Natural Gradient Variational Inference for Continual Learning in Deep Neural Networks

Runa Eschenhagen

Bachelor Thesis

Supervised by Rüdiger Busche M.Sc.
and Prof. Dr. Gordon Pipa.

Institute of Cognitive Science
University of Osnabrück
December 2019

Declaration of Authorship

I hereby certify that the work presented here is, to the best of my knowledge and belief, original and the result of my own investigations, except as acknowledged, and has not been submitted, either in part or whole, for a degree at this or any other university.

Signature

City, Date

Acknowledgements

Most of the work presented in this thesis has been conducted during an internship at RIKEN Center for Advanced Intelligence Project in the Approximate Bayesian Inference (ABI) team, supervised by Emtiyaz Khan. First of all, I would like to thank Emtiyaz Khan for giving me, an inexperienced bachelor student, the opportunity to do research in a great environment, and provide me valuable mentorship.

Additionally, I'm thankful for the support and help from all people who were part of the ABI team during my time there, notably, Anirudh Jain (ISM) and Alexander Immer (EPFL). Many thanks to Harumi Seo for all the administrative help and the RAIDEN (RIKEN AIP compute cluster) team.

Moreover, I'm thankful for many insightful discussions about all things related to continual learning with Siddharth Swaroop (University of Cambridge) and for providing the baselines for the split CIFAR experiment.

Also, thank you to Kazuki Osawa (Tokyo Institute of Technology) for writing a nice and scalable implementation of VOGN for Osawa et al. [2019] which I extended for continual learning, and used for the split CIFAR experiment. Further, I'm very grateful to Rio Yakota (Tokyo Institute of Technology) who provided me access to computing resources (TSUBAME3.0) and made it possible for me to return to Tokyo, to continue to work on continual learning.

Thank you to Christian Schlarmann (University of Tübingen) for reading through a draft of this thesis, pointing out errors, and providing helpful suggestions.

Last but not least, I am very thankful to Rüdiger Busche for being willing to supervise the writing process and for giving me valuable feedback on early drafts. Also, thank you to Prof. Dr. Gordon Pipa for acting as the second supervisor of this thesis.

Abstract

The goal of continual learning is to sequentially learn new tasks without revisiting or forgetting old ones. Specifically, neural networks tend to exhibit catastrophic forgetting when sequentially training on different tasks.

In theory, one approach to enable continual learning is Bayesian inference. In practice however, due to the complexity of neural networks, approximate Bayesian inference has to be used to make the computation feasible.

In this work, we propose to use a principled natural gradient variational inference method, called Variational Online Gauss-Newton (VOGN) [Khan et al., 2018], for continual learning. We provide the necessary background in continual learning, approximate Bayesian inference, and optimisation to understand VOGN. Moreover, building on these foundations, we derive natural gradient variational inference in general, and VOGN specifically.

The focus of this work is on the regularisation based approach to continual learning and on the supervised image classification setting with deep neural networks. We demonstrate empirically that VOGN can be applied to continual learning benchmarks, such as permuted and split MNIST, and split CIFAR10/100. The method consistently matches or outperforms comparable methods.

Additionally, we briefly introduce some potential further improvements to VOGN, such as using a structured covariance approximation.

VOGN is implemented as a plug-and-play optimiser which makes it easy to use; it also naturally scales to larger models and datasets, as demonstrated in Osawa et al. [2019]. We hope this work facilitates further research on natural gradient variational inference for continual learning, and see this as a promising demonstration of the merits of deep learning with Bayesian principles.

Contents

1 Introduction

Humans are able to learn new tasks throughout their whole lifetime without forgetting previously learned tasks. This is a crucial feature of human intelligence since it allows humans to deal with a variety of tasks. Moreover, learning certain tasks makes acquiring some new ones easier or it might increase performance of previously learned tasks. All of this is possible despite the fact that humans have only a limited capacity of memory and computation (the brain) at their disposal.

In contrast, machines have difficulties dealing with inputs in an online fashion; revisiting old data is expensive and sometimes even impossible. Also, even though memory tends to be cheap, not storing data also has other advantages, such as stronger privacy and security guarantees in the case of personal or classified data. In this work, we focus on the setting of continual learning for image classification tasks with neural networks. This setting is commonly used to develop new continual learning approaches and has received increased attention in recent years (cf. 1.1). Mostly, because neural networks tend to exhibit so-called catastrophic forgetting when learning multiple tasks sequentially [McCloskey and Cohen, 1989].

Bayesian inference (2.1) offers a natural way of enabling continual learning (2.1.2). In practice, exact Bayesian inference for neural networks is not tractable but approximate methods are available (2.1.1). Here, we build on a principled natural gradient variational inference algorithm, called Variational Online Gauss Newton (VOGN), which has originally been proposed in Khan et al. [2018]. Moreover, it has been shown to scale well to deep neural networks and offers a lot of promise for out-of-the-box continual learning [Osawa et al., 2019] (one result published in this paper is also part of this work, the rest of the here presented results are unpublished).

We introduce the necessary background to understand the algorithm in section 2, such as variational inference (2.1.3), basic optimisation for deep learning (2.2), and natural gradient descent (2.3). Note, that this is not meant to be a complete or formal review of these topics since all of them are huge active research fields. Rather, the reader should get some basic understanding and intuition about the ideas relevant to VOGN.

In section 3, we derive natural gradient variational inference in general (3.1), and VOGN specifically 3.2. We show how to use VOGN for continual learning (3.3) and how to scale it to larger datasets and model architectures (3.4). Moreover, we briefly introduce two possible structured covariance approximation which can be used with VOGN, SLANG (3.5.1) and K-FAC (3.5.2).

The empirical results and their discussion are presented in section 4. Here, we introduce evaluation metrics (4.1) and datasets (4.2), and present results for three continual learning benchmarks: Permuted (4.3) and split (4.4) MNIST, and split CIFAR10/100 (4.5).

Finally, we briefly summarise the motivation, method, and results in section 5. Additionally, we present potential improvements to VOGN and promising directions for further research in continual learning.

1.1 Related Work on Continual Learning

There are various approaches to facilitate continual learning in neural networks; generally one can differentiate between inference, model, and memory based approaches.

Inference based methods focus on adjusting the training algorithm to either regularise weights [Kirkpatrick et al., 2017, Zenke et al., 2017, Nguyen et al., 2018, Ebrahimi et al., 2019] or outputs [Benjamin et al., 2018, Titsias et al., 2019] to stay close to the ones from the previous tasks.

Model based methods try to enable continual learning by changing the neural network architecture, e.g. adding new layers or units during or in between tasks [Rusu

et al., 2016]. Alternatively, different subnetworks can be learned through pruning [Golkar et al., 2019]; this method still sees a performance decrease after the model capacity is reached and there is no potential for backward transfer. Moreover, learning a hard attention mask for all task has been proposed in Serrà et al. [2018]; the method is called HAT.

In the category of memory based approaches one can either try to select and store a small subset of each task's data [Chaudhry et al., 2019, Aljundi et al., 2019] or learn a generative model to rehearse 'artificial', generated data [Shin et al., 2017, Farquhar and Gal, 2019].

Obviously, there are no hard boundaries between these general approaches and all of them can be combined in some ways, e.g. by adjusting the training algorithm, using a multi-head network, and storing a so-called coreset, as done in Nguyen et al. [2018]. Arguably, a true solution to the continual learning problem might need to incorporate elements of all the different approaches.

In this work, our focus is on methods that modify the training algorithms so that they can help to learn continually; specifically, we focus on methods which regularise in weight space. However, depending on the application, it might make sense to trade-off some memory for increased performance by selecting a small (compared to the dataset size) coreset of each task's data; however, we do not consider this in this work.

2 Background

2.1 Bayesian Inference

Bayesian inference builds upon Bayes' rule which states that the conditional probability of an event A given that B is true, also called the posterior probability, is given by

$$P(A|B) = \frac{P(A,B)}{P(B)} = \frac{P(B|A)P(A)}{P(B)}. \tag{1}$$

Bayes' rule directly follows from the definition of the conditional probability and the product rule.

Bayesian inference frames inference of unknown quantities as calculating the posterior, given by Bayes' rule for probability density functions:

$$p(\mathbf{y}|\mathbf{x}) = \frac{p(\mathbf{x},\mathbf{y})}{p(\mathbf{x})} = \frac{p(\mathbf{x}|\mathbf{y})p(\mathbf{y})}{p(\mathbf{x})} \tag{2}$$

Here, $\mathbf{y} = y_{1:N}$ are unobserved or latent variables and $\mathbf{x} = x_{1:N}$ are observations.

In this work, we want to use Bayesian inference to determine the conditional probability distribution $p(\boldsymbol{\theta}|\mathcal{D})$ over the parameters or weights $\boldsymbol{\theta}$ of a neural network, given the observed data $\mathcal{D}$, and a prior distribution $p(\boldsymbol{\theta})$. We focus on supervised classification tasks where the data is given as a set $\mathcal{D} = \left\{\mathbf{x}_i, y_i\right\}_{i=1}^{N}$, with N identically and independently distributed samples, where $\mathbf{x}_i$ is the input, here often the raw pixel values of an image, and y_i the target class. The goal is to predict the correct label y_i to the corresponding input $\mathbf{x}_i$, i.e. use the neural network as a discriminative model.

Let $\mathbf{X}$ be a matrix with $\mathbf{x}_i$ as the i-th row and $\mathbf{y}$ a vector with y_i as the i-th entry. Since we are interested in the probability distribution over the parameters of the neural network $\boldsymbol{\theta}$, we want to compute

$$\begin{aligned} p(\boldsymbol{\theta}|\mathcal{D}) &= \frac{p(\boldsymbol{\theta},\mathbf{y},\mathbf{X})}{p(\mathbf{y},\mathbf{X})} = \frac{p(\mathbf{y}|\boldsymbol{\theta},\mathbf{X})p(\boldsymbol{\theta}|\mathbf{X})}{p(\mathbf{y}|\mathbf{X})} = \frac{p(\mathbf{y}|\boldsymbol{\theta},\mathbf{X})p(\boldsymbol{\theta})}{p(\mathbf{y}|\mathbf{X})} \\ &\propto p(\mathbf{y}|\boldsymbol{\theta},\mathbf{X})p(\boldsymbol{\theta}), \end{aligned} \tag{3}$$

where $p(\boldsymbol{\theta}|\mathbf{X}) = p(\boldsymbol{\theta})$ because the prior $p(\boldsymbol{\theta})$ is independent of the data. The prior makes it possible to incorporate domain and expert knowledge into the problem. However, in practice the prior is often treated like a hyperparameter (or at least the parameters of the chosen prior distribution).

To predict the label y_* for an unseen data point $\mathbf{x}_*$, we can use the predictive distribution,

$$\begin{aligned} p(y_*|\mathbf{x}_*,\mathcal{D}) &= \int_\Theta p(y_*,\boldsymbol{\theta}|\mathbf{x}_*,\mathcal{D})d\boldsymbol{\theta} = \int_\Theta p(y_*|\boldsymbol{\theta},\mathbf{x}_*,\mathcal{D})p(\boldsymbol{\theta}|\mathbf{x}_*,\mathcal{D})d\boldsymbol{\theta} \\ &= \int_\Theta p(y_*|\boldsymbol{\theta},\mathbf{x}_*)p(\boldsymbol{\theta}|\mathcal{D})d\boldsymbol{\theta}, \end{aligned} \tag{4}$$

which is calculated by using the posterior distribution $p(\boldsymbol{\theta}|\mathcal{D})$ to integrate over the domain $\boldsymbol{\Theta}$ of $\boldsymbol{\theta}$. From now on, we omit the subscript of the integral over parameter values. In contrast to Bayesian inference, one could also use a point estimate of $\boldsymbol{\theta}$. The advantage of Bayesian inference is, that our prediction accounts for the uncertainty in our estimate of $\boldsymbol{\theta}$. In practice, we can use a Monte Carlo approximation (explained in equation 47) to this integral.

The quantity $p(\mathbf{y}|\mathbf{X})$ in equation 3, commonly called model evidence or marginal likelihood, is constant and can be computed by marginalising out all possible parameter values $\boldsymbol{\theta}$:

$$p(\mathbf{y}|\mathbf{X}) = \int p(\mathbf{y}, \boldsymbol{\theta}|\mathbf{X}) d\boldsymbol{\theta} = \int p(\mathbf{y}|\boldsymbol{\theta}, \mathbf{X}) p(\boldsymbol{\theta}) d\boldsymbol{\theta} \tag{5}$$

Since neural networks can have a huge number of parameters, often in the millions, this integral is not tractable for most commonly used models. Therefore, approximate Bayesian inference is necessary in practice. There are many possible ways of approaching this with neural networks, which is briefly summarised in section 2.1.1. In this work, we focus on an approach called variational inference (see section 2.1.3) which can be used for all kind of models even though we only apply it to neural networks.

2.1.1 Bayesian Neural Networks

This section deals with the question of how we can do approximate Bayesian inference with neural networks.

The most naive approach is ignoring the normalising constant $p(\mathbf{y}|\mathbf{X})$ since the remaining term is still proportional to the the true posterior (see equation (3)) and therefore has the same mode, which corresponds to computing the maximum a posteriori (MAP) estimate. Note, that this is similar to using the maximum likelihood (ML) estimate but with a prior distribution $p(\boldsymbol{\theta})$ which corresponds to what is called a regulariser in the common deep learning terminology. If we choose a L2 regulariser, this would be equivalent to a Gaussian prior distribution.

Alternatively, one could use the MAP as the mean and the Hessian at the MAP as the covariance matrix for a Gaussian distribution which has been called Laplace approximation [Mackay, 1991].

Further, more sophisticated approaches to doing approximate Bayesian inference with neural networks exist. There are sampling based approaches like Markov Chain Monte Carlo (MCMC) which have also been used for neural networks but struggle with scalability issues. Moreover, it has been argued that using dropout during test time corresponds to sampling from the posterior distribution [Gal and Ghahramani, 2016].

Also, there is the possibility to perform variational inference with neural networks by using commonly used deep learning optimisation methods like Adam to optimise the evidence lower bound with regards to the mean and covariance of a mean-field Gaussian variational distribution (see section 2.1.3). This has been proposed by Blundell et al. [2015] and is called Bayes-by-Backprop (BBB).

2.1.2 The Bayesian Approach to Continual Learning

Bayesian inference offers a natural solution to the continual learning problem: Since the posterior incorporates all information of the prior distribution, the prior could encode the distribution learned while training on previous tasks. The posterior distribution should capture all information of the previous tasks through this prior and of the current task through the likelihood.

Let k be the index of the current task. Then the posterior after training on task k is given by

$$
\begin{aligned}
p(\boldsymbol{\theta}|\mathcal{D}_{1:k}) &= \frac{p(\mathbf{y}_{1:k}|\boldsymbol{\theta}, \mathbf{X}_{1:k})p(\boldsymbol{\theta})}{p(\mathbf{y}_{1:k}|\mathbf{X}_{1:k})} \\
&= \frac{p(\boldsymbol{\theta})\prod_{i=1}^{k} p(\mathbf{y}_i|\boldsymbol{\theta}, \mathbf{X}_i)}{\prod_{i=1}^{k} p(\mathbf{y}_i|\mathbf{X}_i)} \\
&= \frac{p(\mathbf{y}_k|\boldsymbol{\theta}, \mathbf{X}_k)p(\mathbf{y}_{1:k-1}|\boldsymbol{\theta}, \mathbf{X}_{1:k-1})p(\boldsymbol{\theta})}{p(\mathbf{y}_k|\mathbf{X}_k)p(\mathbf{y}_{1:k-1}|\mathbf{X}_{1:k-1})} \\
&= \frac{p(\mathbf{y}_k|\boldsymbol{\theta}, \mathbf{X}_k)p(\boldsymbol{\theta}|\mathcal{D}_{1:k-1})}{p(\mathbf{y}_k|\mathbf{X}_k)}.
\end{aligned} \tag{6}
$$

Note, that we assume that the likelihood $p(\mathbf{y}_{1:k}|\boldsymbol{\theta}, \mathbf{X}_{1:k})$ and the marginal likelihood $p(\mathbf{y}_{1:k}|\mathbf{X}_{1:k})$ can be factorised, i.e. they are (conditionally) independent.

In this final form it becomes obvious that $p(\boldsymbol{\theta}|\mathcal{D}_{1:k})$, i.e. the posterior obtained after training on task k, can be expressed recursively; $p(\boldsymbol{\theta}|\mathcal{D}_{1:k-1})$, the posterior obtained after training on task $k-1$, becomes the prior for training on task k.

Therefore, in theory, Bayesian inference could prevent catastrophic forgetting but in practice, approximate Bayesian methods have to be used to make the computation feasible which leads to some forgetting.

Still, approximate Bayesian methods for neural networks are able to use the recursive updates of the posterior as described above; the prior acts as a regulariser which helps to avoid changes in parameters important for performance on previous tasks. For example, the Elastic Weight Consolidation (EWC) method [Kirkpatrick et al., 2017] uses past experiences as a prior distribution with a type of Laplace approximation. Ritter et al. [2018] extend this approach to use a Kronecker factored online Laplace approximation. Another method called variational continual learning (VCL) [Nguyen et al., 2018] uses variational inference instead of the Laplace approximation, implemented with BBB. This was extended to use the variance to adapt the learning rate appropriately in a method called uncertainty-guided continual learning with Bayesian neural networks (UCB) [Ebrahimi et al., 2019]. There has also been some recent effort of using Gaussian processes [Rasmussen and Williams, 2006] for *functional* regularisation of neural networks to enable continual learning [Titsias et al., 2019]. While this is out of the scope of this work, using functional regularisation offers some potential benefits over regularisation of the weights since it can exploit well-known weight-space symmetries in neural networks [Benjamin et al., 2018, Bishop, 2006].

2.1.3 Variational Inference

We choose variational inference (VI) [Jordan et al., 1999, Wainwright and Jordan, 2008] as a method of doing approximate Bayesian inference. For a more detailed review of VI, see Blei et al. [2016].

The key idea of variational inference is to formulate Bayesian inference as an optimisation problem. We restrict the posterior distribution to some family of distributions, for example Gaussians, and then maximise the similarity between our chosen variational distribution $q(\boldsymbol{\theta})$ and the true posterior distribution $p(\boldsymbol{\theta}|\mathcal{D})$. As a measure of proximity between two continuous probability distributions $p(x)$ and $q(x)$ we choose the Kullback-Leibler (KL) divergence, defined as

$$
\mathbb{D}_{KL}(p(x) \,\|\, q(x)) = \int_{-\infty}^{\infty} p(x) \log \frac{p(x)}{q(x)} dx = \mathbb{E}_{p(x)}[\log p(x)] - \mathbb{E}_{p(x)}[\log q(x)]. \tag{7}
$$

It is not symmetric (and thereby not a metric) and is greater or equal than zero. It is zero if and only if $p(x) = q(x)$. From now on, if there is no ambiguity, we suppress the boundaries of integrals and the subscript of expectations for the sake of simplicity.

Equipped with this dissimilarity measure, we can formulate the following optimisation problem where we constrain the approximate posterior, the variational distribution $q(\boldsymbol{\theta})$, to be part of the variational family $\mathcal{F}$:

$$\begin{aligned} q^*(\boldsymbol{\theta}) &= \arg\min_{q(\theta)\in\mathcal{F}} \mathbb{D}_{KL}(q(\boldsymbol{\theta}) \,\|\, p(\boldsymbol{\theta}|\mathcal{D})) \\ &= \arg\min_{q(\theta)\in\mathcal{F}} \mathbb{E}[\log q(\boldsymbol{\theta})] - \mathbb{E}[\log p(\boldsymbol{\theta}|\mathcal{D})] \\ &= \arg\min_{q(\theta)\in\mathcal{F}} \mathbb{E}[\log q(\boldsymbol{\theta})] - \mathbb{E}[\log p(\boldsymbol{\theta},\mathcal{D})] + \log p(\mathcal{D}) \end{aligned} \tag{8}$$

From the expansion done in equation (8), it is obvious that we cannot directly optimise this objective since it depends on the logarithm of the marginal probability $p(\mathcal{D})$. Luckily, since it is simply an added constant, we can reformulate an equivalent objective which is independent of $p(\mathcal{D})$. This new function is called evidence lower bound (ELBO) and is the negative version of the previously stated KL divergence with the model evidence added.

$$\begin{aligned} \mathcal{L}(q) &= \mathbb{E}[\log p(\boldsymbol{\theta},\mathcal{D})] - \mathbb{E}[\log q(\boldsymbol{\theta})] \\ &= \mathbb{E}[\log p(\mathcal{D}|\boldsymbol{\theta})] + \mathbb{E}[\log p(\boldsymbol{\theta})] - \mathbb{E}[\log q(\boldsymbol{\theta})] \\ &= \mathbb{E}[\log p(\mathcal{D}|\boldsymbol{\theta})] + \mathbb{D}_{KL}(p(\boldsymbol{\theta}) \,\|\, q(\boldsymbol{\theta})) \\ &= \mathbb{E}[\log p(\mathcal{D}|\boldsymbol{\theta})] - \mathbb{D}_{KL}(q(\boldsymbol{\theta}) \,\|\, p(\boldsymbol{\theta})) \end{aligned} \tag{9}$$

Rewritten in this way, maximising the ELBO has an intuitive interpretation: The first term is the expectation of the log likelihood which is maximised while the second term is the KL divergence between the variational distribution and the prior distribution $p(\boldsymbol{\theta})$ which is minimised. Loosely speaking, optimising this bound corresponds to optimising the fit of the model to the data while encouraging the variational distribution to stay close to the prior.

Using the posterior derived for the supervised continual learning setting in equation 6, the ELBO for task k becomes

$$\begin{aligned} \mathcal{L}_{CL}(q_k) &= \mathbb{E}[\log p(\mathbf{y}_k|\boldsymbol{\theta},\mathbf{X}_k)] - \mathbb{D}_{KL}(q_k(\boldsymbol{\theta}) \,\|\, p(\boldsymbol{\theta}|\mathcal{D}_{1:k-1})) \\ &\approx \mathbb{E}[\log p(\mathbf{y}_k|\boldsymbol{\theta},\mathbf{X}_k)] - \mathbb{D}_{KL}(q_k(\boldsymbol{\theta}) \,\|\, q_{k-1}(\boldsymbol{\theta})), \end{aligned} \tag{10}$$

with $q_0(\boldsymbol{\theta}) = p(\boldsymbol{\theta})$.

The next obvious question is what family of distributions $\mathcal{F}$ we want to constrain $q(\boldsymbol{\theta})$ to be part of. Note, that if we would consider all possible families of distributions, variational inference would result in the exact posterior distribution. Since more complex families are harder to optimise, we need to decide on a a trade off between ease of computation and sufficient complexity for our purposes.

For this work, we only consider Gaussian distributions, especially because they offer many convenient properties and closed form expressions. In particular, we focus on so-called mean-field Gaussian distributions, which take the following form:

$$q(\boldsymbol{\theta}) = \prod_{i=1}^{D} q_i(\theta_i) \tag{11}$$

This leads to a multivariate Gaussian distribution $\mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\mu},\boldsymbol{\Sigma})$ with mean vector $\boldsymbol{\mu}$ and diagonal covariance matrix $\boldsymbol{\Sigma} = \text{diag}(\boldsymbol{\sigma}^2)$, where $\text{diag}(\cdot)$ is an operator that constructs a square diagonal matrix with the argument, a vector, on its diagonal. Later, we also briefly introduce two structured approximations to the covariance matrix in section 3.5 but the main focus remains on the diagonal approximation.

Using the mean-field Gaussian variational family for neural networks means that there now is a univariate Gaussian distribution over every parameter of the neural network. Therefore, the number of factorised Gaussians is equal to the number of parameters of the neural network D; the mean $\boldsymbol{\mu}$ is of dimension D. Also, it is easy to see how a non-diagonal covariance matrix would be huge since it is of size $D \times D$. In the diagonal case, it is sufficient to store and work with the vector $\boldsymbol{\sigma}^2$ of size D. This means that we have to store $2D$ parameters for our neural network instead of just D.

2.2 Optimisation for Deep Learning

The next question when applying variational inference for neural networks is how the ELBO should be optimised. In this section we provide some context on commonly used optimisation methods in deep learning, in particular stochastic gradient descent (SGD) and the popular method Adam [Kingma and Ba, 2015].

Let's consider a common scenario in a deep learning multi-class classification task where the objective ℓ is continuously differentiable, e.g.the cross entropy with the logarithm of the softmax function applied to the outputs of the neural network; we assume that the dataset $\mathcal{D} = \left\{\mathbf{x}_i, y_i\right\}_{i=1}^{N}$ is split into mini-batches of M examples $(\mathbf{x}_i, y_i)$ each. The loss function ℓ is averaged over all examples in a mini batch.

$$\ell(\boldsymbol{\theta}) = \frac{1}{M} \sum_{i=1}^{M} \ell_i(\boldsymbol{\theta}) \tag{12}$$

Now we want to minimise this loss function with help of the gradient of $\ell(\boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$. Therefore, we want to take steps in the opposite direction of the gradient in the parameter space, to iteratively update our estimate of $\boldsymbol{\theta}$.

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha \nabla_\theta \ell(\boldsymbol{\theta}) \tag{13}$$

Here, t indicates the iteration and α is a constant step size, also called learning rate. The gradient $\nabla_\theta \ell(\boldsymbol{\theta})$ is a column vector, as all other vectors in this work.

When the mini-batch size M is equal to the dataset size N, this is full batch gradient descent. If $M = 1$, it is called stochastic gradient descent (SGD). For both, stochastic and full batch gradient descent, convergence guarantees exist, which state that they converge to the global optimum for strongly convex loss functions.

The most common case is a compromise between stochastic and full batch gradient descent, where $1 < M < N$. This offers a balance between the properties of the two extremes: When using the whole dataset for each iteration, the optimisation process is deterministic; this means that common nonlinear optimisation like conjugate gradient and Newton methods can be used. Also, full batch gradient descent can be easily distributed, in contrast to SGD, where the weights are updated much more frequently. However, full batch gradient descent might not be as efficient as SGD, since there might be (approximate) redundancy in the data. This can lead to faster convergence of SGD compared to full batch gradient descent, at least initially. Moreover, large datasets might not even fit in memory. Mini-batch gradient descent can also make use of optimised matrix operations implemented in common deep learning frameworks. See Bottou et al. [2016] for more details on the properties of gradient descent methods.

Main challenges for these simple gradient descent methods are highly non-convex optimisation which is the usual case in deep learning, and how to choose an appropriate learning rate.

In addition to these simple gradient descent methods, many more sophisticated adjustments have been proposed and tested empirically to address these challenges. There are simpler ones like adding a momentum term to the gradient and more

complex ones. Here we focus on Adam [Kingma and Ba, 2015], as it is surprisingly similar to the main method used in this work. The main idea is based on adapting the learning rate through using estimates of the first and second moment, $\mathbf{m}$ and $\mathbf{v}$ respectively, of the gradients $\mathbf{g}$ as exponential moving averages (initialised with zero), which are using the decay factors β_1 and β_2. This effectively changes the learning rate for each individual parameter, which allows steps proportional to the importance of each parameter.

$$\begin{aligned} \mathbf{g}_{t+1} &= \nabla_\theta \ell(\boldsymbol{\theta}_t) \\ \mathbf{m}_{t+1} &= \beta_1 \mathbf{m}_t + (1-\beta_1)\mathbf{g}_{t+1} \\ \mathbf{v}_{t+1} &= \beta_2 \mathbf{v}_t + (1-\beta_2)\mathbf{g}_{t+1}^2 \end{aligned} \tag{14}$$

Since these estimates of the moments are biased, they are corrected:

$$\begin{aligned} \hat{\mathbf{m}}_{t+1} &= \frac{\mathbf{m}_{t+1}}{(1-\beta_1^{t+1})} \\ \hat{\mathbf{v}}_{t+1} &= \frac{\mathbf{v}_{t+1}}{(1-\beta_2)^{t+1}} \end{aligned} \tag{15}$$

And finally, the parameters $\boldsymbol{\theta}$ are updated. A small constant $\epsilon > 0$ is added for numerical stability.

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha \frac{\hat{\mathbf{m}}_{t+1}}{\sqrt{\hat{\mathbf{v}}_{t+1}} + \epsilon} \tag{16}$$

Adam has been empirically shown to work well in many scenarios and is widely used in the deep learning community. One of the advantages for the practical use is that learning rate α and the other hyper-parameters β_1 and β_2 usually do not have to be tuned a lot to get reasonable results.

2.3 Natural Gradient Descent

Now we want to explore an alternative to classical gradient descent methods which is a central part of our main method. Note, that when applying gradient descent methods to minimise an objective function, we want to take steps in the direction of steepest descent. To motivate the notion of natural gradients [Amari, 1998], we first define what steepest descent actually means.

Definition 2.1. (Steepest descent) The direction of steepest descent is the vector $\delta\boldsymbol{\theta}$ that minimises $\ell(\boldsymbol{\theta} + \delta\boldsymbol{\theta})$ under the constraint that $\delta\boldsymbol{\theta}$ has a fixed length $\epsilon\|\mathbf{a}\|$, with ϵ being a small constant and $\mathbf{a}$ a vector with $\|\mathbf{a}\|^2 = 1$. The norm $\|\cdot\|$ is not specified for now.

The crucial point about this definition is that it only makes sense to compare the decrease of the function ℓ that we want to minimise when moving into different directions $\mathbf{a}$ if we move the same distance in each direction, i.e. $\epsilon\|\mathbf{a}\|$.

Remember that we did not decide on a particular norm yet. First, we consider the Euclidean norm with which the constraint becomes $\|\mathbf{a}\|_2^2 = \sum a_i^2 = \mathbf{a}^T\mathbf{a} = 1$. Moreover, we observe that using the first order Taylor expansion, we can write $\ell(\boldsymbol{\theta} + \delta\boldsymbol{\theta}) \approx \ell(\boldsymbol{\theta}) + \epsilon\nabla_\theta\ell(\boldsymbol{\theta})^T\mathbf{a}$, where $\epsilon \to 0$, as long as ℓ is continuously differentiable. Now we can approach this constrained minimisation problem by applying the Lagrangian method, with λ being the Lagrangian multiplier:

$$\frac{\partial}{\partial a_i}\nabla_\theta\ell(\boldsymbol{\theta})^T\mathbf{a} - \lambda\mathbf{a}^T\mathbf{a} = 0 \tag{17}$$

When solving for $\mathbf{a}$, we get

$$\mathbf{a} = \frac{1}{2\lambda}\nabla_\theta \ell(\boldsymbol{\theta}). \tag{18}$$

As you can see, choosing the Euclidean norm results in the regular gradient of ℓ multiplied by a constant $\frac{1}{2\lambda}$ being the direction of steepest descent. If we collapse the constant in our learning rate α we get classical gradient descent, as shown in equation (13).

Remember, that when we are optimising the ELBO, we ultimately want to optimise in the space of probability distributions of the family $\mathcal{F}$. The problem is that the Euclidean distance between parameter vectors $\boldsymbol{\theta}$ and $\boldsymbol{\theta} + \delta\boldsymbol{\theta}$, which we are implicitly chosen when applying classical gradient descent, is generally a bad dissimilarity measure between the two corresponding probability distributions.

This can be easily illustrated by an example from Hoffman et al. [2013]: While the parameters of the two barely overlapping Gaussians $\mathcal{N}(0, 0.01)$ and $\mathcal{N}(0.1, 0.01)$ have an Euclidean distance of only 0.1, the parameters of the two almost identical Gaussians $\mathcal{N}(0, 10000)$ and $\mathcal{N}(10, 10000)$ have an Euclidean distance of 10.

We already know a better dissimilarity measure for probability distributions, namely the KL divergence in equation (7). To define a metric with help of the KL divergence, let us first specify the loss $\ell(\boldsymbol{\theta}) = -\log h(\boldsymbol{\theta})$ with the likelihood $h(\boldsymbol{\theta}) := p(\mathbf{y}|\boldsymbol{\theta}, \mathbf{X})$. Now we approximate the KL divergence between $h(\boldsymbol{\theta})$ and $h(\boldsymbol{\theta}')$ with $\boldsymbol{\theta}' = \boldsymbol{\theta} + \delta\boldsymbol{\theta}$ by the second order Taylor polynomial around the current estimate $\boldsymbol{\theta}$:

$$\begin{aligned}\mathbb{D}_{KL}(h(\boldsymbol{\theta}) \,\|\, h(\boldsymbol{\theta}')) \approx &\mathbb{D}_{KL}(h(\boldsymbol{\theta}) \,\|\, h(\boldsymbol{\theta}')) + \nabla_{\theta'}\mathbb{D}_{KL}(h(\boldsymbol{\theta}) \,\|\, h(\boldsymbol{\theta}'))^T(\boldsymbol{\theta}' - \boldsymbol{\theta}) \\ &+ \frac{1}{2}(\boldsymbol{\theta}' - \boldsymbol{\theta})^T\nabla^2_{\theta'}\mathbb{D}_{KL}(h(\boldsymbol{\theta}) \,\|\, h(\boldsymbol{\theta}'))(\boldsymbol{\theta}' - \boldsymbol{\theta})\end{aligned} \tag{19}$$

with

$$\mathbb{D}_{KL}(h(\boldsymbol{\theta}) \,\|\, h(\boldsymbol{\theta}'))|_{\theta'=\theta} = 0. \tag{20}$$

Additionally, we can rewrite $\nabla_{\theta'}\mathbb{D}_{KL}(h(\boldsymbol{\theta}) \,\|\, h(\boldsymbol{\theta}'))$ as

$$\begin{aligned}\nabla_{\theta'}\mathbb{D}_{KL}(h(\boldsymbol{\theta}) \,\|\, h(\boldsymbol{\theta}')) &= \nabla_{\theta'}\left[\mathbb{E}_{h(\theta)}[\log h(\boldsymbol{\theta})] - \mathbb{E}_{h(\theta)}[\log h(\boldsymbol{\theta}')]\right] \\ &= -\int h(\boldsymbol{\theta})\nabla_{\theta'}\log h(\boldsymbol{\theta}')d\boldsymbol{\theta} \\ &= -\int h(\boldsymbol{\theta})\frac{1}{h(\boldsymbol{\theta}')}\nabla_{\theta'}h(\boldsymbol{\theta}')d\boldsymbol{\theta}\end{aligned} \tag{21}$$

and if we evaluate at $\boldsymbol{\theta}' = \boldsymbol{\theta}$, we get

$$\begin{aligned}\nabla_{\theta'}\mathbb{D}_{KL}(h(\boldsymbol{\theta}) \,\|\, h(\boldsymbol{\theta}'))|_{\theta'=\theta} &= -\int h(\boldsymbol{\theta})\frac{1}{h(\boldsymbol{\theta})}\nabla_\theta h(\boldsymbol{\theta})d\boldsymbol{\theta} \\ &= -\nabla_\theta \int h(\boldsymbol{\theta})d\boldsymbol{\theta} \\ &= -\nabla_\theta 1 \\ &= 0.\end{aligned} \tag{22}$$

The quadratic term can be written as

$$\begin{aligned}\nabla^2_{\theta'}\mathbb{D}_{KL}(h(\boldsymbol{\theta}) \,\|\, h(\boldsymbol{\theta}')) &= -\nabla^2_{\theta'}\int h(\boldsymbol{\theta})\log h(\boldsymbol{\theta}')d\boldsymbol{\theta} \\ &= -\int h(\boldsymbol{\theta})\nabla^2_{\theta'}h(\boldsymbol{\theta}')d\boldsymbol{\theta} \\ &= -\mathbb{E}_{h(\theta)}[\nabla^2_{\theta'}\log h(\boldsymbol{\theta}')]\end{aligned} \tag{23}$$

and evaluated at $\boldsymbol{\theta}' = \boldsymbol{\theta}$, it results in

$$\nabla^2_{\theta'} \mathbb{D}_{KL}(h(\boldsymbol{\theta}) \,\|\, h(\boldsymbol{\theta}'))|_{\theta'=\theta} = -\mathbb{E}_{h(\theta)}[\nabla^2_\theta \log h(\boldsymbol{\theta})]. \tag{24}$$

With all these results, we can write

$$\mathbb{D}_{KL}(h(\boldsymbol{\theta}) \,\|\, h(\boldsymbol{\theta}')) \approx -\frac{1}{2}(\boldsymbol{\theta}' - \boldsymbol{\theta})^T \mathbb{E}_{h(\theta)}[\nabla^2_\theta \log h(\boldsymbol{\theta})](\boldsymbol{\theta}' - \boldsymbol{\theta}). \tag{25}$$

It is important to note, that, as mentioned before, we evaluate at $\boldsymbol{\theta}' = \boldsymbol{\theta}$ since we are interested in infinitesimal small differences in parameter values. Therefore, the KL divergence is locally approximately symmetric which we already learned is generally not the case. This is relevant because as a consequence, the second derivative of the KL divergence is the same when both distributions match and can thereby be used to define a metric.

We now define a metric which we can apply to our derivation of the direction of steepest descent.

Definition 2.2. (Fisher information) Let the Fisher information $\mathbf{F}(\boldsymbol{\theta})$ be defined by the curvature of the KL divergence, so

$$\begin{aligned} \mathbf{F}(\boldsymbol{\theta}) :&= \nabla^2_{\theta'} \mathbb{D}_{KL}(h(\boldsymbol{\theta}) \,\|\, h(\boldsymbol{\theta}'))|_{\theta'=\theta} \\ &= -\mathbb{E}_{h(\theta)}[\nabla^2_\theta \log h(\boldsymbol{\theta})]. \end{aligned} \tag{26}$$

An alternative form of the the Fisher information can be derived through the following algebraic manipulations:

$$\begin{aligned} \mathbf{F}(\boldsymbol{\theta}) &= -\mathbb{E}_{h(\theta)}[\nabla^2_\theta \log h(\boldsymbol{\theta})] \\ &= -\mathbb{E}_{h(\theta)}[\nabla_\theta (\frac{1}{h(\boldsymbol{\theta})} \nabla_\theta h(\boldsymbol{\theta}))] \\ &= -\mathbb{E}_{h(\theta)}[-\frac{1}{h(\boldsymbol{\theta})} \nabla_\theta h(\boldsymbol{\theta}) \frac{1}{h(\boldsymbol{\theta})} \nabla_\theta h(\boldsymbol{\theta})^T + \frac{1}{h(\boldsymbol{\theta})} \nabla^2_\theta h(\boldsymbol{\theta})] \\ &= \mathbb{E}_{h(\theta)}[\nabla_\theta \log h(\boldsymbol{\theta}) \nabla_\theta \log h(\boldsymbol{\theta})^T] - \mathbb{E}_{h(\theta)}[\frac{1}{h(\boldsymbol{\theta})} \nabla^2_\theta h(\boldsymbol{\theta})] \\ &= \mathbb{E}_{h(\theta)}[\nabla_\theta \log h(\boldsymbol{\theta}) \nabla_\theta \log h(\boldsymbol{\theta})^T]. \end{aligned} \tag{27}$$

The last step uses $\mathbb{E}_{h(\theta)}[\frac{1}{h(\boldsymbol{\theta})} \nabla^2_\theta h(\boldsymbol{\theta})] = 0$ which can be derived analogously to equation (22).

Finally, we can go back to our original objective, that is, finding the direction of steepest descent under the assumption of a norm $\|\cdot\|$. Before, we have considered the Euclidean norm. Now, let us consider a non-orthonormal coordinate system, the Riemannian space, where the squared norm with the so-called Riemannian metric tensor, which can be written as a symmetric, positive definite matrix $\mathbf{G}$, is given by the quadratic form

$$\|\mathbf{a}\|^2 = \sum_{i,j} g_{i,j} a_i a_j = \mathbf{a}^T \mathbf{G} \mathbf{a}. \tag{28}$$

Since we have already defined a suitable metric, the Fisher information $\mathbf{F}$, we can choose $\mathbf{G} = \mathbf{F}$ and plug the corresponding norm into the Lagrangian formulation of our constrained optimisation problem in equation (17):

$$\frac{\partial}{\partial a_i} \nabla_\theta \ell(\boldsymbol{\theta})^T \mathbf{a} - \lambda \mathbf{a}^T \mathbf{F} \mathbf{a} = 0 \tag{29}$$

If we now solve for $\mathbf{a}$, we get

$$\mathbf{a} = \frac{1}{2\lambda}\mathbf{F}^{-1}\nabla_\theta \ell(\boldsymbol{\theta}). \tag{30}$$

We absorb the constant $\frac{1}{2\lambda}$ in the learning rate α and get

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha\mathbf{F}^{-1}\nabla_\theta \ell(\boldsymbol{\theta}), \tag{31}$$

called natural gradient descent. Note, that the natural gradient $\tilde{\nabla} = \mathbf{F}^{-1}\nabla$ is equal to the regular gradient ∇ for an orthonormal Euclidean coordinate system since $\mathbf{G}$ would be the identity matrix.

To recap, we have seen that regular gradient descent implicitly uses the Euclidean distance to find the direction of steepest descent. When we optimise in the space of probability distributions this is not an appropriate distance measure, so we use a different metric, the Fisher information, defined by the curvature of the KL divergence which is locally a more appropriate dissimilarity measure between probability distributions. This results in natural gradient descent, where the natural gradient is the inverse Fisher information matrix multiplied by the classical gradient. Thereby, the algorithm utilises the local geometry of the loss landscape to find the direction of steepest descent. The motivation is that this might lead to faster convergence and that it might be easier to overcome local minima in the loss landscape than with classical gradient descent methods; this is supported by some evidence, e.g. Amari [1998].

3 Methods

In this section, we use the results presented in section 2 to derive natural gradient variational inference, and in particular VOGN. Moreover, we show how to use VOGN for continual learning, how to scale it to deep neural networks and bigger datasets, and potential ways to improve it by considering structured approximations to the covariance matrix.

3.1 Natural Gradient Variational Inference

At this point, we can put all the previously discussed methods together: Exact Bayesian inference with neural networks is generally intractable, therefore we want to perform approximate Bayesian inference. We choose to apply variational inference and want to optimise the ELBO through gradient descent. Since classical gradient descent implicitly assumes an Euclidean parameter space even though probability distributions lay on a Riemannian manifold, we use natural gradient descent which respects the information geometry of this manifold.

This results in natural gradient variational inference (NGVI). While there have been earlier results on applying natural gradient descent to variational inference, we follow Khan and Lin [2017] and in particular the extension to practical deep learning algorithms in Khan et al. [2018].

First, we will introduce the general algorithm proposed in Khan and Lin [2017] which assumes the variational distribution $q(\boldsymbol{\theta})$ to be part of the exponential family and can be applied to many models beyond neural networks.

The exponential family is a set of many widely used models such as the set of exponential, Beta, Gamma, Bernoulli, and Poisson distributions. It is formulated as a general parametric form for all those distributions and has convenient algebraic properties. A member of an exponential family takes the following form:

$$q(\mathbf{z}|\boldsymbol{\lambda}) = h(\mathbf{z}) \exp\left(\langle \boldsymbol{\lambda}, \boldsymbol{\phi}(\mathbf{z}) \rangle - A(\boldsymbol{\lambda})\right) \tag{32}$$

where $\mathbf{z}$ is a random variable, $h(\mathbf{z})$ is a positive function called base measure, $\boldsymbol{\phi}$ is a vector of sufficient statistics, $\boldsymbol{\lambda}$ is a vector of natural parameters, $\langle \cdot, \cdot \rangle$ is an inner product, and $A(\cdot)$ is the log-partition function which is automatically determined once the other functions are set since it is normalising the distribution.

The detailed properties go beyond the scope of this work (see Wainwright and Jordan [2008] Chapter 3 for more details); the important point is that we can parameterise any member of the exponential family with natural parameters. A property called minimality means that there is a unique mapping of the natural parameter $\boldsymbol{\lambda}$ to the expectation or mean parameter $\mathbf{m}$. The expectation parameters are the expectation of the sufficient statistics of the distribution.

Now we can state a result from Raskutti and Mukherjee [2015]: If we assume minimality, the natural gradient with regards to $\boldsymbol{\lambda}$ is equal to the regular gradient with regards to $\mathbf{m}$:

$$\tilde{\nabla}_\lambda \mathcal{L}(\boldsymbol{\lambda}) = \mathbf{F}^{-1} \nabla_\lambda \mathcal{L}(\boldsymbol{\lambda}) = \nabla_m \mathcal{L}_*(\mathbf{m}) \tag{33}$$

where $\mathcal{L}(\lambda)$ is the ELBO from equation 9 with variational distribution $q_\lambda(\boldsymbol{\theta})$ and $\mathcal{L}_*(\mathbf{m})$ is a formulation of the same objective in terms of $\mathbf{m}$ (its existence is guaranteed by the minimality assumption).

With this equality, we can now formulate the natural gradient update in the natural parameter space by taking the gradient with regards to $\mathbf{m}$.

$$\boldsymbol{\lambda}_{t+1} = \boldsymbol{\lambda}_t + \beta_t \nabla_m \mathcal{L}_*(\mathbf{m}_t) \tag{34}$$

where β_t is the step size and t the iteration.

In the next step we derive the update equations for the case where the variational distribution $q_\lambda(\boldsymbol{\theta})$ is a Gaussian distribution $\mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ with mean vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$.

The expectation parameters $\mathbf{m}$ of a Gaussian are simply the first and second order moments of a Gaussian random variable. With this, the definitions of the natural and expectation parameters of a Gaussian distribution are as follows:

$$\begin{aligned}
\boldsymbol{\lambda}^{(1)} &= \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu} \\
\boldsymbol{\lambda}^{(2)} &= -\frac{1}{2}\boldsymbol{\Sigma}^{-1} \\
\mathbf{m}^{(1)} &= \mathbb{E}_{q(\theta)}\big[\boldsymbol{\theta}\big] = \boldsymbol{\mu} \\
\mathbf{M}^{(2)} &= \mathbb{E}_{q(\theta)}\big[\boldsymbol{\theta}\boldsymbol{\theta}^T\big] = \boldsymbol{\mu}\boldsymbol{\mu}^T + \boldsymbol{\Sigma}
\end{aligned} \tag{35}$$

Since we want to optimize the mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$ of $q_\lambda(\boldsymbol{\theta})$, we want to express the gradient with respect to $\mathbf{m}^{(1)}$ and $\mathbf{M}^{(2)}$ in terms of the gradients with respect to $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$. First we write $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ in terms of $\mathbf{m}^{(1)}$ and $\mathbf{M}^{(2)}$:

$$\begin{aligned}
\boldsymbol{\mu} &= \mathbf{m}^{(1)} \\
\boldsymbol{\Sigma} &= \mathbf{M}^{(2)} - \boldsymbol{\mu}\boldsymbol{\mu}^T
\end{aligned} \tag{36}$$

Now we can rewrite the gradients using the chain rule:

$$\begin{aligned}
\nabla_{m^{(1)}}\mathcal{L}_* &= \nabla_\mu\mathcal{L} - 2\big[\nabla_\Sigma\mathcal{L}\big]\boldsymbol{\mu} \\
\nabla_{M^{(2)}}\mathcal{L}_* &= \nabla_\Sigma\mathcal{L}
\end{aligned} \tag{37}$$

To derive the final update equations we can simply plug in these gradients and the definition of the natural parameters $\boldsymbol{\lambda}^{(1)}$ and $\boldsymbol{\lambda}^{(2)}$ in equation 34. We will first derive the update for $\boldsymbol{\Sigma}^{-1}$ since $\boldsymbol{\Sigma}_{t+1}^{-1}$ is necessary to update $\boldsymbol{\mu}_t$.

$$\begin{aligned}
-\frac{1}{2}\boldsymbol{\Sigma}_{t+1}^{-1} &= -\frac{1}{2}\boldsymbol{\Sigma}_t^{-1} + \beta_t\nabla_\Sigma\mathcal{L}_t \\
\iff \boldsymbol{\Sigma}_{t+1}^{-1} &= \boldsymbol{\Sigma}_t^{-1} - 2\beta_t\nabla_\Sigma\mathcal{L}_t
\end{aligned} \tag{38}$$

Now we derive the update for $\boldsymbol{\mu}$ in the same way.

$$\begin{aligned}
\boldsymbol{\Sigma}_{t+1}^{-1}\boldsymbol{\mu}_{t+1} &= \boldsymbol{\Sigma}_t^{-1}\boldsymbol{\mu}_t + \beta_t(\nabla_\mu\mathcal{L}_t - 2\big[\nabla_\Sigma\mathcal{L}_t\big]\boldsymbol{\mu}_t) \\
\iff \boldsymbol{\mu}_{t+1} &= \boldsymbol{\Sigma}_{t+1}\big[\boldsymbol{\Sigma}_t^{-1}\boldsymbol{\mu}_t + \beta_t(\nabla_\mu\mathcal{L}_t - 2\big[\nabla_\Sigma\mathcal{L}_t\big]\boldsymbol{\mu}_t)\big] \\
&= \boldsymbol{\Sigma}_{t+1}\big[\boldsymbol{\Sigma}_t^{-1}\boldsymbol{\mu}_t + \beta_t\nabla_\mu\mathcal{L}_t - 2\beta_t\big[\nabla_\Sigma\mathcal{L}_t\big]\boldsymbol{\mu}_t\big] \\
&= \boldsymbol{\Sigma}_{t+1}\big[(\boldsymbol{\Sigma}_t^{-1} - 2\beta_t\big[\nabla_\Sigma\mathcal{L}_t\big])\boldsymbol{\mu}_t + \beta_t\nabla_\mu\mathcal{L}_t\big] \\
&= \boldsymbol{\Sigma}_{t+1}\big[\boldsymbol{\Sigma}_{t+1}^{-1}\boldsymbol{\mu}_t + \beta_t\nabla_\mu\mathcal{L}_t\big] \\
&= \boldsymbol{\mu}_t + \beta_t\boldsymbol{\Sigma}_{t+1}\nabla_\mu\mathcal{L}_t
\end{aligned} \tag{39}$$

These are the natural gradient updates for the mean $\boldsymbol{\mu}$ and precision matrix $\boldsymbol{\Sigma}^{-1}$ of a Gaussian.

At this point we could simply choose a mean-field approximation and use commonly used stochastic gradient methods described in section 2.2 to compute the gradients $\nabla_\mu\mathcal{L}$ and $\nabla_\Sigma\mathcal{L}$. In contrast, the crucial point about the approach described in Khan et al. [2018] is that we only want to use the gradients of

$$f(\boldsymbol{\theta}) := \frac{1}{N}\sum_{i=1}^{N} f_i(\boldsymbol{\theta}) = -\frac{1}{N}\sum_{i=1}^{N} \log p(y_i|\boldsymbol{\theta}, \mathbf{x}_i), \tag{40}$$

where N is the number of data points; this is the maximum likelihood objective commonly used in deep learning. To achieve this, we rewrite the ELBO in 9 as a function of $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ as

$$\mathcal{L}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) := \mathbb{E}_{q(\theta)}\big[-Nf(\boldsymbol{\theta}) + \log p(\boldsymbol{\theta}) - \log q(\boldsymbol{\theta})\big], \tag{41}$$

where the prior $p(\boldsymbol{\theta})$ is also a Gaussian distribution $\mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\mu}_p, \boldsymbol{\Sigma}_p)$.

Now there is a way to replace the gradients $\nabla_\mu \mathcal{L}$ and $\nabla_\Sigma \mathcal{L}$ with an expression which only depends on the gradient $\nabla_\theta f(\boldsymbol{\theta})$ and the Hessian $\nabla^2_\theta f(\boldsymbol{\theta})$ with the help of Bonnet's and Price's theorems [Opper and Archambeau, 2009, Rezende et al., 2014]:

$$\begin{aligned} \nabla_\mu \mathbb{E}_{q(\theta)}\big[f(\boldsymbol{\theta})\big] &= \mathbb{E}_{q(\theta)}\big[\nabla_\theta f(\boldsymbol{\theta})\big] \\ \nabla_\Sigma \mathbb{E}_{q(\theta)}\big[f(\boldsymbol{\theta})\big] &= \frac{1}{2}\mathbb{E}_{q(\theta)}\big[\nabla^2_\theta f(\boldsymbol{\theta})\big] \end{aligned} \tag{42}$$

Using this equality, we can write

$$\begin{aligned} \nabla_\mu \mathcal{L} &= \nabla_\mu \mathbb{E}_{q(\theta)}\big[-Nf(\boldsymbol{\theta}) + \log p(\boldsymbol{\theta}) - \log q(\boldsymbol{\theta})\big] \\ &= \mathbb{E}_{q(\theta)}\big[-N\nabla_\theta f(\boldsymbol{\theta})\big] + \mathbb{E}_{q(\theta)}\big[\nabla_\theta \log p(\boldsymbol{\theta})\big] - \mathbb{E}_{q(\theta)}\big[\nabla_\theta \log q(\boldsymbol{\theta})\big] \\ &= \mathbb{E}_{q(\theta)}\big[-N\nabla_\theta f(\boldsymbol{\theta})\big] + \mathbb{E}_{q(\theta)}\big[\nabla_\theta \log p(\boldsymbol{\theta})\big] \\ &= \mathbb{E}_{q(\theta)}\big[-N\nabla_\theta f(\boldsymbol{\theta})\big] + \mathbb{E}_{q(\theta)}\big[\nabla_\theta - \frac{1}{2}(\boldsymbol{\theta}-\boldsymbol{\mu}_p)^T\boldsymbol{\Sigma}_p^{-1}(\boldsymbol{\theta}-\boldsymbol{\mu}_p)\big] \\ &= \mathbb{E}_{q(\theta)}\big[-N\nabla_\theta f(\boldsymbol{\theta})\big] - \mathbb{E}_{q(\theta)}\big[\boldsymbol{\Sigma}_p^{-1}(\boldsymbol{\theta}-\boldsymbol{\mu}_p)\big] \\ &= \mathbb{E}_{q(\theta)}\big[-N\nabla_\theta f(\boldsymbol{\theta})\big] - \boldsymbol{\Sigma}_p^{-1}(\mathbb{E}_{q(\theta)}\big[\boldsymbol{\theta}\big] - \boldsymbol{\mu}_p) \\ &= -\big[N\mathbb{E}_{q(\theta)}\big[\nabla_\theta f(\boldsymbol{\theta})\big] + \boldsymbol{\Sigma}_p^{-1}(\boldsymbol{\mu}-\boldsymbol{\mu}_p)\big] \end{aligned} \tag{43}$$

and

$$\begin{aligned} \nabla_\Sigma \mathcal{L} &= \nabla_\Sigma \mathbb{E}_{q(\theta)}\big[-Nf(\boldsymbol{\theta}) + \log p(\boldsymbol{\theta}) - \log q(\boldsymbol{\theta})\big] \\ &= \frac{1}{2}\mathbb{E}_{q(\theta)}\big[-N\nabla^2_\theta f(\boldsymbol{\theta})\big] + \frac{1}{2}\mathbb{E}_{q(\theta)}\big[\nabla^2_\theta \log p(\boldsymbol{\theta})\big] - \frac{1}{2}\mathbb{E}_{q(\theta)}\big[\nabla^2_\theta \log q(\boldsymbol{\theta})\big] \\ &= \frac{1}{2}\mathbb{E}_{q(\theta)}\big[-N\nabla^2_\theta f(\boldsymbol{\theta})\big] - \frac{1}{2}\mathbb{E}_{q(\theta)}\big[\nabla_\theta \boldsymbol{\Sigma}_p^{-1}(\boldsymbol{\theta}-\boldsymbol{\mu}_p)\big] + \frac{1}{2}\mathbb{E}_{q(\theta)}\big[\nabla_\theta \boldsymbol{\Sigma}^{-1}(\boldsymbol{\theta}-\boldsymbol{\mu})\big] \\ &= \frac{1}{2}\mathbb{E}_{q(\theta)}\big[-N\nabla^2_\theta f(\boldsymbol{\theta})\big] - \frac{1}{2}\boldsymbol{\Sigma}_p^{-1} + \frac{1}{2}\boldsymbol{\Sigma}^{-1}. \end{aligned} \tag{44}$$

By plugging in these gradients in the NGVI updates from equations 38 and 39 we get

$$\begin{aligned} \boldsymbol{\Sigma}_{t+1}^{-1} &= \boldsymbol{\Sigma}_t^{-1} - 2\beta_t \nabla_\Sigma \mathcal{L}_t \\ &= \boldsymbol{\Sigma}_t^{-1} - \beta_t\big[\mathbb{E}_{q(\theta)}\big[-N\nabla^2_\theta f(\boldsymbol{\theta})\big] - \boldsymbol{\Sigma}_p^{-1} + \boldsymbol{\Sigma}_t^{-1}\big] \\ &= (1-\beta_t)\boldsymbol{\Sigma}_t^{-1} + \beta_t\big[N\mathbb{E}_{q(\theta)}\big[\nabla^2_\theta f(\boldsymbol{\theta})\big] + \boldsymbol{\Sigma}_p^{-1}\big] \end{aligned} \tag{45}$$

and

$$\begin{aligned} \boldsymbol{\mu}_{t+1} &= \boldsymbol{\mu}_t + \beta_t \boldsymbol{\Sigma}_{t+1} \nabla_\mu \mathcal{L}_t \\ &= \boldsymbol{\mu}_t - \beta_t \boldsymbol{\Sigma}_{t+1}\big[N\mathbb{E}_{q(\theta)}\big[\nabla_\theta f(\boldsymbol{\theta})\big] + \boldsymbol{\Sigma}_p^{-1}(\boldsymbol{\mu}_t - \boldsymbol{\mu}_p)\big]. \end{aligned} \tag{46}$$

We still have to approximate the expectations of the gradient and the Hessian of $f(\boldsymbol{\theta})$ with Monte Carlo samples, which means using the law of large numbers to

approximate an expectation of a function $\mathbb{E}_{q(\theta)}\big[f(\boldsymbol{\theta})\big]$ of a random variable $\boldsymbol{\theta}$ by the mean of the function evaluated at samples $\boldsymbol{\theta}_i$ from the distribution $q(\boldsymbol{\theta})$:

$$\mathbb{E}_{q(\theta)}\big[f(\boldsymbol{\theta})\big] \approx \frac{1}{K}\sum_{i=1}^{K} f(\boldsymbol{\theta}_i) \tag{47}$$

For the sake of simplicity we only use one sample $\boldsymbol{\theta}_t \sim \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$ at this point, even though we will use more in practice.

With this approximation of the expectations, the final updates are

$$\begin{aligned}\boldsymbol{\Sigma}_{t+1}^{-1} &= (1-\beta_t)\boldsymbol{\Sigma}_t^{-1} + \beta_t\big[N\nabla_\theta^2 f(\boldsymbol{\theta}_t) + \boldsymbol{\Sigma}_p^{-1}\big] \\ \boldsymbol{\mu}_{t+1} &= \boldsymbol{\mu}_t - \beta_t\boldsymbol{\Sigma}_{t+1}\big[N\nabla_\theta f(\boldsymbol{\theta}_t) + \boldsymbol{\Sigma}_p^{-1}(\boldsymbol{\mu}_t - \boldsymbol{\mu}_p)\big].\end{aligned} \tag{48}$$

By replacing the gradient and Hessian with their stochastic mini batch approximations $\hat{\mathbf{g}}(\boldsymbol{\theta}_t) := \frac{N}{M}\sum_{i=1}^{M}\nabla_\theta f_i(\boldsymbol{\theta}_t)$ and $\hat{\mathbf{H}}(\boldsymbol{\theta}_t) := \frac{N}{M}\sum_{i=1}^{M}\nabla_\theta^2 f_i(\boldsymbol{\theta}_t)$, we get

$$\begin{aligned}\boldsymbol{\Sigma}_{t+1}^{-1} &= (1-\beta_t)\boldsymbol{\Sigma}_t^{-1} + \beta_t\big[\hat{\mathbf{H}}(\boldsymbol{\theta}_t) + \boldsymbol{\Sigma}_p^{-1}\big] \\ \boldsymbol{\mu}_{t+1} &= \boldsymbol{\mu}_t - \beta_t\boldsymbol{\Sigma}_{t+1}\big[\hat{\mathbf{g}}(\boldsymbol{\theta}_t) + \boldsymbol{\Sigma}_p^{-1}(\boldsymbol{\mu}_t - \boldsymbol{\mu}_p)\big].\end{aligned} \tag{49}$$

To summarise, we now have updates that can perform NGVI with the gradient and Hessian of the ML objective instead of the gradients of the ELBO.

Computing the stochastic gradient $\hat{\mathbf{g}}(\boldsymbol{\theta}_t)$ is common practice in deep learning. Even the stochastic approximation of the Hessian $\hat{\mathbf{H}}(\boldsymbol{\theta}_t)$ could be computed by automatic-differentiation software but the problem is that it has to be a positive definite matrix to qualify as a covariance matrix of a non-degenerate Gaussian, i.e. a valid probability density function. The same holds for computing the Hessian with the reparameterisation trick [Kingma and Welling, 2013]. To enforce the constraint on the Hessian, a simple backtracking method could be used, see Appendix D.1. in Khan et al. [2018] for details.

3.2 Variational Online Gauss-Newton (VOGN)

Here, we choose a different approach to deal with this issue. Instead of the Hessian, we write

$$\hat{\mathbf{H}}(\boldsymbol{\theta}_t) \approx \frac{N}{M}\sum_{i=1}^{M}\nabla_\theta f_i(\boldsymbol{\theta}_t)\nabla_\theta f_i(\boldsymbol{\theta}_t)^T =: \hat{\mathbf{F}}(\boldsymbol{\theta}_t) \tag{50}$$

which is an approximation of the Fisher information matrix introduced in equation 27, called the empirical Fisher information. This method is also known as Generalized Gauss-Newton [Schraudolph, 2002, Martens, 2014]. Even though the empirical Fisher is widely used because it is easy to compute as the sum of the outer products of the individual gradients, its theoretical justification is limited. This suggests that there are still gaps in the understanding of methods using the empirical Fisher which are successful in practice. For an excellent discussion on these limitations of the empirical Fisher, see Kunstner et al. [2019].

Finally, using this approximation for the Hessian, the update for $\boldsymbol{\Sigma}^{-1}$ becomes

$$\boldsymbol{\Sigma}_{t+1}^{-1} = (1-\beta_t)\boldsymbol{\Sigma}_t^{-1} + \beta_t\big[\hat{\mathbf{F}}(\boldsymbol{\theta}_t) + \boldsymbol{\Sigma}_p^{-1}\big]. \tag{51}$$

The shape of $\hat{\mathbf{F}}$ is $D \times D$ since it is the outer product of the individual gradients $\nabla_\theta f_i(\boldsymbol{\theta}_t)$ with the shape $D\times 1$. Storing this matrix takes $O(D^2)$ memory and inverting it takes $O(D^3)$ computations. In a neural network D is the number of parameters

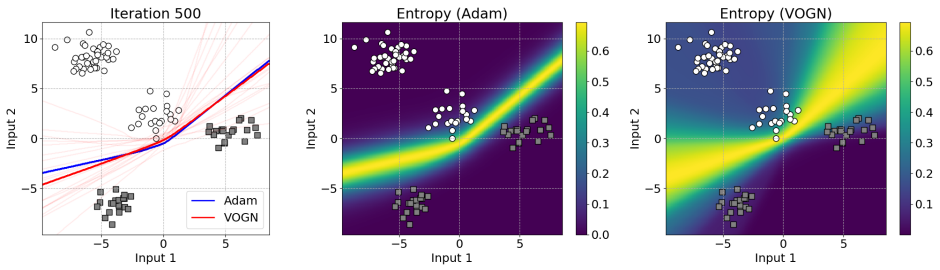

Figure 1: Adam and VOGN on a 2d toy example. The red line indicates the decision boundary based on predictions made with the mean of the posterior distribution obtained by VOGN; the light red lines are predictions made with samples of the posterior. VOGN finds a similar optimum as Adam (blue line) but additionally provides an uncertainty estimate. Code available at `https://github.com/team-approx-bayes/dl-with-bayes/tree/master/toy_example`.

which can be in the millions. Therefore, it is unfeasible to use the full empirical Fisher $\hat{\mathbf{F}}$ for common deep learning models.

As described in section 2.1.3, we constrain the variational distribution $q(\boldsymbol{\theta})$ and the prior $p(\boldsymbol{\theta})$ to have a diagonal precision matrix $\boldsymbol{\Sigma}^{-1} = \text{diag}(\boldsymbol{\sigma}^{-2})$ and prior precision $\boldsymbol{\Sigma}_p^{-1} = \text{diag}(\boldsymbol{\sigma}_p^{-2})$ respectively. This is equivalent to having a univariate Gaussian distribution over each parameter θ_i. By choosing this approximation, we only have to store and invert the vector $\boldsymbol{\sigma}^2$. In section 3.5 we are also briefly introducing structured approximations which might lead to better convergence properties and potentially better continual learning performance.

The version of the NGVI updates with the empirical Fisher as an approximation to the Hessian and a diagonal covariance matrix for $q(\boldsymbol{\theta})$ and $p(\boldsymbol{\theta})$ is called Variational Online Gauss-Newton (VOGN):

$$\begin{aligned} \boldsymbol{\sigma}_{t+1}^{-2} &= (1-\beta_t)\boldsymbol{\sigma}_t^{-2} + \beta_t\big[\text{diag}(\hat{\mathbf{F}}(\boldsymbol{\theta}_t)) + \boldsymbol{\sigma}_p^{-2}\big] \\ \boldsymbol{\mu}_{t+1} &= \boldsymbol{\mu}_t - \alpha_t\boldsymbol{\sigma}_{t+1}^2\big[\hat{\mathbf{g}}(\boldsymbol{\theta}_t) + \boldsymbol{\sigma}_p^{-2}(\boldsymbol{\mu}_t - \boldsymbol{\mu}_p)\big] \end{aligned} \tag{52}$$

where $\text{diag}(\hat{\mathbf{F}}(\boldsymbol{\theta}_t)) = \frac{N}{M}\sum_{i=1}^{M}(\nabla_\theta f_i(\boldsymbol{\theta}_t))^2$, with $(\cdot)^2$ being an element-wise operation. Since we have made several approximations, the updates are not exact NGVI. For this reason, having two distinct learning rates, α_t and β_t which can be tuned separately might be beneficial in practice.

A scalable implementation which was used in Osawa et al. [2019] and can be used as a plug-and-play optimiser in PyTorch is available at `https://github.com/cybertronai/pytorch-sso`.

At this point, it is worth pointing out how similar the updates of VOGN and Adam are. Since we use a slightly different notation for each method, it might not be completely obvious at first. The main differences are that VOGN uses a sample $\boldsymbol{\theta}_t$ (in practice multiple samples) from the current variational distribution to compute the stochastic mini-batch gradients $\hat{\mathbf{g}}(\boldsymbol{\theta}_t)$ and since Adam is a deterministic method, it will only use the gradient at the current values of the parameters. See Figure 1 for an illustration of the different behaviour of VOGN and Adam. The sampling makes VOGN slightly slower than deterministic methods like Adam or SGD.

Also due to being a Bayesian method, VOGN uses a prior distribution; as a consequence, the prior precision $\boldsymbol{\sigma}_p^{-2}$ is added in the update of the precision and $\boldsymbol{\sigma}_p^{-2}(\boldsymbol{\mu}_t - \boldsymbol{\mu}_p)$ is added to the gradient in the update of the mean.

Moreover, Adam uses the squared mini-batch gradient $\hat{\mathbf{g}}^2$ for the computation of the exponential moving average $\mathbf{v}$ while VOGN uses $\text{diag}(\hat{\mathbf{F}}(\boldsymbol{\theta}_t))$ which is the scaled

sum of squares of the individual gradients, i.e. this would only be equivalent for $M = 1$. Since common deep learning frameworks like PyTorch do not provide access to the individual gradients, this is the main source of additional computational complexity of VOGN compared to Adam. Note, that this is simply due to the implementation of currently available tools and not due to theoretical reasons. For a detailed description of how the individual gradients for linear, convolutional, and batch norm layers can be obtained, see Appendix B in Osawa et al. [2019] (based on Goodfellow [2015], which only works for linear layers).

Additionally, Adam uses the square root of the inverse of $\mathbf{v}$ when preconditioning the gradient. Lastly, VOGN does not naturally have an exponential moving average over the gradient and a bias correction which are used in Adam. A variant of VOGN which resembles Adam as closely as possible while still being an approximate Bayesian method has also been introduced together with VOGN and is called Vadam [Khan et al., 2018].

This close connection of these two methods is quite surprising considering the fact that Adam has been developed by largely empirical efforts while VOGN has been derived in a principled manner.

3.3 VOGN for Continual Learning

Now we want to apply VOGN to continual learning, as described in section 2.1.2. This has already been proposed in Tseran et al. [2018] but without any experimental results. Since the only thing that changes is the choice of the prior distribution, the updates remain almost unchanged. To adjust the notation, we introduce an index denoting the task k:

$$\begin{aligned} \boldsymbol{\sigma}_{k,t+1}^{-2} &= (1-\beta)\boldsymbol{\sigma}_{k,t}^{-2} + \beta\big[\mathrm{diag}(\hat{\mathbf{F}}(\boldsymbol{\theta}_{k,t})) + \boldsymbol{\sigma}_{k-1,t_{max}}^{-2}\big] \\ \boldsymbol{\mu}_{k,t+1} &= \boldsymbol{\mu}_{k,t} - \alpha\boldsymbol{\sigma}_{k,t}^{2}\big[\hat{\mathbf{g}}(\boldsymbol{\theta}_{k,t}) + \boldsymbol{\sigma}_{k-1,t_{max}}^{-2}(\boldsymbol{\mu}_{k,t} - \boldsymbol{\mu}_{k-1,t_{max}})\big] \end{aligned} \tag{53}$$

The prior mean $\boldsymbol{\mu}_{k-1,t_{max}}$ and the prior precision $\boldsymbol{\sigma}_{k-1,t_{max}}^{-2}$ for training on task k are the posterior mean and precision of training on all tasks up to $k-1$; t_{max} denotes the last iteration of training on a task.

The prior mean $\boldsymbol{\mu}_0$ and the prior precision $\boldsymbol{\sigma}_0^{-2}$ are the parameters of the prior distribution specified before training on the first task. Usually, $\boldsymbol{\mu}_0 = \mathbf{0}$ and $\boldsymbol{\sigma}_0^{-2} = c\mathbf{1}$ with $c \in \mathbb{R}_{>0}$ and $\mathbf{0}, \mathbf{1}$ are D-dimensional vectors, with every element equal to zero and one respectively.

These updates are optimising the ELBO for continual learning $\mathcal{L}_{CL}$ in equation 10.

3.4 VOGN for Practical Deep Learning

One thing that is important to point out is that VOGN can be simply used as a replacement for an optimiser like Adam, i.e. in principle by changing one line of code. This means that VOGN can be used with any kind of neural network architecture, while other Bayesian deep learning methods like BBB require a custom architecture.

Thereby, the main advantage of VOGN compared to other Bayesian Deep Learning methods such as MC-dropout or Bayes-by-Backprop is that it is theoretically principled, preserves the advantages of Bayesian inference to some extend while being practically usable with deep neural networks on large datasets. This has been shown in Osawa et al. [2019] which contains results for VOGN with ResNet-18 on ImageNet with comparable performance to SGD and Adam. At the same time out-of-distribution uncertainties are improved, predictive probabilities are well-calibrated, and continual learning is enabled. Here, we briefly discuss which techniques can be

applied to make VOGN scale to practically relevant deep learning problems, as demonstrated in Osawa et al. [2019].

To make VOGN useful in practice, common deep learning techniques can be used, i.e. momentum, initialisation best practices, and data augmentation [Sutskever et al., 2013], learning rate scheduling [Goyal et al., 2017], batch normalisation [Ioffe and Szegedy, 2015], and distributed training. Regarding data augmentation, there is a subtle difference when using VOGN. Since in the updates shown in equation 53 $\text{diag}(\hat{\mathbf{F}}(\boldsymbol{\theta}_t))$ and $\hat{\mathbf{g}}(\boldsymbol{\theta}_k)$ are explicitly scaled by the dataset size N, we need to introduce a new scalar hyperparameter ρ to adjust the dataset size to account for the effectively increased number of training samples. It is not clear how to exactly determine ρ but in practice it does not seem to matter as long as the order of magnitude is correct. For more details on using VOGN with deep neural networks and big datasets, see Osawa et al. [2019].

Note, that this scalability and ease of use of VOGN is very desirable for a continual learning method. VOGN can now be applied to all kinds of continual learning problems. How well it performs and how much hyperparameter tuning is necessary is still an open question but we see promising results in section 4.

3.5 Structured Covariance Approximations

One potential direction of further research on improving VOGN in general and VOGN for continual learning in particular, are structured, i.e. non-diagonal, approximations to the covariance matrix $\boldsymbol{\Sigma}$. First, we will introduce a 'low-rank plus diagonal' approximation called SLANG which was proposed in Mishkin et al. [2018] and also show how it can be applied to the prior distribution which is necessary for continual learning. Then we will briefly look a variant of Kronecker-Factored Approximate Curvature (K-FAC) [Martens and Grosse, 2015] which was proposed for natural gradient variational inference in Zhang et al. [2018]; however, here we will not derive the form of the updates with a Kronecker-factored prior covariance.

3.5.1 SLANG

Here, we simply consider the updates for mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$ of the variational distribution $q(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ used in the VOGN (see section 3.2), but allow for some kind of non-diagonal structure in $\boldsymbol{\Sigma}$. The method, developed in Mishkin et al. [2018], is called stochastic, low-rank, approximate, natural gradient (SLANG). It approximates the inverse of the covariance matrix by a diagonal plus low-rank structure:

$$\boldsymbol{\Sigma}_t^{-1} \approx \hat{\boldsymbol{\Sigma}}_t^{-1} := \mathbf{U}_t \mathbf{U}_t^\top + \mathbf{D}_t \tag{54}$$

In the following, we will present the update step of $\mathbf{U}$ and $\mathbf{D}$ in SLANG and show how to extend it to a Gaussian prior with the same low-rank plus diagonal structured precision matrix as $\hat{\boldsymbol{\Sigma}}^{-1}$ instead of an isotropic Gaussian prior with scalar precision parameter λ.

$$\hat{\boldsymbol{\Sigma}}_{t+1}^{-1} := \mathbf{U}_{t+1}\mathbf{U}_{t+1}^\top + \mathbf{D}_{t+1} \approx (1-\beta_t)\hat{\boldsymbol{\Sigma}}_t^{-1} + \beta_t\big[\hat{\mathbf{F}}(\boldsymbol{\theta}_t) + \lambda\mathbf{I}\big] \tag{55}$$

This update cannot be performed exactly without potentially increasing the low-rank of $\mathbf{U}_{t+1}$ (see Mishkin et al. [2018] for details). Therefore, we approximate $\mathbf{U}_{t+1}$ with eigenvalue decomposition of rank L:

$$\begin{aligned}(1-\beta_t)\hat{\boldsymbol{\Sigma}}_t^{-1} + \beta_t\big[\hat{\mathbf{F}}(\boldsymbol{\theta}_t) + \lambda\mathbf{I}\big] &= (1-\beta_t)\mathbf{U}_t\mathbf{U}_t^\top + \beta_t\hat{\mathbf{F}}(\boldsymbol{\theta}_t) + (1-\beta_t)\mathbf{D}_t + \beta_t\lambda\mathbf{I} \\ &\approx \mathbf{Q}_{1:L}\boldsymbol{\Lambda}_{1:L}\mathbf{Q}_{1:L}^\top + (1-\beta_t)\mathbf{D}_t + \beta_t\lambda\mathbf{I},\end{aligned} \tag{56}$$

where $\mathbf{\Lambda}_{1:L}$ is a diagonal matrix with the L largest eigenvalues of $(1-\beta_t)\mathbf{U}_t\mathbf{U}_t^\top + \beta_t\hat{\mathbf{F}}(\boldsymbol{\theta}_t)$ on the diagonal and the columns of $\mathbf{Q}_{1:L}$ are the L corresponding eigenvectors.

Now we introduce a Gaussian prior $p(\boldsymbol{\theta}) \sim \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\mu}_p, \mathbf{\Sigma}_p)$ with a precision matrix with low-rank plus diagonal structure:

$$\mathbf{\Sigma}_p^{-1} := \mathbf{U}_p\mathbf{U}_p^\top + \mathbf{D}_p \tag{57}$$

Since there now is an additional low-rank component in the prior precision, the update has to be adjusted. Even though we know the prior precision factor $\mathbf{U}_p$ which is constant over the duration of the task, we cannot explicitly add it to obtain $\mathbf{U}_{t+1}$ which would add an additional approximation. This is essentially the same problem as with online SLANG in Mishkin et al. [2018]. Therefore, we still approximate the whole term with eigenvalue decomposition (differences to SLANG without vector prior mean and low-rank plus diagonal prior precision are marked in red):

$$\begin{aligned}\hat{\mathbf{\Sigma}}_{t+1}^{-1} &\approx (1-\beta_t)\hat{\mathbf{\Sigma}}_t^{-1} + \beta_t\big[\hat{\mathbf{F}}(\boldsymbol{\theta}_t) + \mathbf{\Sigma}_p^{-1}\big] \\ &= (1-\beta_t)\mathbf{U}_t\mathbf{U}_t^\top + \beta_t\big[\hat{\mathbf{F}}(\boldsymbol{\theta}_t) + \mathbf{U}_p\mathbf{U}_p^\top\big] + (1-\beta_t)\mathbf{D}_t + \beta_t\mathbf{D}_p \\ &\approx \mathbf{Q}_{1:L}\mathbf{\Lambda}_{1:L}\mathbf{Q}_{1:L}^\top + (1-\beta_t)\mathbf{D}_t + \beta_t\mathbf{D}_p\end{aligned} \tag{58}$$

Subsequently, $\mathbf{U}_{t+1}$ can be updated as

$$\mathbf{U}_{t+1} = \mathbf{Q}_{1:L}\mathbf{\Lambda}_{1:L}^{1/2} \tag{59}$$

Similarly, we have to slightly change the update of the diagonal component $\mathbf{D}_t$ of the precision. In the case of the diagonal, it is possible to exactly match the diagonal of both side of the equation for each iteration:

$$\operatorname{diag}\Big[\mathbf{U}_{t+1}\mathbf{U}_{t+1}^\top + \mathbf{D}_{t+1}\Big] = \operatorname{diag}\Big[(1-\beta_t)\mathbf{U}_t\mathbf{U}_t^\top + \beta_t\big[\hat{\mathbf{F}}(\boldsymbol{\theta}_t) + \mathbf{U}_p\mathbf{U}_p^\top\big] + (1-\beta_t)\mathbf{D}_t + \beta_t\mathbf{D}_p\Big] \tag{60}$$

This results in the update

$$\mathbf{D}_{t+1} = (1-\beta_t)\mathbf{D}_t + \beta_t\mathbf{D}_p + \mathbf{\Delta}_t \tag{61}$$

with a "diagonal correction"

$$\mathbf{\Delta}_t = \operatorname{diag}\Big[(1-\beta_t)\mathbf{U}_t\mathbf{U}_t^\top + \beta_t\big[\hat{\mathbf{F}}(\boldsymbol{\theta}_t) + \mathbf{U}_p\mathbf{U}_p^\top\big] - \mathbf{U}_{t+1}\mathbf{U}_{t+1}^\top\Big]. \tag{62}$$

Moreover, the update of the mean $\boldsymbol{\mu}$ also has to be slightly modified since the prior mean $\boldsymbol{\mu}_p$ is not zero anymore and the scalar precision λ is now replaced by $\mathbf{\Sigma}_p^{-1}$:

$$\boldsymbol{\mu}_{t+1} = \boldsymbol{\mu}_t - \alpha_t\mathbf{\Sigma}_{t+1}\big[\hat{\mathbf{g}}(\boldsymbol{\theta}_t) + \mathbf{\Sigma}_p^{-1}(\boldsymbol{\mu}_t - \boldsymbol{\mu}_p)\big] \tag{63}$$

With these adjustments, it is possible to use SLANG for continual learning, in the same way we use VOGN in equation 53.

3.5.2 K-FAC

Another possible approximation for the covariance matrix is the Kronecker-Factored Approximate Curvature (K-FAC) [Martens and Grosse, 2015]. As mentioned before, it has been used for continual learning with an online Laplace approximation [Ritter et al., 2018]. The results are promising, especially one experiment, where the authors ran their algorithm on 50 tasks of permuted MNIST while still achieving comparably high average accuracy. On the other hand, no code is available and the paper lacks some crucial details on how the method was implemented.

Besides this application in continual learning, K-FAC has been also applied to NGVI [Zhang et al., 2018], resulting in an algorithm basically identical to VOGN but with a matrix variate posterior [Louizos and Welling, 2016, Sun et al., 2017] with a Kronecker-factored covariance matrix. This variant of VOGN, called noisy K-FAC, is also scalable and achieves comparable performance to VOGN on ImageNet; it converges faster measured in iterations but since it's slower in terms of computation per iteration, the wall clock time is about the same as VOGN [Osawa et al., 2019].

There has been no work published on noisy K-FAC and continual learning. While the update of the mean is easy to adjust, using a Kronecker-factored prior precision might require some more thought. In Ritter et al. [2018], they do not describe their approach in detail but provide some evidence that K-FAC has the potential to improve Bayesian approaches to continual learning which use a simple diagonal covariance approximation. Therefore, it might be a good direction for further research to try to apply noisy K-FAC to continual learning problems.

(a) MNIST (b) CIFAR10

Figure 2: Random samples of all 10 classes of MNIST (2a) and CIFAR10 (2b) (one class per row).

4 Results and Discussion

As mentioned before, we restrict the domain of problems to supervised image classification tasks, which is common for many continual learning approaches. First, we show that the performance of VOGN is competitive to comparable state-of-the-art approaches on two common continual learning benchmarks, permuted [Srivastava et al., 2013, Goodfellow et al., 2013] and split MNIST [Zenke et al., 2017, Nguyen et al., 2018]. Second, we consider a larger continual learning benchmark, called split CIFAR10/100 [Zenke et al., 2017], discuss the issues we encounter, and how we can overcome them.

As far as baselines are concerned, we only compare against methods which have been used on the same benchmarks with the same model architecture and which are also not necessarily using memory or extending the model architecture. We compare against EWC [Kirkpatrick et al., 2017], SI [Zenke et al., 2017], and VCL [Nguyen et al., 2018] on all benchmarks. On permuted MNIST we also show results for UCB [Ebrahimi et al., 2019] and HAT [Serrà et al., 2018] because the results are obtained with the same model architecture we use (even though HAT does not belong to the class of regularisation based methods). On CIFAR10/100 we also compare against two more simple baselines due to the more challenging nature of the benchmark. For more approaches to continual learning, see section 1.1.

All three benchmarks assume that the task boundaries are known, the tasks are not overlapping, and that on each task multiple passes through the data are possible. The hope is that this work can demonstrate the potential of VOGN for continual learning and offer some insights in how adjustments can be made to scale to bigger and harder problems.

All benchmark datasets are split in training and validation set; we present results for the validation set.

4.1 Evaluation Metrics

We use the standard continual learning metric of average accuracy after training on all tasks, backward transfer, as suggested in Lopez-Paz et al. [2017], and forward transfer, which we define differently than Lopez-Paz et al. [2017].

As mentioned in the introduction, continual learning is not only about avoiding

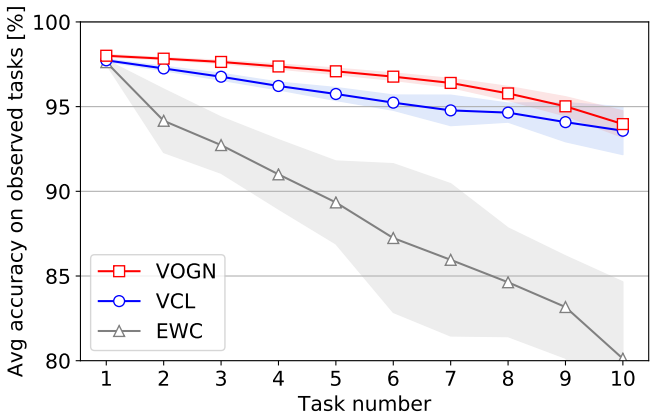

Figure 3: Average accuracy on all tasks up to current task on permuted MNIST. We show the mean as markers and standard deviation as error bands of 20 runs with different random permutations and random seeds for training. VOGN does at least as well as VCL and much better than EWC. Figure adopted from Osawa et al. [2019].

catastrophic forgetting but also about improving performance on previously seen and future data. To formalise these properties, we define backward and forward transfer.

Intuitively, backward transfer measures the average improvement on previous tasks. Note, that it is not clear how to differentiate between backward transfer and a lack of forgetting; maybe it is even impossible to distinguish between the two. As long as there is more forgetting than true backward transfer on average, the backward transfer metric is negative. Forward transfer should measure the average difference the accuracies achieved by a model independently trained from scratch and a model which before was trained on previous tasks, with the previous posterior as prior.

Now we formalise these intuitive descriptions. We train sequentially on K tasks without revisiting old tasks after moving to the next one. After training on task k, we test the accuracy on the validation datasets of all tasks up to k. Let $\mathbf{A} \in \mathbb{R}^{K\times K}$, with $a_{i,j}$ being the validation accuracy on task j after training on task i. This means that $\mathbf{A}$ can be seen as a lower triangular matrix, since we only care about values for $i \geq j$. $\mathbf{s}$ is the vector of test accuracies for every task with the model trained independently from scratch with Adam. Given $\mathbf{A}$ and $\mathbf{s}$, we define the following metrics:

$$\text{Average Accuracy} \qquad ACC = \frac{1}{K}\sum_{i=1}^{K} a_{K,i} \tag{64}$$

$$\text{Backward Transfer} \qquad BWT = \frac{1}{K-1}\sum_{i=1}^{K-1} a_{K,i} - a_{i,i} \tag{65}$$

$$\text{Forward Transfer} \qquad FWT = \frac{1}{K-1}\sum_{i=2}^{K} a_{i,i} - s_i \tag{66}$$

4.2 Datasets

Here, we introduce the datasets which are modified to be used as continual learning benchmarks.

The underlying dataset of permuted and split MNIST is the Modified National Institute of Standards and Technology (MNIST) dataset which contains 60000 train-

Table 1: The average validation accuracy over all tasks of Permuted-MNIST (10 tasks) and Split-MNIST. We report mean and standard deviation over 5 runs for improved VCL and SLANG, 20 runs for VOGN, and use results from Nguyen et al. [2018] for EWC and SI. Results for UCB and HAT are taken from Ebrahimi et al. [2019]. - denotes that no results are known. The best results (within standard deviation) are shown in bold.

Method	**Permuted MNIST**	**Split MNIST**
EWC	84%	63.1%
SI	86%	**98.9**%
UCB	91.4%	-
HAT	91.6%	-
Improved VCL	**93.0**% $\pm$ 1.0	98.4% $\pm$ 0.4
VOGN	**94.0**% $\pm$ 0.8	**98.8**% $\pm$ 0.1
SLANG (L=64)	**93.8**% $\pm$ 1.4	-

ing and 10000 test images [1]. Each image is in grey-scale and 28×28 pixels big and displays a handwritten digit belonging to one of the 10 classes 0 to 9. See Figure 2a for some exemplary images.

The split CIFAR dataset consists of images from CIFAR10 and CIFAR100. The CIFAR10 dataset was created by the Canadian Institute For Advanced Research and contains 60000 $32 \times 32 \times 3$ colour images belonging to ten classes (airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck). The training set contains 50000 images and the test set 10000. See Figure 2b for some exemplary images.

The CIFAR100 dataset, developed by the same institution, also consists of 60000 $32 \times 32 \times 3$ images, with the same split in train and test set. The main difference is that it has 100 classes, grouped in 20 so-called superclasses. Examples of superclasses are *insects, fish, flowers*, and *people*. Examples of corresponding classes are *bee, shark, orchids*, and *baby* [2].

4.3 Permuted MNIST

Permuted MNIST is widely used as a benchmark for new continual learning methods. Each task consists of the whole MNIST dataset with a fixed random permutation of the pixel values. We consider 10 tasks, which means the benchmark consists of 10 subsequent 10-class classification problems. To get comparable results, we follow Kirkpatrick et al. [2017], Zenke et al. [2017], Nguyen et al. [2018] and use a single head, fully-connected two-layer neural network with ReLU activation functions and 100 units per layer. We set the VOGN hyperparameters $\alpha = \beta = 0.001$, and train for 100 epochs per task with a mini batch size of 256. During training, we use 10 MC samples and 100 for validation. We choose a standard normal prior. The mean of the variational distribution is initialised with the initial weights which are sampled according to the default PyTorch initialisation for linear layers. The initial precision is $1e6$. Interestingly, similar to Swaroop et al. [2019] we find that it is crucial for optimal performance to also initialise mean and precision in this way between tasks. We run this experiment with 20 different random seeds of the permutations of the pixel values and model training. We achieve an average accuracy (ACC) of 94.0% with a standard deviation of 0.8%. As we can see in Table 1, VOGN outperforms EWC, SI, UCB, HAT, and achieves similar performance to the improved version of VCL, with slightly slower decrease in average accuracy (see Figure 3).

[1] Available at `http://yann.lecun.com/exdb/mnist/`.

[2] CIFAR10 and CIFAR100 available at `https://www.cs.toronto.edu/~kriz/cifar.html`.

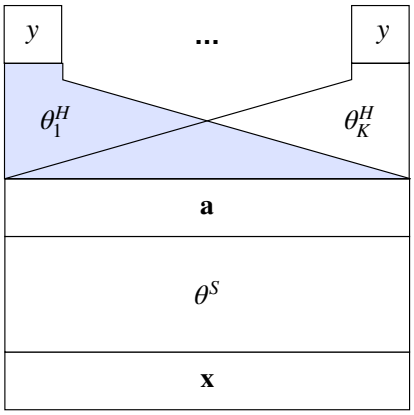

Figure 4: Schematic multi-head neural network for K tasks, with input $\mathbf{x}$, outputs of the last shared layer $\mathbf{a}$, output y, shared parameters $\boldsymbol{\theta}^S$, and head parameters $\boldsymbol{\theta}_k^H$ for task k. Only one head can be used per forward pass; in the figure, the head for task 1 is used, which is indicated by the filling. Figure loosely adopted from Nguyen et al. [2018].

We also run the modified version of SLANG, presented in section 3.5.1, with low-rank $L = 64$ and the same hyperparameters as VOGN but with $\beta = 0.0001$. We use this value for β because of the results for split CIFAR10/100 in section 4.5. For SLANG, it does not seem to make much of a difference; this might be related to the use of a MLP instead of a CNN or the low-rank plus diagonal approximation. Contrary to its theoretical advantage of employing a more sophisticated covariance approximation, we do not see any performance advantage over VOGN. We only run SLANG with 5 different random seeds for the permutations and model training because of the long run times due to increased computational complexity, and achieve $93.8\% \pm 1.4$ average accuracy. There is no obvious reason for the lack of improvement of performance. It might be related to accumulating numerical issues or some not well-understood properties of the approximation. Since this first result does not seem promising and the computational cost is high, we do not run SLANG on the other two benchmarks.

4.4 Split MNIST

This is another standard benchmark problem for continual learning. It consists of five binary classification tasks which are subsets of MNIST. The classes are split in a consequent manner: 0/1, 2/3, 4/5, 6/7, and 8/9.

Unfortunately, for this problem it is necessary to use a multi-head neural network, i.e. a neural network with multiple output layers, each associated with one task. During training and validation of task k, only the output head with parameters $\boldsymbol{\theta}_k^H$ is used; the shared parameters $\boldsymbol{\theta}^S$ are used for training and validation on all tasks (c.f. Figure 4). During training on task k, the prior for $\boldsymbol{\theta}_k^H$ is simply the prior used for training on the first task $\mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0)$. A multi-head neural network assumes that the associated task of a training or validation sample is known for training and prediction. It might not possible to apply this in practice and it is in conflict with the desiderata in Farquhar and Gal [2018]. There might be ways to make a single-head neural network work through, for instance, a wider single-head network as in Golkar et al. [2019].

We follow Swaroop et al. [2019] and use a multi-head fully-connected feed-forward neural network with one hidden layer containing 200 units with ReLU activation functions. We also use a mini batch size of 256 but only train for 100 instead of

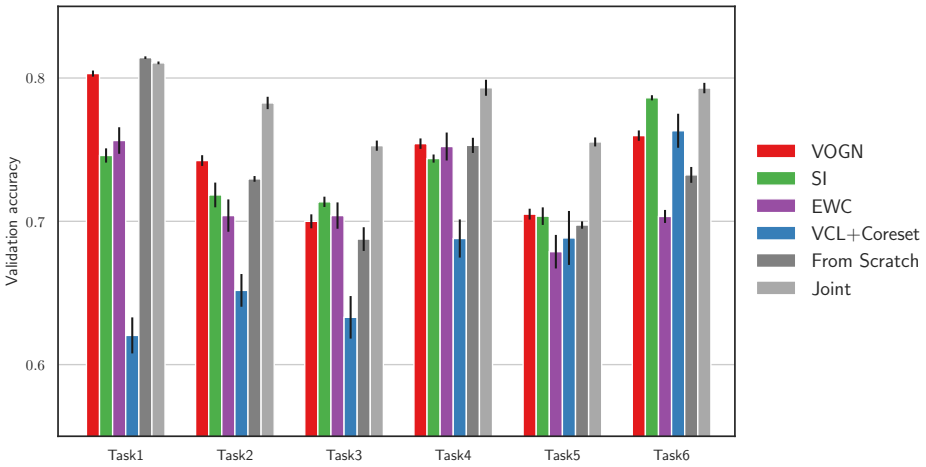

Figure 5: Mean accuracy after training on all 6 tasks of split CIFAR10/100, averaged over 5 runs with standard error. VOGN outperforms VCL + Coreset and EWC, and slightly outperforms SI. VOGN is always comparable or better than training from scratch. VOGN is close to Joint on the first task.

600 epochs per task. We can choose less epochs per task since we expect better convergence properties of VOGN compared to Bayes-by-Backprop due to e.g. the use of natural gradients and also see this confirmed in practice. This is a practical advantage of VOGN over VCL. The hyperparameters of VOGN are set in the same way as for permuted MNIST.

As shown in Table 1, VOGN ($98.8\% \pm 0.1\%$) matches SI, slightly outperforms the average accuracy of the improved result with VCL reported in Swaroop et al. [2019], and outperforms EWC by a large margin.

4.5 Split CIFAR10/100

Split CIFAR10/100 commonly consists of six tasks: The first task is CIFAR10 and the next five tasks are the first 50 classes of CIFAR100 split into five consecutive 10-class classification problems. This problem is significantly harder than permuted and split MNIST. Images are bigger and in colour, $32 \times 32 \times 3$ instead of 28×28 pixels, and the objects shown in the images are harder to classify than the handwritten digits in MNIST. Therefore, for this benchmark a convolutional neural network (CNN) has to be used to achieve good performance.

We choose to follow Zenke et al. [2017] and use a neural network with four convolutional and two fully-connected layers. The only difference is that we do not to use dropout because there is no obvious theoretical interpretation of using VOGN combined with dropout which makes sense and we do not see any benefits in practice. As for split MNIST, we also need to use a multi-head neural network to be able to predict new class labels.

To achieve reasonable results on this benchmark, more hyperparameter tuning than on the simpler problems is necessary. First of all, it is important to realise that the first task (CIFAR10) has ten times as much data as each of the tasks two to six. We use a constant batch size of 256, so to match iterations on all tasks, we choose to train for 60 epochs on the first task and for 600 for the other tasks. Also, we use an initial precision of $4e3$ for the first, and $4e4$ for each of the other tasks. In contrast to the other experiments, we do not reset the weights between tasks. We choose the learning rate $\alpha = 0.001$, a a prior with zero mean and prior precision

Method	**ACC**	**BWT**	**FWT**
EWC	$71.6\% \pm 0.9$	-2.3 ± 1.4	0.17 ± 0.9
SI	$73.5\% \pm 0.5$	*	*
VCL + Coreset	$67.4\% \pm 1.4$	-9.2 ± 1.8	$\mathbf{1.8} \pm 3.1$
VOGN	$\mathbf{74.4}\% \pm 0.4$	$\mathbf{-0.7} \pm 0.5$	$\mathbf{1.8} \pm 0.3$
From Scratch	$73.6\% \pm 0.4$	-	-
Joint	$78.1\% \pm 0.3$	-	-

Table 2: The average validation accuracy (ACC), backward (BWT), and forward (FWT) transfer over all tasks of split CIFAR10/100. We report mean and standard deviation over 5 runs. $*$ means that the metric could not be computed because of lacking access to results. $-$ stands for not applicable. The best results of the continual learning methods (within standard deviation) are in bold.

$\boldsymbol{\sigma}_0^{-2} = 120$. For this benchmark we also use momentum, as suggested in section 3.4, with a momentum rate $\beta_2 = 0.9$. As before, we use 10 MC samples during training, and 100 for validation.

Interestingly, a crucial hyperparameter is the decay rate β of the exponential moving average over the precision. We choose $\beta = 0.0001$ which is smaller than for the other experiments. With $\beta = 0.001$ performance is much worse, almost 10% worse final average accuracy. Choosing a smaller β corresponds to a slower change of the precision, since the current estimate is multiplied by $(1 - \beta)$ and therefore dominate the convex combination. This suggests in the context of this benchmark, that the information contained in the precision vector is crucial for performance. It will be interesting to see if this also applies to other benchmarks where CNNs are used.

Notably, here we compare against VCL with a randomly chosen coreset of 200 samples because VCL without coreset is not able to achieve reasonable performance at all. It might be possible to improve performance by further hyperparameter tuning but even in general, BBB seems to be hard to use with CNNs. As before, we also compare against EWC and SI (SI results taken from Zenke et al. [2017], VCL and EWC baselines kindly provided by Siddharth Swaroop). To get some more perspective on the validation accuracies achieved on each task after training on all tasks, we also consider two more baselines: Training on each task from scratch and training on the data of all tasks jointly, both with the same architecture as the continual learning methods and Adam for optimisation. The final average accuracy of the jointly trained model can be seen as an upper bound for continually learned models. It is not a proper bound for VOGN because we use Adam for the joint training and not VOGN; also, hyperparameter tuning on each task could lead to different results. Still, these two additional baselines provide some more perspective on the properties of the continual learning methods.

As shown in Figure 5, VOGN outperforms VCL + coreset, EWC, and slightly outperforms SI (with overlapping standard deviation, see Table 2 for the exact numbers). As one would hope for, VOGN achieves always almost equal or better accuracy as training from scratch on all tasks. SI almost matches or outperforms training from scratch on tasks 2-6, but is significantly worse on task one. While VCL shows similar forward transfer as VOGN, it does not reach accuracies comparable to training from scratch before task 5, since too much catastrophic forgetting is happening.

Interestingly, VOGN almost matches the accuracy of our approximate upper bound, a jointly trained model, on the first task (CIFAR10); SI, the only method that overall performs almost as well as VOGN, shows significant forgetting of the first task. Conversely, SI outperforms VOGN on the last task. This might be partly explainable by the fact that VOGN is a Bayesian method and the first task has ten times as many training samples as tasks 2-6. In VOGN, we explicitly use the dataset

size N to scale the (square of the individual) stochastic gradients, which practically assigns a proportionally higher importance to the posterior of tasks with more data.

Also, VOGN achieves with -0.7 the best score for backward transfer, which implies that there is very little forgetting or significant transfer to old tasks happening. The problem of catastrophic forgetting is apparent in the BWT score of VCL + Coreset, which is with -9.2 the worst among the tested methods. In contrast, VCL + Coreset achieves the same forward transfer score as VOGN. EWC has BWT and FWT scores in between the two other methods (see Table 2).

Since we do not have access to the exact results for SI, we cannot compute BWT and FWT. Considering the results in Figure 5, we would expect to see a worse BWT, but a better FWT score than VOGN.

5 Conclusion

This work explains an optimisation method for deep neural networks called VOGN, which is using natural gradient variational inference, and provides strong experimental evidence that it can enable continual learning with comparable performance to other state-of-the-art methods. VOGN offers a principled way of doing approximate Bayesian inference which can be scaled to deep neural networks and big datasets. Due to its principled nature, benefits of Bayesian inference are to some extent preserved, which makes it possible to recursively update the posterior when continually learning different tasks. Moreover, through the use of natural gradients, better convergence properties than, for example, Bayes-by-Backprop are expected and also seen in practice, e.g. we only train for 100 epochs per task on permuted and split MNIST while the results with improved VCL are obtained by training 800 and 600 epochs per task respectively. In section 4, we present experimental evidence on some standard continual learning benchmarks for the ability of VOGN to enable continual learning. As discussed before, the results consistently match or outperform other comparable continual learning methods.

Notably, VOGN offers a few natural ways of further improvements: First, as shown with VCL in Nguyen et al. [2018], coresets can easily be integrated into the variational inference framework, which can be used to trade-off some memory and compute for improved performance. Second, VOGN uses a mean field covariance approximation which could be replaced by a structured approximation like SLANG or K-FAC. In practice, SLANG does not show any improvements on permuted MNIST; this might be due to numerical issues or some unknown properties of the algorithm. Using K-FAC is a promising direction, as shown by Ritter et al. [2018], although it is not trivially clear how to use a Kronecker-factored prior precision; some approximation is most likely necessary. Third, the variational family could be extended to something more complex than a simple Gaussian distribution, e.g. a mixture of Gaussians [Lin et al., 2019].

All of these potential improvements represent some trade-off of memory and/or computational cost to improved performance. While this is definitely an interesting direction of further research, if it is appropriate to use these extensions largely depends on the specific application.

Another interesting direction of research in Bayesian deep learning which is loosely related, is to explore the use of more sophisticated priors than the commonly used Gaussian with scalar prior mean and precision. A more appropriate prior for the first task in a continual learning setting might lead to improved performance.

A different direction of potential future work is the application of VOGN to larger and harder continual learning benchmarks. One of the advantages over other methods is that VOGN naturally scales to bigger problems and that there is no additional effort necessary to use VOGN for continually learning. This makes it straight forward to apply it to some continual learning benchmark using bigger datasets like ImageNet. Here, it is interesting to see how much hyperparameter tuning VOGN requires to perform well and if reliable heuristics can be identified. VOGN is available as a plug-and-play PyTorch optimiser, which hopefully facilitates further research in this direction.

Also, all the continual learning benchmarks used in this work make specific and restrictive assumptions. More general scenarios would impose the need to deal with unknown and potentially overlapping task boundaries [Aljundi et al., 2018, Zeno et al., 2018, Jerfel et al., 2018, Aljundi et al., 2019], and might restrict the number of passes through the training data [Lopez-Paz et al., 2017]. Moreover, split dataset tasks with a single-head neural network are an interesting scenario since the task affiliation of the samples during testing has to be known when using a multi-head network; this might be an unrealistic assumption for some practical applications. To come closer to

fulfilling these kind of desiderata for continual learning, e.g. as proposed in Farquhar and Gal [2018] or Lange et al. [2019], more sophisticated benchmarks are necessary.

The development of better and more insightful metrics for continual learning is also crucial to progress in this field; forward and backward transfer as defined in this work are still limited, e.g. it is hard, and maybe impossible, to differentiate between avoiding forgetting and backward transfer. More insight into why algorithms do well and when they fail will help to develop new and better methods. Therefore, the development of new benchmarks and metrics is crucial to progress in continual learning.

All in all, we hope that this work can serve as a starting point for exploring continual learning with natural gradient variational inference in general, and VOGN specifically. Also, it strengthens the point that deep learning and Bayesian inference are not mutually exclusive frameworks; indeed, Bayesian deep learning might offer some potential solutions for common issues with classical deep learning besides catastrophic forgetting, such as fragility with regards to adversarial attacks and overconfident out-of-distribution predictions.

References

Rahaf Aljundi, Klaas Kelchtermans, and Tinne Tuytelaars. Task-free continual learning. In *CVPR*, 2018.

Rahaf Aljundi, Min Lin, Baptiste Goujaud, and Yoshua Bengio. Gradient based sample selection for online continual learning. In *NeurIPS 2019*, 2019.

Shun-Ichi Amari. Natural gradient works efficiently in learning. *Neural computation*, 10(2):251–276, 1998.

Ari S. Benjamin, David Rolnick, and Konrad P. Körding. Measuring and regularizing networks in function space. *ArXiv*, abs/1805.08289, 2018.

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

David M. Blei, Alp Kucukelbir, and Jon D. McAuliffe. Variational Inference: A Review for Statisticians. *arXiv e-prints*, art. arXiv:1601.00670, Jan 2016.

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

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

Arslan Chaudhry, Marcus Rohrbach, Mohamed Elhoseiny, Thalaiyasingam Ajanthan, Puneet Kumar Dokania, Philip H. S. Torr, and Marc'Aurelio Ranzato. Continual learning with tiny episodic memories. *ArXiv*, abs/1902.10486, 2019.

Sayna Ebrahimi, Mohamed Elhoseiny, Trevor Darrell, and Marcus Rohrbach. Uncertainty-guided continual learning with bayesian neural networks. *ArXiv*, abs/1906.02425, 2019.

Sebastian Farquhar and Yarin Gal. Towards robust evaluations of continual learning. *arXiv preprint arXiv:1805.09733*, 2018.

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

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.

Siavash Golkar, Michael Kagan, and Kyunghyun Cho. Continual learning via neural pruning. *CoRR*, abs/1903.04476, 2019.

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

Ian J. Goodfellow, Mehdi Mirza, Xia Da, Aaron C. Courville, and Yoshua Bengio. An empirical investigation of catastrophic forgeting in gradient-based neural networks. *CoRR*, abs/1312.6211, 2013.

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. *ArXiv*, abs/1706.02677, 2017.

Matthew D Hoffman, David M Blei, Chong Wang, and John Paisley. Stochastic variational inference. *The Journal of Machine Learning Research*, 14(1):1303–1347, 2013.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *ArXiv*, abs/1502.03167, 2015.

Ghassen Jerfel, Erin Grant, Thomas L. Griffiths, and Katherine A. Heller. Online gradient-based mixtures for transfer modulation in meta-learning. *ArXiv*, abs/1812.06080, 2018.

Michael I. Jordan, Zoubin Ghahramani, Tommi S. Jaakkola, and Lawrence K. Saul. An introduction to variational methods for graphical models. *Machine Learning*, 37(2):183–233, Nov 1999. ISSN 1573-0565. doi: 10.1023/A:1007665907178. URL `https://doi.org/10.1023/A:1007665907178`.

Mohammad Khan, Didrik Nielsen, Voot Tangkaratt, Wu Lin, Yarin Gal, and Akash Srivastava. Fast and scalable Bayesian deep learning by weight-perturbation in Adam. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2611–2620, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR. URL `http://proceedings.mlr.press/v80/khan18a.html`.

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.

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

Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. *CoRR*, abs/1312.6114, 2013.

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.

Frederik Kunstner, Lukas Balles, and Philipp Hennig. Limitations of the empirical fisher approximation. *ArXiv*, abs/1905.12558, 2019.

Matthias Lange, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Ales Leonardis, Gregory G. Slabaugh, and Tinne Tuytelaars. Continual learning: A comparative study on how to defy forgetting in classification tasks. *ArXiv*, abs/1909.08383, 2019.

Wu Lin, Mohammad Emtiyaz Khan, and Mark Schmidt. Fast and simple natural-gradient variational inference with mixture of exponential-family approximations. In *ICML*, 2019.

David Lopez-Paz et al. Gradient episodic memory for continual learning. In *Advances in Neural Information Processing Systems*, pages 6470–6479, 2017.

Christos Louizos and Max Welling. Structured and efficient variational deep learning with matrix gaussian posteriors. In *International Conference on Machine Learning*, pages 1708–1716, 2016.

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

James Martens. New insights and perspectives on the natural gradient method. *arXiv preprint arXiv:1412.1193*, 2014.

James Martens and Roger Grosse. Optimizing neural networks with Kronecker-factored approximate curvature. In *International Conference on Machine Learning*, pages 2408–2417, 2015.

Michael McCloskey and Neal J. Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In Gordon H. Bower, editor, *Psychology of Learning and Motivation Vol. 24*, volume 24, pages 109 – 165. Academic Press, 1989. doi: https://doi.org/10.1016/S0079-7421(08)60536-8. URL `http://www.sciencedirect.com/science/article/pii/S0079742108605368`.

Aaron Mishkin, Frederik Kunstner, Didrik Nielsen, Mark Schmidt, and Mohammad Emtiyaz Khan. Slang: Fast structured covariance approximations for bayesian deep learning with natural gradient. In *Advances in Neural Information Processing Systems 31*, 2018.

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

M. Opper and C. Archambeau. The variational Gaussian approximation revisited. *Neural Computation*, 21(3):786–792, 2009.

Kazuki Osawa, Siddharth Swaroop, Anirudh Jain, Runa Eschenhagen, Richard E. Turner, Rio Yokota, and Mohammad Emtiyaz Khan. Practical deep learning with bayesian principles, 2019.

Garvesh Raskutti and Sayan Mukherjee. The information geometry of mirror descent. *IEEE Transactions on Information Theory*, 61(3):1451–1457, 2015.

Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic back-propagation and approximate inference in deep generative models. In *International Conference on Machine Learning*, pages 1278–1286, 2014.

Hippolyt Ritter, Aleksandar Botev, and David Barber. Online structured laplace approximations for overcoming catastrophic forgetting. In *International Conference on Learning Representations*, 2018.

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

Nicol N Schraudolph. Fast curvature matrix-vector products for second-order gradient descent. *Neural computation*, 14(7):1723–1738, 2002.

Joan Serrà, Didac Suris, Marius Miron, and Alexandros Karatzoglou. Overcoming catastrophic forgetting with hard attention to the task. *ArXiv*, abs/1801.01423, 2018.

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

Rupesh Kumar Srivastava, Jonathan Masci, Sohrob Kazerounian, Faustino Gomez, and Jürgen Schmidhuber. Compete to compute. In *in Advances in Neural Information Processing Systems, 2013*, pages 2310–2318, 2013.

Shengyang Sun, Changyou Chen, and Lawrence Carin. Learning Structured Weight Uncertainty in Bayesian Neural Networks. In Aarti Singh and Jerry Zhu, editors, *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pages 1283–1292, Fort Lauderdale, FL, USA, 20–22 Apr 2017. PMLR. URL `http://proceedings.mlr.press/v54/sun17b.html`.

Ilya Sutskever, James Martens, George E. Dahl, and Geoffrey E. Hinton. On the importance of initialization and momentum in deep learning. In *ICML (3)*, volume 28 of *JMLR Workshop and Conference Proceedings*, pages 1139–1147. JMLR.org, 2013.

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

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

Hanna Tseran, Mohammad Emtiyaz Khan, Thang Bui, and Tatsui Harada. Natural variational continual learning. In *NeurIPS Workshop on Continual Learning*, 2018.

M. J. Wainwright and M. I. Jordan. Graphical models, exponential families, and variational inference. *Foundations and Trends in Machine Learning*, 1–2:1–305, 2008.

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

Chen Zeno, Itay Golan, Elad Hoffer, and Daniel Soudry. Task agnostic continual learning using online variational bayes. *ArXiv*, abs/1803.10123, 2018.

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

List of Figures

List of Tables