm{diag}(\mathbf{p}_i) - \mathbf{p}_i\mathbf{p}_i^\top$. This covariance is in fact rank $K - 1$ so it cannot be inverted posing a problem for our original Theorems. However, we can give another linear model that has the posterior that is equal to the approximate posterior of our original model at convergence. With $\boldsymbol{\tau}_i \sim \mathcal{N}(0, \boldsymbol{\Lambda}_i)$:

$$\mathbf{y}_i = \mathbf{p}_i + \boldsymbol{\Lambda}_i \boldsymbol{J}(\mathbf{x}_i; \mathbf{w}_*)(\mathbf{w} - \mathbf{w}_*) + \boldsymbol{\tau}_i. \tag{17}$$

Conversion to GP: All the above models work directly in the data domain and preserve the condition that their posterior distribution co-incides with the approximate posterior due to Laplace/VI combined with GGN. The regression problems can be written in parameter or function-space. To convert above formulations into function space, one has to take expectation and covariance of $\mathbf{f}$. The linear problem in Eq. (20), e.g., can be written in function space as

$$\mathrm{y}_i = \mathrm{f}(\mathbf{x}_i) + \tau_i \text{ with } \mathrm{f}(\mathbf{x}) \sim \mathcal{GP}(\mathrm{p} - \Lambda \boldsymbol{J}(\mathbf{x}; \mathbf{w}_*)\mathbf{w}_*, \delta^{-1}\Lambda \boldsymbol{J}(\mathbf{x}; \mathbf{w}_*)\boldsymbol{J}(\mathbf{x}'; \mathbf{w}_*)^\top \Lambda'). \tag{18}$$

Writing the approximate posterior or exact posterior in for this model gives us distribution $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$. Consequently, we can specify the posterior GP as

$$\mathrm{f}(\mathbf{x}) \sim \mathcal{GP}(\mathrm{p} + \Lambda \boldsymbol{J}(\mathbf{x}; \mathbf{w}_*)(\boldsymbol{\mu} - \mathbf{w}_*), \Lambda \boldsymbol{J}(\mathbf{x}; \mathbf{w}_*)\boldsymbol{\Sigma}\boldsymbol{J}(\mathbf{x}'; \mathbf{w}_*)^\top \Lambda'), \tag{19}$$

where $\boldsymbol{\mu} - \mathbf{w}_*$ cancels out for the Laplace approximation, which we use in this work.

A.3 Reducing Complexity in the Multiclass Setting

For the softmax regression with K classes, we will build an individual GP for each class. We utilize $\mathrm{y}_i^{(k)}$ to denote the k-th item of $\mathbf{y}_i$ in Eq. (17). Then $\mathrm{y}_i^{(k)}$ will be:

$$\mathrm{y}_i^{(k)} = \mathrm{p}_i^{(k)} + \mathbf{\Lambda}_i^{(k)} \boldsymbol{J}(\mathbf{x}_i; \mathbf{w}_*)(\mathbf{w} - \mathbf{w}_*) + \tau_i^{(k)}. \tag{20}$$

Here $\mathbf{\Lambda}_i^{(k)}$ denotes the k-th row of the hessian matrix. The above linear problem can be similarly written in the function space as:

$$\mathrm{y}_i^{(k)} = \mathrm{f}^{(k)}(\mathbf{x}_i) + \tau_i^{(k)} \text{ with } \mathrm{f}^{(k)}(\mathbf{x}) \sim \mathcal{GP}(\mathrm{p}^{(k)} - \mathbf{\Lambda}^{(k)} \boldsymbol{J}(\mathbf{x}; \mathbf{w}_*)\mathbf{w}_*, \delta^{-1}\mathbf{\Lambda}^{(k)} \boldsymbol{J}(\mathbf{x}; \mathbf{w}_*)\boldsymbol{J}(\mathbf{x}'; \mathbf{w}_*)^\top [\mathbf{\Lambda}^{(k)}]'). \tag{21}$$

Given the Laplace approximation $\mathcal{N}(\mathbf{w}_*, \mathbf{\Sigma})$, we can write the mean and covariance function of posterior GP as:

$$m^{(k)}(\mathbf{x}_i) = \mathrm{p}_i^{(k)} + \mathbf{\Lambda}^{(k)} \boldsymbol{J}(\mathbf{x}_i; \mathbf{w}_*)(\mathbf{w}_* - \mathbf{w}_*) = \mathrm{p}_i^{(k)} \tag{22}$$

$$D^{(k)}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{\Lambda}_i^{(k)} \boldsymbol{J}(\mathbf{x}_i; \mathbf{w}_*)\mathbf{\Sigma}\boldsymbol{J}(\mathbf{x}_j'; \mathbf{w}_*)^\top [\mathbf{\Lambda}_j^{(k)}]'. \tag{23}$$

The final objective function is:

$$\min_w \quad \tau \sum_{i \in \mathcal{D}_t} \ell(y_i, f_w(\mathbf{x}_i)) + \frac{1}{2} \sum_{s=1}^{t-1} \sum_{k=1}^{K} (\mathbf{m}_{s,w}^{(k)} - \mathbf{m}_{s,w_{t-1}}^{(k)})^\top [\mathbf{D}_{w_{t-1},s}^{(k)}]^{-1} (\mathbf{m}_{s,w}^{(k)} - \mathbf{m}_{s,w_{t-1}}^{(k)}), \tag{24}$$

B Further toy data experiments

This section provides further information and visualisations of toy 2D datasets, as well as hyperparameter settings for VCL.

B.1 Weight-space regularisation's inconsistent behaviour

Table 3: 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% ± 0.2	98.5% ± 0.6	92% ± 10	85% ± 14	68% ± 8	99.70% ± 0.03

Table 3 summarises the performance (measured by train accuracy) of FROMP and VCL+coresets on a toy dataset similar to that in Figure 1. 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 coreset points chosen for VCL. Without coresets, VCL forgets many past tasks, with very low performance. Previous work (Farquhar & Gal, 2019) has argued that this is inevitable for weight-regularisation-only methods such as VCL and EWC.

We now 3 runs with different random seeds of VCL-MP from Table 3: the coreset is chosen 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. Please note that 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 run on them. This behaviour is not specific to coreset choice.

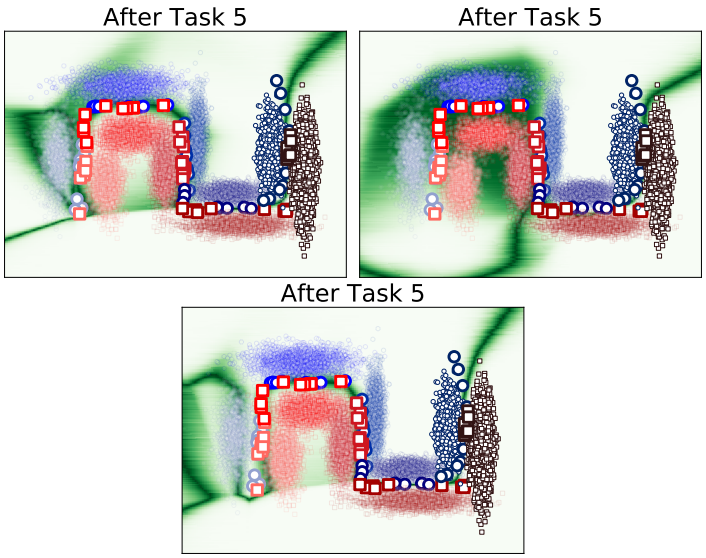


Figure 5: Three runs of VCL-MP on toy 2D data. These are the middle performing 3 runs out of 5 runs with different random seeds. VCL's inconsistent behaviour is clear.

B.2 DATASET VARIATIONS

This section visualises the different dataset variations presented in Table 1. We pick the middle performing FROMP run (out of 5) and batch Adam run to show.

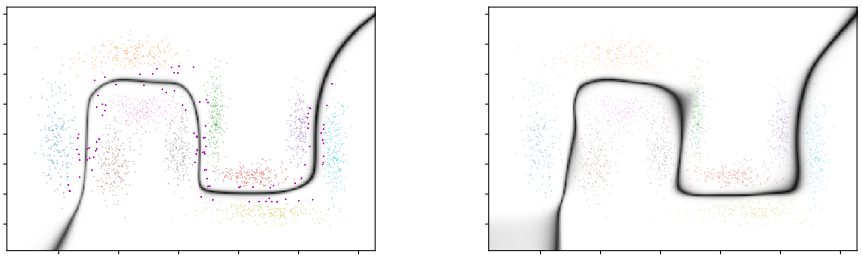


Figure 6: FROMP (middle performing of 5 runs) and batch Adam on a dataset 10x smaller (400 points per task).

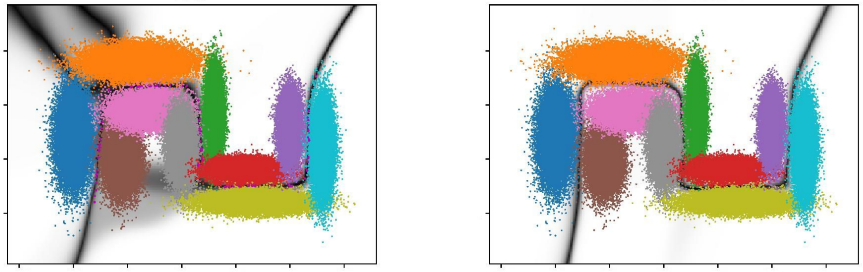


Figure 7: FROMP (middle performing of 5 runs), left, and batch Adam, right, on a dataset 10x larger (40,000 points per task).

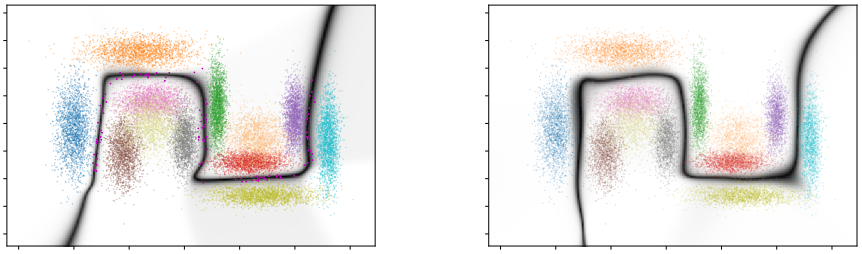


Figure 8: FROMP (middle performing of 5 runs), left, and batch Adam, right, on a dataset with a new, easy, 6th task.

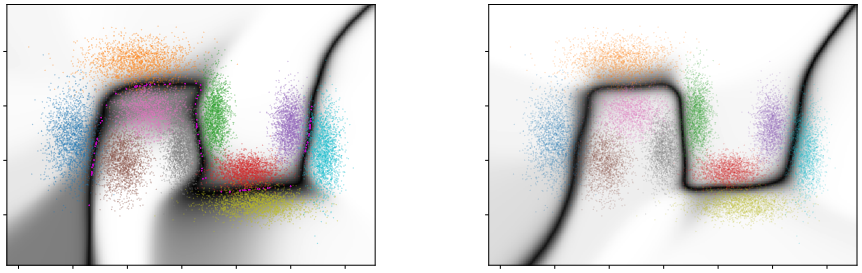


Figure 9: FROMP (middle performing of 5 runs), left, and batch Adam, right, on a dataset with increased standard deviations of each class' points, making classification tougher.

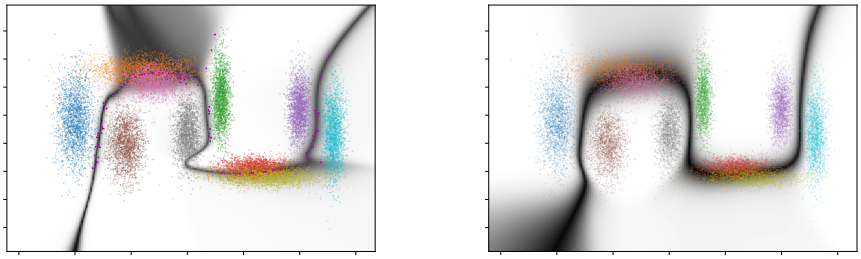


Figure 10: FROMP (middle performing of 5 runs), left, and batch Adam, right, on a dataset with 2 tasks having overlapping data, which is not separable.

B.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 various settings for 5 runs and picking the settings with largest mean train accuracy on the toy dataset in Figure 1. 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 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 all other methods (FROMP and batch Adam).