Statistics and Computing (2023) 33:119
https://doi.org/10.1007/s11222-023-10285-5

ORIGINAL PAPER

Approximate blocked Gibbs sampling for Bayesian neural networks

Theodore Papamarkou[1]

Received: 4 January 2023 / Accepted: 20 July 2023 / Published online: 10 August 2023

Abstract
In this work, minibatch MCMC sampling for feedforward neural networks is made more feasible. To this end, it is proposed to sample subgroups of parameters via a blocked Gibbs sampling scheme. By partitioning the parameter space, sampling is possible irrespective of layer width. It is also possible to alleviate vanishing acceptance rates for increasing depth by reducing the proposal variance in deeper layers. Increasing the length of a non-convergent chain increases the predictive accuracy in classification tasks, so avoiding vanishing acceptance rates and consequently enabling longer chain runs have practical benefits. Moreover, non-convergent chain realizations aid in the quantification of predictive uncertainty. An open problem is how to perform minibatch MCMC sampling for feedforward neural networks in the presence of augmented data.

Keywords Approximate MCMC · Bayesian inference · Bayesian neural networks · Blocked Gibbs sampling · Minibatch sampling · Posterior predictive distribution

1 Introduction

Scope. This paper renders feedforward neural networks more amenable to approximate MCMC sampling of their parameters by splitting the parameters into subgroups. Moreover, it identifies several advantages of such a sampling approach.

Motivation. Why consider approximate MCMC sampling algorithms for deep learning? The answer stems from a general merit of MCMC, namely uncertainty quantification. This work demonstrates how approximate MCMC sampling of neural network parameters quantifies predictive uncertainty in classification problems.

Limitations. Several impediments have inhibited the adoption of MCMC in deep learning; to name three notorious problems, low acceptance rate, high computational cost and lack of convergence typically occur. See Papamarkou et al. (2022) for a relevant review.

Potential. Empirical evidence herein suggests a less dismissive view of approximate MCMC in deep learning. Firstly, a sampling mechanism that takes into account the neural network structure and that partitions the parameter space into smaller parameter blocks retains higher acceptance rate. Secondly, minibatch MCMC sampling of neural network parameters mitigates the computational bottleneck induced by big data. Bayesian marginalization, which is used for making predictions and for assessing predictive performance, is also computationally expensive. However, Bayesian marginalization is embarrassingly parallelizable across test points and along Markov chain length. Thirdly, if assessment of predictive uncertainty via neural networks is the intended outcome, then MCMC convergence in parameter space is viewed as a stepping stone rather than as a pre-requirement for such an outcome. A non-convergent Markov chain acquires valuable predictive information. In fact, it has been shown that the posterior predictive density in Bayesian neural networks can be restricted to a symmetry-free subset of the parameter space (Wiese et al. 2023).

Contributions. The main contribution of this paper is to propose minibatch blocked Gibbs sampling for feedforward neural networks and and to experimentally corroborate the feasibility of such a sampling approach. Without optimizing prior specification, vanishing acceptance rates are overcome by partitioning the parameter space into small blocks. Several observations are drawn from an experimental study of the proposed sampling scheme for feedforward neural networks. Firstly, it is observed that partitioning the parameter space allows to sample from it under increasing width. Secondly, such partitioning alleviates vanishing acceptance rates

✉ Theodore Papamarkou
theo.papamarkou@manchester.ac.uk

[1] Department of Mathematics, The University of Manchester, Manchester, UK

in deeper layers by reducing the proposal variance as depth increases. Thirdly, it is pointed out that increasing the batch size increases the predictive accuracy as expected, as long as the batch size does not become large to the point of yielding vanishing acceptance rates. Fourthly, it is demonstrated that letting the realization of a non-convergent chain run longer increases the predictive accuracy. Fifthly, it is confirmed that one of the open problems is sampling in the presence of augmented data. Finally, it is demonstrated that non-convergent chain realizations aid in the quantification of predictive uncertainty.

Paper structure. The paper is structured as follows. Section 2 reviews the MCMC literature for deep learning. Section 3 revises some basic knowledge, including the Bayesian multilayer perceptron (MLP) model and blocked Gibbs sampling. Section 4 introduces a finer node-blocked Gibbs (FNBG) algorithm to sample MLP parameters. Section 5 utilizes FNBG sampling to fit MLPs to three training datasets, making predictions on three associated test datasets. In Sect. 5, numerous observations are made about the scope of approximate MCMC in MLPs. Section 6 concludes the paper with a discussion about future research directions and about associated limitations.

2 Literature review

This section reviews the literature on MCMC for neural networks. Several other reviews of the topic exist, see for instance Titterington (2004), Wenzel et al. (2020), Izmailov et al. (2021), Papamarkou et al. (2022). New MCMC developments for neural networks, which have appeared after the aforementioned reviews, are included herein.

Four research directions have been mainly taken to develop MCMC algorithms for neural networks. Initially, sequential Monte Carlo (SMC) and reversible jump MCMC were applied on feedforward neural networks. At a second wave of development, minibatch MCMC algorithms became a mainstream approach. More recently, the focus has shifted to Gibbs sampling algorithms and to the construction of priors for Bayesian neural networks.

2.1 SMC and reversible jump MCMC

In early stages of MCMC developments for neural networks, SMC and reversible jump MCMC were applied on MLPs and radial basis function networks (Andrieu et al. 1999; de Freitas 1999; Andrieu et al. 2000; de Freitas et al. 2001). For a historical context of Bayesian approaches to neural networks, see Titterington (2004), Papamarkou et al. (2022).

2.2 Minibatch MCMC

In minibatch MCMC, a target density is evaluated on a subset (minibatch) of the data, thus avoiding the computational cost of MCMC iterations based on the entire data. A stochastic gradient MCMC (SG-MCMC) algorithm is a minibatch MCMC algorithm that uses the gradient of the target density. Welling and Teh (2011) have employed the notion of minibatch to develop a stochastic gradient Langevin dynamics (SG-LD) Monte Carlo algorithm, which is the first instance of SG-MCMC. Chen et al. (2014) have introduced stochastic gradient Hamiltonian Monte Carlo (SG-HMC), which is another instance of SC-MCMC, and applied it to infer the parameters of a Bayesian neural network fitted to the MNIST dataset (Lecun et al. 1998).

SG-LD and SG-HMC are two SG-MCMC algorithms that initiated approximate MCMC research in machine learning. Several variants of SG-MCMC have appeared ever since. Gong et al. (2019) have proposed an SG-MCMC scheme that generalizes Hamiltonian dynamics with state-dependent drift and diffusion, and have demonstrated the performance of this scheme on convolutional and on recurrent neural networks. Zhang et al. (2020) have proposed cyclical SG-MCMC, a tempered version of SG-LD with a cyclical stepsize schedule. Moreover, Zhang et al. (2020) have showcased the performance of cyclical SG-MCMC on a ResNet-18 (He et al. 2016) fitted to the CIFAR-10 and CIFAR-100 datasets (Krizhevsky and Hinton 2009). Alexos et al. (2022) have introduced structured SG-MCMC, a combination of SG-MCMC and structured variational inference (Saul and Jordan 1995). Structured SG-MCMC employs SG-LD or SG-HMC to sample from a factorized variational parameter posterior density. Alexos et al. (2022) have tested the performance of structured SG-MCMC on ResNet-20 (He et al. 2016) architectures fitted to the CIFAR-10, SVHN (Netzer et al. 2011) and fashion MNIST (Xiao et al. 2017) datasets.

2.3 Gibbs sampling

Various Gibbs sampling algorithms have been developed recently with large-scale inference in mind. Bouchard-Côté et al. (2017) have introduced the particle Gibbs split-merge sampler and have explored its performance on four high dimensional datasets. Split Gibbs samplers based on the alternating direction method of multipliers optimization algorithm have been developed to perform Bayesian inference on large datasets and potentially on high-dimensional models (Vono et al. 2019, 2022). Despite not having been applied so far to neural networks, such particle Gibbs and

split Gibbs samplers demonstrate that the idea of splitting parameters or auxiliary variables into subgroups provides one way of attacking the problem of large-scale inference.

Grathwohl et al. (2021) have introduced the Gibbs-with-gradients (GWG) sampler, a general and scalable approximate sampling strategy for probabilistic models with discrete variables. GWG is related to the adaptive Gibbs sampler (Łatuszyński et al. 2013). Grathwohl et al. (2021) have trained GWG on restricted Boltzmann machines, which are generative stochastic neural networks, and have compared GWG to blocked Gibbs sampling, using samples from the latter as the ground truth.

Minibatch MCMC (Sect. 2.2) and Gibbs samplers (current Sect. 2.3) do not constitute two mutually exclusive classes of algorithms. To elaborate on the involved ontology of minibatch MCMC and Gibbs samplers, three remarks are made. Firstly, HMC can be formulated as a Gibbs sampler (Girolami and Calderhead 2011). Secondly, each parameter subgroup in blocked Gibbs sampling can be updated via an MCMC sampling step. For instance, if each parameter subgroup is updated via a Metropolis-Hastings (MH), Langevin dynamics (LD) or HMC sampling step, then the corresponding sampler is known as MH-within-Gibbs, LD-within-Gibbs or HMC-within-Gibbs. Thirdly, the terminology SG-LD and SG-HMC is used in software documentation to refer to algorithms that sample all neural network parameters at one sweep or layer-wise. Nevertheless, when parameter sampling is conducted layer-wise, SG-LD and SG-HMC are misnomers, and the correct sampler names are SG-LD-within-Gibbs and SG-HMC-within-Gibbs, respectively.

2.4 Prior specification

Prior specification for neural networks was considered on the eve of the twenty-first century, see Papamarkou et al. (2022) for a relevant review. Research on prior specification for neural networks has resurged recently, as ridgelet priors (Matsubara et al. 2021) and functional priors (Tran et al. 2022) have been introduced. The functional priors proposed by Tran et al. (2022) have been designed for performing approximate MCMC sampling in Bayesian deep learning.

3 Preliminaries

This section revises two topics, the Bayesian MLP model for supervised classification (Sect. 3.1) and blocked Gibbs sampling (Sect. 3.2). For the Bayesian MLP model, the parameter posterior density and posterior predictive probability mass function (pmf) are stated. Blocked Gibbs sampling provides a starting point in developing the algorithm of Sect. 4 for sampling from the MLP parameter posterior density.

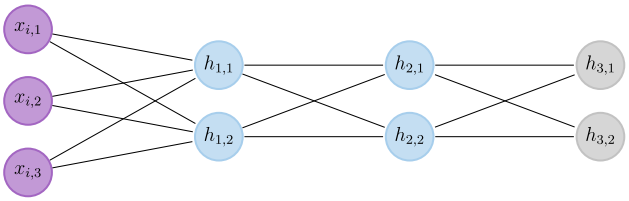


Fig. 1 A graph visualization of MLP(3, 2, 2, 2). Purple, blue and gray nodes correspond to input data, to hidden layer post-activations and to output layer (softmax) post-activations used for making predictions

3.1 The Bayesian MLP model

An MLP is a feedforward neural network comprising an input layer, one or more hidden layers and an output layer (Rosenblatt 1958; Minsky and Papert 1988; Hastie et al. 2016). For a fixed natural number $\rho \geq 2$, an index $j \in \{0, 1, \ldots, \rho\}$ indicates the layer. In particular, $j = 0$ refers to the input layer, $j \in \{1, 2, \ldots, \rho - 1\}$ to one of the $\rho - 1$ hidden layers, and $j = \rho$ to the output layer. Let κ_j be the number of nodes in layer j, and let $\kappa_{0:\rho} = (\kappa_0, \kappa_1, \ldots, \kappa_\rho)$ be the sequence of node counts per layer. MLP($\kappa_{0:\rho}$) denotes an MLP with $\rho - 1$ hidden layers and κ_j nodes at layer j.

An MLP($\kappa_{0:\rho}$) with $\rho - 1$ *hidden layers* and κ_j *nodes* at layer j is defined recursively as

$$g_j(x_i, \theta_{1:j}) = w_j h_{j-1}(x_i, \theta_{1:j-1}) + b_j, \tag{3.1}$$

$$h_j(x_i, \theta_{1:j}) = \phi_j(g_j(x_i, \theta_{1:j})), \tag{3.2}$$

for $j \in \{1, 2, \ldots, \rho\}$. An input data point $x_i \in \mathbb{R}^{\kappa_0}$ is passed to the *input layer* $h_0(x_i) = x_i$, yielding vector $g_1(x_i, \theta_1) = w_1 x_i + b_1$ in the first hidden layer. The parameters $\theta_j = (w_j, b_j)$ at layer j consist of weights w_j and biases b_j. The weight matrix w_j has κ_j rows and κ_{j-1} columns, while the vector b_j of biases has length κ_j. All weights and biases up to layer j are denoted by $\theta_{1:j} = (\theta_1, \theta_2, \ldots, \theta_j)$. An *activation* function ϕ_j is applied elementwise to *pre-activation* vector $g_j(x_i, \theta_{1:j})$, and returns *post-activation* vector $h_j(x_i, \theta_{1:j})$. Concatenating all θ_j, $j \in \{1, 2, \ldots, \rho\}$, gives a parameter vector $\theta = \theta_{1:\rho} \in \mathbb{R}^n$ of length $n = \sum_{j=1}^{\rho} \kappa_j(\kappa_{j-1} + 1)$.

$w_{j,k,l}$ denotes the (k, l)-th element of weight matrix w_j. Analogously, $b_{j,k}$, $x_{i,k}$, $g_{j,k}$ and $h_{j,k}$ correspond to the k-th coordinate of bias b_j, of input x_i, of pre-activation g_j and of post-activation h_j.

MLPs are typically visualized as graphs. For instance, Fig. 1 displays a graph representation of MLP($\kappa_0 = 3, \kappa_1 = 2, \kappa_2 = 2, \kappa_3 = 2$), which has an input layer with $\kappa_0 = 3$ nodes (purple), two hidden layers with $\kappa_1 = \kappa_2 = 2$ nodes each (blue), and an output layer with $\kappa_3 = 2$ nodes (gray). Purple nodes indicate observed variables (input data), whereas blue and gray nodes indicate latent variables (post-activations).

Let $\mathcal{D}_{1:s} = \{(x_i, y_i) : i = 1, 2, \ldots, s\}$ be a training dataset. Each training data point (x_i, y_i) includes an input $x_i \in \mathbb{R}^{\kappa_0}$ and a discrete output (label) $y_i \in \{1, 2, \ldots, \kappa_\rho\}$, $\kappa_\rho \geq 2$. Moreover, let (x, y) be a test point consisting of an input $x \in \mathbb{R}^{\kappa_0}$ and of a label $y \in \{1, 2, \ldots, \kappa_\rho\}$. The *supervised classification problem* under consideration is to predict test label y given test input x and training dataset $\mathcal{D}_{1:s}$. An MLP($\kappa_{0:\rho}$), whose output layer has κ_ρ nodes and applies the *softmax activation function* ϕ_ρ, is used to address this problem. The softmax activation function at the output layer expresses as $\phi_\rho(g_\rho) = \exp(g_\rho)/\sum_{k=1}^{\kappa_\rho} \exp(g_{\rho,k})$.

It is assumed that the training labels $y_{1:s} = (y_1, y_2, \ldots, y_s)$ are outcomes of s independent draws from a categorical pmf with event probabilities given by $\Pr(y_i = k|x_i, \theta) = h_{\rho,k}(x_i, \theta) = \phi_\rho(g_{\rho,k}(x_i, \theta))$, where θ is the set of MLP($\kappa_{0:\rho}$) parameters. It follows that the likelihood function for the MLP($\kappa_{0:\rho}$) model in supervised classification is

$$\mathcal{L}(y_{1:s}|x_{1:s}, \theta) = \prod_{i=1}^{s} \prod_{k=1}^{\kappa_\rho} (h_{\rho,k}(x_i, \theta))^{\mathbb{1}_{\{y_i=k\}}}, \tag{3.3}$$

where $x_{1:s} = (x_1, x_2, \ldots, x_s)$ are the training inputs and $\mathbb{1}$ denotes the indicator function. Interest is in sampling from the parameter posterior density

$$p(\theta|x_{1:s}, y_{1:s}) \propto \mathcal{L}(y_{1:s}|x_{1:s}, \theta)\pi(\theta), \tag{3.4}$$

given the likelihood function $\mathcal{L}(y_{1:s}|x_{1:s}, \theta)$ of Eq. (3.3) and a parameter prior $\pi(\theta)$. For brevity, the parameter posterior density $p(\theta|x_{1:s}, y_{1:s})$ is alternatively denoted by $p(\theta|D_{1:s})$.

By integrating out parameters θ, the posterior predictive pmf of test label y given test input x and training dataset $\mathcal{D}_{1:s}$ becomes

$$p(y|x, \mathcal{D}_{1:s}) = \int \mathcal{L}(y|x, \theta)p(\theta|\mathcal{D}_{1:s})d\theta, \tag{3.5}$$

where $\mathcal{L}$ is the likelihood function of Eq. (3.3) evaluated on (x, y), and $p(\theta|\mathcal{D}_{1:s})$ is the parameter posterior density of Eq. (3.4). The integral in Eq. (3.5) can be approximated via Monte Carlo integration, yielding the approximate posterior predictive pmf

$$\hat{p}(y|x, \mathcal{D}_{1:s}) \simeq \sum_{t=1}^{v} p(y|x, \omega_t), \tag{3.6}$$

where $(\omega_1, \omega_2, \ldots, \omega_v)$ is a Markov chain realization obtained from the parameter posterior density $p(\theta|D_{1:s})$. Maximizing the approximate posterior predictive pmf $\hat{p}(y|x, D_{1:s})$ of Eq. (3.6) yields the prediction

$$\hat{y} = \arg\max_y \{\hat{p}(y|x, D_{1:s})\} \tag{3.7}$$

for test label $y \in \{1, 2, \ldots, \kappa_\rho\}$.

The likelihood function for an MLP model with $\kappa_\rho \geq 2$ output layer nodes, as stated in Eq. (3.3), is suited for multiclass classification with κ_ρ classes. For binary classification, which involves two classes, Eq. (3.3) is related to an MLP with $\kappa_\rho = 2$ output layer nodes. There is an alternative likelihood function based on an MLP model with a single output layer node, which can be used for binary classification; see Papamarkou et al. (2022) for details.

3.2 Blocked Gibbs sampling

A blocked Gibbs sampling algorithm samples groups (blocks) of two or more parameters conditioned on all other other parameters, rather than sampling each parameter individually. The choice of parameter groups affects the rate of convergence (Roberts and Sahu 1997). For instance, breaking down the parameter space into statistically independent groups of correlated parameters speeds up convergence.

To sample from the parameter posterior density $p(\theta|\mathcal{D}_{1:s})$ of an MLP($\kappa_{0:\rho}$) model fitted to a training dataset $\mathcal{D}_{1:s}$, a blocked Gibbs sampling algorithm utilizes a partition $\{\theta_{z(1)}, \theta_{z(2)}, \ldots, \theta_{z(m)}\}$ of the MLP parameters $\theta = (\theta_1, \theta_2, \ldots, \theta_n)$. Due to partitioning $\{\theta_1, \theta_2 \ldots, \theta_n\}$, the parameter subsets $\theta_{z(1)}, \theta_{z(2)}, \ldots, \theta_{z(m)}$ are pairwise disjoint and satisfy $\cup_{q=1}^{m} \theta_{z(q)} = \{\theta_1, \theta_2, \ldots, \theta_n\}$, $m \leq n$. Without loss of generality, it is assumed that each subset $\theta_{z(q)}$ of θ is totally ordered. For any (c, q) such that $1 \leq c \leq q \leq m$, the shorthand notation $\theta_{z(c):z(q)} = (\theta_{z(c)}, \theta_{z(c+1)}, \ldots, \theta_{z(q)})$ is used hereafter. So, the vector $\theta_{z(1):z(m)}$ is a permutation of θ.

Under such a setup, "Appendix A" summarizes blocked Gibbs sampling. At iteration t, for each $q \in \{1, 2, \ldots, m\}$, a blocked Gibbs sampling algorithm draws a sample $\theta_{z(q)}^{(t)}$ of parameter group $\theta_{z(q)}$ from the corresponding conditional density $p(\theta_{z(q)}|\theta_{z(1):z(q-1)}^{(t)}, \theta_{z(q+1):z(m)}^{(t-1)}, \mathcal{D}_{1:s})$. To put it another way, at each iteration, a sample is drawn from the conditional density of each parameter group conditioned on the most recent values of the other parameter groups and on the training dataset.

4 Methodology

This section introduces a blocked Gibbs sampling algorithm for MLPs in supervised classification. MLP parameter blocks are determined by linking parameters to MLP nodes, as elaborated in Sects. 4.1 and 4.2 and as exemplified in Sects. 4.3 and 4.4.

Minibatching and parameter blocking render the proposed Gibbs sampler possible. Blocked Gibbs sampling is typically motivated by increased rates of convergence attained via near-optimal or optimal parameter groupings. Although

low speed of convergence is a problem with MCMC in deep learning, near-zero acceptance rates constitute a more immediate problem. In other words, no mixing is a more pressing issue than slow mixing. By updating a small block of parameters at a time instead of updating all parameters via a single step, each block-specific acceptance rate moves away from zero. So, minibatch blocked Gibbs sampling provides a workaround for vanishing acceptance rates in deep learning. Of course there is no free lunch; increased acceptance rates come at a computational price per Gibbs step, which consists of additional conditional density sampling sub-steps.

In typical SG-LD-within-Gibbs and SG-HMC-within-Gibbs software implementations, one block of parameters is formed for each MLP layer (see Sect. 2.3). A caveat to grouping parameters by MLP layer is that parameter block sizes depend on layer widths. Hence, a parameter block can be large, containing hundreds or thousands of parameters, in which case the problem of low acceptance rate is not resolved. The blocked Gibbs sampler of this paper groups parameters by MLP node and allows to further partition parameters into smaller blocks within each node, thus controlling the number of parameters per block.

While structured SG-MCMC (Alexos et al. 2022) also splits the parameter space into blocks, it uses the parameter blocks to factorize a variational posterior density. Hence, structured SG-MCMC aims to solve the low acceptance and slow mixing problems by factorizing an approximate parameter posterior density. The blocked Gibbs sampler herein factorizes the exact parameter posterior density, relying on finer parameter grouping. Minibatching, which is the only type of approximation employed by the blocked Gibbs sampler of this paper, is an approximation related to the data, not to the MLP model.

The finer node-blocked Gibbs sampler for feedforward neural networks, as presently conceived here, is a minibatch MH-within-Gibbs sampler. The main idea is to update a relatively small block of neural network parameters, thus making it possible to accept states proposed by minibatch MH. Due to taking minibatch MH sampling steps per block of parameters, the sampler is gradient-free. Such a gradient-free approach has been chosen to cap the computational cost. Subject to availability of computing resources, SG-LD or SG-HMC sampling steps can be taken instead of minibatch MH sampling steps.

4.1 Metropolis inside blocks

Blocked Gibbs sampling raises the question how to sample each parameter block from its conditional density. Such conditional densities for MLPs are not available in closed form. Instead, a single Metropolis-Hastings step can be taken to draw a sample from a conditional density. In this case, the resulting blocked Gibbs sampling algorithm is known as Metropolis-within-blocked-Gibbs (MWBG) sampling.

At iteration t of MWBG, a candidate state $\theta^{\star}_{z(q)}$ for parameter block $\theta_{z(q)}$ can be sampled from an isotropic normal proposal density $\mathcal{N}(\theta^{(t-1)}_{z(q)}, \sigma_q^2 I_q)$ centered at state $\theta^{(t-1)}_{z(q)}$ of iteration $t-1$, where I_q is the $|\theta_{z(q)}| \times |\theta_{z(q)}|$ identity matrix, $|\theta_{z(q)}|$ is the number of parameters in block $\theta_{z(q)}$, and $\sigma_q^2 > 0$ is the proposal variance for block $\theta_{z(q)}$. The acceptance probability $a(\theta^{\star}_{z(q)}, \theta^{(t-1)}_{z(q)})$ of candidate state $\theta^{\star}_{z(q)}$ is given by

$$a(\theta^{\star}_{z(q)}, \theta^{(t-1)}_{z(q)}) = \min \left\{ \frac{\pi(\theta^{\star}_{z(q)}) \exp\left(\mathcal{E}(\theta^{(t-1)}, \mathcal{D}_{1:s})\right)}{\pi(\theta^{(t-1)}_{z(q)}) \exp\left(\mathcal{E}(\theta^{\star}, \mathcal{D}_{1:s})\right)}, 1 \right\}, \tag{4.1}$$

where $\mathcal{E}$ denotes the cross-entropy loss function. More details for the acceptance probability $a(\theta^{\star}_{z(q)}, \theta^{(t-1)}_{z(q)})$ are available in "Appendix A".

Algorithm 1 summarizes exact MWBG sampling. To make Algorithm 1 amenable to big data, minibatching can be used by replacing all instances of $\mathcal{D}_{1:s}$ with batches (strict subsets of $\mathcal{D}_{1:s}$); the resulting approximate MCMC algorithm is termed 'minibatch MWBG sampling'.

4.2 Finer blocks

Big data and big models challenge the adaptation of MCMC sampling methods in deep learning. Minibatching provides a way of applying MCMC to big data. It is less clear how to apply MCMC to big neural network models, containing thousands or millions of parameters. Minibatch MWBG sampling proposes a way forward by drawing an analogy between subsetting data and subsetting model parameters. As data batches reduce the dimensionality of data per Gibbs sampling iteration, parameter blocks reduce the dimensionality of parameters per Metropolis-within-Gibbs update.

In an MLP($\kappa_{0:\rho}$) with n parameters, layer j contains $\kappa_j(\kappa_{j-1}+1)$ parameters, of which $\kappa_j\kappa_{j-1}$ are weights and κ_j are biases. So, if parameters are grouped by layer, then the block of layer j contains $\kappa_j(\kappa_{j-1}+1)$ parameters. The number of parameters in the block of layer j grows linearly with the number κ_j of nodes in layer j as well as linearly with the number κ_{j-1} of nodes in layer $j-1$.

If parameters are grouped by node, then each node block in layer j contains $\kappa_{j-1}+1$, of which κ_{j-1} are weights and one is bias. The number of parameters in a node block in layer j does not depend on the number κ_j of nodes in layer j, but it grows linearly with the number κ_{j-1} of nodes in layer $j-1$. MWBG sampling (Algorithm 1) based on parameter grouping by MLP node is termed '(Metropolis-within-)node-blocked-Gibbs (NBG) sampling'.

Algorithm 1 Metropolis-within-blocked-Gibbs (MWBG) sampling based on cross-entropy

Input: training dataset $\mathcal{D}_{1:s}$
Input: initial state $\theta^{(0)}_{z(1):z(m)}$
Input: proposal variances $(\sigma_1^2, \ldots, \sigma_m^2)$ across blocks
Input: number of Gibbs sampling iterations v

for $t = 1, \ldots, v$ **do**
  for $q = 1, \ldots, m$ **do**
    Draw $\theta^{\star}_{z(q)} \sim \mathcal{N}(\theta^{(t-1)}_{z(q)}, \sigma_q^2 I_q)$
    Compute $a(\theta^{\star}_{z(q)}, \theta^{(t-1)}_{z(q)}) = \min\left\{ \dfrac{\pi(\theta^{\star}_{z(q)}) \exp\left(\mathcal{E}(\theta^{(t-1)}, \mathcal{D}_{1:s})\right)}{\pi(\theta^{(t-1)}_{z(q)}) \exp\left(\mathcal{E}(\theta^{\star}, \mathcal{D}_{1:s})\right)}, 1 \right\}$
    Draw $u \sim \mathcal{U}(0, 1)$
    if $u \leq a(\theta^{\star}_{z(q)}, \theta^{(t-1)}_{z(q)})$ **then**
      Set $\theta^{(t)}_{z(q)} = \theta^{\star}_{z(q)}$
    else
      Set $\theta^{(t)}_{z(q)} = \theta^{(t-1)}_{z(q)}$
    end if
  end for
end for

Finer parameter blocks of smaller size can be generated by splitting the $\kappa_{j-1} + 1$ parameters of a node in layer j into β_j subgroups. In this case, each finer parameter block in each node in layer j contains $(\kappa_{j-1} + 1)/\beta_j$ parameters. If hyperparameter β_j is chosen to be a linear function of κ_{j-1}, then the number of parameters per finer block per node in layer j depends neither on the number κ_j of nodes in layer j nor on the number κ_{j-1} of nodes in layer $j-1$. MWBG sampling (Algorithm 1) based on finer parameter grouping per node is termed '(Metropolis-within-)finer-node-blocked-Gibbs (FNBG) sampling'.

Parameter blocks of smaller size increase both the acceptance rate per block and the computational complexity of FNBG sampling. Thus, the number of parameters per block regulates the trade-off between acceptance rates and computational complexity. As a practical guideline, the number of parameters per block can be tuned by reducing it incrementally until non-vanishing acceptance rates are attained in order to make sampling possible. The question of optimal parameter block size for sampling is analogous to the question of optimal learning rate for stochastic optimization. Both of these questions pose hyperparameter optimization problems, which can be approached primarily from an engineering perspective in lieu of theoretical solutions.

4.3 Finer blocks: toy example

The MLP(3, 2, 2, 2) architecture shown in Fig. 1 provides a toy example that showcases layer-based, node-based and finer node-based parameter grouping (more briefly termed 'layer-blocking', 'node-blocking' and 'finer node-blocking'). It is reminded that finer node-based grouping refers to parameter grouping into smaller blocks within each node. Figure 2 shows the directed acyclic graph (DAG) representation of MLP(3, 2, 2, 2), augmenting Fig. 1 with parameter annotations and with a layer consisting of a single node that represents label y_i. Yellow shapes indicate parameters; yellow circles and boxes correspond to biases and weights. Yellow boxes adhere to expository visual conventions of plate models, with each box representing a set of weights. Purple nodes indicate observed variables (input and output data), whereas blue and gray nodes indicate latent variables (post-activations).

Layer-blocking partitions the set of 20 parameters of MLP(3, 2, 2, 2) to three blocks $\theta_{z(1)}$, $\theta_{z(2)}$, $\theta_{z(3)}$, which contain $|\theta_{z(1)}| = 8$, $|\theta_{z(2)}| = 6$, $|\theta_{z(3)}| = 6$ parameters. For instance, the first hidden layer induces block $\theta_{z(1)} = (w_{1,1,1:3}, b_{1,1}, w_{1,2,1:3}, b_{1,2})$, where $w_{j,k,1:l} = (w_{j,k,1}, w_{j,k,2}, \ldots, w_{j,k,l})$.

Node-blocking partitions the set of 20 parameters of MLP(3, 2, 2, 2) to six blocks, as many as the number of hidden and output layer nodes. Each blue or gray node in a hidden layer or in the output layer has its own distinct set of yellow weight and bias parents. Parameters are grouped according to shared parenthood. For instance, the parameters of block $\theta_{z(1)} = (w_{1,1,1:3}, b_{1,1})$, have node $h_{1,1}$ as a common child.

Acceptance probabilities for parameter blocks require likelihood function evaluations. It is not possible to factorize conditional densities to achieve more computationally efficient block updates. For instance, as it can be seen in Fig. 2, changes in block $\theta_{z(1)} = (w_{1,1,1:3}, b_{1,1})$ induced by node $h_{1,1}$ in layer 1 propagate through subsequent layers due to the hierarchical MLP structure, thus prohibiting a factorization of conditional density $p(\theta_{z(1)}|\theta_{z(2):z(6)}, \mathcal{D}_{1:s})$. More formally, each pair of node-based parameter blocks forms

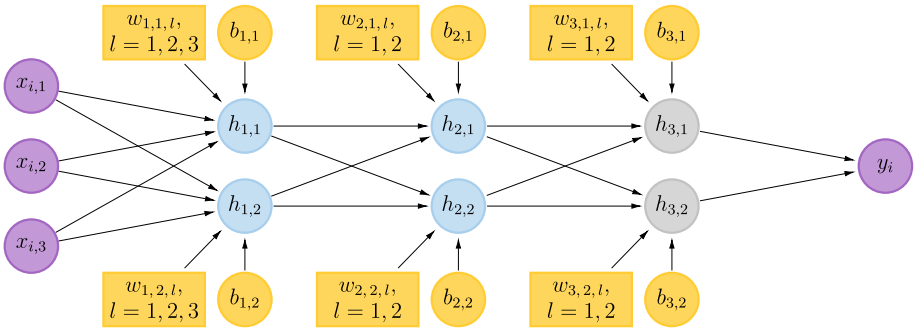


Fig. 2 Visual demonstration of node-based parameter blocking for the MLP(3, 2, 2, 2) architecture. The MLP is expressed as a DAG. Yellow nodes and yellow plates correspond to biases and weights. Each of the blue hidden layer nodes and of the gray output layer nodes is assigned a parameter block of yellow parent nodes in the DAG

a v-structure, having label y_i (purple node) as a descendant. Since training label y_i is observed, such v-structures are activated, and therefore any two node-based parameter blocks are not conditionally independent given label y_i.

As a demonstration of finer node-blocking for MLP(3, 2, 2, 2), set $\beta_1 = 2$ in layer 1. For $\beta_1 = 2$, blocks $\theta_{z(1)} = w_{1,1,1:2}$ and $\theta_{z(2)} = (w_{1,1,3}, b_{1,1})$ are generated within node $h_{1,1}$. Similarly, blocks $\theta_{z(3)} = w_{1,2,1:2}$ and $\theta_{z(4)} = (w_{1,2,3}, b_{1,2})$ are generated within node $h_{1,2}$.

To recap on this toy example, layer-based grouping produces a single block of eight parameters in layer 1, node-based grouping produces two blocks of four parameters each in layer 1, and a case of finer node-based grouping produces four blocks of two parameters each in layer 1. It is thus illustrated that finer blocks per node provide a way to reduce the number of parameters per Gibbs sampling block.

4.4 Finer blocks: MNIST example

After having used MLP(3, 2, 2, 2) as a toy example to describe the basics of finer node-blocking, the wider MLP(784, 10, 10, 10, 10) architecture is utilized to elaborate on the practical relevance of smaller blocks per node. An MLP(784, 10, 10, 10, 10) is fitted to the MNIST (and FMNIST) training dataset in Sect. 5. An MLP(784, 10, 10, 10, 10) contains 8180 parameters, of which 7850, 110, 110 and 110 have children nodes in the first, second, third hidden layer and output layer, respectively.

So, layer-blocking for MLP(784, 10, 10, 10, 10) involves four parameter blocks $\theta_{z(1)}$, $\theta_{z(2)}$, $\theta_{z(3)}$, $\theta_{z(4)}$ of sizes $|\theta_{z(1)}| = 7850$, $|\theta_{z(2)}| = |\theta_{z(3)}| = |\theta_{z(4)}| = 110$. Metropolis-within-Gibbs updates for block $\theta_{z(1)}$ have zero or near-zero acceptance rate due to the large block size of $|\theta_{z(1)}| = 7850$. Although each of blocks $\theta_{z(2)}$, $\theta_{z(3)}$, $\theta_{z(4)}$ has nearly two orders of magnitude smaller size than $\theta_{z(1)}$, a block size of $|\theta_{z(2)}| = |\theta_{z(3)}| = |\theta_{z(4)}| = 110$ might be large enough to yield Metropolis-within-Gibbs updates with prohibitively low acceptance rate.

Node-blocking for MLP(784, 10, 10, 10, 10) entails a block of 785 parameters for each node in the first hidden layer, and a block of 11 parameters for each node in the second and third hidden layer and in the output layer. Thus, node-blocking addresses the low acceptance rate problem related to large parameter blocks for block updates in all layers apart from the first hidden layer.

There is no practical need to carry out finer node-blocking in nodes belonging to the second or third hidden layer or to the output layer of MLP(784, 10, 10, 10, 10), since each block in these layers contains only 11 parameters based on node-blocking. On the other hand, finer node-blocking is useful in nodes belonging to the first hidden layer, since each block related to such nodes contains a large number of 785 parameters. By setting $\beta_1 = 10$, smaller blocks (each consisting of 78 or 79 parameters) are generated in the first hidden layer. So, finer node-blocking disentangles block sizes in the first hidden layer from input data dimensions, making it possible to decrease block sizes and to consequently increase acceptance rates.

5 Experiments

Minibatch FNBG sampling is put into practice to make empirical observations about several characteristics of approximate MCMC in deep learning. In the experiments of this section, parameters of MLPs are sampled. Three datasets are used, namely a simulated noisy version of exclusive-or (Papamarkou et al. 2022), MNIST (Lecun et al. 1998) and fashion MNIST (Xiao et al. 2017). For brevity, exclusive-or and fashion MNIST are abbreviated to XOR and FMNIST. Table 1 displays the correspondence between used datasets and fitted MLPs.

The noisy XOR training and test datasets are visualized in Fig. 9 of "Appendix B". Random perturbations of (0, 0) and of (1, 1), corresponding to gray and yellow points, are mapped to 0 (circles). Moreover, random perturbations of (0, 1) and of (1, 0), corresponding to purple and blue points, are mapped to 1 (triangles). More information about the simulation of noisy XOR can be found in Papamarkou et al. (2022).

Each MNIST and FMNIST image is firstly reshaped, by converting it from a 28×28 matrix to a vector of length

Table 1 Datasets used in the experiments and MLPs fitted to these datasets

Dataset			Neural network	
Name	Sample size		Architecture	# Parameters
	Training	Test		
Noisy XOR	5000	1200	MLP(2, 2, 1)	9
Noisy XOR	5000	1200	MLP(2, 2, 2, 2, 2, 2, 2, 1)	39
MNIST	60000	10000	MLP(784, 10, 10, 10, 10)	8180
FMNIST	60000	10000	MLP(784, 10, 10, 10, 10)	8180

Training and test dataset sample sizes as well as MLP parameter dimensions are shown

$784 = 28 \times 28$, and it is subsequently standardized. This image reshaping explains why the MLP(784, 10, 10, 10, 10) model, which is fitted to MNIST and FMNIST, has an input layer width of 784.

5.1 Experimental configuration

Binary classification for noisy XOR is performed via the likelihood function based on binary cross-entropy, as described in Papamarkou et al. (2022). Multiclass classification for MNIST and FMNIST is performed via the likelihood function given by Eq. (3.3), which is based on cross-entropy.

The sigmoid activation function is applied at each hidden layer of each MLP of Table 1. Furthermore, the sigmoid activation function is also applied at the output layer of MLP(2, 2, 1) and of MLP(2, 2, 2, 2, 2, 2, 2, 1), conforming to the employed likelihood function for binary classification. The softmax activation function is applied at the output layer of MLP(784, 10, 10, 10, 10), in accordance with likelihood function (3.3) for multiclass classification. The same MLP(784, 10, 10, 10, 10) model is fitted to the MNIST and FMNIST datasets.

A normal prior $\pi(\theta) \sim \mathcal{N}(0, 10I)$ is adopted for the parameters $\theta \in \mathbb{R}^n$ of each MLP model shown in Table 1. Thus, a relatively high variance (equal to 10) is assigned a priori to each parameter.

NBG sampling is run upon fitting MLP(2, 2, 1) and MLP(2, 2, 2, 2, 2, 2, 2, 1) to the noisy XOR training set, while FNBG sampling is run upon fitting MLP(784, 10, 10, 10, 10) to the MNIST and FMNIST training sets. So, parameters are grouped by node in MLP(2, 2, 1) and MLP(2, 2, 2, 2, 2, 2, 2, 1), whereas multiple parameter groups per node are formed in the first hidden layer of MLP(784, 10, 10, 10, 10) as elaborated in Sect. 4.4. Parameters are grouped by node from the second hidden layer onwards in MLP(784, 10, 10, 10, 10). All three MLPs of Table 1 are relatively shallow neural networks. However, MLP(784, 10, 10, 10, 10) has two orders of magnitude larger input layer width in comparison to MLP(2, 2, 1) and MLP(2, 2, 2, 2, 2, 2, 2, 1). So, the higher dimension of MNIST and FMNIST input data necessitates finer node-blocking in the first hidden layer of MLP(784, 10, 10, 10, 10). On the other hand, the smaller dimension of noisy XOR input data implies that finer blocks per node are not required in the first hidden layer of MLP(2, 2, 1) or of MLP(2, 2, 2, 2, 2, 2, 2, 1).

A normal proposal density is chosen for each parameter block. The variance of each proposal density is a hyperparameter, thus enabling to tune the magnitude of proposal steps separately for each parameter block. Preliminary FNBG pilot runs have been carried out in order to tune the proposal variances. During this pre-training stage, the proposal variances have been set initially to a single relatively high value across all parameter blocks. Subsequently, the proposal variances of blocks in each hidden layer have been reduced to smaller values in deeper layers until non-vanishing acceptance rates have been attained.

$m = 10$ Markov chains are realized for noisy XOR, whereas $m = 1$ chain is realized for each of MNIST and FMNIST due to computational resource limitations. 110000 iterations are run per chain realization, 10000 of which are discarded as burn-in. Thereby, $v = 100000$ post-burnin iterations are retained per chain realization. Acceptance rates are computed from all 100000 post-burnin iterations per chain.

Monte Carlo approximations of posterior predictive pmfs are computed according to Eq. (3.6) for each data point of each test set. To reduce the computational cost, the last $v = 10{,}000$ iterations of each realized chain are used in Eq. (3.6).

Predictions for noisy XOR are made using the binary classification rule mentioned in Papamarkou et al. (2022). Predictions for MNIST and for FMNIST are made using the multiclass classification rule specified by Eq. (3.7). Given a single chain realization based on a training set, predictions are made for every point in the corresponding test set; the predictive accuracy is then computed as the number of correct predictions over the total number of points in the test set. For the noisy XOR test set, the mean of predictive accuracies across the $m = 10$ realized chains is reported. For the MNIST and FMNIST test sets, the predictive accuracy based on the corresponding single chain realization ($m = 1$) is reported.

5.2 Exact versus approximate MCMC

An illustrative comparison between approximate and exact NBG sampling is made in terms of acceptance rate, predictive accuracy and runtime. The comparison between approximate and exact NBG sampling is carried out in the context of noisy XOR only, since exact MCMC is not feasible for the MNIST and FMNIST examples due to vanishing acceptance rates and high computational requirements.

MLP(2, 2, 2, 2, 2, 2, 2, 1) is fitted to the noisy XOR training set under four scenarios. For scenario 1, approximate NBG sampling is run with a batch size of 100 to simulate $m = 10$ chains. For scenario 2, exact NBG is run to simulate 10 chains. For scenario 3, exact NBG is run until 10 chains are obtained, each having an acceptance rate $\geq 5\%$. For scenario 4, exact NBG is run until 10 chains are acquired, each with an acceptance rate $\geq 20\%$. 11 and 23 chains have been run in total under scenarios 3 and 4, respectively, to get 10 chains that satisfy the acceptance rate lower bounds in each scenario.

It is not suggested to develop a sampling algorithm that relies on some acceptance rate threshold as a criterion for chain retention, since such a criterion would introduce bias in the estimation of the target parameter posterior density. The purpose of this experiment is to showcase that the avoidance of prohibitively low acceptance rates enables the generation of chains with predictive capacity.

For approximate NBG sampling (scenario 1), the proposal variance is set to 0.04. For the three exact NBG sampling scenarios, the proposal variance is lowered to 0.001 in order to mitigate decreased acceptance rates in the presence of increased sample size (5000 training data points) relatively to the batch size of 100 used in approximate sampling.

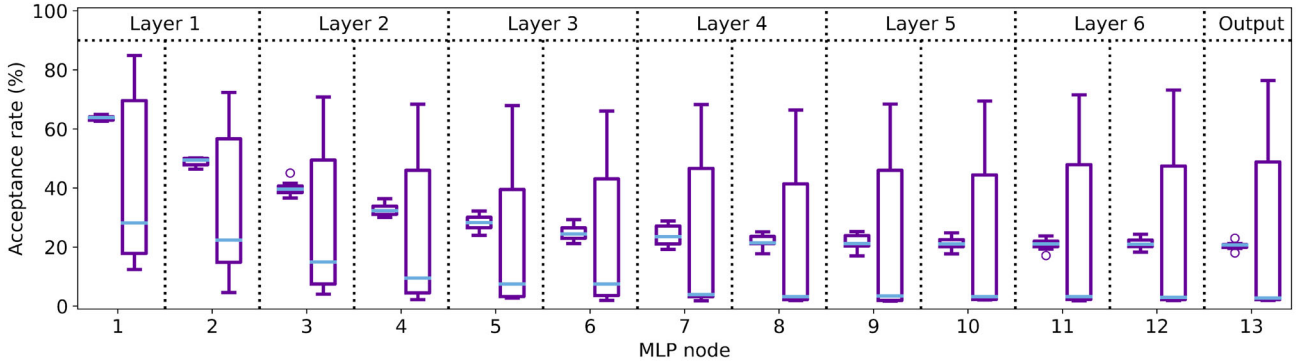
Figure 3a displays boxplots of node-specific acceptance rates for approximate and exact NBG sampling without lower bound conditions on acceptance rates (scenarios 1 and 2). A pair of boxplots is shown for each of the 13 nodes in the six hidden layers and one output layer of MLP(2, 2, 2, 2, 2, 2, 2, 1). The left and right boxplots per pair correspond to approximate and exact NBG sampling. Blue lines represent medians.

Three empirical observations are drawn from Fig. 3a. First of all, approximate NBG attains higher acceptance rates than exact NBG according to the (blue) medians, despite setting higher proposal variance in the former in comparison to the latter (0.04 and 0.001, respectively). Secondly, approximate NBG attains less volatile acceptance rates than exact NBG as seen from the boxplot interquartile ranges. Acceptance rates for exact NBG range from near 0% to about 50% as neural network depth increases, exhibiting lack of stability due to entrapment in local modes in some chain realizations. Thirdly, acceptance rates decrease as depth increases. For instance, exact NBG yields median acceptance rates of 63.83% and 20.72% in nodes 1 and 13, respectively. The attenuation of acceptance rate with depth is further discussed in Sect. 5.3.

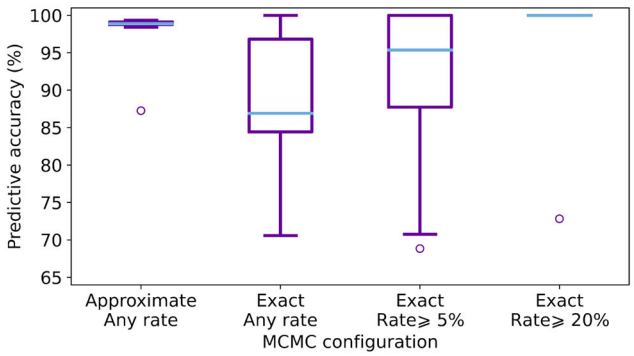
Figure 3b shows boxplots of predictive accuracies for the four scenarios under consideration. Approximate NBG has a median predictive accuracy of 98.88%, with interquartile range concentrated around the median and with a single outlier (87.25%) in 10 chain realizations. Exact NBG without conditions on acceptance rate and exact NBG conditioned on acceptance rate $\geq 5\%$ have lower median predictive accuracies (86.92% and 95.38%) and higher interquartile ranges than exact NBG. Exact NBG conditioned on acceptance rate $\geq 20\%$ attains a median predictive accuracy of 100%; nine out of 10 chain realizations yield 100% accuracy, and one chain gives an outlier accuracy of 72.83%. The overall conclusion is that approximate NBG retains a predictive advantage over exact NBG, since minibatch sampling ensures consistency in terms of high predictive accuracy and reduced predictive variability. Exact NBG conditioned on higher acceptance rates can yield near-perfect predictive accuracy in the low parameter and data dimensions of the toy noisy XOR example, but stability and computational issues arise, as many chains with near-zero acceptance rates are discarded before 10 chains with the required level of acceptance rate ($\geq 20\%$) are obtained.

Figure 3c shows a barplot of runtimes (in hours) for the four scenarios under consideration. Purple bars represent runtimes for the 10 retained chains per scenario, whereas gray bars indicate runtimes for the chains that have been discarded due to unmet acceptance rate requirements. As seen from a comparison between purple bars, approximate NBG has shorter runtime (for retained chains of same length) than exact NBG, which is explained by the fact that minibatching uses a subset of the training set at each approximate NBG iteration. A comparison between gray bars in scenarios 3 and 4 demonstrates that exact NBG runtimes for discarded chains increase with increasing acceptance rate lower bounds. By observing Fig. 3b, c jointly, it is pointed out that predictive accuracy improvements of exact NBG (arising from higher acceptance rate lower bounds) come at higher computational costs.

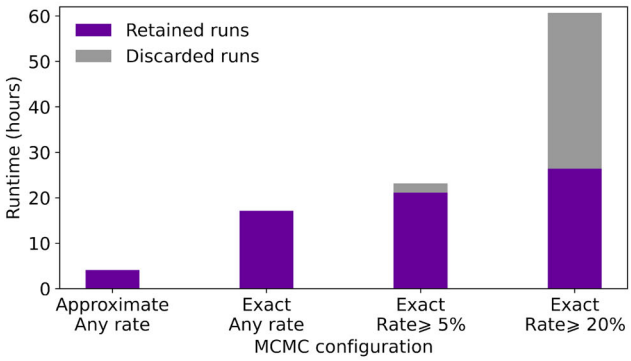
Observation 1 *Exact MCMC algorithms based on the Metropolis-Hastings acceptance mechanism are not feasible for feedforward neural networks due to vanishing acceptance rates and high computational cost. Splitting the parameter space into smaller blocks recovers higher acceptance rates, and minibatch MCMC sampling reduces the computational cost per sampling step. With relatively small penalty in predictive accuracy, minibatch blocked Gibbs sampling makes it possible to traverse the parameter space with reduced computational cost. Being able to shift from no mixing of exact*



(a) Acceptance rate boxplots. The left and right boxplot in each pair correspond to approximate and exact NBG.



(b) Predictive accuracy boxplots.



(c) Runtime barplot.

Fig. 3 A comparison between approximate and exact NBG sampling. MLP(2, 2, 2, 2, 2, 2, 2, 1) is fitted to noisy XOR under four scenarios, acquiring 10 chains per scenario. Scenario 1: approximate NBG with a batch size of 100. Scenario 2: exact NBG. Scenario 3: exact NBG with acceptance rate $\geq$ 5%. Scenario 4: exact NBG with acceptance rate $\geq$ 20%. Chains with acceptance rates below 5% in scenario 3 and below 20% in scenario 4 are discarded until 10 chains are attained in each case

MCMC to slow mixing of approximate MCMC yields gains in predictive accuracy.

5.3 Depth and acceptance rate

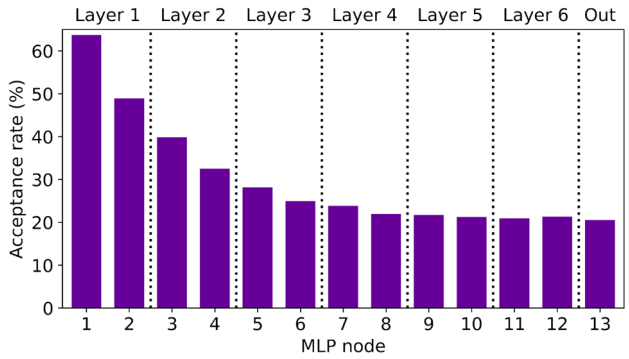
Figure 4 displays mean acceptance rates across $m = 10$ chains realized via minibatch NBG upon fitting MLP(2, 2, 2, 2, 2, 2, 2, 1) to noisy XOR. In particular, Fig. 4a shows the mean acceptance rate for each node in the six hidden layers and one output layer of MLP(2, 2, 2, 2, 2, 2, 2, 1), while Fig. 4b shows the mean acceptance rate for each of these seven (six hidden and one output) layers. A batch size of 100 is used for minibatch NBG. The same set of 10 chains have been used in Figs. 3 and 4.

Figures 3a and 4a provide alternative views of node-specific acceptance rates. The former figure represents such information via boxplots and medians, whereas the latter makes use of a barplot of associated means.

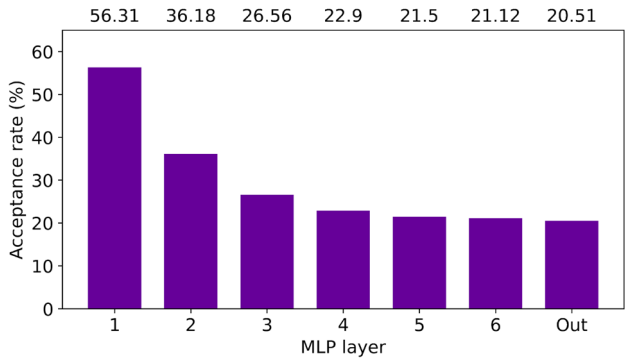
Figure 4 demonstrates that if the proposal variance is the same for all parameter blocks across layers, then the acceptance rate reduces with depth. For instance, it can be seen in Fig. 4b that the acceptance rates for hidden layers 1, 2 and 3 are 56.31%, 36.18% and 26.56%, respectively.

Using a common proposal variance for all parameter blocks across layers generates disparities in acceptance rates, with higher rates in shallower layers and lower rates in deeper layers. These disparities become more pronounced with big data or with high parameter dimensions. For example, sampling MLP(784, 10, 10, 10, 10) parameters with the same proposal variance in all parameter blocks is not feasible in the case of MNIST or FMNIST; the acceptance rates are high in the first hidden layer and drop near zero in the output layer. FNBG sampling enables to reduce the proposal variance for deeper layers, thus avoiding vanishing acceptance rates with increasing depth.

Tables 5 and 6 of "Appendix C" exemplify empirically tuned proposal variances for minibatch FNBG sampling of MLP(784, 10, 10, 10, 10) parameters in the respective cases of MNIST and FMNIST. Batch sizes of 600, 1800, 3000 and 4200 are employed, corresponding to 1%, 3%, 5% and 7% of the MNIST and FMNIST training sets. For each of these four batch sizes and for each training set, the proposal



(a) Mean acceptance rate per node.



(b) Mean acceptance rate per layer.

Fig. 4 Mean acceptance rates (per node and per layer) across 10 chains realized via minibatch NBG sampling of the MLP(2, 2, 2, 2, 2, 2, 2, 1) parameters. The MLP is fitted to noisy XOR. A batch size of 100 is used

variance per layer is reduced during pre-training until the acceptance rate of the layer is not prohibitively low, and subsequently the proposal variance tuned via pre-training is used for computing the acceptance rate of the corresponding layer from a chain realization. Tables 5 and 6 demonstrate that if proposal variances are reduced in deeper layers, then acceptance rates do not vanish with depth. For increasing batch size, acceptance rates drop across all layers, as expected when shifting from approximate towards exact MCMC.

As part of Table 5, a chain is simulated upon fitting MLP(784, 10, 10, 10, 10) to the MNIST training set via minibatch FNBG sampling with a batch size of 3000. Figure 5, which comprises a grid of $4 \times 2 = 8$ traceplots, is produced from that chain. Each row of Fig. 5 is related to one of the 8180 parameters of MLP(784, 10, 10, 10, 10). More specifically, the first, second, third and fourth row correspond to parameter θ_{1005} in hidden layer 1, parameter θ_{7872} in hidden layer 2, parameter θ_{8008} in hidden layer 3 and parameter θ_{8107} in the output layer. A pair of traceplots per parameter is shown in each row; the right traceplot is more zoomed out than the left one. All traceplots in the right column share a common range of $[-8, 8]$ in their vertical axes.

It is observed that the zoomed-in traceplots (left column of Fig. 5) do not exhibit entrapment in local modes irrespective of network depth, agreeing with the non-vanishing acceptance rates of Table 5. Furthermore, it is seen from the zoomed-out traceplots (right column of Fig. 5) that chain scales decrease in deeper layers. For example, the right traceplot of parameter θ_{8107} (output layer) has non-visible fluctuations under a y-axis range of $[-8, 8]$, whereas the right traceplot of parameter θ_{1005} (first hidden layer) fluctuates more widely under the same y-axis range.

Figure 5 suggests that chains of parameters in shallower layers perform more exploration, while chains of parameters in deeper layers carry out more exploitation. This way, chain scales collapse towards point estimates for increasing network depth.

5.4 Batch size and log-likelihood

For each batch size shown in Fig. 6a, the likelihood function of Eq. (3.3) is evaluated on 10 batch samples, which are drawn from the MNIST training set. A boxplot is then generated from the 10 log-likelihood values and it is displayed in Fig. 6a. The log-likelihood function is normalized by batch size in order to obtain visually comparable boxplots across different batch sizes. In `PyTorch`, the normalized log-likelihood is computed via the `CrossEntropyLoss` class initialized with `reduction='mean'`. In each boxplot, the blue line and yellow point correspond to the median and mean of the 10 associated log-likelihood values. The horizontal gray line represents the log-likelihood value based on the whole MNIST training set. Figure 6b is generated using the FMNIST training set, following an analogous setup.

Figures 6a and 6b demonstrate that log-likelihood values are increasingly volatile for decreasing batch size. Furthermore, the volatility of log-likelihood values vanishes as the batch size gets close to the training sample size. So, Fig. 6 confirms visually that the approximate likelihood tends to the exact likelihood for increasing batch size. Thus, the batch size in FNBG sampling is preferred to be as large as possible, up to the point that (finer) block acceptance rates do not become prohibitively low.

5.5 Depth and predictions

Figure 7 explores how network depth affects predictive accuracy in approximate MCMC. Shallower MLP(2, 2, 1), consisting of one hidden layer, and deeper MLP(2, 2, 2, 2, 2, 2, 2, 1), consisting of six hidden layers, are fitted to the noisy XOR training set using minibatch NBG with a batch size of 100 and a proposal variance of 0.04; $m = 10$ chains are realized for each of the two MLPs. Subsequently, the predictive accuracy per chain is evaluated on the noisy XOR test set. One boxplot is generated for each set of 10 chains, as shown in Fig. 7. Blue lines represent medians.

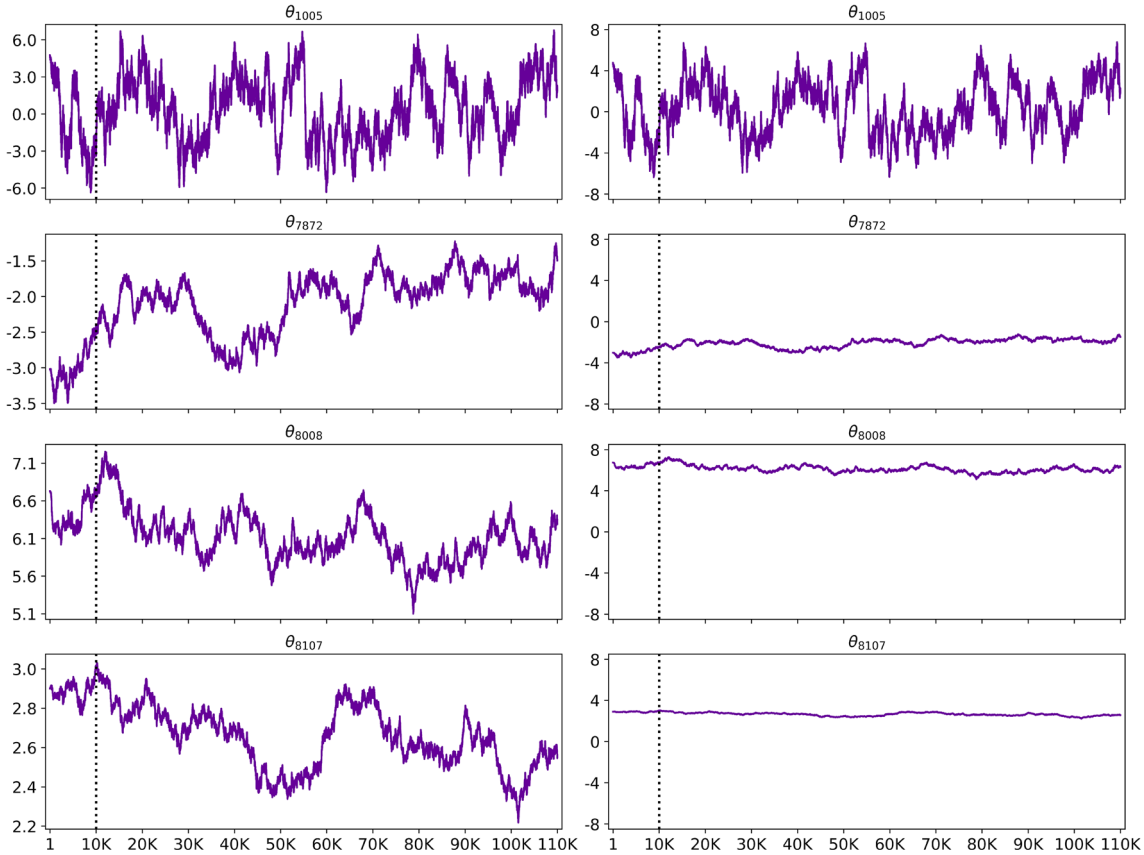


Fig. 5 Markov chain traceplots of four parameter coordinates of MLP(784, 10, 10, 10, 10), which is fitted to MNIST via minibatch FNBG sampling with a batch size of 3000. Each row displays two traceplots of the same chain for a single parameter; the traceplot on the right is more zoomed-out than the one on the left. The traceplots of the right column share a common range on the vertical axes. Vertical dotted lines indicate the end of burnin

The same 10 chains are used to generate relevant plots in Figs. 3b, 4 and 7. In particular, the leftmost boxplot in Fig. 3b and right boxplot in Fig. 7 stem from the same 10 chains and are thus identical. Figure 4 shows mean acceptance rates per node and per layer across the 10 chains that also yield the right boxplot of predictive accuracies in Fig. 7.

MLP(2, 2, 1) and MLP(2, 2, 2, 2, 2, 2, 2, 1) have respective predictive accuracy medians of 86.75% and 98.88% as blue lines indicate in Fig. 7, so predictive accuracy increases with increasing depth. Moreover, the interquartile ranges of Fig. 7 demonstrate that a deeper architecture yields less volatile, and in that sense more stable, predictive accuracy. As an overall empirical observation, increasing the network depth in approximate MCMC seems to produce higher and less volatile predictive accuracy.

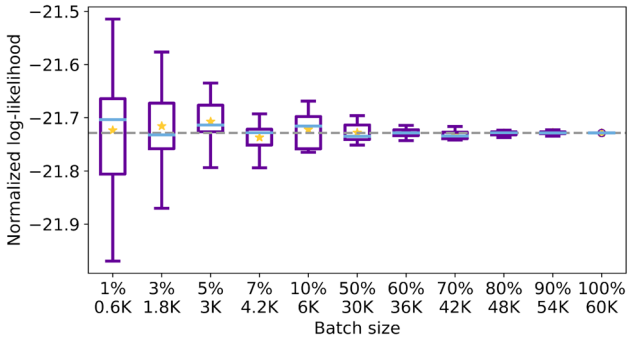
Observation 2 *Increasing the depth of a feedforward neural network increases the predictive accuracy but reduces the acceptance rates for blocks in deeper layers. Reducing the proposal variance in deeper layers helps counter the reduction of acceptance rates. Increasing the network width in initial layers does not have a negative impact on acceptance rates, in contrast to the negative impact of increasing depth on acceptance rates.*

5.6 Batch size and predictions

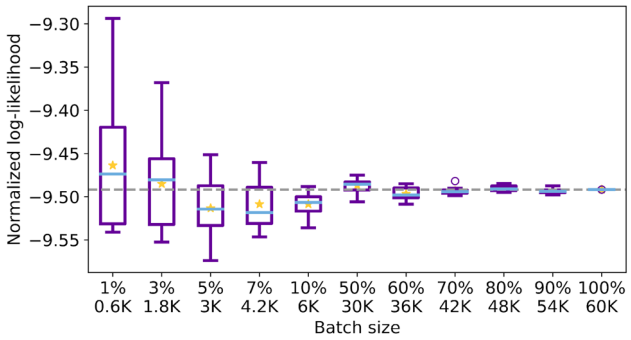
This subsection assesses empirically the effect of batch size on predictive accuracy in approximate MCMC. To this end, MLP(784, 10, 10, 10, 10) is fitted to the MNIST and FMNIST training sets using minibatch FNBG sampling with batch sizes of 600, 1800, 3000 and 4200, which correspond to 1%, 3%, 5% and 7% of each training sample size. One chain is realized per combination of training set and batch size. Table 2 reports the predictive accuracy for each chain.

The same chains are used to compute predictive accuracies in Table 2 as well as acceptance rates in Tables 5 and 6 of "Appendix C". The chain that yields the predictive accuracy for MNIST and for a batch size of 3000 (first row and third column of Table 2) is partly visualized by traceplots in Fig. 5.

According to Table 2, the highest accuracy of 90.75% for MNIST and of 80.89% for FMNIST are attained by employing a batch size of 3000. Overall, predictive accuracy increases as batch size increases. However, predictive



(a) Log-likelihood value boxplots for MNIST.



(b) Log-likelihood value boxplots for FMNIST.

Fig. 6 Boxplots of normalized log-likelihood values for MNIST and FMNIST. Each boxplot summarizes normalized log-likelihood values of 10 batch samples for a given batch size. To normalize, each log-likelihood value is divided by batch size. Blue lines and yellow points correspond to medians and means. Horizontal gray lines represent exact log-likelihood values for batch size equal to training sample size

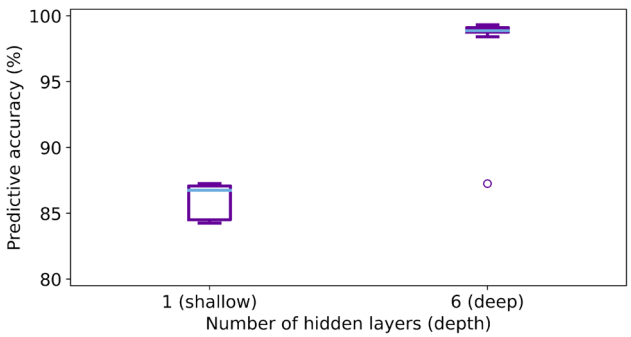


Fig. 7 A comparison between a shallower and a deeper MLP architecture. Each of MLP(2, 2, 1) and MLP(2, 2, 2, 2, 2, 2, 1) is fitted to noisy XOR via minibatch NBG sampling with a batch size of 100. Predictive accuracy boxplots are generated from 10 chains per MLP. Blue lines indicate medians

accuracy decreases when batch size increases from 3000 to 4200; this is explained by the fact that a batch size of 4200 is too large, in the sense that it reduces acceptance rates (see Tables 5 and 6). So, as pointed out in Sect. 5.4, a tuning guideline is to increase the batch size up to the point that no substantial reduction in finer block acceptance rates occurs.

Table 2 Predictive accuracies obtained by fitting MLP(784, 10, 10, 10, 10) to MNIST and to FMNIST via minibatch FNBG sampling with different batch sizes

Dataset	Batch size			
	1% 0.6K	3% 1.8K	5% 3K	7% 4.2K
MNIST	85.99	89.01	90.75	90.43
FMNIST	71.50	80.07	80.89	79.17

An attained predictive accuracy of 90.75% on MNIST demonstrates that non-convergent chains (simulated via minibatch FNBG) learn from data, since data-agnostic guessing based on pure chance has a predictive accuracy of 10%. While stochastic optimization algorithms for deep learning achieve predictive accuracies higher than 90.75% on MNIST, the goal of this work has not been to construct an approximate MCMC algorithm that outperforms stochastic optimization on the predictive front. The main objective has been to demonstrate that approximate MCMC for neural networks learns from data and to uncover associated sampling characteristics, such as diminishing chain ranges (Fig. 5) and diminishing acceptance rates (Tables 5 and 6) for increasing network depth. Similar predictive accuracies in the vicinity of 90% using Hamiltonian Monte Carlo for deep learning have been reported in the literature (Wenzel et al. 2020; Izmailov et al. 2021). Nonetheless, this body of relevant work relies on chain lengths one or two orders of magnitude shorter; for instance, Izmailov et al. (2021) have run up to 900 iterations per chain realization. The present paper proposes to circumvent vanishing acceptance rates by grouping neural network parameters into smaller blocks, thus enabling the generation of lengthier chains.

Observation 3 *Increasing the batch size in minibatch MCMC sampling of feedforward neural network parameters increases the predictive accuracy. This observation is anticipated, in the sense that minibatch MCMC becomes exact MCMC when the batch size is equal to the training sample size. However, the batch size can be increased up to the point that no substantial reduction in acceptance rates occurs.*

5.7 Chain length and predictions

It is reminded that 110000 iterations are run per chain in the experiments herein, of which the first 10000 are discarded as burnin. The last $v = 10000$ (out of the remaining 100000) iterations are used for making predictions via Bayesian marginalization based on Eq. (3.6). Only 10000 iterations are utilized in Eq. (3.6) to cap the computational cost for predictions.

There exists a tractable solution to Bayesian marginalization, since the approximate posterior predictive pmf of

Table 3 Predictive accuracies obtained from different chain lengths

Dataset	Chain length			
	1K	10K	20K	30K
MNIST	88.31	90.75	91.12	91.20
FMNIST	78.93	80.89	81.36	81.53

MLP(784, 10, 10, 10, 10) is fitted to MNIST and to FMNIST via minibatch FNBG sampling with a batch size of 3000. One chain is realized per dataset. Subsequently, predictions are made via Bayesian marginalization using chunks of different length from the end of the realized chains

Eq. (3.6) can be computed in parallel both in terms of Monte Carlo iterations and of test points. The implementation of such a parallel solution is deferred to future work.

In the meantime, it is examined here how chain length affects predictive accuracy. Along these lines, predictive accuracies are computed from the last 1000, 10000, 20000 and 30000 iterations of the chain realized via minibatch FNBG with a batch size of 3000 for each of MNIST and FMNIST (see Table 3). The last 10000 and all 100000 post-burnin iterations of the same chain generate predictive accuracies in Table 2 and acceptance rates in Tables 5 and 6, respectively.

Table 3 demonstrates that predictive accuracy increases (both for MNIST and FMNIST) as chain length increases. So, as a chain traverses the parameter space of a neural network, information of predictive importance accrues despite the lack of convergence. It can also be seen from Table 3 that the rate of improvement in predictive accuracy slows down for increasing chain length.

Observation 4 *Despite the lack of convergence and the slow mixing, increasing the number of approximate MCMC iterations upon sampling from the parameter space of a feedforward neural network increases the predictive accuracy. The rate of improvement in predictive accuracy slows down for increasing chain length.*

5.8 Augmentation and predictions

To assess the effect of data augmentation on predictive accuracy, three image transformations are performed on the MNIST and FMNIST training sets, namely rotations by angle, blurring, and colour inversions. Images are rotated by angles randomly selected between -30 and 30 degrees. Each image is blurred with probability 0.9. Blur is randomly generated from a Gaussian kernel of size 9×9. The standard deviation of the kernel is randomly selected between 1 and 1.5. Each image is colour-inverted with probability 0.5. Figure 10 in "Appendix D" displays examples of MNIST and FMNIST training images that have been rotated, blurred or colour-inverted according to the described transformations.

Table 4 Predictive accuracies obtained from different data augmentation schemes

Dataset	Transform			
	None	Rotation	Blur	Inversion
MNIST	90.75	86.19	85.66	36.87
FMNIST	80.89	6.62	7.46	8.61

MLP(784, 10, 10, 10, 10) is fitted to each of the augmented MNIST and FMNIST training sets via minibatch FNBG sampling with a batch size of 3000. Predictive accuracies are computed on the corresponding non-augmented test sets. The first column reports predictive accuracies based on the non-augmented MNIST and FMNIST training sets

Each of the three transformations is applied to the whole MNIST and FMNIST training sets. Subsequently, MLP(784, 10, 10, 10, 10) is fitted to each transformed training set via minibatch FNBG with a batch size of 3000 and with proposal variances specified in Tables 5 and 6. One chain is simulated per transformed training set. Predictive accuracies are computed on the corresponding untransformed MNIST and FMNIST test sets and are reported in Table 4. Moreover, predictive accuracies based on the untransformed MNIST and FMNIST training sets are available in the first column of Table 4, as previously reported in Table 2.

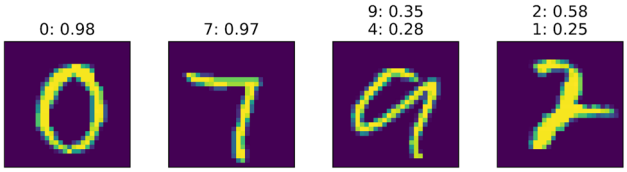
According to Table 4, if data augmentation is performed, then predictive accuracy deteriorates drastically. Notably, data augmentation has catastrophic predictive consequences for FMNIST. These empirical findings agree with the 'dirty likelihood hypothesis' of Wenzel et al. (2020), according to which data augmentation violates the likelihood principle.

Observation 5 *Approximate MCMC sampling of feedforward neural network parameters in the presence of augmented data remains an open problem. Data augmentation violates the likelihood principle and consequently reduces drastically the predictive accuracy.*

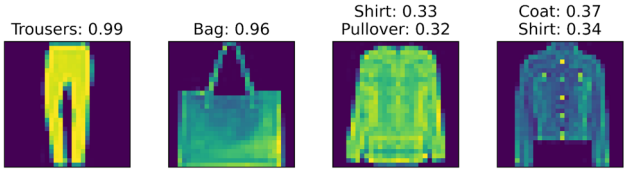
5.9 Uncertainty quantification

Approximate MCMC enables predictive uncertainty quantification (UQ) via Bayesian marginalization. Such a principled approach to UQ constitutes an advantage of approximate MCMC over stochastic optimization in deep learning. This subsection showcases how predictive uncertainty is quantified for neural networks via minibatch FNBG sampling.

Recall that one chain has been simulated for each of MNIST and FMNIST to compute the predictive accuracies of column 3 in Table 2 (see Sect. 5.6). Those chains are used to estimate posterior predictive probabilities for some images in the corresponding test sets, as shown in Fig. 8. All test images in Fig. 8 have been correctly classified via Bayesian marginalization.



(a) Posterior predictive probabilities for MNIST.



(b) Posterior predictive probabilities for FMNIST.

Fig. 8 Demonstration of UQ for some correctly classified MNIST and FMNIST test images. The highest posterior predictive probability is displayed for each image associated with low uncertainty, whereas the two highest posterior predictive probabilities are displayed for each image associated with high uncertainty

The first and second MNIST test images in Fig. 8a show numbers 0 and 7, with corresponding posterior predictive probabilities 0.98 and 0.97 that indicate near-certainty about the classification outcomes. The third MNIST test image in Fig. 8a shows number 9. Attempting to classify this image by eye casts doubt as to whether the number in the image is 9 or 4. While Bayesian marginalization correctly classifies the number as 9, the posterior predictive probability $\hat{p}(y = 9|x, \mathcal{D}_{1:s}) = 0.35$ is relatively low, indicating uncertainty in the prediction. Moreover, the second highest posterior predictive probability $\hat{p}(y = 4|x, \mathcal{D}_{1:s}) = 0.28$ identifies number 4 as a probable alternative, in agreement with human perception. All in all, posterior predictive probabilities and human understanding are aligned in terms of perceived predictive uncertainties and in terms of plausible classification outcomes. Image 4 is aligned with image 3 of Fig. 8a regarding UQ conclusions.

Figure 8b, which entails FMNIST test images, is analogous to Fig. 8a from a UQ point of view. In Fig. 8b, FMNIST test images 1 and 2 show trousers and a bag, with corresponding posterior predictive probabilities 0.99 and 0.96 that indicate near-certainty about the classification outcomes. The third FMNIST test image of Fig. 8b shows a shirt. It is not visually clear whether this image depicts a shirt or a pullover. While Bayesian marginalization correctly identifies the object as a shirt, the posterior predictive probabilities $\hat{p}(y = \text{shirt}|x, \mathcal{D}_{1:s}) = 0.33$ and $\hat{p}(y = \text{pullover}|x, \mathcal{D}_{1:s}) = 0.32$ capture human uncertainty and identify the two most plausible classification outcomes. Image 4 is analogous to image 3 of Fig. 8b in terms of UQ conclusions.

Observation 6 *A non-convergent chain realization via approximate MCMC sampling of feedforward neural network parameters can help with the assessment of predictive uncertainty meaningfully, that is in agreement with human insights.*

6 Future work

Several future research directions emerge from this paper; three software engineering extensions are planned, three methodological developments are proposed, and one theoretical question is posed.

To start with possible software engineering work, Bayesian marginalization can be parallelized across test points and across FNBG iterations per test point. Additionally, an adaptive version of FNBG sampling can be implemented based on existing Gibbs sampling methods for proposal variance tuning (Andrieu and Thoms 2008), thus automating tuning and reducing tuning computational requirements. Moreover, FNBG sampling can be implemented with a subsampling mechanism that sets the batch size adaptively (Bardenet et al. 2014).

In terms of methodological developments, alternative ways of grouping parameters in FNBG sampling may be considered. For example, parameters may be grouped according to their covariance structure, as estimated from pilot FNBG runs. Furthermore, functional priors proposed by Tran et al. (2022) or adaptations of them may be utilized in conjunction with FNBG. Moreover, FNBG sampling may be developed for neural network architectures other than MLPs. To this end, DAG representations of other neural network architectures will be devised and fine parameter blocks will be identified from the DAGs.

A theoretical question of interest is how to construct lower bounds of predictive accuracy for minibatch FNBG (and for minibatch MCMC more generally) as a function of the distance between the exact and approximate parameter posterior density. It has been observed empirically that minibatch FNBG has predictive capacity, yet theoretical guarantees for predictive accuracy have not been established.

The proposed sampling approach and future developments face two main limitations. Firstly, it remains an open question how to sample neural network parameters given augmented training data, as previously pointed out by the 'dirty likelihood hypothesis' of Wenzel et al. (2020). Secondly, as the depth of a feedforward neural network increases, the proposal variance of FNBG is reduced for deeper layers. Thus, the proposal variance for deeper layers may be set to a value too close to zero from a practical point of view.

Software and data

The FNBG sampler for MLPs has been implemented under the `eeyore` package using `Python` and `PyTorch`. `eeyore` is available at https://github.com/papamarkou/eeyore. Source code for the examples of Sect. 5 can be found in `dmcl_examples`, forming a separate `Python` package based on `eeyore`. `dmcl_examples` can be downloaded from https://github.com/papamarkou/dmcl_examples.

Acknowledgements The author would like to acknowledge the assistance given by Research IT and the use of the Computational Shared Facility at The University of Manchester. This work used the Cirrus UK National Tier-2 HPC Service at EPCC (http://www.cirrus.ac.uk) funded by the University of Edinburgh and EPSRC (EP/P020267/1). The author would like to thank Google for the provision of free credit on Google Cloud Platform. This work was presented at two seminars supported by a travel grant from the Dame Kathleen Ollerenshaw Trust, which is gratefully acknowledged. The author would like to dedicate this paper to the memory of his mother, who died as this paper was being developed.

Author Contributions T.P. conceived and implemented this research work, as well as wrote and reviewed the manuscript.

Declarations

Conflict of interest The authors declare no competing interests.

Appendix A: Blocked Gibbs

Algorithm 2 summarizes blocked Gibbs sampling in the context of sampling MLP parameters, as set out in Sect. 3.2. Remarks 1 and 2 provide expressions for the acceptance probability of candidate state $\theta^{\star}_{z(q)}$ of Algorithm 1 (MWBG sampling), as stated in Eq. (4.1) of Sect. 4.1.

Remark 1 Consider an MLP($\kappa_{0:\rho}$) with likelihood function $\mathcal{L}(y_{1:s}|x_{1:s},\theta)$ specified by Eq. (3.3), where $\{(x_i, y_i) : i = 1, 2, \ldots, s\}$ is a training dataset related to a supervised classification problem and θ are the MLP parameters. Let $\pi(\theta) = \prod_{q=1}^{m} \pi(\theta_{z(q)})$ be a parameter prior density based on a partition $\{\theta_{z(1)}, \theta_{z(2)}, \ldots, \theta_{z(m)}\}$ of θ. A MWBG version of Algorithm 2 is used for sampling from the target density $p(\theta|x_{1:s}, y_{1:s})$. At iteration t, a candidate state $\theta^{\star}_{z(q)}$ for parameter block $\theta_{z(q)}$ is drawn from the isotropic normal proposal density $\mathcal{N}(\theta^{(t-1)}_{z(q)}, \sigma_q^2 I_q)$. The acceptance probability $a(\theta^{\star}_{z(q)}, \theta^{(t-1)}_{z(q)})$ of $\theta^{\star}_{z(q)}$ is given by

Algorithm 2 Blocked Gibbs sampling

Input: training dataset $\mathcal{D}_{1:s}$
Input: initial state $\theta^{(0)}_{z(1):z(m)}$
Input: number of sampling iterations v

for $t = 1, \ldots, v$ **do**
  Draw $\theta^{(t)}_{z(1)} \sim p(\theta_{z(1)}|\theta^{(t-1)}_{z(2):z(m)}, \mathcal{D}_{1:s})$
  Draw $\theta^{(t)}_{z(2)} \sim p(\theta_{z(2)}|\theta^{(t)}_{z(1)}, \theta^{(t-1)}_{z(3):z(m)}, \mathcal{D}_{1:s})$
  $\vdots$
  Draw $\theta^{(t)}_{z(q)} \sim p(\theta_{z(q)}|\theta^{(t)}_{z(1):z(q-1)}, \theta^{(t-1)}_{z(q+1):z(m)}, \mathcal{D}_{1:s})$
  $\vdots$
  Draw $\theta^{(t)}_{z(m)} \sim p(\theta_{z(m)}|\theta^{(t)}_{z(1):z(m-1)}, \mathcal{D}_{1:s})$
end for

$$a(\theta^{\star}_{z(q)}, \theta^{(t-1)}_{z(q)}) = \min\left\{\frac{\mathcal{L}(y_{1:s}|x_{1:s},\theta^{\star})\pi(\theta^{\star}_{z(q)})}{\mathcal{L}(y_{1:s}|x_{1:s},\theta^{(t-1)})\pi(\theta^{(t-1)}_{z(q)})}, 1\right\}, \tag{A1}$$

where $\theta^{(t-1)}$ and $\theta^{\star}$ denote the values of θ obtained by inverting the permutations $(\theta^{(t)}_{z(1):z(q-1)}, \theta^{(t-1)}_{z(q):z(m)})$ and $(\theta^{(t)}_{z(1):z(q-1)}, \theta^{\star}_{z(q)}, \theta^{(t-1)}_{z(q+1):z(m)})$, respectively.

Remark 2 Consider an MLP($\kappa_{0:\rho}$) with cross-entropy loss function $\mathcal{E}(\theta, \mathcal{D}_{1:s})$, where $\mathcal{D}_{1:s} = \{(x_i, y_i) : i = 1, 2, \ldots, s\}$ is a training dataset related to a supervised classification problem and θ are the MLP parameters. It is assumed that $\mathcal{E}$ is unnormalized, which means that it is not scaled by batch size. Under the sampling setup of Remark 1, the acceptance probability of $\theta^{\star}_{z(q)}$, expressed in terms of cross-entropy loss function $\mathcal{E}$, is given by

$$a(\theta^{\star}_{z(q)}, \theta^{(t-1)}_{z(q)}) = \min\left\{\frac{\pi(\theta^{\star}_{z(q)})\exp\left(\mathcal{E}(\theta^{(t-1)}, \mathcal{D}_{1:s})\right)}{\pi(\theta^{(t-1)}_{z(q)})\exp\left(\mathcal{E}(\theta^{\star}, \mathcal{D}_{1:s})\right)}.1\right\}. \tag{A2}$$

The relation between the cross-entropy loss function

$$\mathcal{E}(\theta, \mathcal{D}_{1:s}) = -\sum_{i=1}^{s}\sum_{k=1}^{\kappa_\rho} \mathbb{1}_{\{y_i=k\}} \log\left(h_{\rho,k}(x_i,\theta)\right) \tag{A3}$$

and the likelihood function of Eq. (3.3) is given by

$$\mathcal{L}(y_{1:s}|x_{1:s},\theta) = \exp\left(-\mathcal{E}(\theta,\mathcal{D}_{1:s})\right). \tag{A4}$$

Combining Eqs. (A1) and (A4) yields Eq. (A2).

Remark 1 states the acceptance probability in statistical terms using the likelihood function, whereas Remark 2 states it in deep learning terms using the cross-entropy loss function. Remark 2 is practical in the sense that deep learning software frameworks, being geared towards optimization, provide implementations of cross-entropy loss. For example, the unnormalized cross-entropy loss $\mathcal{E}$, as stated in Eq. (A3), can be computed in `PyTorch` via the `CrossEntropyLoss` class initialized with `reduction='sum'`.

Appendix B: Noisy XOR

Figure 9 shows the noisy XOR training and test datasets used in Sect. 5. Information about how these noisy XOR datasets have been simulated is available in Papamarkou et al. (2022).

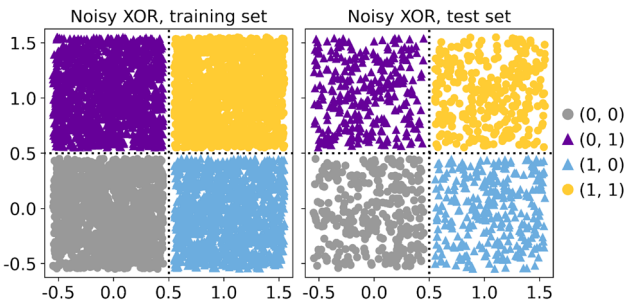


Fig. 9 Noisy XOR training set (left) and test set (right) consisting of 5000 and 1200 data points, respectively

Appendix C: Tuning

Tables 5 and 6 show that acceptance rates obtained from minibatch FNBG sampling can be retained at non-vanishing levels in deeper layers by reducing the proposal variances corresponding to these layers. MLP(784, 10, 10, 10, 10) is fitted to MNIST and to FMNIST via minibatch FNBG sampling with different batch sizes. The acceptance rate per layer is computed from one chain for each batch size. Tables 5 and 6 report the obtained acceptance rates for MNIST and for FMNIST, respectively.

Table 5 Acceptance rate per layer obtained by fitting MLP(784, 10, 10, 10, 10) to MNIST via minibatch FNBG sampling with different batch sizes

Layer		σ	Rate
Batch size = 600 (1%)			
Hidden	1st	$5 \cdot 10^{-2}$	45.56
	2nd	$5 \cdot 10^{-4}$	26.43
	3rd	$5 \cdot 10^{-4}$	26.28
Output		$5 \cdot 10^{-5}$	29.18
Batch size = 1800 (3%)			
Hidden	1st	$2 \cdot 10^{-2}$	41.41
	2nd	$2 \cdot 10^{-4}$	30.68
	3rd	$2 \cdot 10^{-4}$	31.92
Output		$2 \cdot 10^{-5}$	35.66
Batch size = 3000 (5%)			
Hidden	1st	10^{-2}	54.95
	2nd	10^{-4}	45.73
	3rd	10^{-4}	44.98
Output		10^{-5}	51.54
Batch size = 4200 (7%)			
Hidden	1st	10^{-2}	31.68
	2nd	10^{-4}	20.17
	3rd	10^{-4}	19.76
Output		10^{-5}	22.22

σ denotes the proposal standard deviation

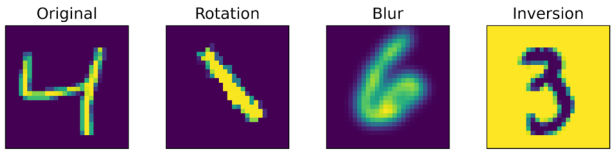
Appendix D: Augmentation

Figure 10 shows examples of images from the MNIST and FMNIST training sets transformed by rotation, blurring and colour inversion. These transformations are used in Sect. 5.8 to assess the effect of data augmentation on predictive accuracy. Details about the performed transformations are available in Sect. 5.8.

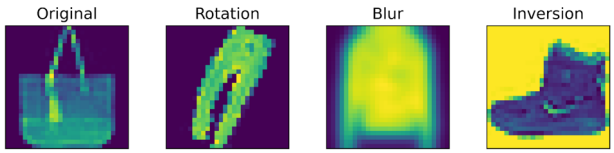
Table 6 Acceptance rate per layer obtained by fitting MLP(784, 10, 10, 10, 10) to FMNIST via minibatch FNBG sampling with different batch sizes

Layer		σ	Rate
Batch size = 600 (1%)			
Hidden	1st	$5 \cdot 10^{-2}$	47.86
	2nd	$5 \cdot 10^{-4}$	34.61
	3rd	$5 \cdot 10^{-4}$	32.99
Output		$5 \cdot 10^{-5}$	37.73
Batch size = 1800 (3%)			
Hidden	1st	$2 \cdot 10^{-2}$	60.34
	2nd	$2 \cdot 10^{-4}$	46.78
	3rd	$2 \cdot 10^{-4}$	45.91
Output		$2 \cdot 10^{-5}$	52.07
Batch size = 3000 (5%)			
Hidden	1st	10^{-2}	66.94
	2nd	10^{-4}	57.40
	3rd	10^{-4}	58.48
Output		10^{-5}	64.64
Batch size = 4200 (7%)			
Hidden	1st	10^{-2}	55.28
	2nd	10^{-4}	47.10
	3rd	10^{-4}	47.19
Output		10^{-5}	52.75

σ denotes the proposal standard deviation



(a) Examples of transformed MNIST training images.



(b) Examples of transformed FMNIST training images.

Fig. 10 Examples of MNIST and of FMNIST training images transformed by rotation, blurring and colour inversion. Such transformations are deployed in the data augmentation experiments of Sect. 5.8. Examples of untransformed MNIST and FMNIST training images are also displayed

References

Alexos, A., Boyd, A.J., Mandt, S.: Structured stochastic gradient MCMC. In: Proceedings of the 39th International Conference on Machine Learning, vol. 162, pp. 414–434. PMLR, Baltimore (2022)

Andrieu, C., de Freitas, J.F.G., Doucet, A.: Sequential Bayesian Estimation and Model Selection Applied to Neural Networks, Cambridge (1999)

Andrieu, C., de Freitas, N., Doucet, A.: Reversible jump MCMC simulated annealing for neural networks. In: Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence, pp. 11–18 (2000)

Andrieu, C., Thoms, J.: A tutorial on adaptive MCMC. Stat. Comput. **18**(4), 343–373 (2008)

Bardenet, R., Doucet, A., Holmes, C.: Towards scaling up Markov chain Monte Carlo: an adaptive subsampling approach. In: Proceedings of the 31st International Conference on Machine Learning, vol. 32, pp. 405–413. PMLR (2014)

Bouchard-Côté, A., Doucet, A., Roth, A.: Particle Gibbs split-merge sampling for Bayesian inference in mixture models. J. Mach. Learn. Res. **18**(28), 1–39 (2017)

Chen, T., Fox, E., Guestrin, C.: Stochastic gradient Hamiltonian Monte Carlo. In: Proceedings of the 31st International Conference on Machine Learning, vol. 32, pp. 1683–1691. PMLR (2014)

de Freitas, N.: Bayesian methods for neural networks. PhD thesis, University of Cambridge (1999)

de Freitas, N., Andrieu, C., Højen-Sørensen, P., Niranjan, M., Gee, A.: Sequential Monte Carlo methods for neural networks, pp. 359–379. Springer, New York (2001)

Girolami, M., Calderhead, B.: Riemann manifold Langevin and Hamiltonian Monte Carlo methods. J. Roy. Stat. Soc. Ser. B (Stat. Methodol.) **73**(2), 123–214 (2011)

Gong, W., Li, Y., Hernández-Lobato, J.M.: Meta-learning for stochastic gradient MCMC. In: International Conference on Learning Representations. PMLR (2019)

Grathwohl, W., Swersky, K., Hashemi, M., Duvenaud, D., Maddison, C.: Oops i took a gradient: scalable sampling for discrete distributions. In: Proceedings of the 38th International Conference on Machine Learning, vol. 139, pp. 3831–3841. PMLR (2021)

Hastie, T., Tibshirani, R., Friedman, J.: The Elements of Statistical Learning: Data Mining, Inference and Prediction, 2nd edn. Springer, New York (2016)

He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778 (2016)

Izmailov, P., Vikram, S., Hoffman, M.D., Wilson, A.G.G.: What are Bayesian neural network posteriors really like? In: Proceedings of the 38th International Conference on Machine Learning, vol. 139, pp. 4629–4640. PMLR, Vienna (2021)

Krizhevsky, A., Hinton, G.: Learning multiple layers of features from tiny images. Technical report. University of Toronto, Toronto (2009)

Łatuszyński, K., Roberts, G.O., Rosenthal, J.S.: Adaptive Gibbs samplers and related MCMC methods. Ann. Appl. Probab. **23**(1), 66–98 (2013)

Lecun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proc. IEEE **86**(11), 2278–2324 (1998)

Matsubara, T., Oates, C.J., Briol, F.-X.: The ridgelet prior: a covariance function approach to prior specification for Bayesian neural networks. J. Mach. Learn. Res. **22**(157), 1–57 (2021)

Minsky, M.L., Papert, S.A.: Perceptrons. MIT Press, Cambridge (1988)

Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., Ng, A.Y.: Reading digits in natural images with unsupervised feature learning.

In: NIPS Workshop on Deep Learning and Unsupervised Feature Learning (2011)
Papamarkou, T., Hinkle, J., Young, M.T., Womble, D.: Challenges in Markov chain Monte Carlo for Bayesian neural networks. Stat. Sci. **37**(3), 425–442 (2022)
Roberts, G.O., Sahu, S.K.: Updating schemes, correlation structure, blocking and parameterization for the Gibbs sampler. J. Roy. Stat. Soc. Ser. B (Stat. Methodol.) **59**(2), 291–317 (1997)
Rosenblatt, F.: The perceptron: a probabilistic model for information storage and organization in the brain. Psychol. Rev. **65**(6), 386 (1958)
Saul, L., Jordan, M.: Exploiting tractable substructures in intractable networks. In: Advances in Neural Information Processing Systems, vol. 8. MIT Press, Denver (1995)
Titterington, D.M.: Bayesian methods for neural networks and related models. Stat. Sci. **19**(1), 128–139 (2004)
Tran, B.-H., Rossi, S., Milios, D., Filippone, M.: All you need is a good functional prior for Bayesian deep learning. J. Mach. Learn. Res. **23**(74), 1–56 (2022)
Vono, M., Dobigeon, N., Chainais, P.: Split-and-augmented Gibbs sampler-application to large-scale inference problems. IEEE Trans. Signal Process. **67**(6), 1648–1661 (2019)
Vono, M., Paulin, D., Doucet, A.: Efficient MCMC sampling with dimension-free convergence rate using ADMM-type splitting. J. Mach. Learn. Res. **23**(25), 1–69 (2022)
Welling, M., Teh, Y.W.: Bayesian learning via stochastic gradient Langevin dynamics. In: Proceedings of the 28th International Conference on Machine Learning, pp. 681–688 (2011)
Wenzel, F., Roth, K., Veeling, B., Swiatkowski, J., Tran, L., Mandt, S., Snoek, J., Salimans, T., Jenatton, R., Nowozin, S.: How good is the Bayes posterior in deep neural networks really? In: Proceedings of the 37th International Conference on Machine Learning, vol. 119, pp. 10248–10259. PMLR, Vienna (2020)
Wiese, J.G., Wimmer, L., Papamarkou, T., Bischl, B., Günnemann, S., Rügamer, D.: Towards efficient MCMC sampling in Bayesian neural networks by exploiting symmetry. In: European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases. Springer, Turin (2023)
Xiao, H., Rasul, K., Vollgraf, R.: Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms. arXiv preprint arXiv:1708.07747 (2017)
Zhang, R., Li, C., Zhang, J., Chen, C., Wilson, A.G.: Cyclical stochastic gradient MCMC for Bayesian deep learning. In: International Conference on Learning Representations (2020)