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

Jurijs Nazarovs$^{1,2}$ Ronak R. Mehta$^{1,2}$ Vishnu Suresh Lokhande$^{1,2}$ 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 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.

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 $Q_\theta$ of our choice, which depends on learned parameters $\theta$. Let $W_\theta = (W^1_\theta, \ldots, W^D_\theta)$ denote a random vector with a distribution $Q_\theta$ and pdf $q_\theta$. VI seeks to find $\theta$ such that $Q_\theta$ is as close as possible to the real (unknown) posterior $P(W|y, x)$, accomplished by minimizing the $KL$ divergence between $Q_\theta$ 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 $W^d_\theta$, for $d \in 1, \ldots, D$, i.e. $p(W) = \prod_{d=1}^D p^d(W^d)$ and $q_\theta(W_\theta) = \prod_{d=1}^D q_\theta^d(W^d_\theta)$,

$$\theta^* = \arg \min \theta \ KL (q_\theta||p) - E_{q_\theta}[\ln p(y|W, x)]$$

$$KL (q_\theta||p) = \sum_{d=1}^D E_{q_\theta^d} [\ln q_\theta^d(w)] - E_{q_\theta^d} [\ln p^d(w)]$$

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

$$E_{q_θ}[g(w)] = \int g(w)q_θ(w)dw \approx \frac{1}{M} \sum_{i=1}^{M} g(w_i),$$

where $w_i \sim Q_θ$. (3)

Expected value terms in (3) 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 $\hat{KL}$ 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.

1 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 naive 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.
Table 1: Summary list of approximate posterior distributions $q_{\theta}$ 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 $q_{\theta}$ and $p(w)$. Reference: Radial [Farquhar et al., 2020], Horseshoe [Ghosh and Doshi-Velez, 2017].

<table>
<thead>
<tr>
<th>Sampling: $W(\theta, \xi)$</th>
<th>Approximate Posterior p.d.f. $q_{\theta}$</th>
<th>Prior p.d.f. $p(w)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scaling property family:</td>
<td>Exponential(\theta)</td>
<td>Standard Wald(\theta)</td>
</tr>
<tr>
<td>$W(\theta, \xi) = \theta \xi$</td>
<td>Rayleigh(\theta)</td>
<td>Weibull(k, \theta)</td>
</tr>
<tr>
<td></td>
<td>Erlang(k, \theta)</td>
<td>Gamma(k, \theta)</td>
</tr>
<tr>
<td></td>
<td>Error(\alpha, \theta, c)</td>
<td>Log-Gamma(k, \theta)</td>
</tr>
<tr>
<td></td>
<td>Inverse-Gamma(\theta, \xi)</td>
<td></td>
</tr>
<tr>
<td>Location-Scale family:</td>
<td>Normal(\mu, \sigma)</td>
<td>Logistic(\mu, \sigma)</td>
</tr>
<tr>
<td>$W(\theta, \xi) = \mu + \sigma \xi$,</td>
<td>Lognormal(\mu, \sigma)</td>
<td>Horseshoe(\mu, \sigma)</td>
</tr>
<tr>
<td>$\theta = (\mu, \sigma)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Horseshoe(\mu, \sigma)</td>
<td></td>
</tr>
<tr>
<td>Corollary</td>
<td>Log-Normal(\mu, \sigma)</td>
<td>Dirichlet</td>
</tr>
</tbody>
</table>

3 COMPUTATION GRAPHS FOR MC ITERATIONS

Despite the ability to approximate the expectation in principle, the minimization in \cite{11} via \cite{12} is difficult for common architectures, and relies on gradient computations at each iterate. Standard implementations make use of automatic differentiation based on computation graphs \cite{Griewank}. 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.

\textbf{Definition 1} Consider $w$ as sampled based on a parameter $\theta$ and an ancillary random variable $\xi$, 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 \textbf{parameterization tuple} for the function $F$, where $\circ$ is the Hadamard product. Let $d_{P}$ be the dimension of $\circ t$, corresponding to the number of nodes requiring gradients with respect to $\theta$.

To demonstrate the application of the Def.\cite{11} as an example, consider the computation graph for the MC approximation of the function $g(w) = w^2$ in \cite{3} and given one weight $W_{\theta} = N(\mu, \sigma^2)$. Applying the reparameterization trick: $W_{\theta} = \mu + \sigma \xi, \xi \sim N(0, 1)$, the Python form is,

\begin{verbatim}
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
\end{verbatim}

The computation graph, a function of both the parameters $\theta = (\mu, \sigma)$ and of the auxiliary samples $\xi_{1}, \xi_{2}$, and $\xi_{3}$, generated by PyTorch/AutoGrad for $M = 3$ iterations of this loop is shown in Figure \ref{fig:MC_i}. According to Def.\cite{11} $d_{P} = 9$ and

\begin{align*}
    n(\theta) &= (\mu^2, 2\mu \sigma, \sigma^2, \mu^2, 2\mu \sigma, \sigma^2, \mu^2, 2\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{align*}

Naively, the graph size grows linearly $O(M)$ with the number of MC iterations, as in the direct implementation (Fig. \ref{fig:MC_i}). For Bayesian VI in DNNs, this is a problem. We need to perform MC approximations of $KL$ terms at every layer. Also, \cite{Miller} 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. \ref{fig:MC_i}). For the right parameterization tuple $P$, we can achieve $d_{P} = 3$. This leads us to,

\textbf{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 \( Q_\theta \) 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

\[
p(w; \zeta) = h(w) \exp(\eta(\zeta)T(w) - A(\zeta))
\]

(4)

where \( \zeta \) 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 \( Q_\theta \). 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 \( q(w) = \log q_\theta(w) / \log p(w) \). Assume that the factorization form \( 4 \) of distributions \( q_\theta(w) \) (and similarly \( p(w) \)) 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:

\[
\frac{1}{M} \sum_{i=1}^{M} \{ \log h(w(\theta, \xi_i)) + \eta(\theta)T(w(\theta, \xi_i)) \} - A(\theta)
\]

(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 \( \theta \), i.e., number of parameters defining the distribution \( Q_\theta \). For example, for the Exponential(\( \lambda \)): \( S = 1 \) and \( \theta = (\lambda) \); for Gaussian(\( \mu, \sigma \)): \( S = 2 \) 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 2 applies to \( W(\theta, \xi) \) and \( g(W(\theta, \xi)) \) if \( g(w(u)) \) 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 \( d_P = \left( k + S - 1 \right) / S - 1 \).

### 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 naive implementation. For example, consider \( \sum_{i=1}^{M} (\mu + \sigma \xi)^2 \). When \( M = 1 \), the naive construction would have \( d_P = 2 \), \( (n = (\mu, \sigma), t = (1, \xi)) \), while a “nicer” tuple may have \( d_P = 3 \) independent of \( M \) \( (n = (\mu, 2\mu \sigma, \sigma^2), t = (1, \xi, \xi^2)) \).

### 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 \( g(w(u)) \) is \( w^k \).

### 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., \( \text{LogNormal}(\mu, \sigma^2) \) can be generated as \( 4^{\text{exp}(N(\mu, \sigma^2))} \). Table 1 summarizes the choice of \( q_\theta \) 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

\[
d_P = \left( K + S - 1 \right) / S - 1.
\]

(7)
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 \( M_{\text{max}} \), such an approximation of \( g(w) \) allows a tradeoff between accuracy of running just \( M_{\text{max}} \) iterations for the real \( g(w) \) versus approximating \( g(w) \) with \( K(M_{\text{max}}) \) terms and running \( M \gg M_{\text{max}} \) 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 \( y \sim N(u^b, \sigma) \), where \( \sigma \) is fixed. Then,

\[
\log p(y_b \mid w, x_b) = \log \left( \frac{1}{\sqrt{2\pi}} \exp\left( -\frac{1}{2} (y_b - u_b^L)^2 \right) \right) = \log \frac{1}{\sqrt{2\pi}} - \frac{1}{2} y_b^2 - y_b u_b^L + \frac{1}{2} (u_b^L)^2.
\]

**Classification:** Consider a binary classification problem.
Then, \( y \sim \text{Bern}(p) \), where \( p = \frac{1}{1 + \exp(-u^L)} \). Thus,

\[
\log p (y_b | w, x_b) = \log \left( p(y_b | 1 - p)^{1 - y_b} \right) = -\log \left( 1 + e^{-u_b^L} \right) - (1 - y_b) u_b^L
\]

\[-\log(2) + \frac{u_b^L}{2} - \frac{(u_b^L)^2}{4} + O \left( (u_b^L)^3 \right) \]

\[-(1 - y_b) u_b^L.
\]

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 \( \mathbb{E}_{q_b} [\log p(y|w, x)] \) term for the minibatch (of size \( B \)) can be written as

\[
S_1 := \frac{1}{B} \sum_{b=1}^{B} \mathbb{E}_{q_b} [\log p(y_b|w, x_b)].
\]

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 \( V(S_1) \),

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

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 Densenets [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: \( \xi \sim \mathcal{MVN}(0, I), r \sim N(0, 1) \) 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 \( q_b \) 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 [ ] 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. [ ]). 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. [ ] we are able to run for \( M = 1000 \) or more. (b) **The significant reduction of model size on the**
GPU results in a reduction of training time per batch, up to $5 \times$ (Figure 6), 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) reases 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).
We train the model on with several runs of training when we are allowed to use with pre-processed MR images of size . We look at a specific setting where we have individuals, and validate on the re-

<table>
<thead>
<tr>
<th>Confidence</th>
<th>0.5</th>
<th>0.6</th>
<th>0.7</th>
<th>0.8</th>
<th>0.9</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>m = 1</td>
<td>63.07 ± 1.47</td>
<td>62.14 ± 1.59</td>
<td>63.01 ± 0.64</td>
<td>64.13 ± 4.15</td>
<td>59.59 ± 4.40</td>
<td>60.71 ± 15.15</td>
</tr>
<tr>
<td>m = 100</td>
<td>64.39 ± 4.59</td>
<td>66.28 ± 4.19</td>
<td>66.05 ± 1.00</td>
<td>67.77 ± 4.00</td>
<td>66.82 ± 1.24</td>
<td>87.50 ± 17.68</td>
</tr>
<tr>
<td>∆</td>
<td>1.33</td>
<td>4.09</td>
<td>3.04</td>
<td>3.64</td>
<td>7.23</td>
<td>26.79</td>
</tr>
</tbody>
</table>

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 9(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.

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


