Practical Deep Learning with Bayesian Principles

Kazuki Osawa,[1] **Siddharth Swaroop,**[2,*] **Anirudh Jain,**[3,*,†] **Runa Eschenhagen,**[4,†]
Richard E. Turner,[2] **Rio Yokota,**[1] **Mohammad Emtiyaz Khan**[5,‡]**.**

[1] Tokyo Institute of Technology, Tokyo, Japan
[2] University of Cambridge, Cambridge, UK
[3] Indian Institute of Technology (ISM), Dhanbad, India
[4] University of Osnabrück, Osnabrück, Germany
[5] RIKEN Center for AI Project, Tokyo, Japan

Abstract

Bayesian methods promise to fix many shortcomings of deep learning, but they are impractical and rarely match the performance of standard methods, let alone improve them. In this paper, we demonstrate practical training of deep networks with natural-gradient variational inference. By applying techniques such as batch normalisation, data augmentation, and distributed training, we achieve similar performance in about the same number of epochs as the Adam optimiser, even on large datasets such as ImageNet. Importantly, the benefits of Bayesian principles are preserved: predictive probabilities are well-calibrated, uncertainties on out-of-distribution data are improved, and continual-learning performance is boosted. This work enables practical deep learning while preserving benefits of Bayesian principles. A PyTorch implementation[1] is available as a plug-and-play optimiser.

1 Introduction

Deep learning has been extremely successful in many fields such as computer vision [29], speech processing [17], and natural-language processing [39], but it is also plagued with several issues that make its application difficult in many other fields. For example, it requires a large amount of high-quality data and it can overfit when dataset size is small. Similarly, sequential learning can cause forgetting of past knowledge [27], and lack of reliable confidence estimates and other robustness issues can make it vulnerable to adversarial attacks [6]. Ultimately, due to such issues, application of deep learning remains challenging, especially for applications where human lives are at risk.

Bayesian principles have the potential to address such issues. For example, we can represent uncertainty using the posterior distribution, enable sequential learning using Bayes' rule, and reduce overfitting with Bayesian model averaging [19]. The use of such Bayesian principles for neural networks has been advocated from very early on. Bayesian inference on neural networks were all proposed in the 90s, e.g., by using MCMC methods [41], Laplace's method [35], and variational inference (VI) [18, 2, 49, 1]. Benefits of Bayesian principles are even discussed in machine-learning textbooks [36, 3]. Despite this, they are rarely employed in practice. This is mainly due to computational concerns, unfortunately overshadowing their theoretical advantages.

The difficulty lies in the computation of the posterior distribution, which is especially challenging for deep learning. Even approximation methods, such as VI and MCMC, have historically been difficult

* These two authors contributed equally.
† This work is conducted during an internship at RIKEN Center for AI project.
‡ Corresponding author: `emtiyaz.khan@riken.jp`
[1] The code is available at `https://github.com/team-approx-bayes/dl-with-bayes`.

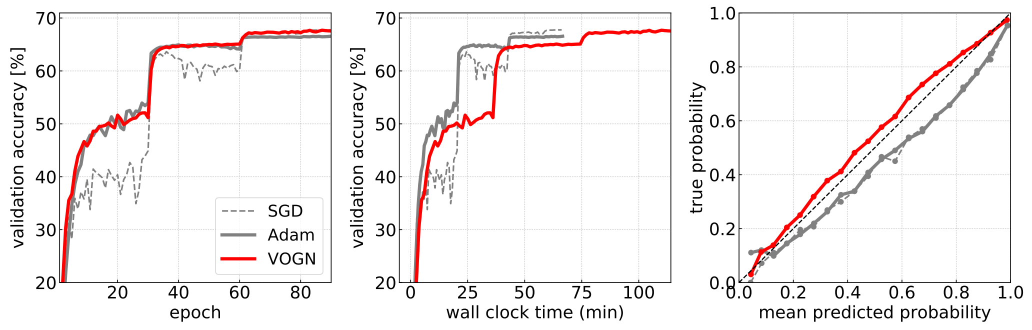


Figure 1: Comparing VOGN [24], a natural-gradient VI method, to Adam and SGD, training ResNet-18 on ImageNet. The two left plots show that VOGN and Adam have similar convergence behaviour and achieve similar performance in about the same number of epochs. VOGN achieves 67.38% on validation compared to 66.39% by Adam and 67.79% by SGD. Run-time of VOGN is 76 seconds per epoch compared to 44 seconds for Adam and SGD. The rightmost figure shows the calibration curve. VOGN gives calibrated predictive probabilities (the diagonal represents perfect calibration).

to scale to large datasets such as ImageNet [47]. Due to this, it is common to use less principled approximations, such as MC-dropout [9], even though they are not ideal when it comes to fixing the issues of deep learning. For example, MC-dropout is unsuitable for continual learning [27] since its posterior approximation does not have mass over the whole weight space. It is also found to perform poorly for sequential decision making [45]. The form of the approximation used by such methods is usually rigid and cannot be easily improved, e.g., to other forms such as a mixture of Gaussians. The goal of this paper is to make more principled Bayesian methods, such as VI, practical for deep learning, thereby helping researchers tackle its key limitations.

We demonstrate practical training of deep networks by using recently proposed natural-gradient VI methods. These methods resemble the Adam optimiser, enabling us to leverage existing techniques for initialisation, momentum, batch normalisation, data augmentation, and distributed training. As a result, we obtain similar performance in about the same number of epochs as Adam when training many popular deep networks (e.g., LeNet, AlexNet, ResNet) on datasets such as CIFAR-10 and ImageNet (see Fig. 1). The results show that, despite using an approximate posterior, the training methods preserve the benefits coming from Bayesian principles. Compared to standard deep-learning methods, the predictive probabilities are well-calibrated, uncertainties on out-of-distribution inputs are improved, and performance for continual-learning tasks is boosted. Our work shows that practical deep learning is possible with Bayesian methods and aims to support further research in this area.

Related work. Previous VI methods, notably by Graves [14] and Blundell et al. [4], require significant implementation and tuning effort to perform well, e.g., on convolution neural networks (CNN). Slow convergence is found to be especially problematic for sequential problems [45]. There appears to be no reported results with complex networks on large problems, such as ImageNet. Our work solves these issues by applying deep-learning techniques to natural-gradient VI [24, 56].

In their paper, Zhang et al. [56] also employed data augmentation and batch normalisation for a natural-gradient method called Noisy K-FAC (see Appendix A) and showed results on VGG on CIFAR-10. However, a mean-field method called Noisy Adam was found to be unstable with batch normalisation. In contrast, we show that a similar method, called Variational Online Gauss-Newton (VOGN), proposed by Khan et al. [24], works well with such techniques. We show results for distributed training with Noisy K-FAC on Imagenet, but do not provide extensive comparisons since tuning it is time-consuming. Many of our techniques can speed-up Noisy K-FAC, which is promising.

Many other approaches have recently been proposed to compute posterior approximations by training deterministic networks [46, 37, 38]. Similarly to MC-dropout, their posterior approximations are not flexible, making it difficult to improve the accuracy of their approximations. On the other hand, VI offers a much more flexible alternative to apply Bayesian principles to deep learning.

2 Deep Learning with Bayesian Principles and Its Challenges

The success of deep learning is partly due to the availability of scalable and practical methods for training deep neural networks (DNNs). Network training is formulated as an optimisation problem where a loss between the data and the DNN's predictions is minimised. For example, in a supervised learning task with a dataset $\mathcal{D}$ of N inputs $\mathbf{x}_i$ and corresponding outputs $\mathbf{y}_i$ of length K, we minimise a loss of the following form: $\bar{\ell}(\mathbf{w}) + \delta \mathbf{w}^\top \mathbf{w}$, where $\bar{\ell}(\mathbf{w}) := \frac{1}{N}\sum_i \ell(\mathbf{y}_i, \mathbf{f}_w(\mathbf{x}_i))$, $\mathbf{f}_w(\mathbf{x}) \in \mathbb{R}^K$ denotes the DNN outputs with weights $\mathbf{w}$, $\ell(\mathbf{y}, \mathbf{f})$ denotes a differentiable loss function between an output $\mathbf{y}$ and the function $\mathbf{f}$, and $\delta > 0$ is the L_2 regulariser.[2] Deep learning relies on stochastic-gradient (SG) methods to minimise such loss functions. The most commonly used optimisers, such as stochastic-gradient descent (SGD), RMSprop [53], and Adam [25], take the following form[3] (all operations below are element-wise):

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \alpha_t \frac{\hat{\mathbf{g}}(\mathbf{w}_t) + \delta \mathbf{w}_t}{\sqrt{\mathbf{s}_{t+1}} + \epsilon}, \qquad \mathbf{s}_{t+1} \leftarrow (1-\beta_t)\mathbf{s}_t + \beta_t \left(\hat{\mathbf{g}}(\mathbf{w}_t) + \delta \mathbf{w}_t\right)^2, \tag{1}$$

where t is the iteration, $\alpha_t > 0$ and $0 < \beta_t < 1$ are learning rates, $\epsilon > 0$ is a small scalar, and $\hat{\mathbf{g}}(\mathbf{w})$ is the stochastic gradients at $\mathbf{w}$ defined as follows: $\hat{\mathbf{g}}(\mathbf{w}) := \frac{1}{M}\sum_{i \in \mathcal{M}_t} \nabla_w \ell(\mathbf{y}_i, \mathbf{f}_w(\mathbf{x}_i))$ using a minibatch $\mathcal{M}_t$ of M data examples. This simple update scales extremely well and can be applied to very large problems. With techniques such as initialisation protocols, momentum, weight-decay, batch normalisation, and data augmentation, it also achieves good performance for many problems.

In contrast, the full Bayesian approach to deep learning is computationally very expensive. The posterior distribution can be obtained using Bayes' rule: $p(\mathbf{w}|\mathcal{D}) = \exp\left(-N\bar{\ell}(\mathbf{w})/\tau\right) p(\mathbf{w})/p(\mathcal{D})$ where $0 < \tau \le 1$.[4] This is costly due to the computation of the marginal likelihood $p(\mathcal{D})$, a high-dimensional integral that is difficult to compute for large networks. Variational inference (VI) is a principled approach to more scalably estimate an approximation to $p(\mathbf{w}|\mathcal{D})$. The main idea is to employ a parametric approximation, e.g., a Gaussian $q(\mathbf{w}) := \mathcal{N}(\mathbf{w}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$. The parameters $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ can then be estimated by maximising the *evidence lower bound (ELBO)*:

$$\text{ELBO:} \quad \mathcal{L}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) := -N\mathbb{E}_q\left[\bar{\ell}(\mathbf{w})\right] - \tau \mathbb{D}_{KL}[q(\mathbf{w}) \,\|\, p(\mathbf{w})], \tag{2}$$

where $\mathbb{D}_{KL}[\cdot]$ denotes the Kullback-Leibler divergence. By using more complex approximations, we can further reduce the approximation error, but at a computational cost. By formulating Bayesian inference as an optimisation problem, VI enables a practical application of Bayesian principles.

Despite this, VI has remained impractical for training large deep networks on large datasets. Existing methods, such as Graves [14] and Blundell et al. [4], directly apply popular SG methods to optimise the variational parameters in the ELBO, yet they fail to get a reasonable performance on large problems, usually converging very slowly. The failure of such direct applications of deep-learning methods to VI is not surprising. The techniques used in one field may not directly lead to improvements in the other, but it will be useful if they do, e.g., if we can optimise the ELBO in a way that allows us to exploit the tricks and techniques of deep learning and boost the performance of VI. The goal of this work is to do just that. We now describe our methods in detail.

3 Practical Deep Learning with Natural-Gradient Variational Inference

In this paper, we propose natural-gradient VI methods for practical deep learning with Bayesian principles. The natural-gradient update takes a simple form when estimating exponential-family approximations [23, 22]. When $p(\mathbf{w}) := \mathcal{N}(\mathbf{w}|0, \mathbf{I}/\delta)$, the update of the natural-parameter $\boldsymbol{\lambda}$ is performed by using the stochastic gradient of the *expected regularised-loss*:

$$\boldsymbol{\lambda}_{t+1} = (1-\tau\rho)\boldsymbol{\lambda}_t - \rho \nabla_\mu \mathbb{E}_q\left[\bar{\ell}(\mathbf{w}) + \tfrac{1}{2}\tau\delta\mathbf{w}^\top\mathbf{w}\right], \tag{3}$$

[2]This regulariser is sometimes set to 0 or a very small value.

[3]Alternate versions with weight-decay and momentum differ from this update [34]. We present a form useful to establish the connection between SG methods and natural-gradient VI.

[4]This is a tempered posterior [54] setup where τ is set $\neq 1$ when we expect model misspecification and/or adversarial examples [10]. Setting $\tau = 1$ recovers standard Bayesian inference.

p(w): prior.q(w): variational appro

I don't quite get this.

Ah OK, so the used method VOGN is actually

where $\rho > 0$ is the learning rate, and we note that the stochastic gradients are computed with respect to $\boldsymbol{\mu}$, the *expectation parameters* of q. The *moving average* above helps to deal with the stochasticity of the gradient estimates, and is very similar to the moving average used in deep learning (see (1)). When τ is set to 0, the update essentially minimises the regularised loss (see Section 5 in Khan et al. [24]). These properties of natural-gradient VI makes it an ideal candidate for deep learning.

Recent work by Khan et al. [24] and Zhang et al. [56] further show that, when q is Gaussian, the update (3) assumes a form that is strikingly similar to the update (1). For example, the Variational Online Gauss-Newton (VOGN) method of Khan et al. [24] estimates a Gaussian with mean $\boldsymbol{\mu}_t$ and a diagonal covariance matrix $\boldsymbol{\Sigma}_t$ using the following update:

$$\boldsymbol{\mu}_{t+1} \leftarrow \boldsymbol{\mu}_t - \alpha_t \frac{\hat{\mathbf{g}}(\mathbf{w}_t) + \tilde{\delta}\boldsymbol{\mu}_t}{\mathbf{s}_{t+1} + \tilde{\delta}}, \quad \mathbf{s}_{t+1} \leftarrow (1 - \tau\beta_t)\mathbf{s}_t + \beta_t \frac{1}{M} \sum_{i \in \mathcal{M}_t} \left(\mathbf{g}_i(\mathbf{w}_t)\right)^2, \qquad (4)$$

where $\mathbf{g}_i(\mathbf{w}_t) := \nabla_w \ell(y_i, f_{w_t}(\mathbf{x}_i))$, $\mathbf{w}_t \sim \mathcal{N}(\mathbf{w}|\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$ with $\boldsymbol{\Sigma}_t := \text{diag}(1/(N(\mathbf{s}_t + \tilde{\delta})))$, $\tilde{\delta} := \tau\delta/N$, and $\alpha_t, \beta_t > 0$ are learning rates. Operations are performed element-wise. Similarly to (1), the vector $\mathbf{s}_t$ adapts the learning rate and is updated using a moving average.

A major difference in VOGN is that the update of $\mathbf{s}_t$ is now based on a Gauss-Newton approximation [14] which uses $\frac{1}{M}\sum_{i\in\mathcal{M}_t}(\mathbf{g}_i(\mathbf{w}_t))^2$. This is fundamentally different from the SG update in (1) which instead uses the gradient-magnitude $(\frac{1}{M}\sum_{i\in\mathcal{M}_t}\mathbf{g}_i(\mathbf{w}_t) + \delta\mathbf{w}_t)^2$ [5]. The first approach uses the sum *outside* the square while the second approach uses it *inside*. VOGN is therefore a second-order method and, similarly to Newton's method, does not need a square-root over $\mathbf{s}_t$. Implementation of this step requires an additional calculation (see Appendix B) which makes VOGN a bit slower than Adam, but VOGN is expected to give better variance estimates (see Theorem 1 in Khan et al. [24]).

The main contribution of this paper is to demonstrate practical training of deep networks using VOGN. Since VOGN takes a similar form to SG methods, we can easily borrow existing deep-learning techniques to improve performance. We will now describe these techniques in detail. Pseudo-code for VOGN is shown in Algorithm 1.

Batch normalisation: Batch normalisation [20] has been found to significantly speed up and stabilise training of neural networks, and is widely used in deep learning. BatchNorm layers are inserted between neural network layers. They help stabilise each layer's input distribution by normalising the running average of the inputs' mean and variance. In our VOGN implementation, we simply use the existing implementation with default hyperparameter settings. We do not apply L2 regularisation and weight decay to BatchNorm parameters, like in Goyal et al. [13], or maintain uncertainty over the BatchNorm parameters. This straightforward application of batch normalisation works for VOGN.

Data Augmentation: When training on image datasets, data augmentation (DA) techniques can improve performance drastically [13]. We consider two common real-time data augmentation techniques: random cropping and horizontal flipping. After randomly selecting a minibatch at each iteration, we use a randomly selected cropped version of all images. Each image in the minibatch has a 50% chance of being horizontally flipped.

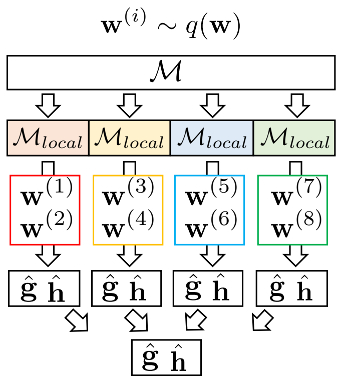
? We find that directly applying DA gives slightly worse performance than expected, and also affects the calibration of the resulting uncertainty. However, DA increases the effective sample size. We therefore modify it to be ρN where $\rho \geq 1$, improving performance (see step 2 in Algorithm 1). The reason for this performance boost might be due to the complex relationship between the regularisation δ and N. For the regularised loss $\bar{\ell}(\mathbf{w}) + \delta\mathbf{w}^\top\mathbf{w}$, the two are unidentifiable, i.e., we can multiply δ by a constant and reduce N by the same constant without changing the minimum. However, in a Bayesian setting (like in (2)), the two quantities are separate, and therefore changing the data might also change the optimal prior variance hyperparameter in a complicated way. This needs further theoretical investigations, but our simple fix of scaling N seems to work well in the experiments.

We set ρ by considering the specific DA techniques used. When training on CIFAR-10, the random cropping DA step involves first padding the 32x32 images to become of size 40x40, and then taking randomly selected 28x28 cropped images. We consider this as effectively increasing the dataset size by a factor of 5 (4 images for each corner, and one central image). The horizontal flipping DA step doubles the dataset size (one dataset of unflipped images, one for flipped images). Combined, this gives $\rho = 10$. Similar arguments for ImageNet DA techniques give $\rho = 5$. Even though ρ is another hyperparameter to set, we find that its precise value does not matter much. Typically, after setting an estimate for ρ, tuning δ a little seems to work well (see Appendix E).

This seems somewhat strange to me.

Algorithm 1: Variational Online Gauss Newton (VOGN)

```
1:  Initialise $\mu_0$, $s_0$, $m_0$.
2:  $N \leftarrow \rho N$, $\tilde{\delta} \leftarrow \tau\delta/N$.
3:  repeat
4:      Sample a minibatch $\mathcal{M}$ of size $M$.
5:      Split $\mathcal{M}$ into each GPU (local minibatch $\mathcal{M}_{local}$).
6:      for each GPU in parallel do
7:          for $k = 1, 2, \ldots, K$ do
8:              Sample $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$.
9:              $\mathbf{w}^{(k)} \leftarrow \boldsymbol{\mu} + \epsilon\boldsymbol{\sigma}$ with $\boldsymbol{\sigma} \leftarrow (1/(N(\mathbf{s} + \tilde{\delta} + \gamma)))^{1/2}$.
10:             Compute $\mathbf{g}_i^{(k)} \leftarrow \nabla_w \ell(\mathbf{y}_i, \mathbf{f}_{w^{(k)}}(\mathbf{x}_i)), \forall i \in \mathcal{M}_{local}$
                using the method described in Appendix B.
11:             $\hat{\mathbf{g}}_k \leftarrow \frac{1}{M}\sum_{i\in\mathcal{M}_{local}} \mathbf{g}_i^{(k)}$.
12:             $\hat{\mathbf{h}}_k \leftarrow \frac{1}{M}\sum_{i\in\mathcal{M}_{local}} (\mathbf{g}_i^{(k)})^2$ .
13:         end for
14:         $\hat{\mathbf{g}} \leftarrow \frac{1}{K}\sum_{k=1}^{K} \hat{\mathbf{g}}_k$ and $\hat{\mathbf{h}} \leftarrow \frac{1}{K}\sum_{k=1}^{K} \hat{\mathbf{h}}_k$.
15:     end for
16:     AllReduce $\hat{\mathbf{g}}$, $\hat{\mathbf{h}}$.
17:     $\mathbf{m} \leftarrow \beta_1 \mathbf{m} + (\hat{\mathbf{g}} + \tilde{\delta}\boldsymbol{\mu})$.
18:     $\mathbf{s} \leftarrow (1 - \tau\beta_2)\mathbf{s} + \beta_2\hat{\mathbf{h}}$.
19:     $\boldsymbol{\mu} \leftarrow \boldsymbol{\mu} - \alpha\mathbf{m}/(\mathbf{s} + \tilde{\delta} + \gamma)$.
20: until stopping criterion is met
```



$\mathbf{w}^{(i)} \sim q(\mathbf{w})$
$\mathcal{M}$
$\mathcal{M}_{local}$
$\mathcal{M}_{local}$
$\mathcal{M}_{local}$
$\mathcal{M}_{local}$
$\mathbf{w}^{(1)}$ $\mathbf{w}^{(2)}$
$\mathbf{w}^{(3)}$ $\mathbf{w}^{(4)}$
$\mathbf{w}^{(5)}$ $\mathbf{w}^{(6)}$
$\mathbf{w}^{(7)}$ $\mathbf{w}^{(8)}$
$\hat{\mathbf{g}}$ $\hat{\mathbf{h}}$

Learning rate	α
Momentum rate	β_1
Exp. moving average rate	β_2
Prior precision	δ
External damping factor	γ
Tempering parameter	τ
# MC samples for training	K
Data augmentation factor	ρ

Figure 2: A pseudo-code for our distributed VOGN algorithm is shown in Algorithm 1, and the distributed scheme is shown in the right figure. The computation in line 10 requires an extra calculation (see Appendix B), making VOGN slower than Adam. The bottom table gives a list of algorithmic hyperparameters needed for VOGN.

Momentum and initialisation: It is well known that both momentum and good initialisation can improve the speed of convergence for SG methods in deep learning [51]. Since VOGN is similar to Adam, we can implement momentum in a similar way. This is shown in step 17 of Algorithm 1, where β_1 is the momentum rate. We initialise the mean $\boldsymbol{\mu}$ in the same way the weights are initialised in Adam (we use `init.xavier_normal` in PyTorch [11]). For the momentum term $\mathbf{m}$, we use the same initialisation as Adam (initialised to 0). VOGN requires an additional initialisation for the variance $\boldsymbol{\sigma}^2$. For this, we first run a forward pass through the first minibatch, calculate the average of the squared gradients and initialise the scale $\mathbf{s}_0$ with it (see step 1 in Algorithm 1). This implies that the variance is initialised to $\boldsymbol{\sigma}_0^2 = \tau/(N(\mathbf{s}_0 + \tilde{\delta}))$. For the tempering parameter τ, we use a schedule where it is increased from a small value (e.g., 0.1) to 1. With these initialisation protocols, VOGN is able to mimic the convergence behaviour of Adam in the beginning.

Learning rate scheduling: A common approach to quickly achieve high validation accuracies is to use a specific learning rate schedule [13]. The learning rate (denoted by α in Algorithm 1) is regularly decayed by a factor (typically a factor of 10). The frequency and timings of this decay are usually pre-specified. In VOGN, we use the same schedule used for Adam, which works well.

Distributed training: We also employ distributed training for VOGN to perform large experiments quickly. We can parallelise computation both over data and Monte-Carlo (MC) samples. Data parallelism is useful to split up large minibatch sizes. This is followed by averaging over multiple MC samples and their losses on a single GPU. MC sample parallelism is useful when minibatch size is small, and we can copy the entire minibatch and process it on a single GPU. Algorithm 1 and Figure 2 illustrate our distributed scheme. We use a combination of these two parallelism techniques with different MC samples for different inputs. This theoretically reduces the variance during training (see Equation 5 in Kingma et al. [26]), but sometimes requires averaging over multiple MC samples to get a sufficiently low variance in the early iterations. Overall, we find that this type of distributed training is essential for fast training on large problems such as ImageNet.

Implementation of the Gauss-Newton update in VOGN: As discussed earlier, VOGN uses the Gauss-Newton approximation, which is fundamentally different from Adam. In this approximation, the gradients on individual data examples are first squared and then averaged afterwards (see step

This still seems like quite a lot of work?

12 in Algorithm 1 which implements the update for $\mathbf{s}_t$ shown in (4)). We need extra computation to get access to individual gradients, due to which, VOGN is slower Adam or SGD (e.g., in Fig. 1). However, this is not a theoretical limitation and this can be improved if a framework enables an easy computation of the individual gradients. Details of our implementation are described in Appendix B. This implementation is much more efficient than a naive one where gradients over examples are stored and the sum over the square is computed sequentially. Our implementation usually brings the running time of VOGN to within 2-5 times of the time that Adam takes.

Tuning VOGN: Currently, there is no common recipe for tuning the algorithmic hyperparameters for VI, especially for large-scale tasks like ImageNet classification. One key idea we use in our experiments is to start with Adam hyperparameters and then make sure that VOGN training closely follows an Adam-like trajectory in the beginning of training. To achieve this, we divide the tuning into an *optimisation part* and a *regularisation part*. In the *optimisation part*, we first tune the hyperparameters of a deterministic version of VOGN, called the online Gauss-Newton (OGN) method. This method, described in Appendix C, is more stable than VOGN since it does not require MC sampling, and can be used as a stepping stone when moving from Adam/SGD to VOGN. After reaching a competitive performance to Adam/SGD by OGN, we move to the *regularisation part*, where we tune the prior precision δ, the tempering parameter τ, and the number of MC samples K for VOGN. We initialise our search by setting the prior precision δ using the L2-regularisation parameter used for OGN, as well as the dataset size N. Another technique is to warm-up the parameter τ towards $\tau = 1$ (also see the "momentum and initialisation" part). Setting τ to smaller values usually stabilises the training, and increasing it slowly also helps during tuning. We also add an *external damping factor* $\gamma > 0$ to the moving average $\mathbf{s}_t$. This increases the lower bound of the eigenvalues of the diagonal covariance $\mathbf{\Sigma}_t$ and prevents the noise and the step size from becoming too large. We find that a mix of these techniques works well for the problems we considered.

4 Experiments

In this section, we present experiments on fitting several deep networks on CIFAR-10 and ImageNet. Our experiments demonstrate practical training using VOGN on these benchmarks and show performance that is competitive with Adam and SGD. We also assess the quality of the posterior approximation, finding that benefits of Bayesian principles are preserved.

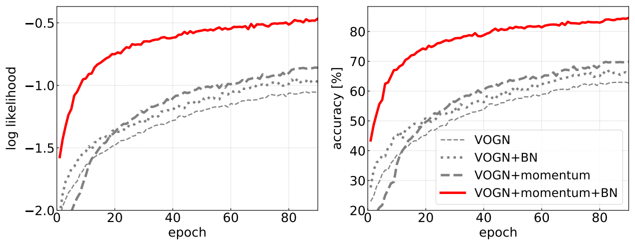
CIFAR-10 [28] contains 10 classes with 50,000 images for training and 10,000 images for validation. For ImageNet, we train with 1.28 million training examples and validate on 50,000 examples, classifying between 1,000 classes. We used a large minibatch size $M = 4,096$ and parallelise them across 128 GPUs (NVIDIA Tesla P100). We compare the following methods on CIFAR-10: Adam, MC-dropout [9]. For ImageNet, we also compare to SGD, K-FAC, and Noisy K-FAC. We do not consider Noisy K-FAC for other comparisons since tuning is difficult. We compare 3 architectures: LeNet-5, AlexNet, ResNet-18. We only compare to Bayes by Backprop (BBB) [4] for CIFAR-10 with LeNet-5 since it is very slow to converge for larger-scale experiments. We carefully set the hyperparameters of all methods, following the best practice of large distributed training [13] as the initial point of our hyperparameter tuning. The full set of hyperparameters is in Appendix D.

4.1 Performance on CIFAR-10 and ImageNet

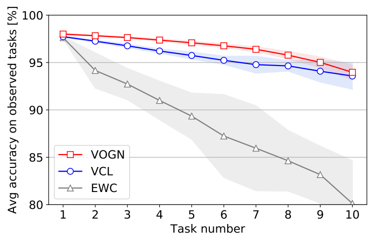
We start by showing the effectiveness of momentum and batch normalisation for boosting the performance of VOGN. Figure 3a shows that these methods significantly speed up convergence and performance (in terms of both accuracy and log likelihoods).

Figures 1 and 4 compare the convergence of VOGN to Adam (for all experiments), SGD (on ImageNet), and MC-dropout (on the rest). VOGN shows similar convergence and its performance is competitive with these methods. We also try BBB on LeNet-5, where it converges prohibitively slowly, performing very poorly. We are not able to successfully train other architectures using this approach. We found it far simpler to tune VOGN because we can borrow all the techniques used for Adam. Figure 4 also shows the importance of DA in improving performance.

Table 1 gives a final comparison of train/validation accuracies, negative log likelihoods, epochs required for convergence, and run-time per epoch. We can see that the accuracy, log likelihoods, and the number of epochs are comparable. VOGN is 2-5 times slower than Adam and SGD. This



(a) Effect of momentum and batch normalisation.



(b) Continual Learning

Figure 3: Figure (a) shows that momentum and batch normalisation improve the performance of VOGN. The results are for training ResNet-18 on CIFAR-10. Figure (b) shows comparison for a continual-learning task on the Permuted MNIST dataset. VOGN performs at least as well (average accuracy) as VCL over 10 tasks. We also find that, for each task, VOGN converges much faster, taking only 100 epochs per task as opposed to 800 epochs taken by VCL (plots not shown).

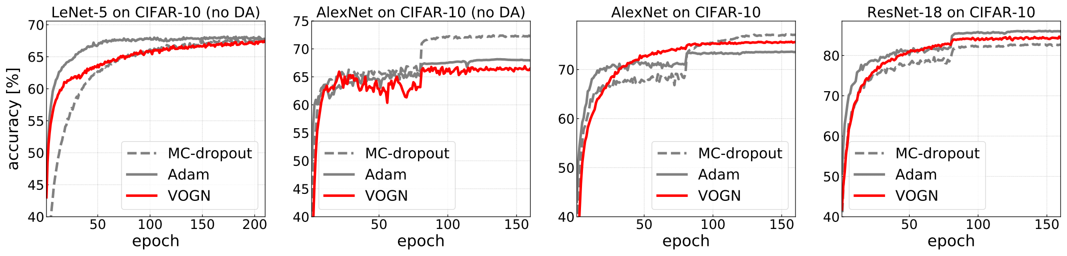


Figure 4: Validation accuracy for various architectures trained on CIFAR-10 (DA: Data Augmentation). VOGN's convergence and validation accuracies are comparable to Adam and MC-dropout.

is mainly due to the computation of individual gradients required in VOGN (see the discussion in Section 3). We clearly see that by using deep-learning techniques on VOGN, we can perform practical deep learning. This is not possible with methods such as BBB.

Due to the Bayesian nature of VOGN, there are some trade-offs to consider. Reducing the prior precision (δ in Algorithm 1) results in higher validation accuracy, but also larger train-test gap (more overfitting). This is shown in Appendix E for VOGN on ResNet-18 on ImageNet. As expected, when the prior precision is small, performance is similar to non-Bayesian methods. We also show the effect of changing the effective dataset size ρ in Appendix E: note that, since we are going to tune the prior variance δ anyway, it is sufficient to set ρ to its correct order of magnitude. Another trade-off concerns the number of Monte-Carlo (MC) samples, shown in Appendix F. Increasing the number of training MC samples (up to a limit) improves VOGN's convergence rate and stability, but also increases the computation. Increasing the number of MC samples during testing improves generalisation, as expected due to averaging.

Finally, a few comments on the performance of the other methods. Adam regularly overfits the training set in most settings, with large train-test differences in both validation accuracy and log likelihood. One exception is LeNet-5, which is most likely due to the small architecture which results in underfitting (this is consistent with the low validation accuracies obtained). In contrast to Adam, MC-dropout has small train-test gap, usually smaller than VOGN's. However, we will see in Section 4.2 that this is because of underfitting. Moreover, the performance of MC-dropout is highly sensitive to the dropout rate (see Appendix G for a comparison of different dropout rates). On ImageNet, Noisy K-FAC performs well too. It is slower than VOGN, but it takes fewer epochs. Overall, wall clock time is about the same as VOGN.

4.2 Quality of the Predictive Probabilities

In this section, we compare the quality of the predictive probabilities for various methods. For Bayesian methods, we compute these probabilities by averaging over the samples from the posterior approximations (see Appendix H for details). For non-Bayesian methods, these are obtained using the

Not clear if VOGN is better than MC-dropout IMO.

Dataset/ Architecture	Optimiser	Train/Validation Accuracy (%)	Validation NLL	Epochs	Time/ epoch (s)	ECE	AUROC
CIFAR-10/ LeNet-5 (no DA)	Adam	71.98 / **67.67**	**0.937**	210	6.96	**0.021**	0.794
	BBB	66.84 / 64.61	1.018	800	11.43[†]	0.045	0.784
	MC-dropout	68.41 / **67.65**	0.99	210	6.95	0.087	**0.797**
	VOGN	70.79 / **67.32**	**0.938**	210	18.33	0.046	**0.8**
CIFAR-10/ AlexNet (no DA)	Adam	100.0 / 67.94	2.83	161	3.12	0.262	0.793
	MC-dropout	97.56 / **72.20**	1.077	160	3.25	0.140	**0.818**
	VOGN	79.07 / 69.03	**0.93**	160	9.98	**0.024**	0.796
CIFAR-10/ AlexNet	Adam	97.92 / 73.59	1.480	161	3.08	0.262	0.793
	MC-dropout	80.65 / **77.04**	**0.667**	160	3.20	0.114	0.828
	VOGN	81.15 / 75.48	0.703	160	10.02	**0.016**	**0.832**
CIFAR-10/ ResNet-18	Adam	97.74 / **86.00**	0.55	160	11.97	0.082	**0.877**
	MC-dropout	88.23 / 82.85	0.51	161	12.51	0.166	0.768
	VOGN	91.62 / 84.27	**0.477**	161	53.14	**0.040**	**0.876**
ImageNet/ ResNet-18	SGD	82.63 / **67.79**	**1.38**	90	44.13	0.067	0.856
	Adam	80.96 / 66.39	1.44	90	44.40	0.064	0.855
	MC-dropout	72.96 / 65.64	1.43	90	45.86	**0.012**	0.856
	OGN	85.33 / 65.76	1.60	90	63.13	0.128	0.854
	VOGN	73.87 / **67.38**	**1.37**	90	76.04	0.029	0.854
	K-FAC	83.73 / 66.58	1.493	60	133.69	0.158	0.842
	Noisy K-FAC	72.28 / 66.44	1.44	60	179.27	0.080	0.852

Table 1: Performance comparisons on different dataset/architecture combinations. Out of the 15 metrics (NLL, ECE, and AUROC on 5 dataset/architecture combinations), VOGN performs the best or tied best on 10 ,and is second-best on the other 5. Here DA means 'Data Augmentation', NLL refers to 'Negative Log Likelihood' (lower is better), ECE refers to 'Expected Calibration Error' (lower is better), AUROC refers to 'Area Under ROC curve' (higher is better). BBB is the Bayes By Backprop method. For ImageNet, the reported accuracy and negative log likelihood are the median value from the final 5 epochs. All hyperparameter settings are in Appendix D. See Table 3 for standard deviations. [†] BBB is not parallelised (other methods have 4 processes), with 1 MC sample used for the convolutional layers (VOGN uses 6 samples per process).

point estimate of the weights. We compare the probabilities using the following metrics: validation negative log-likelihood (NLL), area under ROC (AUROC) and expected calibration curves (ECE) [40, 15]. For the first and third metric, a lower number is better, while for the second, a higher number is better. See Appendix H for an explanation of these metrics. Results are summarised in Table 1. VOGN's uncertainty performance is more consistent and marginally better than the other methods, as expected from a more principled Bayesian method. Out of the 15 metrics (NLL, ECE and AUROC on 5 dataset/architecture combinations), VOGN performs the best or tied best on 10, and is second-best on the other 5. In contrast, both MC-dropout's and Adam's performance varies significantly, sometimes performing poorly, sometimes performing decently. MC-dropout is best on 4, and Adam is best on 1 (on LeNet-5; as argued earlier, the small architecture may result in underfitting). We also show calibration curves [7] in Figures 1 and 14. Adam is consistently over-confident, with its calibration curve below the diagonal. Conversely, MC-dropout is usually under-confident. On ImageNet, MC-dropout performs well on ECE (all methods are very similar on AUROC), but this required an excessively tuned dropout rate (see Appendix G).

We also compare performance on out-of-distribution datasets. When testing on datasets that are different from the training datasets, predictions should be more uncertain. We use experimental protocol from the literature [16, 31, 8, 32] to compare VOGN, Adam and MC-dropout on CIFAR-10. We also borrow metrics from other works [16, 30], showing predictive entropy histograms and also reporting AUROC and FPR at 95% TPR. See Appendix I for further details on the datasets and metrics. Ideally, we want predictive entropy to be high on out-of-distribution data and low on in-distribution data. Our results are summarised in Figure 5 and Appendix I. On ResNet-18 and AlexNet, VOGN's predictive entropy histograms show the desired behaviour: a spread of entropies for the in-distribution data, and high entropies for out-of-distribution data. Adam has many predictive entropies at zero, MC-dropout seems to perform at least as well as VOGN

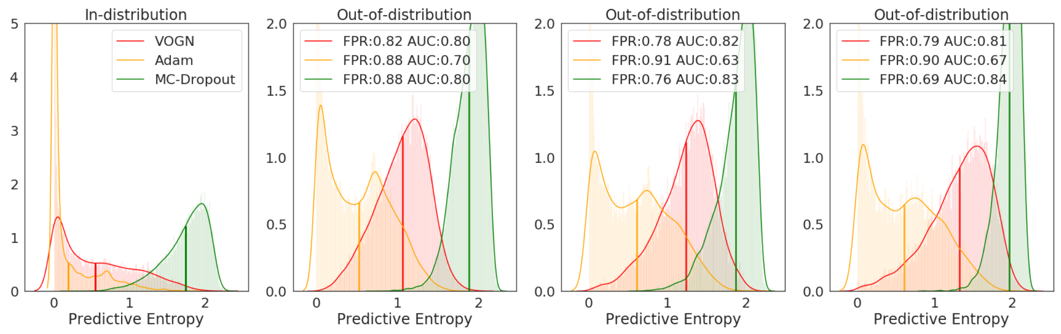


Figure 5: Histograms of predictive entropy for out-of-distribution tests for ResNet-18 trained on CIFAR-10. Going from left to right, the inputs are: the in-distribution dataset (CIFAR-10), followed by out-of-distribution data: SVHN, LSUN (crop), LSUN (resize). Also shown are the FPR at 95% TPR metric (lower is better) and the AUROC metric (higher is better), averaged over 3 runs. We clearly see that VOGN's predictive entropy is generally low for in-distribution and high for out-of-distribution data, but this is not the case for other methods. Solid vertical lines indicate the mean predictive entropy. The standard deviations are small and therefore not reported.

indicating Adam tends to classify out-of-distribution data too confidently. Conversely, MC-dropout's predictive entropies are generally high (particularly in-distribution), indicating MC-dropout has too much noise. On LeNet-5, we observe the same result as before: Adam and MC-dropout both perform well. The metrics (AUROC and FPR at 95% TPR) do not provide a clear story across architectures.

4.2.1 Performance on a Continual-learning task

The goal of continual learning is to avoid forgetting of old tasks while sequentially observing new tasks. The past tasks are never visited again, making it difficult to remember them. The field of continual learning has recently grown, with many approaches proposed to tackle this problem [27, 33, 43, 48, 50]. Most approaches consider a simple setting where the tasks (such as classifying a subset of classes) arrive sequentially, and all the data from that task is available. We consider the same setup in our experiments.

We compare to Elastic Weight Consolidation (EWC) [27] and a VI-based approach called Variational Continual Learning (VCL) [43]. VCL employs BBB for each task, and we expect to boost its performance by replacing BBB by VOGN. Figure 3b shows results on a common benchmark called Permuted MNIST. We use the same experimental setup as in Swaroop et al. [52]. In Permuted MNIST, each task consists of the entire MNIST dataset (10-way classification) with a different fixed random permutation applied to the input images' pixels. We run each method 20 times, with different random seeds for both the benchmark's permutations and model training. See Appendix D.2 for hyperparameter settings and further details. We see that VOGN performs at least as well as VCL, and far better than a popular approach called EWC [27]. Additionally, as found in the batch learning setting, VOGN is much quicker than BBB: we run VOGN for only 100 epochs per task, whereas VCL requires 800 epochs per task to achieve best results [52].

5 Conclusions

We successfully train deep networks with a natural-gradient variational inference method, VOGN, on a variety of architectures and datasets, even scaling up to ImageNet. This is made possible due to the similarity of VOGN to Adam, enabling us to boost performance by borrowing deep-learning techniques. Our accuracies and convergence rates are comparable to SGD and Adam. Unlike them, however, VOGN retains the benefits of Bayesian principles, with well-calibrated uncertainty and good performance on out-of-distribution data. Better uncertainty estimates open up a whole range of potential future experiments, for example, small data experiments, active learning, adversarial experiments, and sequential decision making. Our results on a continual-learning task confirm this. Another potential avenue for research is to consider structured covariance approximations.

Acknowledgements

We would like to thank Hikaru Nakata (Tokyo Institute of Technology) and Ikuro Sato (Denso IT Laboratory, Inc.) for their help on the PyTorch implementation. We are also thankful for the RAIDEN computing system and its support team at the RIKEN Center for AI Project which we used extensively for our experiments. This research used computational resources of the HPCI system provided by Tokyo Institute of Technology (TSUBAME3.0) through the HPCI System Research Project (Project ID:hp190122). K. O. is a Research Fellow of JSPS and is supported by JSPS KAKENHI Grant Number JP19J13477.

References

[1] James R Anderson and Carsten Peterson. A mean field theory learning algorithm for neural networks. *Complex Systems*, 1:995–1019, 1987.

[2] David Barber and Christopher M Bishop. Ensemble learning in Bayesian neural networks. *Generalization in Neural Networks and Machine Learning*, 168:215–238, 1998.

[3] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006. ISBN 0387310738.

[4] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural networks. In *International Conference on Machine Learning*, pages 1613–1622, 2015.

[5] Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *arXiv preprint arXiv:1606.04838*, 2016.

[6] John Bradshaw, Alexander G de G Matthews, and Zoubin Ghahramani. Adversarial examples, uncertainty, and transfer testing robustness in Gaussian process hybrid deep networks. *arXiv preprint arXiv:1707.02476*, 2017.

[7] Morris H. DeGroot and Stephen E. Fienberg. The comparison and evaluation of forecasters. *The Statistician: Journal of the Institute of Statisticians*, 32:12–22, 1983.

[8] Terrance DeVries and Graham W. Taylor. Learning confidence for out-of-distribution detection in neural networks. *arXiv preprint arXiv:1802.04865*, 2018.

[9] Yarin Gal and Zoubin Ghahramani. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In *International Conference on Machine Learning*, pages 1050–1059, 2016.

[10] S. Ghosal and A. Van der Vaart. *Fundamentals of nonparametric Bayesian inference*, volume 44. Cambridge University Press, 2017.

[11] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[12] Ian Goodfellow. Efficient Per-Example Gradient Computations. *ArXiv e-prints*, October 2015.

[13] Priya Goyal, Piotr Dollár, Ross B. Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch SGD: training imagenet in 1 hour. *CoRR*, abs/1706.02677, 2017.

[14] Alex Graves. Practical variational inference for neural networks. In *Advances in Neural Information Processing Systems*, pages 2348–2356, 2011.

[15] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1321–1330. JMLR. org, 2017.

[16] Dan Hendrycks and Kevin Gimpel. A baseline for detecting misclassified and out-of-distribution examples in neural networks. In *International Conference on Learning Representations*, 2017.

[17] Geoffrey Hinton, Li Deng, Dong Yu, George Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Brian Kingsbury, et al. Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal processing magazine*, 29, 2012.

[18] Geoffrey E Hinton and Drew Van Camp. Keeping the neural networks simple by minimizing the description length of the weights. In *Annual Conference on Computational Learning Theory*, pages 5–13, 1993.

[19] Jennifer A Hoeting, David Madigan, Adrian E Raftery, and Chris T Volinsky. Bayesian model averaging: a tutorial. *Statistical science*, pages 382–401, 1999.

[20] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015. URL `http://arxiv.org/abs/1502.03167`.

[21] Mohammad Khan. *Variational learning for latent Gaussian model of discrete data*. PhD thesis, University of British Columbia, 2012.

[22] Mohammad Emtiyaz Khan and Wu Lin. Conjugate-computation variational inference: converting variational inference in non-conjugate models to inferences in conjugate models. In *International Conference on Artificial Intelligence and Statistics*, pages 878–887, 2017.

[23] Mohammad Emtiyaz Khan and Didrik Nielsen. Fast yet simple natural-gradient descent for variational inference in complex models. In *2018 International Symposium on Information Theory and Its Applications (ISITA)*, pages 31–35. IEEE, 2018.

[24] Mohammad Emtiyaz Khan, Didrik Nielsen, Voot Tangkaratt, Wu Lin, Yarin Gal, and Akash Srivastava. Fast and scalable Bayesian deep learning by weight-perturbation in Adam. In *International Conference on Machine Learning*, pages 2616–2625, 2018.

[25] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

[26] Diederik P Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick. In *Advances in Neural Information Processing Systems*, pages 2575–2583, 2015.

[27] 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.

[28] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[29] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[30] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Advances in Neural Information Processing Systems 30*, pages 6402–6413. Curran Associates, Inc., 2017.

[31] Kimin Lee, Honglak Lee, Kibok Lee, and Jinwoo Shin. Training confidence-calibrated classifiers for detecting out-of-distribution samples. In *International Conference on Learning Representations*, 2018.

[32] Shiyu Liang, Yixuan Li, and R. Srikant. Enhancing the reliability of out-of-distribution image detection in neural networks. In *International Conference on Learning Representations*, 2018.

[33] David Lopez-Paz and Marc Aurelio Ranzato. Gradient episodic memory for continual learning. In *NIPS*, 2017.

[34] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

[35] David Mackay. *Bayesian Methods for Adaptive Models*. PhD thesis, California Institute of Technology, 1991.

[36] David JC MacKay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

[37] Wesley Maddox, Timur Garipov, Pavel Izmailov, Dmitry Vetrov, and Andrew Gordon Wilson. A simple baseline for Bayesian uncertainty in deep learning. *arXiv preprint arXiv:1902.02476*, 2019.

[38] Stephan Mandt, Matthew D Hoffman, and David M Blei. Stochastic gradient descent as approximate Bayesian inference. *Journal of Machine Learning Research*, 18:1–35, 2017.

[39] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[40] Mahdi Pakdaman Naeini, Gregory F. Cooper, and Milos Hauskrecht. Obtaining well calibrated probabilities using Bayesian binning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI'15, pages 2901–2907. AAAI Press, 2015.

[41] Redford M Neal. *Bayesian learning for neural networks*. PhD thesis, University of Toronto, 1995.

[42] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*, 2011.

[43] Cuong V Nguyen, Yingzhen Li, Thang D Bui, and Richard E Turner. Variational continual learning. *arXiv preprint arXiv:1710.10628*, 2017.

[44] Kazuki Osawa, Yohei Tsuji, Yuichiro Ueno, Akira Naruse, Rio Yokota, and Satoshi Matsuoka. Second-order optimization method for large mini-batch: Training resnet-50 on imagenet in 35 epochs. *CoRR*, abs/1811.12019, 2018.

[45] Carlos Riquelme, George Tucker, and Jasper Snoek. Deep Bayesian bandits showdown: An empirical comparison of Bayesian deep networks for Thompson sampling. *arXiv preprint arXiv:1802.09127*, 2018.

[46] Hippolyt Ritter, Aleksandar Botev, and David Barber. A scalable Laplace approximation for neural networks. In *International Conference on Learning Representations*, 2018.

[47] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[48] Andrei A. Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

[49] Lawrence K Saul, Tommi Jaakkola, and Michael I Jordan. Mean field theory for sigmoid belief networks. *Journal of Artificial Intelligence Research*, 4:61–76, 1996.

[50] Jonathan Schwarz, Jelena Luketina, Wojciech M. Czarnecki, Agnieszka Grabska-Barwinska, Yee Whye Teh, Razvan Pascanu, and Raia Hadsell. Progress & compress: A scalable framework for continual learning. In *International Conference on Machine Learning*, 2018.

[51] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147, 2013.

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

[53] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-RMSprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning 4*, 2012.

[54] V. G. Vovk. Aggregating strategies. In *Proceedings of the Third Annual Workshop on Computational Learning Theory*, COLT '90, pages 371–386, San Francisco, CA, USA, 1990. Morgan Kaufmann Publishers Inc. ISBN 1-55860-146-5.

[55] Fisher Yu, Yinda Zhang, Shuran Song, Ari Seff, and Jianxiong Xiao. LSUN: construction of a large-scale image dataset using deep learning with humans in the loop. *CoRR*, abs/1506.03365, 2015.

[56] Guodong Zhang, Shengyang Sun, David K. Duvenaud, and Roger B. Grosse. Noisy natural gradient as variational inference. *arXiv preprint arXiv:1712.02390*, 2018.

A Noisy K-FAC algorithm

Noisy K-FAC [56] attempts to approximate the structure of the full covariance matrix, and therefore the updates are a bit more involved than VOGN (see Equation 4). Assuming a fully-connected layer, we denote the weight matrix of layer by $\mathbf{W}$. The Noisy K-FAC method estimates the parameters of a matrix-variate Gaussian distribution $q_t(\mathbf{W}) = \mathcal{MN}(\mathbf{W}|\mathbf{M}_t, \mathbf{\Sigma}_{2,t} \otimes \mathbf{\Sigma}_{1,t})$ by using the following updates:

$$\mathbf{M}_{t+1} \leftarrow \mathbf{M}_t - \alpha \left[\mathbf{A}_{t+1}^{\gamma}\right]^{-1} \left(\nabla_W E\left[\ell(y_i, f_W(\mathbf{x}_i))\right] + \tilde{\delta}\mathbf{W}_t\right) \left[\mathbf{S}_{t+1}^{\gamma}\right]^{-1}, \tag{5}$$

$$\mathbf{A}_{t+1} \leftarrow (1-\tilde{\beta}_t)\mathbf{A}_t + \tilde{\beta}_t E\left[\mathbf{a}_t\mathbf{a}_t^{\top}\right], \quad \mathbf{S}_{t+1} \leftarrow (1-\tilde{\beta}_t)\mathbf{S}_t + \tilde{\beta}_t E\left[\mathbf{g}_t\mathbf{g}_t^{\top}\right], \tag{6}$$

where $\mathbf{W}_t \sim q_t(\mathbf{W})$, $\mathbf{g}_t := \nabla_s \ell(y_i, f_W(\mathbf{x}_i))$ with $s = \mathbf{s}_t := \mathbf{W}_t^{\top}\mathbf{a}_t$, $\mathbf{a}_t$ is the input vector (the activation of the previous layer), $E[\cdot]$ is the average over the minibatch. $\tilde{\beta} := \beta\tau/N$, and $\gamma := \tilde{\gamma} + \gamma_{ex}$ with some *external* damping factor γ_{ex}. The covariance parameters are set to $\mathbf{\Sigma}_{2,t}^{-1} := \tau\mathbf{A}_t^{\gamma}/N$ and $\mathbf{\Sigma}_{1,t}^{-1} := \mathbf{S}_t^{\gamma}$, where $\mathbf{A}_t^{\gamma} := \mathbf{A}_t + \pi_t\sqrt{\gamma}\mathbf{I}$ and $\mathbf{S}_t^{\gamma} := \mathbf{S}_t + \frac{1}{\pi_t}\sqrt{\gamma}\mathbf{I}$. $\pi_t^2 (\pi_t > 0)$ is the average eigenvalue of $\mathbf{A}_t$ divided by that of $\mathbf{S}_t$. Similarly to the VOGN update in Equation 4, the gradients are scaled by matrices $\mathbf{A}_t$ and $\mathbf{S}_t$, which are related to the precision matrix of the approximation.

B Details on fast implementation of the Gauss-Newton approximation

Current codebases are only optimised to directly return the average of gradients over the minibatch. In order to efficiently compute the Gauss-Newton (GN) approximation, we modify the backward-pass to efficiently calculate the gradient per example in the minibatch, and extend the solution in Goodfellow [12] to both convolutional and batch normalisation layers.

B.1 Convolutional layer

Consider a convolutional layer with a weight matrix $\mathbf{W} \in \mathbb{R}^{C_{out} \times C_{in}k^2}$ (ignore bias for simplicity) and an input tensor $\mathbf{A} \in \mathbb{R}^{C_{in} \times H_{in} \times W_{in}}$, where C_{out}, C_{in} are the number of output, input channels, respectively, H_{in}, W_{in} are the spatial dimensions, and k is the kernel size. For any stride and padding, by applying `torch.nn.functional.unfold` function in PyTorch[5], we get the extended matrix $\mathbf{M}_A \in \mathbb{R}^{C_{in}k^2 \times H_{out}W_{out}}$ so that the output tensor $\mathbf{S}$ is calculated by a matrix multiplication:

$$\mathbf{M}_A \leftarrow \mathrm{unfold}(\mathbf{A}) \in \mathbb{R}^{C_{in}k^2 \times H_{out}W_{out}}, \tag{7}$$

$$\mathbf{M}_S \leftarrow \mathbf{W}\mathbf{M}_A \in \mathbb{R}^{C_{out} \times H_{out}W_{out}}, \tag{8}$$

$$\mathbf{S} \leftarrow \mathrm{reshape}(\mathbf{M}_S) \in \mathbb{R}^{C_{out} \times H_{out} \times W_{out}}, \tag{9}$$

where H_{out}, W_{out} are the spatial dimensions of the output feature map. Using the matrix $\mathbf{M}_A$, we can also get the gradient per example by a matrix multiplication:

$$\nabla_{M_S}\ell(y_i, f_W(\mathbf{x}_i)) \leftarrow \mathrm{reshape}(\nabla_S \ell(y_i, f_W(\mathbf{x}_i))) \in \mathbb{R}^{C_{out} \times H_{out}W_{out}}, \tag{10}$$

$$\nabla_W \ell(y_i, f_W(\mathbf{x}_i)) \leftarrow \nabla_{M_S}\ell(y_i, f_W(\mathbf{x}_i))\mathbf{M}_A^{\top} \in \mathbb{R}^{C_{out} \times C_{in}k^2}. \tag{11}$$

Note that in PyTorch, we can access to the inputs $\mathbf{A}$ and the gradient $\nabla_S \ell(y_i, f_W(\mathbf{x}_i))$ per example in the computational graph during a forward-pass and a backward-pass, respectively, by using the `Function Hooks` [6]. Hence, to get the gradient $\nabla_W \ell(y_i, f_W(\mathbf{x}_i))$ per example, we only need to perform (7), (10), and (11) after the backward-pass for this layer.

B.2 Batch normalisation layer

Consider a batch normalisation layer follows a fully-connected layer, which activation is $\mathbf{a} \in \mathbb{R}^d$, with the scale parameter $\boldsymbol{\gamma} \in \mathbb{R}^d$ and the shift parameter $\boldsymbol{\beta} \in \mathbb{R}^d$, we get the output of this layer

[5] `https://pytorch.org/docs/stable/nn.functional.html#torch.nn.functional.unfold`

[6] `https://pytorch.org/tutorials/beginner/former_torchies/nnft_tutorial.html#forward-and-backward-function-hooks`

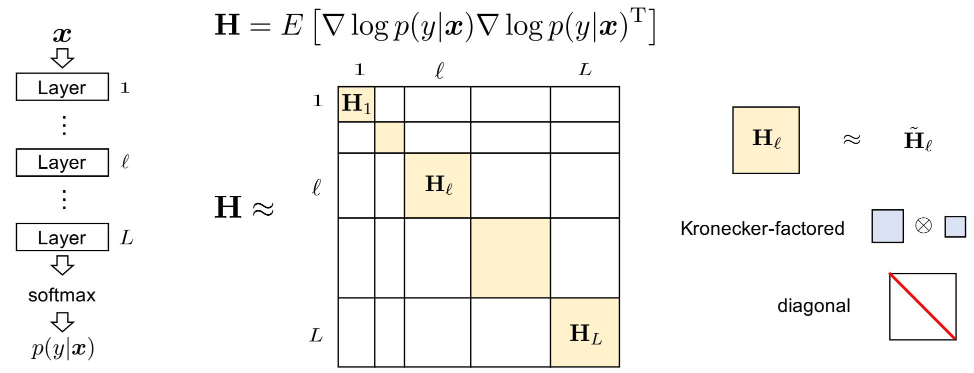


Figure 6: Layer-wise block-diagonal Gauss-Newton approximation

$\mathbf{s} \in \mathbb{R}^d$ by,

$$\boldsymbol{\mu} \leftarrow E\left[\mathbf{a}\right] \in \mathbb{R}^d\,, \tag{12}$$

$$\boldsymbol{\sigma}^2 \leftarrow E\left[(\mathbf{a} - \boldsymbol{\mu})^2\right] \in \mathbb{R}^d\,, \tag{13}$$

$$\hat{\mathbf{a}} \leftarrow \frac{\mathbf{a} - \boldsymbol{\mu}}{\sqrt{\boldsymbol{\sigma}^2}} \in \mathbb{R}^d\,, \tag{14}$$

$$\mathbf{s} \leftarrow \boldsymbol{\gamma}\hat{\mathbf{a}} + \boldsymbol{\beta} \in \mathbb{R}^d\,, \tag{15}$$

where $E\left[\cdot\right]$ is the average over the minibatch and $\hat{\mathbf{a}}$ is the normalised input. We can find the gradient with respect to parameters $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ per example by,

$$\nabla_\gamma \ell(y_i, f_W(\mathbf{x}_i) \leftarrow \nabla_s \ell(y_i, f_W(\mathbf{x}_i) \circ \hat{\mathbf{a}}\,, \tag{16}$$

$$\nabla_\beta \ell(y_i, f_W(\mathbf{x}_i) \leftarrow \nabla_s \ell(y_i, f_W(\mathbf{x}_i)\,. \tag{17}$$

We can obtain the input $\mathbf{a}$ and the gradient $\nabla_s \ell(y_i, f_W(\mathbf{x}_i)$ per example from the computational graph in PyTorch in the same way as a convolutional layer.

B.3 Layer-wise block-diagonal Gauss-Newton approximation

Despite using the method above, it is still intractable to compute the Gauss-Newton matrix (and its inverse) with respect to the weights of large-scale deep neural networks. We therefore apply two further approximations (Figure 6). First, we view the Gauss-Newton matrix as a layer-wise block-diagonal matrix. This corresponds to ignoring the correlation between the weights of different layers. Hence for a network with L layers, there are L diagonal blocks, and $\mathbf{H}_\ell$ is the diagonal block corresponding to the ℓ-th layer ($\ell = 1, \ldots, L$). Second, we approximate each diagonal block $\mathbf{H}_\ell$ with $\tilde{\mathbf{H}}_\ell$, which is either a Kronecker-factored or diagonal matrix. Using a Kronecker-factored matrix as $\tilde{\mathbf{H}}_\ell$ corresponds to K-FAC; a diagonal matrix corresponds to a mean-field approximation in that layer. By applying these two approximations, the update rule of the Gauss-Newton method can be written in a layer-wise fashion:

$$\boldsymbol{W}_{\ell,t+1} = \boldsymbol{W}_{\ell,t} - \alpha_t \tilde{\mathbf{H}}_\ell(\boldsymbol{\theta}_t)^{-1} \mathbf{g}_\ell(\boldsymbol{\theta}_t) \;\; (\ell = 1, \ldots, L)\,, \tag{18}$$

where $\boldsymbol{W}_\ell$ is the weights in ℓ-th layer, and

$$\boldsymbol{\theta} = \begin{pmatrix} \mathrm{vec}(\boldsymbol{W}_1)^{\mathrm{T}} & \cdots & \mathrm{vec}(\boldsymbol{W}_\ell)^{\mathrm{T}} & \cdots & \mathrm{vec}(\boldsymbol{W}_{\mathrm{L}})^{\mathrm{T}} \end{pmatrix}^{\mathrm{T}}\,. \tag{19}$$

Since the cost of computing $\tilde{\mathbf{H}}_\ell^{-1}$ is much cheaper compared to that of computing $\mathbf{H}^{-1}$, our approximations make Gauss-Newton much more practical in deep learning.

In the distributed setting (see Figure 2), each parallel process (corresponding to 1 GPU) calculates the GN matrix for its local minibatch. Then, one GPU adds them together and calculates the inverse. This inversion step can also be parallelised after making the block-diagonal approximation to the GN matrix. After inverting the GN matrix, the standard deviation $\boldsymbol{\sigma}$ is updated (line 9 in Algorithm 1), and sent to each parallel process, allowing each process to draw independently from the posterior.

In the Noisy K-FAC case, a similar distributed scheme is used, except each parallel process now has both matrices $\mathbf{S}$ and $\mathbf{A}$ (see Appendix A). When using K-FAC approximations to the Gauss-Newton blocks for other layers, Osawa et al. [44] empirically showed that the BatchNorm layer can be approximated with a diagonal matrix without loss of accuracy, and we find the same. We therefore use diagonal $\tilde{\mathbf{H}}_\ell$ with K-FAC and Noisy K-FAC in BatchNorm layers (see Table 2). For further details on how to efficiently parallelise K-FAC in the distributed setting, please see Osawa et al. [44].

optimiser	convolution	fully-connected	Batch Normalisation
OGN	diagonal	diagonal	diagonal
VOGN	diagonal	diagonal	diagonal
K-FAC	Kronecker-factored	Kronecker-factored	diagonal
Noisy K-FAC	Kronecker-factored	Kronecker-factored	diagonal

Table 2: The approximation used for each layer type's diagonal block $\tilde{\mathbf{H}}_\ell$ for the different optimisers tested this paper.

C OGN: A deterministic version of VOGN

To easily apply the tricks and techniques of deep-learning methods, we recommend to first test them on a deterministic version of VOGN, which we call the online Gauss-Newton (OGN) method. In this method, we approximate the gradients at the mean of the Gaussian, rather than using MC samples[7]. This results in an update without any sampling, as shown below (we have replaced $\boldsymbol{\mu}_t$ by $\mathbf{w}_t$ since there is no distinction between them):

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \alpha_t \frac{\hat{\mathbf{g}}(\mathbf{w}_t) + \tilde{\delta}\mathbf{w}_t}{\mathbf{s}_{t+1} + \tilde{\delta}}, \quad \mathbf{s}_{t+1} \leftarrow (1 - \tau\beta_t)\mathbf{s}_t + \beta_t \frac{1}{M} \sum_{i \in \mathcal{M}_t} \left(\mathbf{g}_i(\mathbf{w}_t)\right)^2. \tag{20}$$

At each iteration, we still get a Gaussian $\mathcal{N}(\mathbf{w}|\mathbf{w}_t, \hat{\boldsymbol{\Sigma}}_t)$ with $\hat{\boldsymbol{\Sigma}}_t := \text{diag}(1/(N(\mathbf{s}_t + \tilde{\delta})))$. It is easy to see that, like SG methods, this algorithm converges to a local minima of the loss function, thereby obtaining a Laplace approximation instead of a variational approximation. The advantage of OGN is that this can be used as a stepping stone, when switching from Adam to VOGN. Since it does not involve sampling, the tricks and techniques applied to Adam are easier to apply to OGN than VOGN. However, due to the lack of averaging over samples, this algorithm is less effective to preserve the benefits of Bayesian principles, and gives slightly worse uncertainty estimates.

D Hyperparameter settings

Hyperparameters for all results shown in Table 1 are given in Table 4. The settings for distributed VI training are given in Table 5. Please see Goyal et al. [13] and Osawa et al. [44] for best practice on these hyperparameter values.

D.1 Bayes by Backprop for CIFAR-10/LeNet-5 training

We use hyperparameter settings and training procedure for Bayes by Backprop (BBB) [4] as suggested by Swaroop et al. [52]. This includes using the local reparameterisation trick, initialising means and variances at small values, using 10 MC samples per minibatch during training for linear layers (1 MC sample per minibatch for convolutional layers) and 100 MC samples per minibatch during testing for linear layers (10 MC samples per minibatch for convolutional layers). Note that BBB has twice as many parameters to optimise than Adam or SGD (means and variances for each weight in the deep neural network). The fewer MC samples per minibatch for convolutional layers speed up training time per epoch while empirically not reducing convergence rate.

[7]This gradient approximation used here is referred to as the *zeroth-order delta approximation* where $\mathbb{E}_q[\hat{\mathbf{g}}(\mathbf{w})] \approx \hat{\mathbf{g}}(\boldsymbol{\mu})$ (see Appendix A.6 in Khan [21] for details).

Dataset/ Architecture	Optimiser	Train Acc (%)	Train NLL	Validation Acc (%)	Validation NLL	ECE	AUROC	Epochs	Time/ epoch (s)
CIFAR-10/ LeNet-5 (no DA)	Adam	71.98 ± 0.117	0.733 ± 0.021	**67.67** ± **0.513**	**0.937** ± **0.012**	**0.021** ± **0.002**	0.794 ± 0.001	210	6.96
	BBB	66.84 ± 0.003	0.957 ± 0.006	64.61 ± 0.331	1.018 ± 0.006	0.045 ± 0.005	0.784 ± 0.003	800	11.43
	MC-dropout	68.41 ± 0.581	0.870 ± 0.101	**67.65** ± **1.317**	0.99 ± 0.026	0.087 ± 0.009	**0.797** ± **0.006**	210	6.95
	VOGN	70.79 ± 0.763	0.880 ± 0.02	**67.32** ± **1.310**	**0.938** ± **0.024**	0.046 ± 0.002	**0.8** ± **0.002**	210	18.33
CIFAR-10/ AlexNet (no DA)	Adam	100.0 ± 0	0.001 ± 0	67.94 ± 0.537	2.83 ± 0.02	0.262 ± 0.005	0.793 ± 0.001	161	3.12
	MC-dropout	97.56 ± 0.278	0.058 ± 0.014	**72.20** ± **0.177**	1.077 ± 0.012	0.140 ± 0.004	**0.818** ± **0.002**	160	3.25
	VOGN	79.07 ± 0.248	0.696 ± 0.020	69.03 ± 0.419	**0.931** ± **0.017**	**0.024** ± **0.010**	0.796 ± 0	160	9.98
CIFAR-10/ AlexNet	Adam	97.92 ± 0.140	0.057 ± 0.006	73.59 ± 0.296	1.480 ± 0.015	0.262 ± 0.005	0.793 ± 0.001	161	3.08
	MC-dropout	80.65 ± 0.615	0.47 ± 0.052	**77.04** ± **0.343**	**0.667** ± **0.012**	0.114 ± 0.002	0.828 ± 0.002	160	3.20
	VOGN	81.15 ± 0.259	0.511 ± 0.039	75.48 ± 0.478	0.703 ± 0.006	**0.016** ± **0.001**	**0.832** ± **0.002**	160	10.02
CIFAR-10/ ResNet-18	Adam	97.74 ± 0.140	0.059 ± 0.012	**86.00** ± **0.257**	0.55 ± 0.01	0.082 ± 0.002	**0.877** ± **0.001**	160	11.97
	MC-dropout	88.23 ± 0.243	0.317 ± 0.045	82.85 ± 0.208	0.51 ± 0	0.166 ± 0.025	0.768 ± 0.004	161	12.51
	VOGN	91.62 ± 0.07	0.263 ± 0.051	84.27 ± 0.195	**0.477** ± **0.006**	**0.040** ± **0.002**	**0.876** ± **0.002**	161	53.14
ImageNet/ ResNet-18	SGD	82.63 ± 0.058	0.675 ± 0.017	**67.79** ± **0.017**	**1.38** ± **0**	0.067	0.856	90	44.13
	Adam	80.96 ± 0.098	0.723 ± 0.015	66.39 ± 0.168	1.44 ± 0.01	0.064	0.855	90	44.40
	MC-dropout	72.96	1.12	65.64	1.43	**0.012**	0.856	90	45.86
	OGN	85.33 ± 0.057	0.526 ± 0.005	65.76 ± 0.115	1.60 ± 0.00	0.128 ± 0.004	0.8543 ± 0.001	90	63.13
	VOGN	73.87 ± 0.061	1.02 ± 0.01	**67.38** ± **0.263**	**1.37** ± **0.01**	0.0293 ± 0.001	0.8543 ± 0	90	76.04
	K-FAC	83.73 ± 0.058	0.571 ± 0.016	66.58 ± 0.176	1.493 ± 0.006	0.158 ± 0.005	0.842 ± 0.005	60	133.69
	Noisy K-FAC	72.28	1.075	66.44	1.44	0.080	0.852	60	179.27

Table 3: Comparing optimisers on different dataset/architecture combinations, means and standard deviations over three runs. DA: Data Augmentation, Acc.: Accuracy (higher is better), NLL: Negative Log Likelihood (lower is better), ECE: Expected Calibration Error (lower is better), AUROC: Area Under ROC curve (higher is better), BBB: Bayes By Backprop. For ImageNet results, the reported accuracy and negative log likelihood are the median value from the final 5 epochs.

Dataset/ Architecture	Optimiser	α_{init}	α	Epochs to decay α	β_1	β_2	Weight decay	L2 reg
CIFAR-10/ LeNet-5 (no DA)	Adam	-	1e-3	-	0.1	0.001	1e-2	-
	BBB	-	1e-3	-	0.1	0.001	-	-
	MC-dropout	-	1e-3	-	0.9	-	-	1e-4
	VOGN	-	1e-2	-	0.9	0.999	-	-
CIFAR-10/ AlexNet (no DA)	Adam	-	1e-3	[80, 120]	0.1	0.001	1e-4	-
	MC-dropout	-	1e-1	[80, 120]	0.9	-	-	1e-4
	VOGN	-	1e-4	[80, 120]	0.9	0.999	-	-
CIFAR-10/ AlexNet	Adam	-	1e-3	[80, 120]	0.1	0.001	1e-4	-
	MC-dropout	-	1e-1	[80, 120]	0.9	-	-	1e-4
	VOGN	-	1e-4	[80, 120]	0.9	0.999	-	-
CIFAR-10/ ResNet-18	Adam	-	1e-3	[80, 120]	0.1	0.001	5e-4	-
	MC-dropout	-	1e-1	[80, 120]	0.9	-	-	1e-4
	VOGN	-	1e-4	[80, 120]	0.9	0.999	-	-
ImageNet/ ResNet-18	SGD	1.25e-2	1.6	[30, 60, 80]	0.9	-	-	1e-4
	Adam	1.25e-5	1.6e-3	[30, 60, 80]	0.1	0.001	1e-4	-
	MC-dropout	1.25e-2	1.6	[30, 60, 80]	0.9	-	-	1e-4
	OGN	1.25e-5	1.6e-3	[30, 60, 80]	0.9	0.9	-	1e-5
	VOGN	1.25e-5	1.6e-3	[30, 60, 80]	0.9	0.999	-	-
	K-FAC	1.25e-5	1.6e-3	[15, 30, 45]	0.9	0.9	-	1e-4
	Noisy K-FAC	1.25e-5	1.6e-3	[15, 30, 45]	0.9	0.9	-	-

Table 4: Hyperparameters for all results in Table 1

Optimiser	Dataset/ Architecture	M	# GPUs	K	τ	ρ	N_{orig}	δ	$\tilde{\delta}$	γ
VOGN	CIFAR-10/ LeNet-5 (no DA)	128	4	6	$0.1 \to 1$	1	50,000	100	2e-4 $\to$ 2e-3	1e-3
	CIFAR-10/ AlexNet (no DA)	128	8	3	$0.05 \to 1$	1	50,000	0.5	5e-7 $\to$ 1e-5	1e-3
	CIFAR-10/ AlexNet	128	8	3	$0.5 \to 1$	10	50,000	0.5	5e-7 $\to$ 1e-5	1e-3
	CIFAR-10/ ResNet-18	256	8	5	1	10	50,000	50	1e-3	1e-3
	ImageNet/ ResNet-18	4096	128	1	1	5	1,281,167	133.3	2e-5	1e-4
Noisy K-FAC	ImageNet/ ResNet-18	4096	128	1	1	5	1,281,167	133.3	2e-5	1e-4

Table 5: Settings for distributed VI training

K: number of monte carlo samples duri

D.2 Continual learning experiment

Following the setup of Swaroop et al. [52], all models are run with two hidden layers, of 100 hidden units each, with ReLU activation functions. VCL is run with the same hyperparameter settings as in Swaroop et al. [52]. We perform a grid search over EWC's λ hyperparameter, finding that $\lambda = 100$ performs the best, exactly like in Nguyen et al. [43].

VOGN is run for 100 epochs per task. Parameters are initialised before training with the default PyTorch initialisation for linear layers. The initial precision is 1e6. A standard normal initial prior is used, just like in VCL. Between tasks, the mean and precision are initialised in the same way as for the first task. The learning rate $\alpha = 1e-3$, the batch size $M = 256$, $\beta_1 = 0$, $\beta_2 = 1e-3$, 10 MC samples are used during training and 100 for testing. We run each method 20 times, with different random seeds for both the benchmark's permutation and for model training.

E Effect of prior variance and dataset size reweighting factor

We show the effect of changing the prior variance (δ^{-1} in Algorithm 1) in Figures 8 and 9. We can see that increasing the prior variance improves validation performance (accuracy and log likelihood). However, increasing prior variance also always increases the train-test gap, without exceptions, when the other hyperparameters are held constant. As an example, training VOGN on ResNet-18 on ImageNet with a prior variance of $7.5e-4$ has train-test accuracy and log likelihood gaps of 2.29 and 0.12 respectively. When the prior variance is increased to $7.5e-3$, the respective train-test gaps increase to 6.38 and 0.34 (validation accuracy and validation log likelihood also increase, see Figure 8).

With increased prior variance, VOGN (and Noisy K-FAC) reach converged solutions more like their non-Bayesian counterparts, where overfitting is an issue. This is as expected from Bayesian principles.

Figure 10 shows the combined effect of the dataset reweighting factor ρ and prior variance. When ρ is set to a value in the correct order of magnitude, it does not affect performance so much: instead, we should tune δ. This is our methodology when dealing with ρ. Note that we set ρ for ImageNet to be smaller than that for CIFAR-10 because the data augmentation cropping step uses a higher portion of the initial image than in CIFAR-10: we crop images of size 224x224 from images of size 256x256.

F Effect of number of Monte Carlo samples on ImageNet

In the paper, we report results for training ResNet-18 on ImageNet using 128 GPUs, with 1 independent Monte-Carlo (MC) sample per process during training (`mc=128x1`), and 10 MC samples per validation image (`val_mc`= 10). We now show that increasing either of training or testing MC samples improves performance (validation accuracy and log likelihood) at the cost of increased computation time. See Figure 11.

Increasing the number of training MC samples per process reduces noise during training. This effect is observed when training on CIFAR-10, where multiple MC samples per process are required to stabilise training. On ImageNet, we have much larger minibatch size (4,096 instead of 256) and more parallel processes (128 not 8), and so training with 1 MC sample per process is still stable. However, as shown in Figure 11, increasing the number of training MC samples per process to from 1 to 2 speeds up convergence per epoch, and reaches a better converged solution. The time per epoch (and hence total runtime) also increases by approximately a factor of 1.5. Increasing the number of train MC samples per process to 3 does not increase final test performance significantly.

Increasing the number of testing MC samples from 10 to 100 (on the same trained model) also results in better generalisation: the train accuracy and log likelihood are unchanged, but the validation accuracy and log likelihood increase. However, as we run an entire validation on each epoch, increasing validation MC samples also increases run-time.

These results show that, if more compute is available to the user, they can improve VOGN's performance by improving the MC approximation at either (or both) train-time or test-time (up to a limit).

G MC-dropout's sensitivity to dropout rate

We show MC-dropout's sensitivity to dropout rate, p, in this Appendix. We tune MC-dropout as best as we can, finding that $p = 0.1$ works best for all architectures trained on CIFAR-10 (see Figure 12 for the dropout rate's sensitivity on LeNet-5 as an example). On ResNet-18 trained on ImageNet, we find that MC-dropout is extremely sensitive to dropout rate, with even $p = 0.1$ performing badly. We therefore use $p = 0.05$ for MC-dropout experiments on ImageNet. This high sensitivity to dropout rate is an issue with MC-dropout as a method.

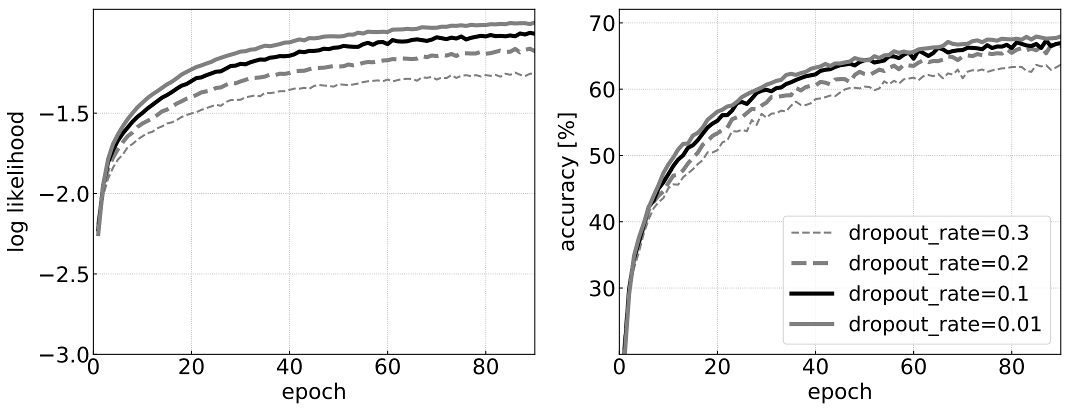


Figure 12: Effect of changing the dropout rate in MC-dropout, training LeNet-5 on CIFAR-10. When $p = 0.01$, the train-test gap on accuracy and log likelihood is very high (10.3% and 0.34 respectively). When $p = 0.1$, gaps are 1.4% and 0.04 respectively. When $p = 0.2$, the gaps are -7.71% and -0.02 respectively. We therefore choose $p = 0.1$ as it has high accuracy and log likelihood, and small train-test gap.

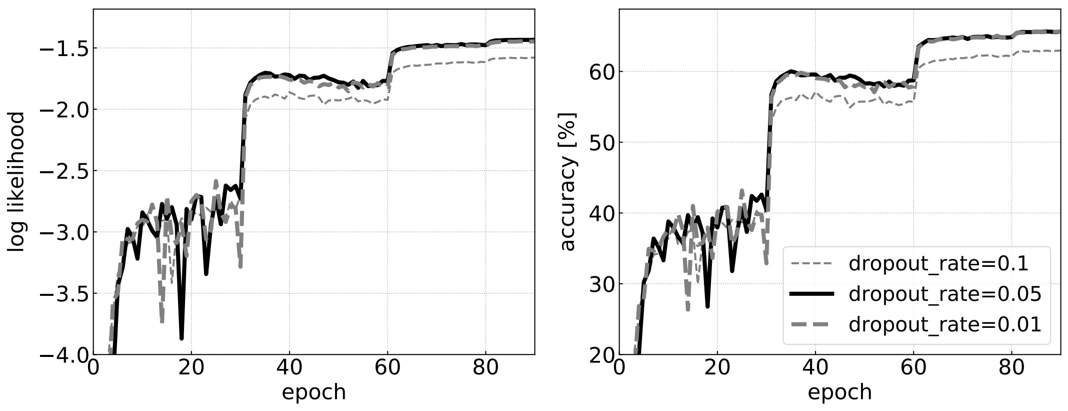


Figure 13: Effect of changing the dropout rate in MC-dropout, training Resnet-18 on ImageNet. We use $p = 0.05$ for our results.

H Uncertainty metrics

We use several approaches to compare uncertainty estimates obtained by each optimiser. We follow the same methodology for all optimisers: first, tune hyperparameters to obtain good accuracy on the validation set. Then, test on uncertainty metrics. For multi-class classification problems, all of these are based on the predictive probabilities. For non-Bayesian approaches, we compute the probabilities for a validation input $\mathbf{x}_i$ as $\hat{p}_{ik} := p(y_i = k|\mathbf{x}_i, \mathbf{w}_*)$, where $\mathbf{w}_*$ is the weight vector of the DNN whose uncertainty we are estimating. For Bayesian methods, we can compute the predictive probabilities for each validation example $\mathbf{x}_i$ as follows:

$$\hat{p}_{ik} := \int p(y_i = k|\mathbf{x}_i, \mathbf{w})p(\mathbf{w}|\mathcal{D})d\mathbf{w} \approx \int p(y_i = k|\mathbf{x}_i, \mathbf{w})q(\mathbf{w})d\mathbf{w} \approx \frac{1}{C}\sum_{c=1}^{C} p(y_i = k|\mathbf{x}_i, \mathbf{w}^{(c)}),$$

where $\mathbf{w}^{(c)} \sim q(\mathbf{w})$ are samples from the Gaussian approximation returned by a variational method. We use 10 MC samples at validation-time for VOGN and MC-dropout (the effect of changing number of validation MC samples is shown in Appendix F). This increases the computational cost during testing for these methods when compared to Adam or SGD.

Using the estimates $\hat{p}_{ik}$, we use three methods to compare uncertainties: validation log loss, AUROC and calibration curves. We also compare uncertainty performance by looking at model outputs when exposed to out-of-distribution data.

Validation log likelihood. Log likelihood (or log loss) is a common uncertainty metric. We consider a validation set of N_{Va} examples. For an input $\mathbf{x}_i$, denote the true label by $\mathbf{y}_i$, a 1-of-K encoded vector with 1 at the true label and 0 elsewhere. Denote the full vector of all validation outputs by $\mathbf{y}$. Similarly, denote the vector of all probabilities $\hat{p}_{ik}$ by $\mathbf{p}$, where $k \in \{1, ..., K\}$. The validation log likelihood is defined as $\ell(\mathbf{y}, \hat{\mathbf{p}}) := \frac{1}{N_{Va}} \sum_{i=1}^{N_{Va}} \sum_{k=1}^{K} y_{ik} \log \hat{p}_{ik}$.

Tables 1 and 3 show final validation (negative) log likelihood. VOGN performs very well on this metric (aside from on CIFAR-10/AlexNet, with or without DA, where MC-dropout performs the best). All final validation log likelihoods are very similar, with VOGN usually performing similarly to the other best-performing optimisers (usually MC-dropout).

Area Under ROC curves (AUROC). We consider Receiver Operating Characteristic (ROC) curves for our multi-way classification tasks. A potential way that we may care about uncertainty measurements would be to discard uncertain examples by thresholding each validation input's predicted class' softmax output, marking them as too ambiguous to belong to a class. We can then consider the remaining validation inputs to either be correctly or incorrectly classified, and calculate the True Positive Rate (TPR) and False Positive Rate (FPR) accordingly. The ROC curve is summarised by its Area Under Curve (AUROC), reported in Table 1. This metric is useful to compare uncertainty performance in conjunction with the other metrics we use. The AUROC results are very similar between optimisers, particularly on ImageNet, although MC-dropout performs marginally better than the others, including VOGN. On all but one CIFAR-10 experiment (AlexNet, without DA), VOGN performs the best, or tied best. Adam performs the worst, but is surprisingly good in CIFAR-10/ResNet-18.

Calibration Curves. Calibration curves [7] test how well-calibrated a model is by plotting true accuracy as a function of the model's predicted accuracy $\hat{p}_{ik}$ (we only consider the predicted class' $\hat{p}_{ik}$). Perfectly calibrated models would follow the $y = x$ diagonal line on a calibration curve. We approximate this curve by binning the model's predictions into $M = 20$ bins, as is often done. We show calibration curves in Figures 1 and 14. We can also consider the **Expected Calibration Error (ECE)** metric [40, 15], reported in Table 1. ECE calculates the expected error between the true accuracy and the model's predicted accuracy, averaged over all validation examples, again approximated by using M bins. Across all datasets and architectures, with the exception of LeNet-5 (which we have argued causes underfitting), VOGN usually has better calibration curves and better ECE than competing optimisers. Adam is consistently over-confident, with the calibration curve below the diagonal. Conversely, MC-dropout is usually under-confident, with too much noise, as mentioned earlier. The exception to this is on ImageNet, where MC-dropout performs well: we excessively tuned the MC-dropout rate to achieve this (see Appendix G).

I Out-of-distribution experimental setup and additional results

We use experiments from the out-of-distribution tests literature [16, 31, 8, 32], comparing VOGN to Adam and MC-dropout. Using trained architectures (LeNet-5, AlexNet and ResNet-18) on CIFAR-10, we test on SVHN, LSUN (crop) and LSUN (re-size) as out-of-distribution datasets, with the in-distribution data given by the validation set of CIFAR-10 (10,000 images). The entire training set of SVHN (73,257 examples, 10 classes) [42] is used. The test set of LSUN (Large-scale Scene UNderstanding dataset [55], 10,000 images from 10 different scenes) is randomly cropped to obtain LSUN (crop), and is down-sampled to obtain LSUN (re-size). These out-of-distribution datasets have no similar classes to CIFAR-10.

Similar to the literature [16, 30], we use 3 metrics to test performance on out-of-distribution data. Firstly, we plot histograms of predictive entropy for the in-distribution and out-of-distribution datasets, seen in Figure 5, 15, 16 and 17. Predictive entropy is given by $\sum_{k=1}^{K} -\hat{p}_{ik} \log \hat{p}_{ik}$. Ideally, on out-of-distribution data, a model would have high predictive entropy, indicating it is unsure of which class the input image belongs to. In contrast, for in-distribution data, good models should have many examples with low entropy, as they should be confident of many input examples' (correct) class. We also compare AUROC and FPR at 95% TPR, also reported in the figures. By thresholding the most likely class' softmax output, we assign high uncertainty images to belong to an unknown class. This allows us to calculate the FPR and TPR, allowing the ROC curve to be plotted, and the AUROC to be calculated.

We show results on AlexNet in Figure 15 and 16 (trained on CIFAR-10 with DA and without DA respectively) and on LeNet-5 in Figure 17. Results on ResNet-18 is in Figure 5. These results are discussed in Section 4.2.

J Author contributions statement

List of Authors: Kazuki Osawa, Siddharth Swaroop, Anirudh Jain, Runa Eschenhagen, Richard E. Turner, Rio Yokota, Mohammad Emtiyaz Khan.

M.E.K., A.J., and R.E. conceived the original idea. This was also discussed with R.Y. and K.O. and then with S.S. and R.T. Eventually, all authors discussed and agreed with the main focus and ideas of this paper.

The first proof-of-concept was done by A.J. using LeNet-5 on CIFAR-10. This was then extended by K.O. who wrote the main PyTorch implementation, including the distributed version. R.E. fixed multiple issues in the implementation, and also pointed out an important issue regarding data augmentation. S.S., A.J., K.O., and R.E. together fixed this issue. K.O. conducted most of the large experiments (shown in Fig. 1 and 4). The results shown in Fig. 3a was done by both K.O. and A.J. The BBB implementation was written by S.S.

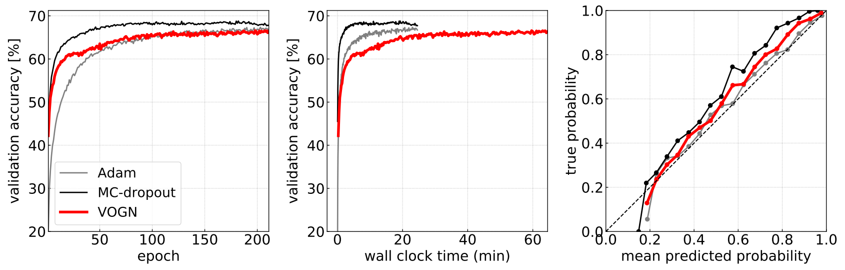
The experiments in Section 4.2 were performed by A.J. and S.S. The main ideas behind the experiments were conceived by S.S., A.J., and M.E.K. with many helpful suggestions from R.T. R.E. performed the permuted MNIST experiment using VOGN for the continual-learning experiments, and S.S. obtained the baseline results for the same.

The main text of the paper was written by M.E.K. and S.S. The section on experiments was first written by S.S. and subsequently improved by A.J., K.O., and M.E.K. R.T. helped edit the manuscript. R.E. also helped in writing parts of the paper.

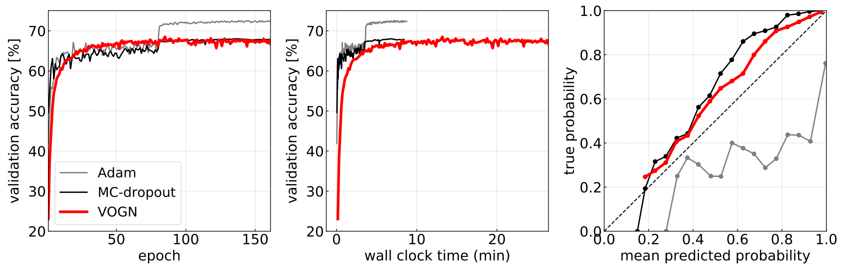
M.E.K. led the project with a significant help from S.S.. Computing resources and access to the HPCI systems were provided by R.Y.

K Changes in the camera-ready version compared to the submitted version

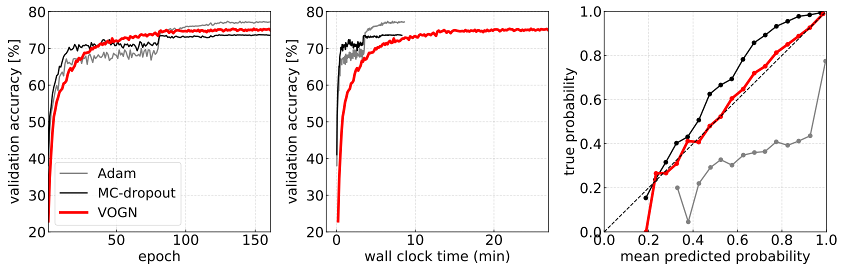
- We added an additional experiment on a continual learning task to show the effectiveness of VOGN (Figure 3b).
- In our experiments, we were using a damping factor γ. This was unfortunately missed in the submitted version, and we have now added it in Section 3.
- We modified the notation for Noisy K-FAC algorithm at Appendix A.
- We updated the description of our implementation of the Gauss-Newton approximation at Appendix B. Previous description had some missing parts and was a bit unclear.
- We added a description on a new method OGN which we were using to tune hyperparameters of VOGN. We have added its results in Table 1 and Table 3. The method details are in Appendix C.
- We added a description on how to tune VOGN to get good performance.
- We listed all training curves (epoch/time vs accuracy), including K-FAC, Noisy K-FAC, and OGN, along with the corresponding calibration curves in Figure 7.



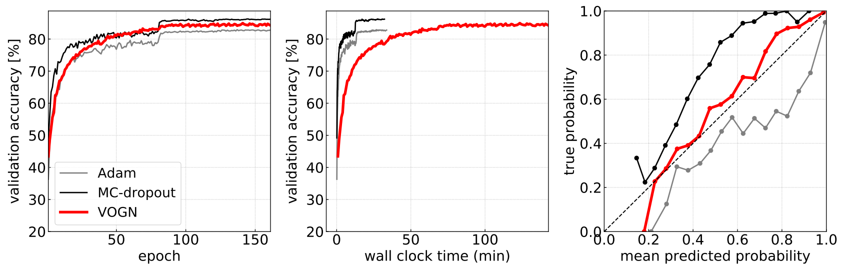
(a) LeNet-5 on CIFAR-10 (no DA)



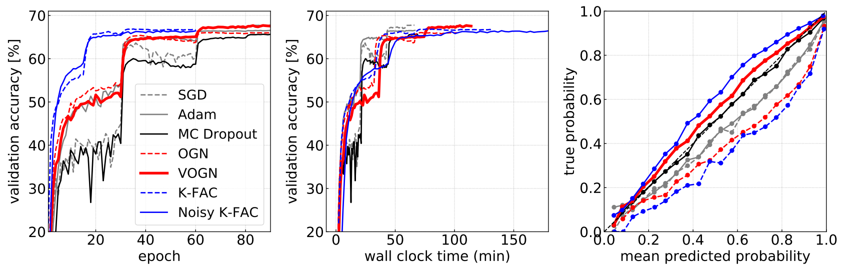
(b) AlexNet on CIFAR-10 (no DA)



(c) AlexNet on CIFAR-10



(d) ResNet-18 on CIFAR-10



(e) ResNet-18 on ImageNet

Figure 7: All results in Table 1

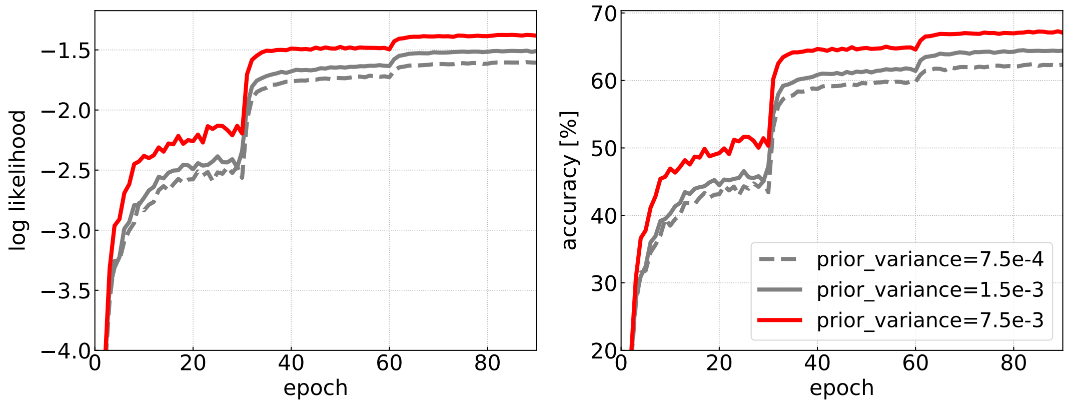


Figure 8: Effect of prior variance on VOGN training ResNet-18 on ImageNet.

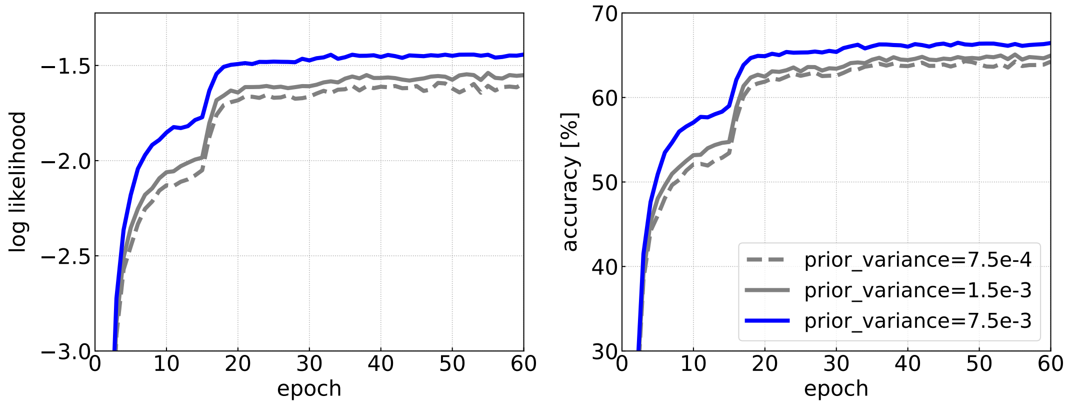


Figure 9: Effect of prior variance on Noisy K-FAC training ResNet-18 on ImageNet.

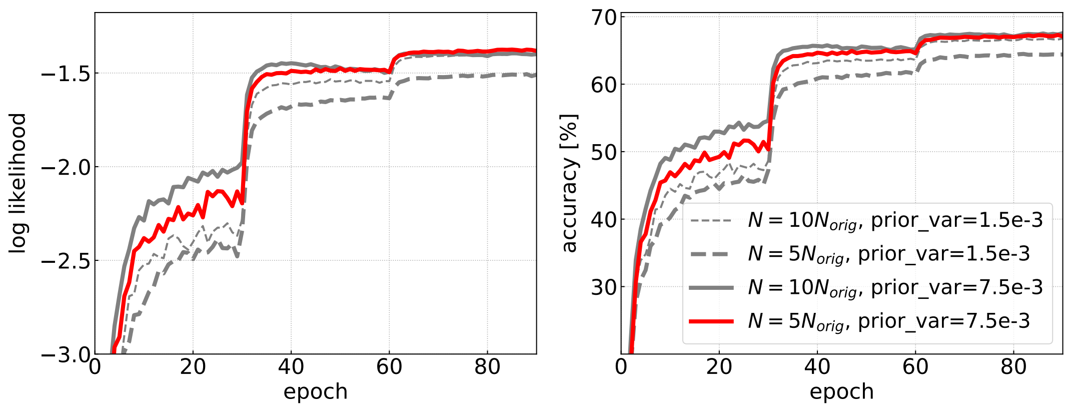


Figure 10: Effect of changing the dataset size reweighting factor ρ and prior variance on VOGN training ResNet-18 on ImageNet.

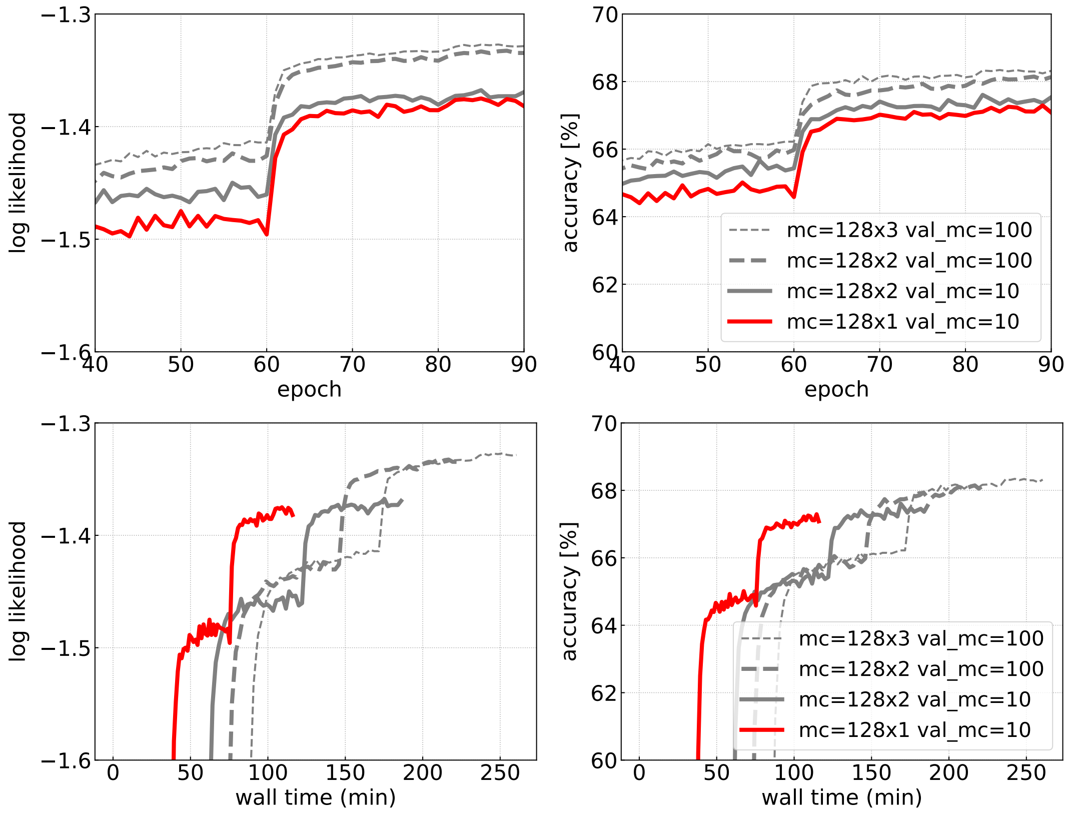


Figure 11: Effect of number of training and testing Monte Carlo samples on validation accuracy and log loss for VOGN on ResNet-18 on ImageNet.

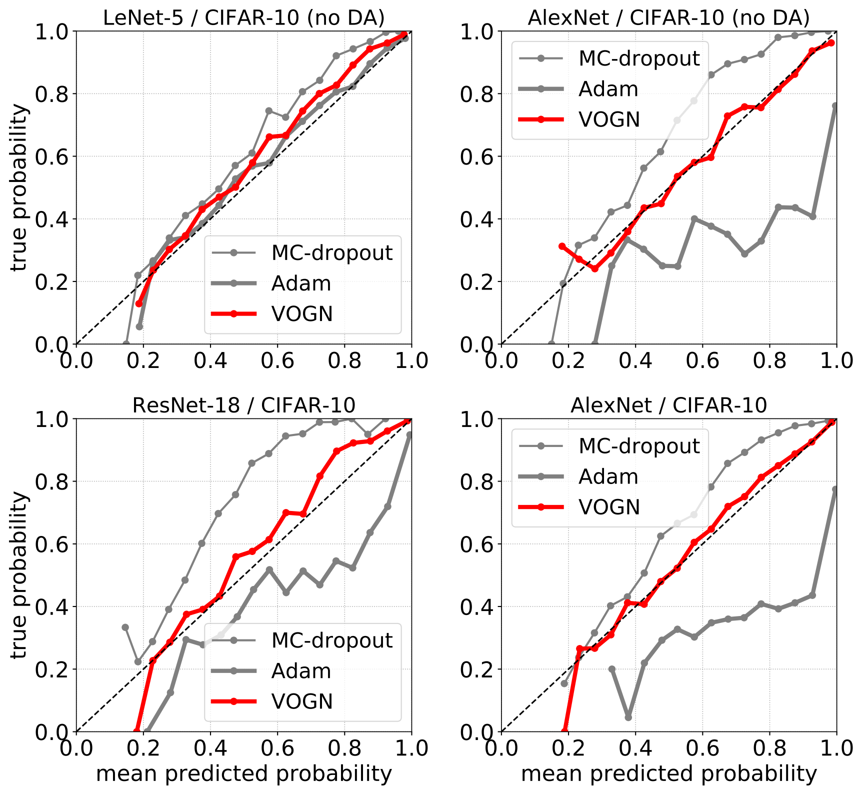


Figure 14: Calibration curves comparing VOGN, Adam and MC-dropout for final trained models trained on CIFAR-10. VOGN is extremely well-calibrated compared to the other two optimisers (except for LeNet-5, where all optimisers peform well). The calibration curve for ResNet-18 trained on ImageNet is in Figure 1.

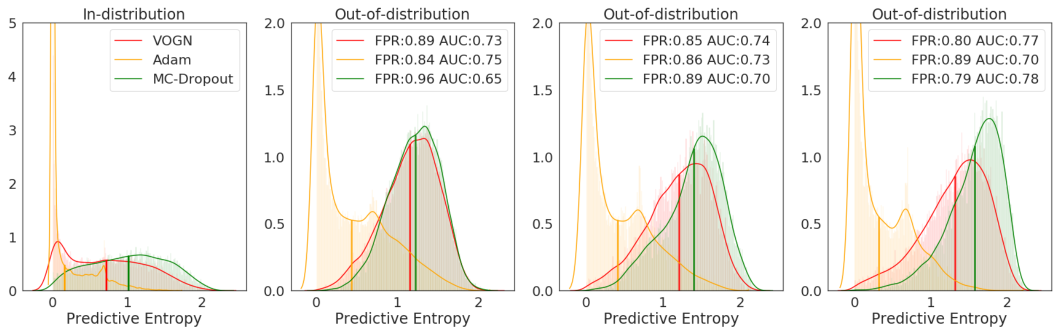


Figure 15: Histograms of predictive entropy for out-of-distribution tests for AlexNet trained on CIFAR-10 with data augmentation. Going from left to right, the inputs are: the in-distribution dataset (CIFAR-10), followed by out-of-distribution data: SVHN, LSUN (crop), LSUN (resize). Also shown are the AUROC metric (higher is better) and FPR at 95% TPR metric (lower is better), averaged over 3 runs. The standard deviations are very small and so not reported here.

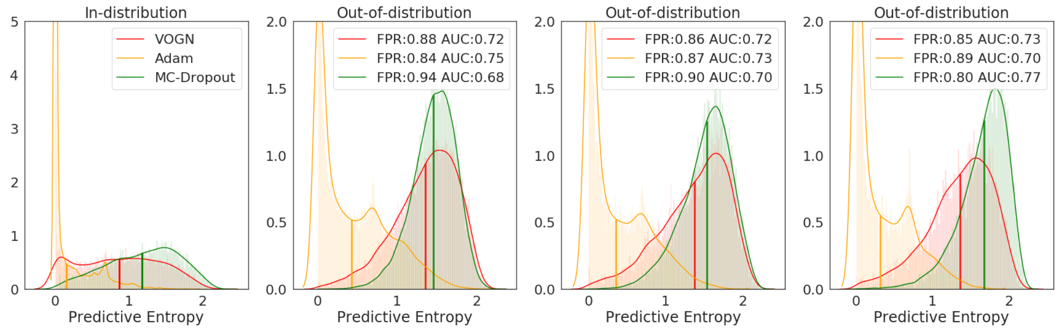


Figure 16: Histograms of predictive entropy for out-of-distribution tests for AlexNet trained on CIFAR-10 without data augmentation. Going from left to right, the inputs are: the in-distribution dataset (CIFAR-10), followed by out-of-distribution data: SVHN, LSUN (crop), LSUN (resize). Also shown are the AUROC metric (higher is better) and FPR at 95% TPR metric (lower is better), averaged over 3 runs. The standard deviations are very small and so not reported here.

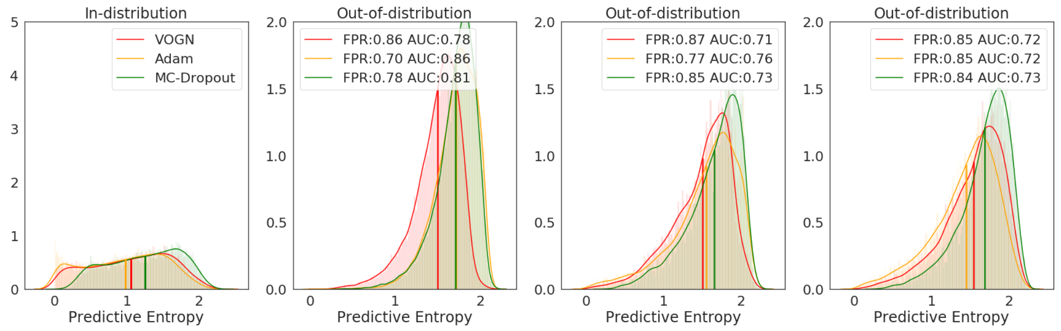


Figure 17: Histograms of predictive entropy for out-of-distribution tests for LeNet-5 trained on CIFAR-10 without data augmentation. Going from left to right, the inputs are: the in-distribution dataset (CIFAR-10), followed by out-of-distribution data: SVHN, LSUN (crop), LSUN (resize). Also shown are the AUROC metric (higher is better) and FPR at 95% TPR metric (lower is better), averaged over 3 runs. The standard deviations are very small and so not reported here.