A. Background

This section provides a background explanation of using BOL to approximate the posterior of the model parameters and overcome catastrophic forgetting, commonly in large-scale supervised classification. We apply this approach to our recursion in Eq. (5).

The posterior is typically intractable due to the enormous size of the modern neural network architectures. This leads to the requirement for a good approximation of the posterior of the model parameters. A particularly suitable candidate for this purpose is the Laplace approximation (MacKay, 1992; Ritter et al., 2018b), as it simply adds a quadratic regulariser to the training objective. Variational continual learning (Nguyen et al., 2018) is another possible method to obtain an approximation for the posterior of the model parameters.

A.1. Bayesian Online Learning

Upon the arrival of the new dataset $\mathfrak{D}_{t+1}$, we consider the posterior $p(\theta|\mathfrak{D}_{1:t+1})$ of the parameters θ of a neural network. Using Bayes' rule on the posterior gives the recursive formula

$$p(\theta|\mathfrak{D}_{1:t+1}) \propto p(\mathfrak{D}_{t+1}|\theta)p(\theta|\mathfrak{D}_{1:t}) \tag{15}$$

where Eq. (15) follows from the assumption that each dataset is independent given θ. As the normalised posterior $p(\theta|\mathfrak{D}_{1:t})$ is usually intractable, it can be approximated by a parametric distribution q with parameter ϕ_t. The BOL framework consists of the *update step* and the *projection step* (Opper, 1998). The update step uses the approximate posterior $q(\theta|\phi_t)$ obtained from the previous step for an update in the form of Eq. (15):

$$p(\theta|\mathfrak{D}_{1:t+1}, \phi_t) \propto p(\mathfrak{D}_{t+1}|\theta)q(\theta|\phi_t). \tag{16}$$

The new posterior $p(\theta|\mathfrak{D}_{1:t+1}, \phi_t)$ might not belong to the same parametric family as $q(\theta|\phi_t)$. In this case, the new posterior has to be projected into the same parametric family to obtain $q(\theta|\phi_{t+1})$. Opper (1998) performs this projection by minimising the KL-divergence between the new posterior and the parametric q, while Ritter et al. (2018a) use Laplace approximation and Nguyen et al. (2018) use variational inference.

A.2. Laplace Approximation

We consider finding a MAP estimate following from Eq. (15):

$$\theta^*_{t+1} = \arg\max_\theta p(\theta|\mathfrak{D}_{1:t+1}) = \arg\max_\theta \{\log p(\mathfrak{D}_{t+1}|\theta) + \log p(\theta|\mathfrak{D}_{1:t})\}. \tag{17}$$

Since the posterior $p(\theta|\mathfrak{D}_{1:t})$ of a neural network is intractable except for small architectures, the unnormalised posterior $\tilde{p}(\theta|\mathfrak{D}_{1:t})$ is considered instead. Performing Taylor expansion on the logarithm of the unnormalised posterior around a mode θ^*_t gives

$$\log \tilde{p}(\theta|\mathfrak{D}_{1:t}) \approx \log \tilde{p}(\theta|\mathfrak{D}_{1:t})\big|_{\theta=\theta^*_t} - \frac{1}{2}(\theta - \theta^*_t)^T A_t(\theta - \theta^*_t), \tag{18}$$

where A_t denotes the Hessian matrix of the negative log-posterior evaluated at θ^*_t. The expansion in Eq. (18) suggests using a Gaussian approximate posterior. Given the parameter $\phi_t = \{\mu_t, \Lambda_t\}$, a mean μ_{t+1} for step $t+1$ can be obtained by finding a mode of the approximate posterior as follows via a standard gradient-based optimisation:

$$\mu_{t+1} = \arg\max_\theta \left\{ \log p(\mathfrak{D}_{t+1}|\theta) - \frac{1}{2}(\theta - \mu_t)^T \Lambda_t(\theta - \mu_t) \right\}. \tag{19}$$

The precision matrix is updated as $\Lambda_{t+1} = H_{t+1} + \Lambda_t$, where H_{t+1} is the Hessian matrix of the negative log-likelihood for $\mathfrak{D}_{t+1}$ evaluated at μ_{t+1} with entries

$$H^{ij}_{t+1} = -\frac{\partial^2}{\partial\theta^{(i)}\partial\theta^{(j)}} \log p(\mathfrak{D}_{t+1}|\theta)\bigg|_{\theta=\mu_{t+1}}. \tag{20}$$

For a neural network model, gradient-based optimisation methods such as SGD (Robbins & Monro, 1951) and Adam (Kingma & Ba, 2015) are the standard gradient-based methods in finding a mode for Laplace approximation in Eq. (19). We show in Section 4.1 that this provides a well-suited skeleton to implement Bayesian online meta-learning in Eq. (5) with the mode-seeking optimisation procedure.

A.3. Block-Diagonal Hessian Approximation

Since the full Hessian matrix in Eq. (20) is intractable for large neural networks, we seek for an efficient and relatively close approximation to the Hessian matrix. Diagonal approximations (Denker & LeCun, 1991; Kirkpatrick et al., 2017) are memory and computationally efficient, but sacrifice approximation accuracy as they ignore the interaction between parameters. Consider instead separating the Hessian matrix into blocks where different blocks are associated to different layers of a neural network. A particular diagonal block corresponds to the Hessian for a particular layer of the neural network. The block-diagonal Kronecker-factored approximation (Martens & Grosse, 2015; Grosse & Martens, 2016; Botev et al., 2017) utilises the fact that each diagonal block of the Hessian is Kronecker-factored for a single data point. This provides a better Hessian approximation as it takes the parameter interactions within a layer into consideration.

A.3.1. Kronecker-Factored Approximation

Consider a neural network with L layers and parameter $\theta = [\mathrm{vec}(W_1)^T, \dots, \mathrm{vec}(W_L)^T]^T$ where W_ℓ is the weight of layer ℓ for $\ell = \{1, \dots, L\}$ and vec denotes stacking the columns of a matrix into a vector. We denote the input of the neural network as $a_0 = x$ and the output of the neural network as h_L. As the input passes through each layer of the neural network, we have the pre-activation for layer ℓ as $h_\ell = W_\ell a_{\ell-1}$ and the activation as $a_\ell = f_\ell(h_\ell)$ where f_ℓ is the activation function of layer ℓ. If a bias vector is applicable in calculating the pre-activation of a layer, we append the bias vector to the last column of the weight matrix and append a scalar one to the last element of the activation. The gradient g_ℓ of loss $L_\theta(x, y) = -\log p(y|x, \theta)$ with respect to h_ℓ for an input-target pair (x, y) is the pre-activation gradient for layer ℓ.

Martens & Grosse (2015) show that the ℓ-th diagonal block F_ℓ of the Fisher information matrix F can be approximated by the Kronecker product between the expectation of the outer product of the $(\ell - 1)$-th layer activation and the ℓ-th layer pre-activation gradient:

$$\begin{aligned} F_\ell &= \mathbb{E}_{x,y}[a_{\ell-1}a_{\ell-1}^T \otimes g_\ell g_\ell^T] && (21)\\ &\approx \mathbb{E}_x[a_{\ell-1}a_{\ell-1}^T] \otimes \mathbb{E}_{y|x}[g_\ell g_\ell^T] && (22)\\ &= A_{\ell-1} \otimes G_\ell, && (23) \end{aligned}$$

where $A_{\ell-1} = \mathbb{E}_x[a_{\ell-1}a_{\ell-1}^T]$ and $G_\ell = \mathbb{E}_{y|x}[g_\ell g_\ell^T]$. Grosse & Martens (2016) extend the block-diagonal Kronecker-factored Fisher approximation for fully-connected layers to that for convolutional layers. For batch normalisation layers, we adopt the unit-wise approximation (Osawa et al., 2020). The Gaussian log-probability term can be calculated efficiently without expanding the Kronecker product using the identity

$$(A_{\ell-1} \otimes G_\ell)\,\mathrm{vec}(W_\ell - W_\ell^*) = \mathrm{vec}(G_\ell(W_\ell - W_\ell^*)A_{\ell-1}^T). \tag{24}$$

As we mentioned in Section 4.2, approximating the Hessian with the one-step SGD inner loop assumption results in having terms that multiply two or more Kronecker products together. The ℓ-th diagonal block of $\widetilde{F}$ in Eq. (11) is

$$\widetilde{F}_\ell = \frac{1}{M}\sum_{m=1}^{M}(I - A_{\ell-1}^m \otimes G_\ell^m)(\widetilde{A}_{\ell-1}^m \otimes \widetilde{G}_\ell^m)(I - A_{\ell-1}^m \otimes G_\ell^m)^T, \tag{25}$$

where $\widetilde{A}_{\ell-1}^m \otimes \widetilde{G}_\ell^m$ is the Kronecker product corresponding to the non-Jacobian terms in Eq. (11) for task m, and $A_{\ell-1}^m \otimes G_\ell^m$ is the Kronecker product corresponding to the Hessian in Eq. (12). We expand $\widetilde{F}_\ell$ using the Kronecker product property:

$$(A_{\ell-1}^m \otimes G_\ell^m)(\widetilde{A}_{\ell-1}^m \otimes \widetilde{G}_\ell^m) = A_{\ell-1}^m\widetilde{A}_{\ell-1}^m \otimes G_\ell^m\widetilde{G}_\ell^m. \tag{26}$$

This gives

$$\widetilde{F}_\ell = \frac{1}{M}\sum_{m=1}^{M}\Big\{\widetilde{A}_{\ell-1}^m \otimes \widetilde{G}_\ell^m - A_{\ell-1}^m\widetilde{A}_{\ell-1}^m \otimes G_\ell^m\widetilde{G}_\ell^m - \widetilde{A}_{\ell-1}^m(A_{\ell-1}^m)^T \otimes \widetilde{G}_\ell^m(G_\ell^m)^T + A_{\ell-1}^m\widetilde{A}_{\ell-1}^m(A_{\ell-1}^m)^T \otimes G_\ell^m\widetilde{G}_\ell^m(G_\ell^m)^T\Big\}. \tag{27}$$

Finally, moving the meta-batch averaging into the Kronecker factors gives the approximation:

$$\widetilde{F}_\ell \approx \widetilde{\boldsymbol{A}}_{\ell-1} \otimes \widetilde{\boldsymbol{G}}_\ell - \boldsymbol{A}_{\ell-1}\widetilde{\boldsymbol{A}}_{\ell-1} \otimes \boldsymbol{G}_\ell\widetilde{\boldsymbol{G}}_\ell - \widetilde{\boldsymbol{A}}_{\ell-1}(\boldsymbol{A}_{\ell-1})^T \otimes \widetilde{\boldsymbol{G}}_\ell(\boldsymbol{G}_\ell)^T + \boldsymbol{A}_{\ell-1}\widetilde{\boldsymbol{A}}_{\ell-1}(\boldsymbol{A}_{\ell-1})^T \otimes \boldsymbol{G}_\ell\widetilde{\boldsymbol{G}}_\ell(\boldsymbol{G}_\ell)^T, \tag{28}$$

where $\widetilde{\boldsymbol{A}}_{\ell-1} = \frac{1}{M}\sum_m \widetilde{A}_{\ell-1}^m$, $\widetilde{\boldsymbol{G}}_\ell = \frac{1}{M}\sum_m \widetilde{G}_\ell^m$, $\boldsymbol{A}_{\ell-1}\widetilde{\boldsymbol{A}}_{\ell-1} = \frac{1}{M}\sum_m A_{\ell-1}^m\widetilde{A}_{\ell-1}^m$, and so on.

A.3.2. Posterior Regularising Hyperparameter for Precision Update

Ritter et al. (2018a) use a hyperparameter λ as a multiplier to the Hessian when updating the precision:

$$\Lambda_{t+1} = \lambda H_{t+1} + \Lambda_t. \tag{29}$$

In the large-scale supervised classification setting, this hyperparameter has a regularising effect on the Gaussian posterior approximation for a balance between having a good performance on a new dataset and maintaining the performance on previous datasets (Ritter et al., 2018a). A large λ results in a sharply peaked Gaussian posterior and is therefore unable to learn new datasets well, but can prevent forgetting previously learned datasets. A small λ in contrast gives a dispersed Gaussian posterior and allows better performance on new datasets by sacrificing the performance on the previous datasets.

A.4. Variational Continual Learning

The variational continual learning method (Nguyen et al., 2018) also provides a suitable meta-training framework for BOML in Eq. (5). Consider approximating the posterior q by minimising the KL-divergence between the parametric q and the new posterior as in the projection step in Eq. (16), where q belongs to some pre-determined approximate posterior family $\mathcal{Q}$ with parameters ϕ_t:

$$q(\theta|\phi_{t+1}) = \arg\min_{q\in\mathcal{Q}} D_{\mathrm{KL}}(q(\theta|\phi)\|p(\mathfrak{D}_{t+1}|\theta)q(\theta|\phi_t)) \tag{30}$$

$$= \arg\min_{q\in\mathcal{Q}} \big\{ -\mathbb{E}_{q(\theta|\phi)}[\log p(\mathfrak{D}_{t+1}|\theta)] + D_{\mathrm{KL}}(q(\theta|\phi)\|q(\theta|\phi_t))\big\}. \tag{31}$$

The optimisation in Eq. (31) leads to the objective

$$\phi_{t+1} = \arg\min_{\phi} \big\{ -\mathbb{E}_{q(\theta|\phi)}[\log p(\mathfrak{D}_{t+1}|\theta)] + D_{\mathrm{KL}}(q(\theta|\phi)\|q(\theta|\phi_t))\big\}. \tag{32}$$

One can use a Gaussian mean-field approximate posterior $q(\theta|\phi_t) = \prod_{d=1}^{D} N(\mu_{t,d}, \sigma_{t,d}^2)$, where $\phi_t = \{\mu_{t,d}, \sigma_{t,d}\}_{d=1}^{D}$ and $D = \dim(\theta)$. The first term in Eq. (32) can be estimated via simple Monte Carlo with local reparameterisation trick (Kingma et al., 2015), and the second KL-divergence term has a closed form for Gaussian distributions.

Algorithm 1 Bayesian online meta-learning with Laplace approximation (BOMLA)

```
 1: Require: sequential base sets (or tasks) $\mathcal{D}_1, \ldots, \mathcal{D}_T$, learning rate $\alpha$, posterior regulariser $\lambda$, number of meta-training
    iterations (or epochs) $J$, meta-batch size (or number of mini-batches) $M$
 2: Initialise: $\mu_0$, $\Lambda_0$, $\theta$
 3: for $t = 1$ to $T$ do
 4:     for $i = 1, \ldots, J$ do                                   ▷ meta-training on base set (or task) $\mathcal{D}_t$
 5:         for $m = 1$ to $M$ do
 6:             Sample task (or split the batch) $\mathcal{D}_t^m = \mathcal{D}_t^{m,S} \cup \mathcal{D}_t^{m,Q}$
 7:             Inner update $\tilde{\theta}^m = SGD_k(\mathcal{L}(\theta, \mathcal{D}_t^{m,S}))$
 8:         end for
 9:         Evaluate loss $f_t^{\text{BOMLA}}(\theta, \mu_{t-1}, \Lambda_{t-1})$ in Eq. (8)
10:         Outer update $\theta \leftarrow \theta - \alpha \nabla_\theta f_t^{\text{BOMLA}}(\theta, \mu_{t-1}, \Lambda_{t-1})$
11:     end for
12:     Update mean $\mu_t \leftarrow \theta$                                    ▷ update posterior mean
13:     For sequential datasets, sample a number of tasks for Hessian approximation
14:     Run inner update in line 7 for each sampled task (or for each batch)
15:     Approximate $\widetilde{H}_t$ with block-diagonal Kronecker-factored approximation to $\widetilde{F}$ in Eq. (11)
16:     Update precision $\Lambda_t \leftarrow \lambda \widetilde{H}_t + \Lambda_{t-1}$                      ▷ update posterior precision
17: end for
```

Algorithm 2 Bayesian online meta-learning with variational inference (BOMVI)

```
 1: Require: sequential base sets (or tasks) $\mathcal{D}_1, \ldots, \mathcal{D}_T$, learning rate $\alpha$, number of meta-training iterations (or epochs) $J$,
    meta-batch size (or number of mini-batches) $M$
 2: Initialise: $\phi_0 = \{\mu_0, \sigma_0\}$
 3: for $t = 1$ to $T$ do
 4:     for $i = 1, 2, \ldots, J$ do                                ▷ meta-training on base set (or task) $\mathcal{D}_t$
 5:         for $m = 1$ to $M$ do
 6:             Sample task (or split the batch) $\mathcal{D}_t^m = \mathcal{D}_t^{m,S} \cup \mathcal{D}_t^{m,Q}$
 7:             Inner update $\tilde{\theta}^m = SGD_k(\mathcal{L}(\theta, \mathcal{D}_t^{m,S}))$
 8:         end for
 9:         Evaluate loss $f_t^{\text{BOMVI}}(\phi, \phi_{t-1})$ in Eq. (14)
10:         Outer update $\mu \leftarrow \mu - \alpha \nabla_\mu f_t^{\text{BOMVI}}(\phi, \phi_{t-1})$, and $\sigma \leftarrow \sigma - \alpha \nabla_\sigma f_t^{\text{BOMVI}}(\phi, \phi_{t-1})$
11:     end for
12:     Update $\mu_t \leftarrow \mu$ and $\sigma_t \leftarrow \sigma$                          ▷ update posterior parameters
13: end for
```

B. Algorithms

B.1. BOMLA and BOMVI

Algorithm 1 gives the pseudo-code of the BOMLA algorithm, with the corresponding variation for the Section 6.2 Omniglot sequential tasks setting in *brackets*. The algorithm is formed of three main elements: meta-training on a specific base set or task (line 4 – 11), updating the Gaussian mean (line 12) and updating the Gaussian precision (line 13 – 16). For the precision update, we approximate the Hessian using block-diagonal Kronecker-factored approximation.

Algorithm 2 gives the pseudo-code of the BOMVI algorithm, with the corresponding variation for the Section 6.2 Omniglot sequential tasks setting in *brackets*. The algorithm is formed of two main elements: meta-training on a specific base set or task (line 4 – 11) and updating the parameters of the Gaussian mean-field approximate posterior (line 12).

B.2. BOMVI Monte Carlo Estimator

Recall that the BOMVI objective is:

$$f_{t+1}^{\text{BOMVI}}(\phi, \phi_t) = -\frac{1}{M}\sum_{m=1}^{M}\mathbb{E}_{q(\theta|\phi)}\big[\log p(\mathcal{D}_{t+1}^{m,Q}|\tilde{\theta}^m)\big] - \frac{1}{M}\sum_{m=1}^{M}\mathbb{E}_{q(\theta|\phi)}\big[\log p(\mathcal{D}_{t+1}^{m,S}|\theta)\big] + D_{\text{KL}}(q(\theta|\phi)\|q(\theta|\phi_t)),$$

where $\tilde{\theta}^m = SGD_k(\mathcal{L}(\theta, \mathcal{D}_{t+1}^{m,S}))$ for $m = 1, \dots, M$. The Monte Carlo estimator for the first term of the BOMVI objective is difficult to compute, as every sampled meta-parameters θ_r for $r = 1, \dots, R$ has to undergo a few-shot quick adaptation prior to the log-likelihood evaluation. As a consequence the estimator is prone to a large variance. Moreover, every quickly-adapted sample θ_r contributes to the meta-learning gradients of the posterior mean and covariance, resulting in a high computational cost when taking the meta-gradients.

To solve these impediments, we introduce a slight modification to the SGD quick adaptation $\widetilde{\theta}^m$. Instead of taking the gradients with respect to the sampled meta-parameters, we consider the gradients with respect to the *posterior mean*. A one-step SGD quick adaptation, for instance, becomes:

$$\widetilde{\theta}^m = \theta - \alpha\nabla_{\mu_t}\mathcal{L}(\mu_t, \mathcal{D}_{t+1}^{m,S}). \tag{33}$$

This gives $\widetilde{\theta}^m \sim N(\widetilde{\mu}_t, \text{diag}(\sigma_t^2))$ where

$$\widetilde{\mu}_t = \mu_t - \alpha\nabla_{\mu_t}\mathcal{L}(\mu_t, \mathcal{D}_{t+1}^{m,S}), \tag{34}$$

since $\theta \sim N(\mu_t, \text{diag}(\sigma_t^2))$. A quick adaptation with more steps works in a similar fashion. With this modification, we can calculate the Monte Carlo estimate for the first term using the local reparameterisation trick as usual.

C. Experiments

C.1. Triathlon and Pentathlon

In these experiments, we use the model architecture proposed by Vinyals et al. (2016) that takes 4 modules with 64 filters of size 3×3, followed by a batch normalisation, a ReLU activation and a 2×2 max-pooling. A fully-connected layer is appended to the final module before getting the class probabilities with softmax. Tables 1 and 2 are the hyperparameters used in these experiment.

Omniglot: Omniglot (Lake et al., 2011) comprises 1623 characters from 50 alphabets and each character has 20 instances. We use 1100 characters for meta-training, 100 characters for validation and the remaining for meta-evaluation. New classes with rotations in the multiples of 90° are formed after splitting the characters as mentioned.

***mini*QuickDraw:** QuickDraw (Ha & Eck, 2017) comprises 345 categories of drawings collected from the players in the game "Quick, Draw!". We generate *mini*QuickDraw by randomly sampling 1000 instances in each class of QuickDraw.

CIFAR-FS: CIFAR-FS (Bertinetto et al., 2019) has 100 classes of objects and each class comprises 600 images. We use the same split as Bertinetto et al. (2019): 64 classes for meta-training, 16 classes for validation and 20 classes for meta-evaluation.

***mini*ImageNet:** *mini*ImageNet (Vinyals et al., 2016) takes 100 classes and 600 instances in each class from the ImageNet dataset. We use the same split as Ravi & Larochelle (2017): 64 classes for meta-training, 16 classes for validation and 20 classes for meta-evaluation.

VGG-Flowers: VGG-Flowers (Nilsback & Zisserman, 2008) comprises 102 different types of flowers as the classes. We randomly split 66 classes for meta-training, 16 classes for validation and 20 classes for meta-evaluation.

Aircraft: Aircraft (Maji et al., 2013) is a fine-grained dataset consisting of 100 aircraft models as the classes and each class has 100 images. We randomly split 64 classes for meta-training, 16 classes for validation and 20 classes for meta-evaluation.

Table 1. Hyperparameters for the triathlon and pentathlon experiments (same value for all datasets)

Hyperparameter	BOMLA	BOMVI
Posterior regulariser λ	(various values)	-
Precision initialisation values	$10^{-4} \sim 10^{-2}$	-
Number of tasks sampled for Hessian approx.	5000	-
Covariance initialisation values	-	$\exp(-5)$
Number of Monte Carlo samples	-	20
Meta-batch size M	32	32
Number of query samples per class	15	15
Number of iterations per dataset	5000	5000
Outer loop optimiser	Adam	Adam
Outer loop learning rate	0.001	0.001
Number of tasks sampled for meta-evaluation	100	100

Table 2. Hyperparameters for the triathlon and pentathlon experiments (individual datasets)

Hyperparameter	Omniglot	*mini*QuickDraw	CIFAR-FS	*mini*ImageNet	VGG-Flowers	Aircraft
Number of inner SGD steps in meta-training (k)	1	3	5	5	5	5
Inner SGD learning rate (α)	0.4	0.2	0.1	0.1	0.1	0.1
Outer learning rate decay schedule (none for BOMVI)	-	$\times 0.1$ halfway	$\times 0.1$ halfway	$\times 0.1$ halfway	$\times 0.1$ every 1000 iterations	$\times 0.1$ halfway
Number of inner SGD steps in meta-evaluation	3	5	10	10	10	10

C.2. Pentathlon: Analysing the Change in Approximate Posterior Covariance

We visualise the covariance of the meta-parameters approximate posterior from BOMVI to better understand how the uncertainty in the algorithm prevents catastrophic forgetting in few-shot classification problems. We follow the pentathlon sequence going from left to right of the figure: Omniglot $\rightarrow$ CIFAR-FS $\rightarrow$ *mini*ImageNet $\rightarrow$ VGG-Flowers $\rightarrow$ Aircraft. The Gaussian mean-field approximate posterior becomes increasingly concentrated in general as it learns on more datasets. This is especially true for the earlier layers (Conv 1 and Conv 2), meaning that the posterior progressively becomes very confident on the meta-parameters of the raw-level filters. The covariance for the layer closest to the classifier (Conv 4) remains large in general, although there are some filters with decreasing covariance. As the convolutional layer gets closer to the classifying layer, a larger fine-tuning in the meta-parameters is needed (Ravi & Beatson, 2019) to cope with few-shot tasks from different knowledge domains. The approximate posterior covariance from BOMLA is too large for visualisation as it is block-diagonal. The BOMLA covariance for each convolutional layer has dimension $D \times D$ where D is the number of parameters in a convolutional layer. In theory, the BOMLA covariance should also follow the same pattern as the BOMVI covariance.

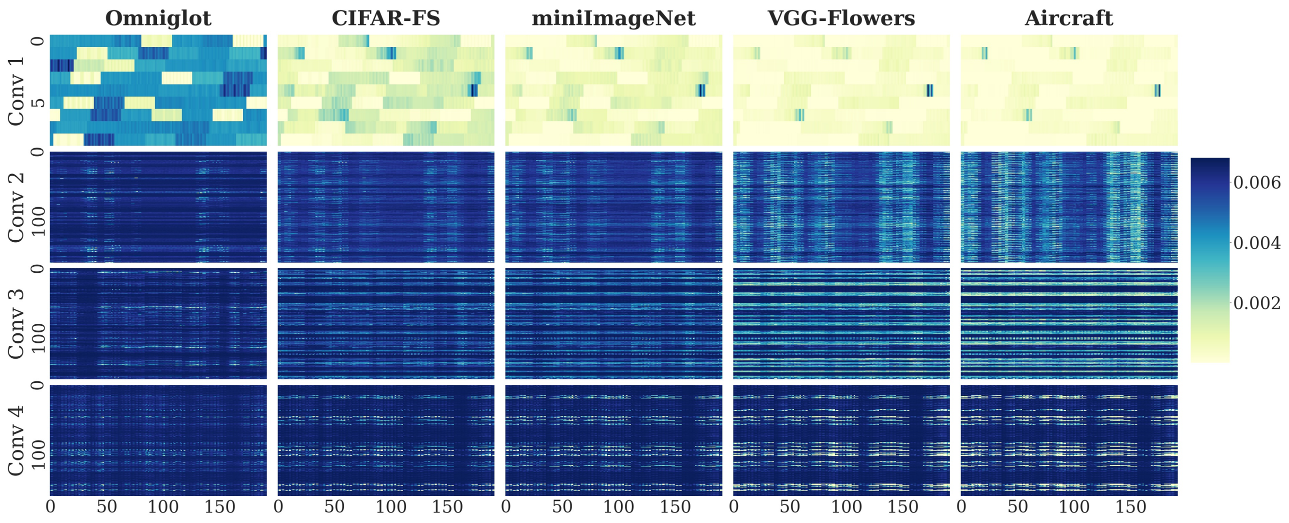

Figure 9. The change in the approximate posterior covariance after meta-training is completed on each dataset. Going from left to right are the pentathlon sequence of datasets. Going from top to bottom are the convolutional layers of the neural network which gets closer to the classifying layer.

C.3. Pentathlon: the Comparison between BOMVI and BOMLA with Different Values of λ

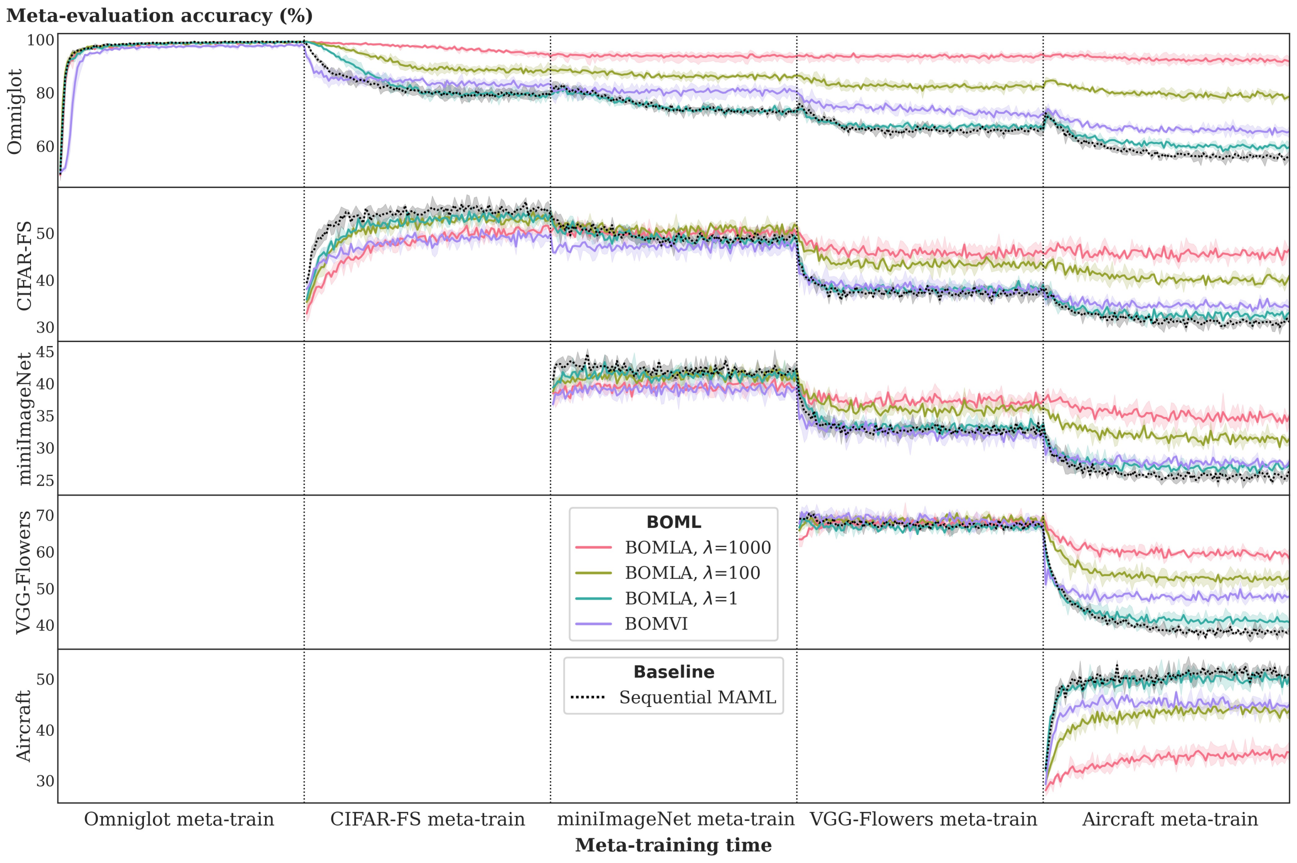

Figure 10. Meta-evaluation accuracy across 3 seed runs on each dataset along meta-training. Higher accuracy values indicate better results with less forgetting as we proceed to new datasets. BOMLA with a large $\lambda = 1000$ gives better performance in the off-diagonal plots (retains performances on previously learned datasets) but worse performance in the diagonal plots (does not learn well on new datasets). A small $\lambda = 1$ gives better performance in the diagonal plots (learns well on new datasets) but worse performance in the off-diagonal plots (forgets previously learned datasets). BOMVI is also able to retain performance on previous datasets, although it may be unable to perform as good as BOMLA due to sampling and estimator variance.

Tuning the posterior regulariser λ mentioned in Section 4.2 and Appendix A.3.2 corresponds to balancing between a smaller performance trade-off on a new dataset and less forgetting on previous datasets. As shown in the figure above, a larger $\lambda = 1000$ results in a more concentrated Gaussian posterior and is therefore unable to learn new datasets well, but can better retain the performances on previous datasets. A smaller $\lambda = 1$ on the other hand gives a widespread Gaussian posterior and learns better on new datasets by sacrificing the performance on the previous datasets. In this experiment, the value $\lambda = 100$ gives the best balance between old and new datasets. Ideally we seek for a good performance on both old and new datasets, but in reality there is a trade-off between retaining performance on old datasets and learning well on new datasets due to posterior approximation errors.

C.4. Omniglot: Sequential Tasks from a Stationary Task Distribution

In this experiment, we use the model architecture proposed by Vinyals et al. (2016) that takes 4 modules with 64 filters of size 3×3, followed by a batch normalisation, a ReLU activation and a 2×2 max-pooling. A fully-connected layer is appended to the final module before getting the class probabilities with softmax. Table 3 shows the hyperparameters used in this experiment.

The Omniglot dataset comprises 50 alphabets (super-classes). Each alphabet has numerous characters (classes) and each character has 20 instances. As the meta-training alphabets arrive sequentially, we form **non-overlapping** sequential tasks from each arriving alphabet, and the tasks also do not overlap in the characters. We use 35 alphabets for meta-training, 7 alphabets for validation and 8 alphabets for meta-evaluation. The alphabet splits are as follows:

35 alphabets for meta-training:

```
Alphabet_of_the_Magi, Angelic, Armenian, Asomtavruli_(Georgian), Atlantean,
Aurek-Besh, Avesta, Balinese, Bengali, Braille, Burmese_(Myanmar), Early_Aramaic,
Grantha, Gujarati, Gurmukhi, Hebrew, Inuktitut_(Canadian_Aboriginal_Syllabics),
Japanese_(hiragana), Japanese_(katakana), Kannada, Keble, Korean,
Latin, Malayalam, Malay_(Jawi_-_Arabic), Manipuri, Mongolian,
Ojibwe_(Canadian_Aboriginal_Syllabics), Old_Church_Slavonic_(Cyrillic), Oriya,
Sanskrit, Sylheti, Tengwar, Tifinagh, ULOG
```

7 alphabets for validation:

```
Anglo-Saxon_Futhorc, Arcadian, Blackfoot_(Canadian_Aboriginal_Syllabics),
Cyrillic, Ge_ez, Glagolitic, N_Ko
```

8 alphabets for meta-evaluation:

```
Atemayar_Qelisayer, Futurama, Greek, Mkhedruli_(Georgian), Syriac_(Estrangelo),
Syriac_(Serto), Tagalog, Tibetan
```

Table 3. Hyperparameters for the Omniglot sequential tasks experiment.

Hyperparameter	**BOMLA**	**BOMVI**
Posterior regulariser λ	0.01	-
Precision initialisation values	$10^{-4} \sim 10^{-2}$	-
Covariance initialisation values	-	$\exp(-10)$
Number of Monte Carlo samples	-	5
Number of mini-batches M	1	1
Number of query samples per class (meta-evaluation)	15	15
Number of epochs per task	50	50
Number of inner SGD steps in meta-training (k)	5	5
Inner SGD learning rate (α)	0.1	0.1
Outer loop optimiser	Adam	Adam
Outer loop learning rate	0.001	0.001
Number of tasks sampled for meta-evaluation	100	100
Number of inner SGD steps in meta-evaluation (k)	10	10