Graph Reparameterizations for Enabling 1000+ Monte Carlo Iterations in Bayesian Deep Neural Networks

Jurijs Nazarovs[1,2] **Ronak R. Mehta**[3,2] **Vishnu Suresh Lokhande**[3,2] **Vikas Singh**[2,3]

[1]Department of Statistics, University of Wisconsin Madison
[2]Department of Biostatistics & Med. Info., University of Wisconsin Madison
[3] Department of Computer Science, University of Wisconsin Madison

Abstract

Uncertainty estimation in deep models is essential in many real-world applications and has benefited from developments over the last several years. Recent evidence [Farquhar et al., 2020] suggests that existing solutions dependent on simple Gaussian formulations may not be sufficient. However, moving to other distributions necessitates Monte Carlo (MC) sampling to estimate quantities such as the KL divergence: it could be expensive and scales poorly as the dimensions of both the input data and the model grow. This is directly related to the structure of the computation graph, which can grow linearly as a function of the number of MC samples needed. Here, we construct a framework to describe these computation graphs, and identify probability families where the graph size can be **independent** or only weakly dependent on the number of MC samples. These families correspond directly to large classes of distributions. Empirically, we can run a much larger number of iterations for MC approximations for larger architectures used in computer vision with gains in performance measured in confident accuracy, stability of training, memory and training time.

1 INTRODUCTION

Motivated by the need to provide measures of uncertainty in the deployment of deep neural networks in mission critical and medical applications, there has been a strong recent interest in deep Bayesian learning. While deep Bayesian learning provides many methods to estimate posterior distributions, Variational Inference (VI) is a convenient choice for many problem settings [Blundell et al., 2015]. Many libraries such as Tensorflow Probability [Dillon et al., 2017] are also now available that offer a rich set of features.

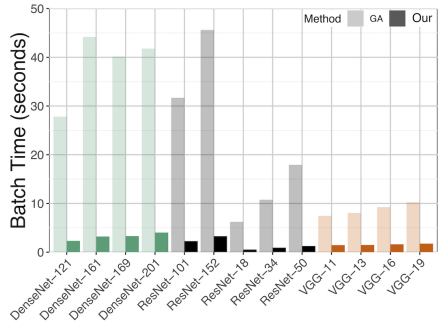


Figure 1: MC sampling is significantly slower in existing neural network libraries incorporating Gradient Accumulation (GA). In contrast, our proposed MC reparameterization reduces the compute time up to $14\times$ for some networks.

Denote the observed data as (x, y), where x is an input to the network, and y is a corresponding response (in autoencoder settings we may have $y = x$). When using VI in Bayesian Neural Networks (BNNs), one considers all weights $W = (W^1, \ldots, W^D)$ as a random vector and approximates the true unknown posterior distribution $P(W|y, x)$ with an *approximate posterior* distribution Q_θ of *our choice*, which depends on learned parameters θ. Let $W_\theta = (W_\theta^1, \ldots, W_\theta^D)$ denote a random vector with a distribution Q_θ and pdf q_θ. VI seeks to find θ such that Q_θ is as close as possible to the real (unknown) posterior $P(W|y, x)$, accomplished by minimizing the KL divergence between Q_θ and $P(W|y, x)$. Given a prior pdf of weights p, along with a likelihood term $p(y|W, x)$, and a common *mean field* assumption of independence for W^d and W_θ^d, for $d \in 1, \ldots, D$, i.e. $p(W) = \prod_{d=1}^D p^d(W^d)$ and $q_\theta(W_\theta) = \prod_{d=1}^D q_\theta^d(W_\theta^d)$,

$$\boldsymbol{\theta}^* = \arg\min_\theta KL\left(q_\theta||p\right) - \mathbb{E}_{q_\theta}\left[\ln p(y|W, x)\right] \tag{1}$$

$$KL\left(q_\theta||p\right) = \sum_{d=1}^D \mathbb{E}_{q_\theta^d}\left[\ln q_\theta^d(w)\right] - \mathbb{E}_{q_\theta^d}\left[\ln p^d(w)\right] \tag{2}$$

Accepted for the 37th Conference on Uncertainty in Artificial Intelligence (UAI 2021).

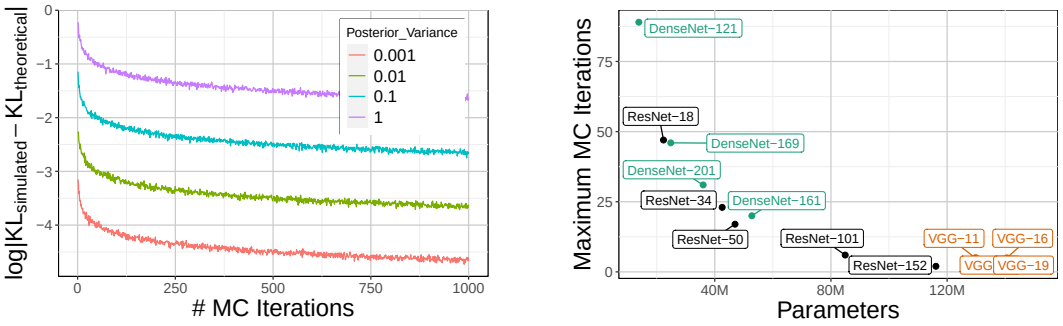


Figure 2: **(left)** Approximation error (log) of simulated KL for the single-parameter Bayesian Neural Network at different variance values of approximate posterior distribution, **(right)** Maximum number of feasible MC iterations required for training Bayesian versions of different neural networks on a single GPU .

A key consideration in VI is the choice of prior p and the approximate posterior q_θ. This choice does not drastically change the computation of the likelihood term $p(y|W, x)$ which is influenced more by the problem and the complexity of the network instead of W (e.g., it is Gaussian for regression problems). But it strongly impacts the computation of KL term. For example, a common choice for p, and q_θ is Gaussian, which allows calculating (2) in a closed form. However, there is emerging evidence [Farquhar et al., 2020, Fortuin et al., 2020] that the Gaussian assumption may not work well on medium/large scale Bayesian NNs. [Farquhar et al., 2020] attributes this to the probability mass in high-dimensional Gaussian distributions concentrating in a narrow "soap-bubble" far from the mean. Choosing a correct distribution is an open problem [Ghosh and Doshi-Velez, 2017, Farquhar et al., 2020, McGregor et al., 2019, Krishnan et al., 2019], and unfortunately, more complex distributions frequently lack closed form solutions for (2).

Numerical approximations. When the integrals for these expectations cannot be solved in closed form, an approximation is used [Ranganath et al., 2014, Paisley et al., 2012, Miller et al., 2017]. One strategy is Monte Carlo (MC) sampling, which gives an unbiased estimator with variance $O(\frac{1}{M})$ where M is number of samples. For a function $g(\cdot)$:

$$\mathbb{E}_{q_\theta}\left[g(w)\right] = \int g(w) q_\theta(w) dw \approx \frac{1}{M}\sum_{i=1}^{M} g(w_i), \quad \text{where } w_i \sim Q_\theta. \tag{3}$$

Expected value terms in (2) can be estimated by applying the scheme in (3) and in fact, even if a closed form expression can be computed, with enough samples an MC approximation may perform similarly [Blundell et al., 2015]. Unfortunately, MC procedures are costly, and may need many samples (i.e., iterations) for a good estimation as the model size grows: [Miller et al., 2017] shows this relationship for small networks, and demonstrates that using fewer samples leads to large variances in the approximation. In general, for deep BNNs, computation of both KL and expectation of log-likelihood requires numerical approximation with MC sampling, but for now, we will only focus on the KL term.

How does M affect the KL approximation necessary for large scale VI? Consider a standard Gaussian distribution for the approximate posterior q_θ and prior p for the weights of an arbitrary BNN, and also consider an MC approximation of the KL term in (2). In this case, we have a closed form solution for KL, which allows checking the approximation quality: the gap between the MC approximation $\widehat{KL}$ and the closed form KL.

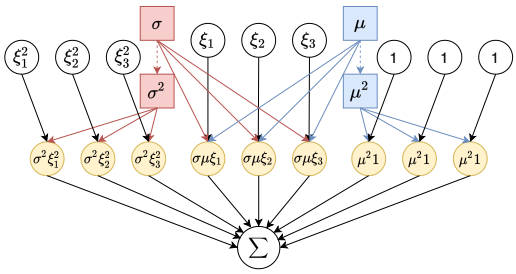
(a) Figure 2 (left) shows this gap for different variances of the approximate posterior for a BNN. While decreasing the variance of the posterior distribution indeed reduces the variance of an estimator, with such a small variance on weights, the model is essentially deterministic. Clearly by increasing M, we decrease the error. However, in current DNNs, increasing the number of MC iterations not only slows down computation, but severely limits GPU memory. **(b)** Figure 2 (right) presents the maximum number of iterations possible on a single GPU (Nvidia 2080 TI) with a direct implementation of MC approximation for Bayesian versions of popular DNN architectures: ResNet, DenseNet and VGG (more details in §3). Extrapolating Figure 2, we see clearly that Bayes versions of these networks will result in large variances. This raises the question: is there a way to increase the number of MC iterations for deep networks without sacrificing performance, memory, or time?

Contributions. This work makes two contributions. **(a)** We propose a new framework to construct an MC estimator for the KL term, which significantly decreases GPU memory needs and improves runtime. Memory savings allow us to run up to 1000× more MC iterations on a single GPU, resulting in smaller variances of the MC estimators, improving both training convergence and final accuracy, especially on subsets of data where the model is not confident. We show feasibility for popular architectures including ResNets [He et al., 2016], DenseNets [Huang et al., 2017], VGG [Simonyan and Zisserman, 2014] and U-Net [Ronneberger

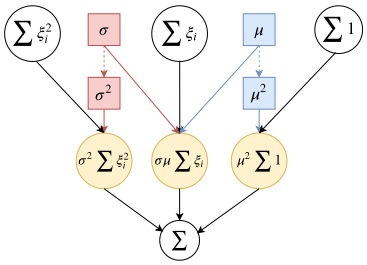
```
model = AlexNet(n_classes=10, n_channels=3,
                approx_post="Radial",
                kl_method="repar",
                n_mc_iter=1000)
```

Figure 3: Proposed MC reparameterization presented as an API. Only a minimal change in an existing programming interface is required to incorporate our method. See the supplement for details.

et al., 2015] – strategies for successfully training Bayesian versions of many of these (deep) networks remain limited [Dusenberry et al., 2020]. **(b)** From the user perspective, we provide a simple interface for implementing and estimating BNNs (Figure. 3). **(c)** On the technical side, we obtain a scheme under which we can determine whether our *reparameterization* can be applied. The result covers a broad class of distributions used in VI as an approximate posterior and prior. Inspired by the Pitman–Koopman–Darmois theorem [Koopman, 1936], we show that our method is effective when an exponential family is used as a prior on weights in deep BNNs estimated via VI, and the approximate posterior is modeled as location-scale or certain other distributions, expanding the range of distributions that can be used.



(a) AutoGrad implementation, $d_P = 9$



(b) $d_P = 3$

Figure 4: Two computation graphs of the same MC expression $\sum_{i=1}^{3}(\mu + \sigma\xi_i)^2$, with different parameterizations. Filled squares represent elements of the vector function $n(\theta)$, clear circles represent functions of auxiliary variables $t(\xi)$, yellow circles represent the Hadamard product $n(\theta) \circ t(\xi)$. Clearly, parameterization affects the size of the graph, and there exists parameterization (b) where the size is independent of the number of MC iterations M. **Note:** We slightly modify the computational graph presentation for space and clarity. Actual computation graphs from PyTorch convey the same message.

2 RELATED WORK

In addition to VI, the literature provides a broad range of ways to estimate posterior predictive distributions. Ensemble methods [Lakshminarayanan et al., 2017, Pearce et al., 2018, Newton et al., 2018] can be applied to common networks with minimal modifications; however, they require many forward passes, often similar in terms of space/time to a standard *gradient accumulation* schemes (we provide a PyTorch code snippet in Figure 5). Figure 1 provides experimental results showing that gradient accumulation is much slower. Other methods like Deterministic Variational Inference [Wu et al., 2019] and Probabilistic Backpropagation [Hernández-Lobato and Adams, 2015], improve over naïve MC implementations of VI, but often approximate the posterior of a neural network with a Gaussian distribution. However, [Farquhar et al., 2020] shows that Gaussians are sensitive to hyperparameter choices, among other problems during training. For this reason, a non-Gaussian distribution can be used as an approximate posterior in the traditional VI setup, but its lack of a closed form solution for the KL term ends up needing MC approximation. This is where our proposal offers value. Also, note some other issues that emerge in Deterministic Variational Inference and Probabilistic Backpropagation: (a) the methods need non-trivial modification of the network to perform a moment matching and (b) replacing the Gaussian assumption with another distribution requires new analytical solutions of closed forms. This is more complicated than a MC approximation.

Our work is distinct from other works that also target MC estimation in neural networks. For example, one may seek to derive new estimators with an explicit goal of variance reduction (e.g., [Miller et al., 2017]). Here, we do not obtain a new estimator replacing the MC procedure with a smaller variance procedure. Instead, we study a scheme that makes the computation graph mostly independent of the number of samples, and is applicable to ideas such as those in [Miller et al., 2017] as well.

```
optimizer.zero_grad()
for _ in range(n_mc_iter):
    output = model(inputs)
    loss = computeLoss(output, targets)
    loss.backward()
optimizer.step()
```

Figure 5: PyTorch implementation of "gradient accumulation" technique, a standard method to collect gradient from several different forward passes. Memory consumption is equivalent to 1 forward pass, but time complexity is proportional to number of forward passes.

Sampling: $W(\theta, \xi)$	Approximate Posterior p.d.f. q_θ		Prior p.d.f. $p(w)$		
Scaling property family: $W(\theta, \xi) = \theta\xi$ and related – Corollary 1	Exponential(θ) Rayleigh(θ) Erlang(k, θ) Error(a, θ, c) Inverse-Gamma(k, θ)	Standard Wald(θ) Weibull(k, θ) Gamma(k, θ) Log-Gamma(k, θ)	Exponential Dirichlet Inverse-Gamma Log-normal Inverse-Gaussian	Standard Wald Chi-squared Gamma Error Normal	Rayleigh Pareto Erlang Weibull
Location-Scale family: $W(\theta, \xi) = \mu + \sigma\xi$, $\theta = (\mu, \sigma)$	Normal(μ, σ) Logistic(μ, σ) Radial(μ, σ)	Laplace(μ, σ) Horseshoe(μ, σ)	Logistic	Exponential Laplace	Normal
			Normal variations, e.g., Horseshoe, Radial		
Corollary 2	Log-Normal(μ, σ)		Dirichlet	Pareto	

Table 1: Summary list of approximate posterior distributions q_θ and priors $p(w)$, which allows to define a parameterization tuple P for MC estimation, such that d_P is independent of M. For every cell in "Sampling: $W(\theta, \xi)$" we can select any combination of q_θ and $p(w)$. Reference: Radial [Farquhar et al., 2020], Horseshoe [Ghosh and Doshi-Velez, 2017].

3 COMPUTATION GRAPHS FOR MC ITERATIONS

Despite the ability to approximate the expectation in principle, the minimization in (1) via (3) is difficult for common architectures, and relies on gradient computations at each iterate. Standard implementations make use of automatic differentiation based on computation graphs [Griewank, 2012]. *Computation graphs* are directed acyclic graphs, where nodes are the inputs/outputs and edges are the operations. If there is a single input to an operation that requires a gradient, its output will also require a gradient. As noted in PyTorch manual (cf. Autograd mechanics), a backward computation is never performed for subgraphs where no nodes require gradients. This allows us to replace such a subgraph with one output node and to define the size of the computation graph as the minimal number of nodes necessary to perform backpropagation: the number of nodes which require gradients. Modern neural networks lead to graphs where the number of nodes range from a few hundred to millions. To define the size of a graph, accounting for the probabilistic nature of the MC approximation, we propose the following construction.

Definition 1 *Consider w as sampled based on a parameter θ and an ancillary random variable ξ, i.e., $w = W(\theta, \xi)$. If there exist functions G, n, and t such that a function $F(w_1, \ldots, w_n)$ can be expressed as $G(n(\theta) \circ t(\xi_1, \ldots, \xi_n))$, then we say $P := (G, n, t)$ is a **parameterization tuple** for the function F, where $\circ$ is the Hadamard product. Let d_P be the dimension of $n \circ t$, corresponding to the number of nodes requiring gradients with respect to θ.*

To demonstrate the application of the Def. 1, as an example, consider the computation graph for the MC approximation of the function $g(w) = w^2$ in (3) and given one weight $W_\theta \sim N(\mu, \sigma^2)$. Applying the reparameterization trick: $W_\theta = \mu + \sigma\xi$, $\xi \sim \mathrm{N}(0, 1)$, the Python form is,

```
for i in range(M):
    # sample 1 observation from N(0, 1)
    sample = sampler_normal.sample()
    w = mu * 1 + sigma * sample
    loss += w^2 / M
```

The computation graph, a function of both the parameters $\theta = (\mu, \sigma)$ and of the auxiliary samples ξ_1, ξ_2, and ξ_3, generated by PyTorch/AutoGrad for $\mathbf{M = 3}$ **iterations** of this loop is shown in Figure 4a. According to Def. 1, $d_P = 9$ and

$$
\begin{aligned}
n(\theta) &= (\mu^2, 2\sigma\mu, \sigma^2, \mu^2, 2\sigma\mu, \sigma^2, \mu^2, 2\sigma\mu, \sigma^2), \\
t(\xi) &= (1, \xi_1, \xi_1^2, 1, \xi_2, \xi_2^2, 1, \xi_3, \xi_3^2), \\
G(n(\theta) \circ t(\xi)) &= n_1(\theta)t_1(\xi) + \cdots + n_9(\theta)t_9(\xi)
\end{aligned}
$$

Naïvely, the graph size grows linearly $O(M)$ with the number of MC iterations, as in the direct implementation (Fig. 4a). For Bayesian VI in DNNs, this is a problem. We need to perform MC approximations of KL terms at every layer. Also, [Miller et al., 2017] shows that iterating over a large number of samples M might be important for convergence. This constrains model sizes given limited hardware resources. One might suspect that a "for" loop is a poor way to evaluate this expectation and instead the expression should be vectorized. Indeed, creating a vector of size M and summing it will clearly help runtime. But the loop does not change the computation graph; all trainable parameters maintain the same corresponding connections to samples, and rapidly exhaust memory.

But graphs for the same function can be constructed differently (see Fig. 4b). For the right parameterization tuple P, we can achieve $d_P = 3$. This leads us to,

Remark 1 *For computation graph of MC approximation $\sum_{i=1}^{M} g(w_i)$ and specific g, there exists a parameterization tuple $P = (G, n, t)$, such that d_P is independent of M.*

For which class of distributions Q_θ and functions $g(\cdot)$ can we always construct reparameterizations of the MC estimation (3), maintaining the size of the computation graph d_P as **independent** of number of iterations M? We explore this in the next section.

4 MC REPARAMETERIZATION ENABLES FEASIBLE TRAINING

Our approach is partly inspired by a vast literature on known distributional families and their use within VI. For example, in VI, commonly one chooses distributions that fall within *exponential families* (e.g., Gaussian, Laplace, Horseshoe). With this assumption on the prior, we can express

$$p(w;\zeta) = h(w)\exp(\eta(\zeta)'T(w) - A(\zeta)) \tag{4}$$

where ζ is a parameter defining w. The sufficient statistics $T(w)$ and natural parameters $\eta(\zeta)$ completely define a specific distribution.

Relevance of PKD theorem. While the foregoing discussion links our approach to well-known statistical concepts, it does not directly yield our proposed scheme. To see this, recall that the Pitman-Koopman-Darmois (PKD) theorem states that for exponential families in (4), there exist sufficient statistics such that the number of scalar components does not increase as the sample size increases. However, in approximating (2) with MC, we need to compute not only terms containing the sufficient statistics $T(w)$ but also $\frac{1}{M}\sum_{i=1}^{M}\log h(w_i)$. Regardless, even though the PKD result cannot be applied directly in our case, it still suggests considering members of the exponential family as candidates for Q_θ. We derive technical results for the forms of $W(\theta,\xi)$ and $g(\cdot)$, where the graph size is not affected by MC sampling.

To approximate KL in (2), we need to compute MC estimation (3) for $g(w) = \log q_\theta(w)$ (or $\log p(w)$). Assume that the factorization form (4) of distributions $q_\theta^d(w)$ (and similarly $p^d(w)$) and recall that the weights of NN are parameterized as $w \sim W(\theta,\xi)$. Then, $\mathbb{E}_\theta \log q_\theta(w)$ is approximated as:

$$\frac{1}{M}\sum_{i=1}\{\log h(w(\theta,\xi_i)) + \eta(\theta)'T(w(\theta,\xi_i))\} - A(\theta) \tag{5}$$

To keep the graph size agnostic of M, we must handle the initial two terms in (5). Checking distributions from Tab. 1, our work reduces to functions of the form w^k and $\log(w)$.

Denote S as the dimension of θ, i.e., number of parameters defining the distribution Q_θ. For example, for the Exponential(λ): $S = 1$ and $\theta = (\lambda)$; for Gaussian(μ, σ): $S = 2$ and $\theta = (\mu, \sigma)$. Denote k to be a positive integer.

Theorem 1 *If $W(\theta,\xi) = \eta(\theta)T(\xi)$ ($S = 1$), then there exists a parameterization tuple P with $d_P = 1$ for the following functions $g(w)$: w^k, $\log(w)$, and $\frac{1}{w^k}$.*

Corollary 1 *If $W(\theta,\xi)' = f(W(\theta,\xi))$ and $W(\theta,\xi) = \eta(\theta)T(\eta)$, then Theorem 1 applies to $W(\theta,\xi)'$ and $g(W(\theta,\xi)')$ if $g(w'(w))$ is: w^k, $\log(w)$, and $\frac{1}{w^k}$.*

Theorem 2 *If $W(\theta,\xi) = \sum_{s=1}^{S}\eta_s(\theta)T_s(\xi)$, and $g(w) = w^k$, then there exists a parameterization tuple P with*

$$d_P = \binom{k+S-1}{S-1}. \tag{6}$$

Remark 2 *As long as $d_P < M$, it is possible to create a computation graph of a smaller size by reparameterization, compared to a direct implementation of the MC approximation. Note that for a small M it is still possible for a parameterization tuple to generate a graph larger than a naïve implementation. For example, consider $\sum_{i=1}^{M}(\mu+\sigma\xi)^2$. When $M = 1$, the naïve construction would have $d_P = 2, (n = (\mu,\sigma), t = (1,\xi)$, while a "nicer" tuple may have $d_P = 3$ independent of M $(n = (\mu^2, 2\mu\sigma, \sigma^2), t = (1,\xi,\xi^2))$.*

Corollary 2 *If $W(\theta,\xi)' = f(W(\theta,\xi))$ and $W(\theta,\xi) = \sum_{s=1}^{S}\eta_s(\theta)T_s(\eta)$, where $S \geq 2$, then Theorem 2 applies to $W(\theta,\xi)'$ and $g(W(\theta,\xi)')$ if $g(w'(w)) = w^k$.*

Relevance of results: (1) Thm. 1 can be applied when $W(\theta,\xi)$ represents a distribution with scaling property: any positive real constant times a random variable having this distribution comes from the same distributional family. (2) Thm. 2 can be applied, when $W(\theta,\xi)$ is a member of the location-scale family. (3) Corollaries 1 and 2 are useful when random variables can be presented as a transformation of other distributions, e.g. $\mathrm{LogNormal}(\mu,\sigma^2)$ can be generated as $\exp(\mathrm{N}(\mu,\sigma^2))$. Table 1 summarizes the choice of q_θ and p for Bayesian VI, which lead to the computation graph size d_P being independent of M in MC estimation.

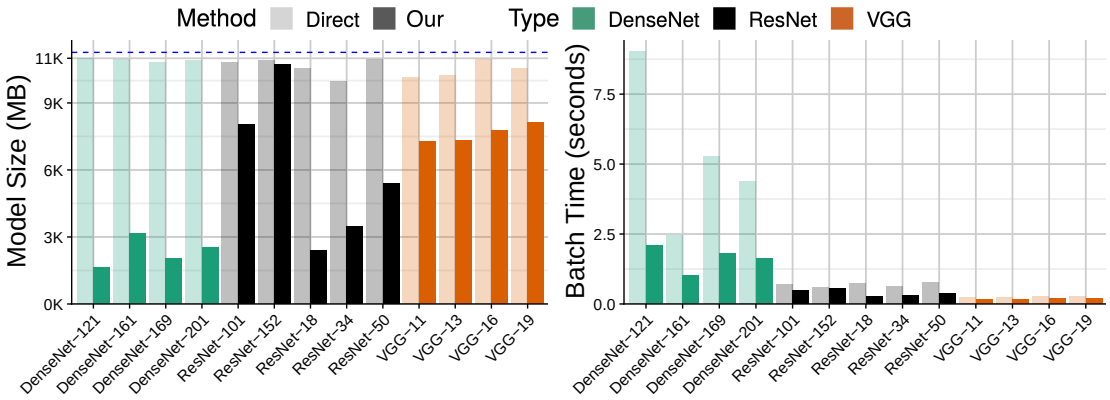
Although Theorem 2 does not suggest that there are no nice parameterization tuples for the case where $g(w) = \log w, 1/w^k$, empirically we did not find tuples that allow for d_P to be independent of M. But it is interesting to consider an approximation which does allow for this independence.

4.1 TAYLOR APPROXIMATED MONTE CARLO

Our results extend to the generic polynomial case where $g(w) = p_K(w)$, an arbitrary polynomial of degree K:

Corollary 3 *If $W = \sum_{s=1}^{S}\eta_s(\theta)T_s(\xi)$, and $g(w) = p_K(w)$, then there exists parameterization tuple P, such that for any M iterations*

$$d_P = \binom{K+S}{S} - 1. \tag{7}$$



(a) (b)

Figure 6: For maximum possible number of MC iterations for a given model via the direct MC method, we show: **(a)** Model size (dashed blue line indicates GPU capacity, 11GB), **(b)** Training time. For some networks, our method occupies less than 25% of memory and 5 times faster.

So, can we find a parameterization tuple for any $g(w)$ that we can approximate via a polynomial Taylor expansion?

Theorem 3 *Let $W = \sum_{s=1}^{S} \eta_s(\theta) T_s(X)$, $S \geq 2$. If an approximation of $g(w)$ uses K Taylor terms, then Cor. 3 applies.*

Practical implications. If one is limited to running a maximum number of MC iterations M_{max}, such an approximation of $g(w)$ allows a tradeoff between accuracy of running just M_{max} iterations for the real $g(w)$ versus approximating $g(w)$ with $K(M_{max})$ terms and running $M \gg M_{max}$ iterations instead, since d_p is independent of M. This strategy may not work for approximating non-polynomial functions, and is a "fall-back" that could be used for arbitrary distributions.

Example 1. Let $W = \mu + \sigma\xi \implies S = 2$ and $g(w) = \log w$, then

$$\sum_{i=1}^{M} g(w_i) = \sum_{i=1}^{M} \log(w_i) \approx \sum_{i=1}^{M} \sum_{k=0}^{K} \frac{1}{k!} (\mu + \sigma\xi_i - 1)^k$$

where we take the Taylor expansion of $\log(w)$ around $w = 1$. This is clearly a polynomial function of order K, and applying Corollary 3, we have $d_P = \frac{1}{2}(K+1)(K+2) - 1$ interactions. For example, if one is able to run just 9 direct MC iterations, it is possible to approximate $g(w)$ with $K = 3$ terms, allowing any number of MC iterations M.

4.2 APPLYING REPARAMETERIZATION IN BAYESIAN NN

Recall that training a Bayesian NN via VI requires the approximation of both the KL term and expected value of log-likelihood in (2). While it is clear how MC reparameterization can be applied to approximate the KL term, what can we say about the likelihood term? In general, this term cannot be handled by the ideas described so far although some practical strategies are possible.

Usually, estimating the expectation of the likelihood term is based on [Kingma and Welling, 2013, Kingma et al., 2015], where for every data item b in the minibatch (of size B), one MC sample is selected, which results in B different samples – in fact, [Kingma and Welling, 2013] suggests that the number of samples per data item can be set to one if the minibatch size is "large enough" which we will discuss more shortly. If a large B is feasible, then our scheme might not contribute substantially in estimating the likelihood term. However, if B is small, then our scheme can provide some empirical benefits, described next.

Let (x, y) be the observed data and (x_b, y_b) be the observed b-th data point. Let w correspond to the weights of NN with L layers. We can use $w(l, \cdot)$ to index the weights of layer l. Note that we can draw a unique sample of w for each data point b which we denote as $w(l, b)$. When M samples are drawn for b, these will be indexed by $w_i(l, b)$ for $i = 1, \cdots, M$. Notice that $w_1(l, b)$ is the same as $w(l, b)$. In the forward pass, u_b^l is the output for the b-th data point and u_b^L is the output of the last layer for data point x_b.

Observation 1 (Likelihood form in BNN) *Consider the following form for regression and classification tasks,*
Regression: *Consider $y \sim N(u^L, \widehat{\sigma})$, where $\widehat{\sigma}$ is fixed. Then,*

$$\begin{aligned} \log p\left(y_b \mid w, x_b\right) &= \log\left(\frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(y_b - u_b^L\right)^2\right)\right) \\ &= \log\frac{1}{\sqrt{2\pi}} - \frac{1}{2} y_b^2 - y_b u_b^L + \frac{1}{2}(u_b^L)^2. \end{aligned}$$

Classification: *Consider a binary classification problem.*

Then, $y \sim$ Bern (p), where $p = \frac{1}{1+\exp(-u^L)}$. Thus,

$$\begin{aligned}\log p\left(y_b \mid w, x_b\right) &= \log\left(p^{y_b}(1-p)^{1-y_b}\right) \\ &= -\log\left(1+e^{-u_b^L}\right) - (1-y_b)u_b^L \\ &= -\log(2) + \frac{u_b^L}{2} - \frac{\left(u_b^L\right)^2}{4} + O\left(\left(u_b^L\right)^3\right) \\ &\quad - (1-y_b)u_b^L.\end{aligned}$$

Based on the above description, let us assume that the final layer output u^L corresponds to a convolution or a fully connected layer with no activation function. Then, the log-likelihood term in a regression and classification setup can be expressed as

$$\log p(y_b|w, x_b) = \text{polynomial}(u_b^{L-1}w(L)).$$

SGVB Estimator. Following [Kingma and Welling, 2013], the $\mathbb{E}_{q_\theta}\left[\log p(y|w,x)\right]$ term for the minibatch (of size B) can be written as

$$S_1 \coloneqq \frac{1}{B}\sum_{b=1}^{B} \mathbb{E}_{q_\theta}\left[\log p(y_b|w, x_b)\right].$$

To approximate the expectation, we use 1 sample $w(\cdot, b)$ for each data point (x_b, y_b), which results in $S_1 = \frac{1}{B}\sum_{b=1}^{B} \log p(y_b|w(\cdot, b), x_b)$. Substituting in polynomial$(u_b^{L-1}w(L, b))$ into $\log p(y_b|w(\cdot, b), x_b)$ leads to the following form for variance $V(S_1)$,

$$\frac{1}{B}\left(\, V\left(w(L,b)\right)\mathbb{E}\left[\left(u_b^{L-1}\right)^2\right] + V(u_b^{L-1})\mathbb{E}^2\left[w(L,b)\right]\,\right), \tag{8}$$

plus higher order terms which decreases as B grows. By efficiently evaluating the KL term, we can utilize the memory savings to increase the batch size B and thus, to decrease the variance of S_1.

MC Reparameterization estimator of likelihood. The above strategy is practically sufficient. However, if B is limited by hardware, we can use the memory savings for more MC samples (higher M) for improving the estimate of the log likelihood term. This reduces the variance of first term in (8) by a factor of M, but the scheme described is restricted to the last layer.

5 EXPERIMENTS: BAYESIAN DENSENET, U-NET, AND OTHER NETWORKS

We perform experiments on Bayesian forms of several architectures and show that training is feasible. While we expect some drop in overall accuracy compared to a deterministic version of the network, these experiments shed light on the benefits/ limitations of increasing MC iterations. Since model uncertainty is important in scientific applications, we also study the feasibility of training such models for classifying high-resolution brain images from a public dataset.

Setup. For deterministic comparisons, we run several variations of PreActResNet [He et al., 2016] and Densenet [Huang et al., 2017] (9 in total) on CIFAR10. For brain images, we use a simple modification of 3D U-Net [Ronneberger et al., 2015]. Since our method is most relevant when a closed form for KL is unavailable, we select the approximate posterior to be a Radial distribution, where samples can be generated as: $\mu + \sigma * \frac{\xi}{||\xi||} * |r|$, where $\xi \sim MVN(0, I)$, $r \sim N(0, 1)$ and the prior of our weights is a Normal distribution. This satisfies the conditions of Thm. 2, allowing us to find a parameterization tuple that does not grow with respect to M: we can run $1000+$ MC iterations with almost no additional GPU memory cost compared to 1 MC iteration. Another reason for choosing the Radial distribution as our approximate posterior q_θ is because Gaussian-approximate posteriors do not perform well in high-dimensional settings [Farquhar et al., 2020]. Empirically, we find this to be the case as well; we were not able to train any models with a standard Gaussian assumption without any ad-hoc fixes such as pretraining, burn-in, or KL-reweighting (common in many implementations).

Parameter settings/hardware. All experiments used Nvidia 2080 TIs. The code was implemented in PyTorch, using the Adam optimizer [Kingma and Ba, 2014] for all models, with training data augmented via standard transformations: normalization, random re-cropping, and random flipping. All models were run for 100 epochs.

5.1 TIME AND SPACE CONSIDERATIONS

We first examine whether our MC-reparameterization leads to meaningful benefits in model size or runtime. We should expect a competitive advantage in model size as the number of MC iterations grows, which may come at the cost of significantly increased runtime. To allow ease of comparison, we fix the batch size for all models to be 32. We determine the maximum number of MC iterations able to run on a single GPU for a given model via the classical direct method. For DenseNet-121, we are able to run 89 MC iterations, while for VGG-16 we are only able to run 5 MC iterations.

Figure 6 shows a comparison of computational performance between our method and the direct approach. **(a)** With **our** construction, we significantly **reduce model size** on the GPU (Fig. 6a). For smaller models like DenseNet, for the same number of MC iterations our method uses less than 25% of GPU memory, which allows for a significant **increase in batch size**. Since the size of the computation graph in our construction is independent of M, for the memory used in Fig. 6a we are able to run for $M = 1000$ or more. **(b)** The significant reduction of model size on the

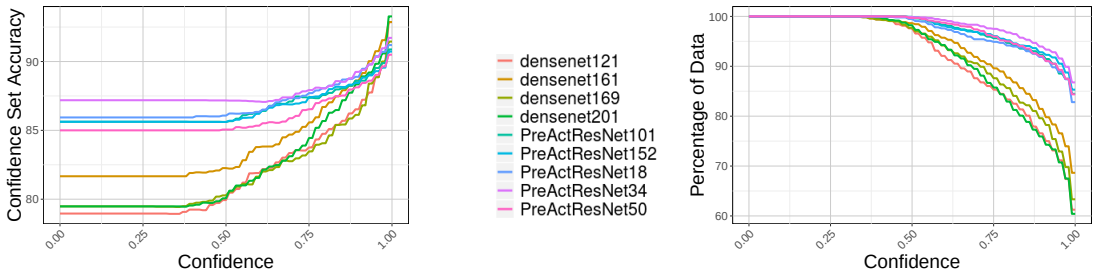


Figure 7: Confidence Set Accuracy and Confidence Sets on CIFAR-10 for a variety of ResNet and DenseNet models with 100 MC iterations (not previously possible). Both ResNet and Densenet achieve accuracy of more than 90% with 100% confidence, but ResNet is 100% confident on almost 90% of the data.

GPU results in a reduction of training time per batch, up to $5\times$ (Figure 6b); the generated computation graph has fewer parameters (nodes on the path) during backpropagation.

5.2 PREDICTION CONFIDENCE/ACCURACY AND HOW MANY MC ITERATIONS?

For our next set of experiments, we run a Bayesian version of PreActResNet and DenseNet with 100 MC iterations, which is feasible.

(a) We evaluate the accuracy concurrently with the confidence of the prediction, offered directly by the model. We expect that the model has a higher accuracy for those samples where it highly confident. This is indeed the case – Figure 7 shows the accuracy for varying levels of confidence over the entire validation set for a number of models. At high confidence levels, all models perform well, competing strongly with state of the art results. Additionally, we observe the proportion of data for which the model is confident is large (Figure 7 right). We can see that Bayesian model is at least 75% confident on 85%–95% of data.

(b) One issue in Bayesian networks is evaluating the expected drop in accuracy (compared to its deterministic versions), a behavior common in both shallow and deep models [Wenzel et al., 2020]. Figure 7 (left) reassures us that the drop in performance for a number of widely used architectures is not that significant even when the model is not confident.

(c) To understand the effect of increasing the number of MC iterations, we run replications of experiment on ResNet-50 for 3 different number of MC iterations, Figure 8(left): 1 iteration (black), 17 iterations – maximum possible on GPU with the traditional method – (blue), and 100 iterations (red) possible to run due to our method. In all cases, as the threshold increases, model confidence increases and as expected, the accuracy does as well. However, we see that training with 100 MC iterations, consistently provides higher accuracy for the entire range of confidences. In contrast, with 1 MC iteration, accuracy has higher variance for the non-confident set.

5.3 NEUROIMAGING: PREDICTIVE UNCERTAINTY IN BRAIN IMAGING ANALYSIS

While we demonstrated advantages of our reparameterization in traditional image classification settings and benchmarks – mostly as a proof of feasibility – a real need for BNNs is in scientific/biomedical domains: where high confidence and accurate predictions may inform diagnosis/treatment. To evaluate applicability, we focus on a learning task with brain imaging data.

Data. Data used in our experiments were obtained from the Alzheimer's Disease Neuroimaging Initiative (ADNI). As such, the investigators within the ADNI contributed to the design and implementation of ADNI and/or provided data but did not participate in analysis or writing of this report. A complete listing of ADNI investigators can be found in [ADNI, 2020a]. The primary goal of ADNI has been to test whether serial magnetic resonance imaging (MRI), positron emission tomography (PET), other biological markers, and clinical and neuropsychological assessment can be combined to measure the progression of mild cognitive impairment (MCI) and early Alzheimer's disease (AD).

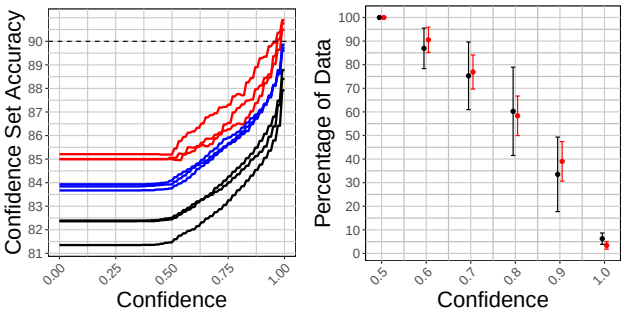


Figure 8: **(left)** Replicated Confidence Set Accuracy on CIFAR-10 for Resnet-50 with different number of MC iterations: 1 (black), 17 (blue, maximum allowed on GPU with standard method) and 100 (red). With $M = 100$ the accuracy is higher for any confidence. **(right)** Distributions of Confidence Set Size for a number of replications, with 1 MC iteration (black) and 100 MC iterations (red). With 100 MC iterations variance is smaller.

	Confidence					
	0.5	0.6	0.7	0.8	0.9	1
$m = 1$	63.07 ± 1.47	62.14 ± 1.59	63.01 ± 0.64	64.13 ± 4.15	59.59 ± 4.40	60.71 ± 15.15
$m = 100$	64.39 ± 4.59	66.23 ± 4.19	66.05 ± 1.00	67.77 ± 4.00	66.82 ± 1.24	87.50 ± 17.68
Δ	1.33	4.09	3.04	3.64	7.23	26.79

Table 2: Average validation accuracy per model confidence for 2 values of MC iterations. $\Delta = A_{100} - A_1$, where A_i is validation accuracy for i MC iterations. With 100 MC iterations we got on average much better results, especially when prediction is highly confident.

For up-to-date information, see [ADNI, 2020b]. Classifying healthy and diseased individuals via their MR images, similar to ADNI, is common in the literature However, overfitting when using deep models remains an issue for two reasons: small dataset size and a large feature space. Here, we look at a specific setting where we have 388 individuals with pre-processed MR images of size $105 \times 127 \times 105$. Preprocessing. All MR images were registered to MNI space using SPM12 with default settings.

Network. We use a slightly modified version of the encoder from an off-the-shelf 3D U-Net architecture [Ronneberger et al., 2015], demonstrated in Figure 9, to learn a classifier for cognitively normal (CN) and Alzheimer's Disease (AD) subjects. We note that while this architecture is not competitive with those which achieve state-of-the-art classification accuracy on ADNI, our aim here is to demonstrate feasibility of training deep Bayesian models in this setting and evaluate the value of accurate confidence estimation.

We train the model on 300 individuals, and validate on the remaining 88. Additional experimental details can be found in the appendix. Since the input to the network is a mini-batch of high dimensional images, when we take into account the memory already needed by a deterministic model, we already reach the limits of the GPU memory. While we cannot perform more than 1 MC iteration with the standard method, we can successfully perform more than 100 with our scheme. We evaluate the consistency of performance with several runs of training when we are allowed to use 1 versus 100 MC iterations. **(a)** Table 2 shows the average validation accuracy for the choice of MC iterations and their difference. We see that for every confidence threshold, training with 100 MC iterations provides higher accuracy on average. This is especially noticeable on a high confident set, where the difference approaches 26.7%. **(b)** In addition to accuracy, it is important to understand how consistent the estimation is. Figure 8 (right) demonstrates the distribution of the size of confident set. While on average, the size of the "confident set" of the two models is similar, the variance is significantly smaller when we use a larger number of MC iterations, consistent with our hypothesis in §1. In cases where this confidence needs to be measured as accurately as possible, one obtains benefits over a single MC iteration.

```
Conv3D_Block(1, 16)
MaxPool3d((3,3,3))
Conv3D_Block(16, 32, stride=1)
MaxPool3d((2,2,2))
Conv3D_Block(32, 64, stride=1)
MaxPool3d((2,2,2))
Conv3D_Block(64, 128, stride=3),
MaxPool3d((2,2,2))
Conv3D_Block(128, 256, stride=3)
Linear(256, 2)
```

Figure 9: Structure of the model we used for ADNI classification.

6 CONCLUSIONS

While a broad variety of neural network architectures are used in vision and medical imaging, successfully training them in a Bayesian setting poses challenges. Part of the reason has to do with distributional assumptions. Moving to a broader class of distributions involves MC estimations but direct implementations pose serious demands on memory and run-time. In this work, we identify that different computation graphs can be constructed for different parameterizations of the target function. Specifically when one is attempting a Monte Carlo approximation, these graphs can grow linearly with the number of MC iterations needed, which is undesirable. By directly characterizing the parameterizations that lead to different graphs, we analyze situations where it is possible for graphs to be constructed independent of this sampling rate (number of MC iterations). Evaluating our parameterization empirically, we find that it is feasible to run a large number of MC iterations for large networks in vision, with a nominal drop in accuracy (compared to deterministic versions). The code is available at `https://github.com/vsingh-group/mcrepar`.

ACKNOWLEDGMENTS

This work was supported in part by NIH grants RF1 AG059312 and RF1 AG062336. RRM was supported in part by NIH Bio-Data Science Training Program T32 LM012413 grant to the University of Wisconsin Madison.

References

ADNI. ADNI Authors, 2020a. URL http://adni.loni.usc.edu/wp-content/uploads/how_to_apply/ADNI_Acknowledgement_List.pdf.

ADNI. ADNI Info, 2020b. URL www.adni-info.org.

Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural networks. arXiv preprint arXiv:1505.05424, 2015.

Joshua V Dillon, Ian Langmore, Dustin Tran, Eugene Brevdo, Srinivas Vasudevan, Dave Moore, Brian Patton, Alex Alemi, Matt Hoffman, and Rif A Saurous. Tensorflow distributions. arXiv preprint arXiv:1711.10604, 2017.

Michael Dusenberry, Ghassen Jerfel, Yeming Wen, Yian Ma, Jasper Snoek, Katherine Heller, Balaji Lakshminarayanan, and Dustin Tran. Efficient and scalable bayesian neural nets with rank-1 factors. In International conference on machine learning, pages 2782–2792. PMLR, 2020.

Sebastian Farquhar, Michael A Osborne, and Yarin Gal. Radial bayesian neural networks: Beyond discrete support in large-scale bayesian deep learning. stat, 1050:7, 2020.

Vincent Fortuin, Adrià Garriga-Alonso, Florian Wenzel, Gunnar Ratsch, Richard E Turner, Mark van der Wilk, and Laurence Aitchison. Bayesian neural network priors revisited. In "I Can't Believe It's Not Better!"NeurIPS 2020 workshop, 2020.

Soumya Ghosh and Finale Doshi-Velez. Model selection in bayesian neural networks via horseshoe priors. arXiv preprint arXiv:1705.10388, 2017.

Andreas Griewank. Who invented the reverse mode of differentiation. Documenta Mathematica, Extra Volume ISMP, pages 389–400, 2012.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In European conference on computer vision, pages 630–645. Springer, 2016.

José Miguel Hernández-Lobato and Ryan Adams. Probabilistic backpropagation for scalable learning of bayesian neural networks. In International Conference on Machine Learning, pages 1861–1869, 2015.

Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 4700–4708, 2017.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. arXiv preprint arXiv:1312.6114, 2013.

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

Bernard Osgood Koopman. On distributions admitting a sufficient statistic. Transactions of the American Mathematical society, 39(3):399–409, 1936.

Ranganath Krishnan, Mahesh Subedar, and Omesh Tickoo. Efficient priors for scalable variational inference in bayesian deep neural networks. In Proceedings of the IEEE International Conference on Computer Vision Workshops, pages 0–0, 2019.

Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In Advances in neural information processing systems, pages 6402–6413, 2017.

Felix McGregor, Arnu Pretorius, Johan du Preez, and Steve Kroon. Stabilising priors for robust bayesian deep learning. arXiv preprint arXiv:1910.10386, 2019.

Andrew Miller, Nick Foti, Alexander D'Amour, and Ryan P Adams. Reducing reparameterization gradient variance. In Advances in Neural Information Processing Systems, pages 3708–3718, 2017.

Michael Newton, Nicholas G Polson, and Jianeng Xu. Weighted bayesian bootstrap for scalable bayes. arXiv preprint arXiv:1803.04559, 2018.

John Paisley, David Blei, and Michael Jordan. Variational bayesian inference with stochastic search. arXiv preprint arXiv:1206.6430, 2012.

Tim Pearce, Mohamed Zaki, and Andy Neely. Bayesian neural network ensembles. arXiv preprint arXiv:1811.12188, 2018.

Rajesh Ranganath, Sean Gerrish, and David Blei. Black box variational inference. In Artificial Intelligence and Statistics, pages 814–822, 2014.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In International Conference on Medical image computing and computer-assisted intervention, pages 234–241. Springer, 2015.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556, 2014.

Florian Wenzel, Kevin Roth, Bastiaan S. Veeling, Jakub Swiatkowski, Linh Tran, Stephan Mandt, Jasper Snoek, Tim Salimans, Rodolphe Jenatton, and Sebastian Nowozin. How good is the bayes posterior in deep neural networks really?, 2020.

A Wu, S Nowozin, E Meeds, RE Turner, JM Hernández-Lobato, and AL Gaunt. Deterministic variational inference for robust bayesian neural networks. In 7th International Conference on Learning Representations, ICLR 2019, 2019.

Graph Reparametrizations for Enabling 1000+ Monte Carlo Iterations In Bayesian Deep Neural Networks Appendix

In this document we provide more details about experiments, introduce our interactive application to analyze the quality of KL approximation, and give examples of computation graphs of KL terms for different distributions, in comparison between direct implementation and our parameterization technique. Proofs of the results in the main paper can also be found towards the end of the document.

1 EXPERIMENTS DETAILS

A working version of the code is attached in the directory "main_code". In our experiments, we follow the reweighting scheme for mini-batches proposed by [Graves, 2011] as $\beta = \frac{1}{B}$, where B is number of mini-batches. For all experiments with VGG, we decrease the number of nodes by half in the last dense layers to fit the Bayesian model on a single GPU. We choose an exponential family with 2 parameters, which results in doubling the number of parameters compared to the original networks.

2 MAKING YOUR OWN BAYESIAN NETWORK, USING OUR API

Figure 1 provides an example of how to implement your own Bayesian neural network with our API.

3 COMPUTATION GRAPHS

In this section we demonstrate computation graphs corresponding to the MC estimation of one of the expectation terms in KL (sometimes it is called KL cross-entropy): $E_{Q_\theta} \log p(w)$, where Q_θ is the approximate posterior distribution with pdf q_θ, and $p(w)$ is the prior distribution on w. We compare the size of computation graphs for different numbers of MC iterations for a direct implementation and our reparameterization method.

3.1 APPROXIMATE POSTERIOR: RADIAL(μ, σ^2); PRIOR: GAUSSIAN(0, 1)

For the following setup there is no closed form solution for KL term, and approximation with MC sampling is required. Samples from approximate posterior Q_θ can be generated as $\mu + \sigma\xi$, where $\xi = \frac{w}{|w|}r$, $w \sim \text{MVN}(0, I)$, $r \sim N(0, 1)$. Assumption about the prior gives us the following term to estimate $E_{Q_\theta} \log(\exp(-w^2))$. Figure 2 shows computation graphs which correspond to different number of MC iterations. We can see that with the direct implementation, the size is proportional to the number of MC iterations, while our approach constructs a graph whose size is independent of the number of MC iterations.

4 INTERACTIVE APPLICATION TO EVALUATE MC APPROXIMATION OF KL TERMS

4.1 EXAMPLE

To demonstrate the relationship between MC estimation quality of the KL term and number of MC iterations, we provide an interactive Shiny application in this supplement. If one assumes that approximate posterior and prior are Gaussian distributed, in this setting, we can calculate the "ground truth" KL. The main purpose here is to evaluate MC approximation by calculating the sample variance. The main parameters which will influence the quality of the MC approximation are: number of MC iterations, choice of variance of the approximate posterior distribution, and the size of the model (i.e. number of parameters in Neural Network). All these parameters can be set in our application. In addition, we provide an option to plot results in log-scale (for a better comparison). Additionally, graphs can be zoomed in, by highlighting a selected zone on the plot and double clicking (to zoom out, double click again).

Sample runs of our application (if the reader cannot run

Accepted for the 37th Conference on Uncertainty in Artificial Intelligence (UAI 2021).

```
import bayes_layers as bl
class AlexNet(nn.Module):
    def __init__(self, num_classes, in_channels,
                 **bayes_args):
        super(AlexNet, self).__init__()
        self.conv1 = bl.Conv2d(in_channels, 64,
                               kernel_size=11, stride=4,
                               padding=5,
                               **bayes_args)
        self.classifier = bl.Linear(1*1*128,
                                    num_classes,
                                    **bayes_args)
        ...

    def forward(self, x):
        kl = 0
        for layer in self.layers:
            tmp = layer(x)
            if isinstance(tmp, tuple):
                x, kl_ = tmp
                kl += kl_
            else:
                x = tmp

        x = x.view(x.size(0), -1)
        logits, _kl = self.classifier.forward(x)
        kl += _kl
```

Figure 1: An example of how to implement your own version of the bayesian neural network, using our API. We need to import bayesian layers module, which provides new functional for convolution1d, convolution2d, convolution3d and fully connected layers. In addition we need to redefine forward function, as shown.

the tool) appear in Figure 3. We fix variance equal to 10^{-4}, number of MC iterations up to 10^3, number of simulations per MC equals to 10^2 (to smooth the variance estimation). Then we compare the variance of 4 different models with number of parameters: $10^2, 10^4, 10^6$ and 10^8, and plot the results on the bottom figure with a log-scale. We see that despite the small variance, with growing size of the model it is necessary to increase number of MC iterations to decrease the variance of the MC estimator for KL.

4.2 INSTALLATION

Files are located in the directory "interactive_app". There are two ways to prepare our application for execution, both of them are handled by the integer parameter “method":

```
install_shiny_mc_repar.sh method
```

“method" can be one of 2 values: 1 or 2

1. If you have R installed, then the following packages are required to be installed and their installation is handled automatically:

   ```
   c("shiny", "RColorBrewer",
     "dplyr", "ggplot2", "latex2exp")
   ```

2. If you would like to avoid installing R, but you have Docker installed, the script creates a Docker image with all necessary dependencies. It will take about 1.8GB of space and can be checked by running

   ```
   docker images
   ```

4.3 EXECUTION

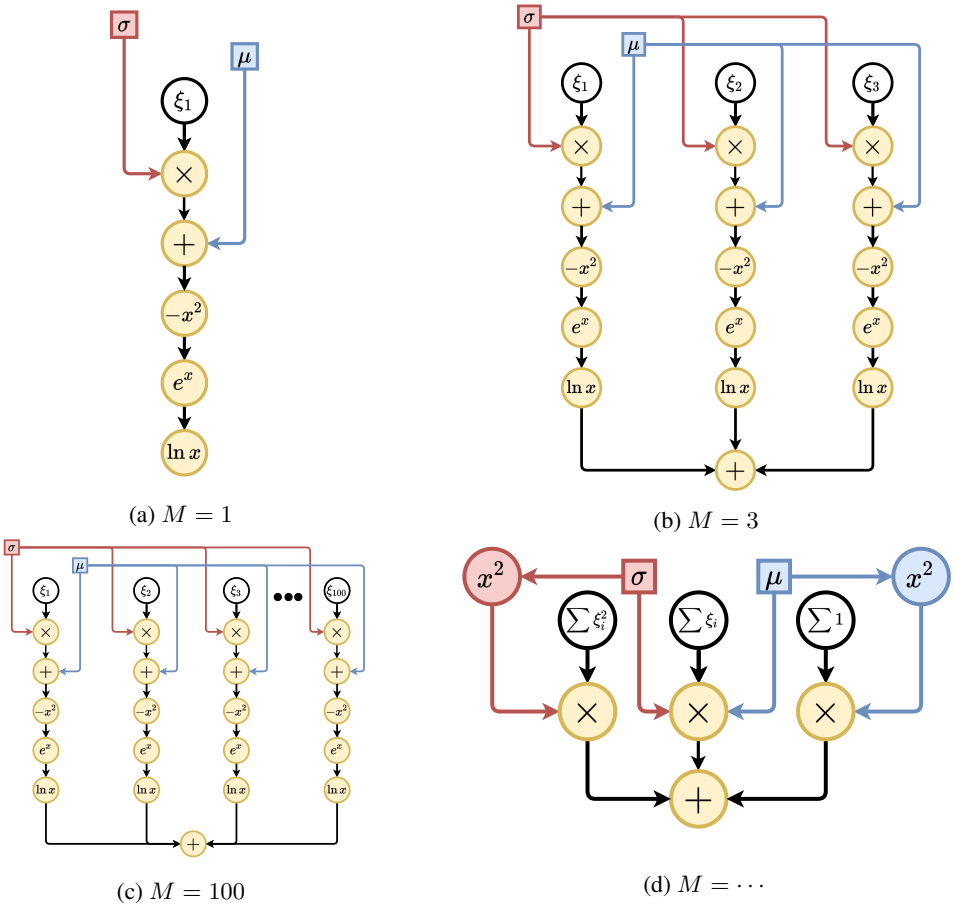
After installation is successful, to run the application, execute the following script with a new “method" parameter:

```
run_shiny_app.sh method
```

“method" can be one of 2 values: 1 and 2

1. (R route) It will start app automatically in a browser.

2. (Docker route) In this case script starts shiny application and provides a local address, which you can access through the browser (this is address on your local machine, not external: http://localhost:3838)



(a) $M = 1$

(b) $M = 3$

(c) $M = 100$

(d) $M = \cdots$

Figure 2: Computation graphs corresponding to MC approximation of KL term. (a)-(c) direct implementation, (d) - our method for any number of MC iterations

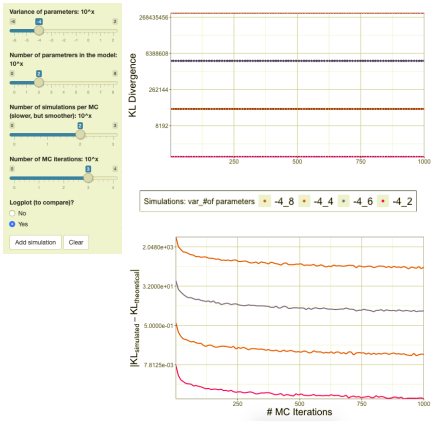


Figure 3: Demonstration of interactive Shiny application to evaluate the performance of MC estimation.

4.4 ADDITIONAL INFORMATION ABOUT VERSION OF PACKAGES, WHICH WERE USED TO RUN APPLICATION

```
> sessionInfo()
```

```
R version 3.4.2 (2017-09-28)
Platform: x86_64-apple-darwin15.6.0
Running under: macOS  10.14.6

attached packages:
latex2exp_0.4.0
ggplot2_2.2.1
dplyr_0.7.4
RColorBrewer_1.1-2
shiny_1.0.5
```

5 THEORY: PROOFS AND CLARIFICATIONS

5.1 PROOFS OF MAIN RESULTS

Theorem 1 *If $W(\theta, \xi) = \eta(\theta)T(\xi)$ $(S = 1)$, then there exists a parametrization tuple P with $d_P = 1$ for following functions $g(w)$: w^k, $\log(w)$, and $\frac{1}{w^k}$.*

First, we are going to show that for $g(w)$: w^k, $\log(w)$, and $\frac{1}{w^k}$ there exists a parametrization tuple P of a specific form,

and then we show that for these tuples $d_P = 1$.

Case (1). $\mathbf{g(w) = w^k}$

$$\begin{aligned}\sum_{i=1}^{M} g(w_i) &= \sum_{i=1}^{M} w_i^k = \sum_{i=1}^{M} (\eta(\theta)T(\xi_i))^k \\ &= \eta^k(\theta)\sum_{i=1}^{M} T^k(\xi_i) = \eta^k(\theta)T^k(\xi) \\ &= n(\theta)t(\xi)\end{aligned}$$

Case (2). $\mathbf{g(w) = \log(w)}$

$$\begin{aligned}\sum_{i=1}^{M} g(w_i) &= \sum_{i=1}^{M} \log(w_i) = \sum_{i=1}^{M} \log(\eta(\theta)T(\xi_i)) \\ &= M\log(\eta(\theta)) + \sum_{i=1}^{M} \log(T(\xi_i)) \\ &= M\log(\eta(\theta)) + \log(T(\xi)) \\ &= n(\theta)t(\xi)\end{aligned}$$

Case (3). $\mathbf{g(w) = \frac{1}{w^k}}$

$$\begin{aligned}\sum_{i=1}^{M} g(w_i) &= \sum_{i=1}^{M} \frac{1}{w_i^k} = \sum_{i=1}^{M} \frac{1}{(\eta(\theta)T(\xi_i))^k} \\ &= \frac{1}{\eta^k(\theta)}\sum_{i=1}^{M} \frac{1}{T^k(\xi_i)} \\ &= n(\theta)t(\xi)\end{aligned}$$

We see that for all $g(w)$ from the list, we identify a parametrization tuple $P = (G(n,t), n(\theta), t(\xi))$, such that $G(n,t) = nt$ and $n(\theta)$, $t(\theta)$ depends on choice of $g(w)$. Clearly, for all these parametrization tuples P, $d_P = 1$.

Theorem 2 *If $W(\theta,\xi) = \sum_{s=1}^{S} \eta_s(\theta)T_s(\xi)$, and $g(w) = w^k$, then there exists a parametrization tuple P with $d_P = \binom{k+S-1}{S-1}$.*

Consider an MC expression $\frac{1}{M}\sum_{i=1}^{M} g(w_i)$. Given assumptions on $g(w) = w^k$ and $W = \sum_{s=1}^{S} \eta_s(\theta)T_s(\xi)$, we get:

$$\sum_{i=1}^{M} g(w_i) = \sum_{i=1}^{M} w_i^k = \sum_{i=1}^{M} \left(\sum_{s=1}^{S} \eta_s(\theta)T_s(\xi_i)\right)^k$$

We observe that $(\sum_{s=1}^{S} \eta_s(\theta)T_s(\xi_i))^k$ is a polynomial of order k with S indeterminates $T_s(\xi_i)$, and coefficients $\eta_s(\theta)$ independent of ξ_i. Let us denote the polynomial as $p_k^S(T(\xi_i);\eta(\theta))$. Since coefficients are independent of ξ_i for all i, then

$$\sum_{i=1}^{M} p_k^S(T(\xi_i);\eta(\theta)) = p_k^S(T(\xi);\eta(\theta)),$$

where $\xi = (\xi_1, \ldots, \xi_M)$. Which results in the following parametrization tuple $P = (G, n(\theta), t(\xi))$, such that $G(n,t) = \sum_{i=1}^{d_P} n_i t_i$, and n_i, t_i are coefficients and indeterminates of the polynomial $p_k^S(T(\xi);\eta(\theta))$.
The expansion of a polynomial of order k with S indeterminates has $\binom{k+S-1}{S-1}$ coefficients, exactly the number of interactions d_P.

Corollary 3 *If $W(\theta,\xi) = \sum_{s=1}^{S} \eta_s(\theta)T_s(\xi)$, and $g(w) = p_K(w)$, then there exists parametrization tuple P, such that for any M iterations*

$$d_P = \binom{K+S}{S} - 1. \tag{1}$$

Note. We consider a polynomial $p_K(w) = \sum_{k=0}^{K} a_k w^k$, such that coefficients a_k do not depend on optimized parameter θ. Since the first element a_0 of the polynomial $p_k(w)$ is not important for the analysis of d_P, we can ignore it and compute d_P as presented in Eq. 1. However, if there is a need to consider a_0, then one only needs to add $+1$ to the Eq. 1.

Lemma 1 *Consider a polynomial of order k with $S+1$ indeterminates*

$$\left(\sum_{s=1}^{S} \eta_s(\theta)T_s(\xi) - a\right)^k,$$

where a is a constant. If there $\exists j : T_j(\xi) = c = const$, then

$$\left(\sum_{s=1}^{S} \eta_s(\theta)T_s(\xi) - a\right)^k = \left(\sum_{s=1}^{S} \eta_s^*(\theta)T_s(\xi)\right)^k$$

where $\eta^* = (\eta_1^*, \ldots, \eta_S^*) : \forall s \neq j,\ \eta_s^*(\theta) = \eta_s(\theta)$ and $\eta_j^*(\theta) = \eta_j(\theta) - a/c$.

The proof is direct by substituting $\eta_j^*(\theta)$.

Theorem 3 *Let $W(\theta,\xi) = \sum_{s=1}^{S} \eta_s(\theta)T_s(X)$, $S \geq 2$. If an approximation of $g(w)$ is made with K Taylor terms, then Corollary 3 applies.*

Consider Taylor approximation of $g(w)$, with proper a and K terms:

$$g(w) = \sum_{k=0}^{K} \frac{g^{(k)}(a)}{k!}(w-a)^k, \text{ where } K \text{ can be } \infty.$$

Then,

$$\begin{aligned}\sum_{i=1}^{M} g(w_i) &= \sum_{i=1}^{M} g(w_i) \\ &= \sum_{i=1}^{M}\sum_{k=0}^{K} c_k (w_i - a)^k \\ &= \sum_{k=0}^{K}\sum_{i=1}^{M} c_k (w_i - a)^k \\ &= \sum_{k=0}^{K}\sum_{i=1}^{M} c_k \left(\sum_{s=1}^{S} \eta_s(\theta) T_s(\xi_i) - a\right)^k \end{aligned} \quad (2)$$

From this point there are 2 ways to apply Corollary 3:

1. Polynomial of order K with $S+1$ indeterminates
Eq. (2) can be considered as a polynomial of order K with $S+1$ indeterminates. Then according to Corollary 3, for corresponding parametrization tuple P, $d_P = \frac{(K+1)\binom{K+S+1}{S}}{S+1} - 1$. However, since a in Eq. 2 is constant, then $p_K(w-a)$ has K terms depending just on a and can be disregarded in the computation graph, since there is no interaction with parameters. This results in

$$d_P = \frac{(K+1)\binom{K+S+1}{S}}{S+1} - (K+1)$$

2. Polynomial of order K with S indeterminates
In some cases Eq. (2) can be considered as polynomial of order K with S new indeterminates, following Lemma 1. Then according to Corollary 3, in addition to a node responsible for reparameterization of $\eta^*(\theta)$, we get

$$d_P = \frac{(K+1)\binom{K+S}{S-1}}{S} + 1$$

which reduces to the form in Eq. (1).

References

Alex Graves. Practical variational inference for neural networks. In Advances in neural information processing systems, pages 2348–2356, 2011.