Engineering Applications of Artificial Intelligence 94 (2020) 103700

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

journal homepage: www.elsevier.com/locate/engappai

# Surrogate-assisted parallel tempering for Bayesian neural learning

Rohitash Chandra [a,b,*], Konark Jain [d,b], Arpit Kapoor [b,c], Ashray Aman [b,e]

[a] School of Mathematics and Statistics, University of New South Wales, Sydney, NSW 2052, Australia
[b] Centre for Translational Data Science, The University of Sydney, Sydney, NSW 2006, Australia
[c] Department of Computer Science and Engineering, SRM Institute of Science and Technology, Chennai, Tamil Nadu, India
[d] Department of Electronics and Electrical Engineering, Indian Institute of Technology Guwahati, Assam, India
[e] Department of Mathematics, Indian Institute of Technology Delhi, Delhi, India

ARTICLE INFO

*Keywords:*
Bayesian neural networks
Parallel tempering
MCMC
Surrogate-assisted optimization
Parallel computing

ABSTRACT

Due to the need for robust uncertainty quantification, Bayesian neural learning has gained attention in the era of deep learning and big data. Markov Chain Monte-Carlo (MCMC) methods typically implement Bayesian inference which faces several challenges given a large number of parameters, complex and multimodal posterior distributions, and computational complexity of large neural network models. Parallel tempering MCMC addresses some of these limitations given that they can sample multimodal posterior distributions and utilize high-performance computing. However, certain challenges remain given large neural network models and big data. Surrogate-assisted optimization features the estimation of an objective function for models which are computationally expensive. In this paper, we address the inefficiency of parallel tempering MCMC for large-scale problems by combining parallel computing features with surrogate assisted likelihood estimation that describes the plausibility of a model parameter value, given specific observed data. Hence, we present surrogate-assisted parallel tempering for Bayesian neural learning for simple to computationally expensive models. Our results demonstrate that the methodology significantly lowers the computational cost while maintaining quality in decision making with Bayesian neural networks. The method has applications for a Bayesian inversion and uncertainty quantification for a broad range of numerical models.

## 1. Introduction

Although neural networks have gained significant attention due to the deep learning revolution (Schmidhuber, 2015), several limitations exist. The challenge widens for uncertainty quantification in decision making given the development of new neural network architectures and learning algorithms. Bayesian neural learning provides a probabilistic viewpoint with the representation of neural network weights as probability distributions (MacKay, 1995; Robert, 2014). Rather than single-point estimates by gradient-based learning methods, the probability distributions naturally account for uncertainty in parameter estimates.

Through Bayesian neural learning, uncertainty regarding the data and model can be propagated into the decision making process. Markov Chain Monte Carlo (MCMC) sampling methods implement Bayesian inference (Hastings, 1970; Metropolis et al., 1953; Tarantola et al., 1982; Mosegaard and Tarantola, 1995) by constructing a Markov chain, such that the desired distribution becomes the equilibrium distribution after a given number of steps (Raftery and Lewis, 1996; van Ravenzwaaij et al., 2016). MCMC methods provide numerical approximations of multi-dimensional integrals (Banerjee et al., 2014). MCMC methods have not gained as much attention in neural learning when compared to gradient-based counterparts since convergence becomes computationally expensive for a large number of model parameters and multimodal posteriors (Robert et al., 2018). MCMC methods typically require thousands of samples to be drawn depending on the model which becomes a major limitation in applications such as deep learning (Schmidhuber, 2015; Gal and Ghahramani, 2016; Kendall and Gal, 2017). Hence, other methods for implementing Bayesian inference exist, such as variational inference (Blei et al., 2017; Damianou et al., 2016) which has been used for deep learning (Blundell et al., 2015). Variational inference has the advantage of faster convergence when compared to MCMC methods for large models. However, in computationally expensive models, variational inference methods would have a similar problem as MCMC, since both would need to evaluate model samples for the likelihood; hence, both would benefit from surrogate-based models.

Parallel tempering is an MCMC method that (Swendsen and Wang, 1987; Marinari and Parisi, 1992; Geyer and Thompson, 1995) features

* Corresponding author.
*E-mail addresses:* rohitash.chandra@unsw.edu.au (R. Chandra), konark145@gmail.com (K. Jain), kapoor.arpit97@gmail.com (A. Kapoor), ashray17aman@gmail.com (A. Aman).
*URL:* https://research.unsw.edu.au/people/dr-rohitash-chandra (R. Chandra).

https://doi.org/10.1016/j.engappai.2020.103700
Received 29 September 2019; Received in revised form 29 April 2020; Accepted 7 May 2020
Available online xxxx
0952-1976/

multiple replicas to provide global and local exploration which makes them suitable for irregular and multi-modal distributions (Patriksson and van der Spoel, 2008; Hukushima and Nemoto, 1996). During sampling, parallel tempering features the exchange of neighbouring replicas to feature exploration and exploitation in the search space. The replicas with higher temperature values ensures that there is enough exploration, while replicas with lower temperature values exploit the promising areas found during the exploration. In contrast to canonical MCMC sampling methods, we can more easily implement parallel tempering in a multi-core or parallel computing architecture (Lamport, 1986). In the case of neural networks, parallel tempering was used for inference of restricted Boltzmann machines (RBMs) (Salakhutdinov et al., 2007; Fischer and Igel, 2015) where it was shown that (Desjardins et al., 2010a) parallel tempering is more effective than Gibbs sampling by itself. Parallel tempering for RBMs has been improved by featuring the efficient exchange of information among the replicas (Brakel et al., 2012). These studies motivated the use of parallel tempering in Bayesian neural learning for pattern classification and time series prediction (Chandra et al., 2019a).

Surrogate assisted optimization (Hicks and Henne, 1978; Jin, 2011) considers the use of machine learning methods such as Gaussian process and neural network models to estimate the objective function during optimization. This is handy when the evaluation of the objective function is too time-consuming. In the past, metaheuristic and evolutionary optimization methods have been used in surrogate assisted optimization (Ong et al., 2003; Zhou et al., 2007). Surrogate assisted optimization has been useful for the fields of engine and aerospace design to replicate computationally expensive models (Ong et al., 2005; Jeong et al., 2005; Samad et al., 2008; Hicks and Henne, 1978). The optimization literature motivates improving parallel tempering using a low-cost replica of the actual model via a surrogate to lower the computational costs.

In the case of conventional Bayesian neural learning, much of the literature concentrated on smaller problems such as datasets and network architecture (Richard and Lippmann, 1991; MacKay, 1996, 1995; Robert, 2014) due to computational efficiency of MCMC methods. Therefore, parallel computing has been used in the implementation of parallel tempering for Bayesian neural learning (Chandra et al., 2019a), where the computational time was significantly decreased due to parallelization. Besides, the method achieved better prediction accuracy and convergence due to the exploration features of parallel tempering. We believe that this can be further improved through incorporating notions from surrogate assisted optimization in parallel tempering, where the likelihood function at certain times is estimated rather than evaluated in a high-performance computing environment.

We note that some work has been done using surrogate assisted Bayesian inference. Zeng et al. (Wang et al., 2016) presented a method for material identification using surrogate assisted Bayesian inference for estimating the parameters of advanced high strength steel used in vehicles. Ray and Myer (2019) used Gaussian process-based surrogate models with MCMC for geophysical inversion problems. The benefits of surrogate assisted methods for computationally expensive optimization problems motivate parallel tempering for computationally expensive models. To our knowledge, there is no work on parallel tempering with surrogate-models implemented via parallel computing for machine learning problems. In the case of parallel tempering that uses parallel computing, the challenge would be in developing a paradigm where different replicas can communicate efficiently. Besides, the task of training the surrogate model from data across multiple replicas in parallel poses further challenges.

In this paper, we present surrogate-assisted parallel tempering for Bayesian neural learning where a surrogate is used to estimate the likelihood rather than evaluating the actual model that feature a large number of parameters and datasets. We present a framework that seamlessly incorporates the decision making by a master surrogate for parallel processing cores that execute the respective replicas of parallel tempering MCMC. Although the framework is intended for general computationally expensive models, we demonstrate its effectiveness using a neural network model for classification problems. The major contribution of this paper is to address the limitations of parallel tempering given computationally expensive models.

The rest of the paper is organized as follows. Section 2 provides background and related work, while Section 3 presents the proposed methodology. Section 4 presents experiments and results and Section 5 concludes the paper with discussion for future research.

## 2. Related work

### 2.1. Bayesian neural learning

In Bayesian inference, we update the probability for a hypothesis as more evidence or information becomes available (Freedman, 1963). We estimate the posterior distribution by sampling using prior distribution and a 'likelihood function' that evaluates the model with observed data. A probabilistic perspective treats learning as equivalent to maximum likelihood estimation (MLE) (White, 1982). Given that the neural network is the model, we base the prior distribution on belief or expert opinion without observing the evidence or training data (Richard and Lippmann, 1991). An example of information or belief for the prior in the case of neural networks is the concept of *weight decay* that states that smaller weights are better for generalization (Krogh and Hertz, 1992; MacKay, 1995; Neal, 2012; Auld et al., 2007).

Due to limitations in MCMC sampling methods, progress in development and applications of Bayesian neural learning has been slow, especially when considering larger neural network architectures, big data and deep learning. Several techniques have been applied to improve MCMC sampling methods by incorporating approaches from the optimization literature. Neal et al. (2011) presented Hamiltonian dynamics that involve using gradient information for constructing efficient MCMC proposals during sampling. Gradient-based learning using Langevin dynamics refer to use of gradient information with Gaussian noise (Welling and Teh, 2011); Chandra et al. (2017) employed Langevin dynamics for Bayesian neural networks for time series prediction. Hinton et al. (2006) used complementary priors for deep belief networks to form an undirected associative memory for handwriting recognition. Furthermore, parallel tempering has been used for the Gaussian Bernoulli Restricted Boltzmann Machines (RBMs) (Cho et al., 2011). Prior to this, Cho et al. (2010) demonstrated the efficiency of parallel tempering in RBMs. Desjardins et al. (2010b) utilized parallel tempering for maximum likelihood training of RBMs and later used it for deep learning using RBMs (Desjardins et al., 2014). Recently, Chandra and Kapoor (2020) employed Langevin dynamics for Bayesian multi-source transfer learning.

In Bayesian neural learning, we estimate the posterior distribution by MCMC sampling using a 'likelihood function' that evaluates the model given the observed data and prior distribution. Given input features or covariates ($\mathbf{x}_t$), we compute $f(\mathbf{x}_t)$ by a feedforward neural network with one hidden layer,

$$f(\mathbf{x}_t) = g\left(\delta_o + \sum_{h=1}^{H} v_{ho} g\left(\delta_h + \sum_{d=1}^{I} (w_{dh}\mathbf{x}_t)\right)\right) \quad (1)$$

where $\delta_o$ and $\delta_h$ are the bias for the output $o$ and hidden $h$ layer, respectively. $v_{ho}$ is the weight which maps the hidden layer $h$ to the output layer. $w_{dh}$ is the weight which maps $\mathbf{x}_t$ to the hidden layer $h$ and $g$ is the activation function for the hidden and output layer units.

Let $\boldsymbol{\theta} = (\mathbf{w}, \mathbf{v}, \boldsymbol{\delta}, \tau^2)$, with $\boldsymbol{\delta} = (\delta_o, \delta_h)$, and $\mathcal{L}$ as the number of parameters that includes weights and biases. $\tau^2$ is a single parameter to represent the noise in the predictions given by the neural network model. $I, H, O$ refers to number of input, hidden and output neurons, respectively. We assume a Gaussian prior distribution using,

$$\log(p(\boldsymbol{\theta})) = -\frac{\mathcal{L}}{2}\log(\hat{\sigma}^2)$$

$$-\frac{1}{2\hat{\sigma}^2}\left(\sum_{h=1}^{H}\sum_{d=1}^{I} w_{dh}^2 + \sum_{h=1}^{H}(\delta_h^2 + v_h^2) + \delta_o^2\right) \tag{2}$$

where the variance ($\sigma^2$) is user-defined, gathered by information (prior belief) regarding the distribution of weights of trained neural networks in similar applications.

Given data ($\mathbf{y}$), we use Bayes rule for the posterior $p(\boldsymbol{\theta}|\mathbf{y})$ which is proportional to the likelihood $p(\mathbf{y}|\boldsymbol{\theta})$ times the prior $p(\boldsymbol{\theta})$.

$$p(\boldsymbol{\theta}|\mathbf{y}) \propto p(\mathbf{y}|\boldsymbol{\theta}) \times p(\boldsymbol{\theta})$$

Then, the log-posterior transforms to

$$\log\left(p(\boldsymbol{\theta}|\mathbf{y})\right) = \log\left(p(\boldsymbol{\theta})\right) + \log\left(p(\mathbf{y}|\boldsymbol{\theta})\right)$$

To implement the likelihood given in Eqs. (1) and (2), we use the multinomial likelihood function for the classification problems as shown in Eq. (3).

$$\log\left(p(\mathbf{y}|\boldsymbol{\theta})\right) = \sum_{t\in T}\sum_{k=1}^{K} z_{t,k}\log \pi_k \tag{3}$$

for classes $k = 1, \ldots, K$, where $\pi_k$ is the output of the neural network after applying the transfer function, and $T$ is the number of training samples. In this case, the transfer function is the softmax function (Bishop et al., 1995),

$$\pi_k = \frac{\exp(f(x_p))}{\sum_{k=1}^{K}\exp(f(x_k))} \tag{4}$$

for $k = 1, \ldots, K$. $z_{t,k}$ is an indicator variable for the given sample $t$ and the class $k$ as given in the dataset and defined by,

$$z_{t,k} = \begin{cases} 1, & \text{if } y_t = k \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

### 2.2. Parallel tempering MCMC

Parallel tempering MCMC features an ensemble of chains (known as replicas) executed at different *temperature levels* defined by a *temperature ladder* that determine the extent of exploration and exploitation (Swendsen and Wang, 1986; Hukushima and Nemoto, 1996; Hansmann, 1997; Sambridge, 2014). The replicas with higher temperature values ensure that there is enough exploration, while replicas with lower temperature values exploit the promising areas found by exploration. Typically, we exchange the neighbouring replicas given the Metropolis–Hastings criterion. In some implementations, we can also consider non-neighbouring replicas for exchange (Sambridge, 2014; Ray et al., 2016). In such cases, we calculate the acceptance probabilities of all possible moves a *priori*. The specific exchange move is then selected, which is useful when a limited number of replicas are available (Calvo, 2005). Although geometrically uniform temperature levels have been typically used for the respective replicas, determining the optimal tempering assigned for each of the replicas has been a challenge that attracted some attention in the literature (Rathore et al., 2005; Katzgraber et al., 2006; Bittner et al., 2008; Patriksson and van der Spoel, 2008). Typically, gradient-free proposals within chains are used for proposals for exploring multimodal and discontinuous posteriors (Sen and Stoffa, 1996; Maraschini and Foti, 2010); however, it is possible to incorporate gradient-based information for developing effective proposal distributions (Chandra et al., 2019a).

Although denoted "parallel", the replicas can be executed sequentially in a single processing unit (Ray et al., 2013); however, multi-core or high-performance computing systems can feature parallel implementations improving the computational time (Ray et al., 2016, 2018; Li et al., 2009). There are challenges since parallel tempering features exchange or transition between neighbouring replicas, and we need to consider efficient strategies that take into account inter-process communication (Li et al., 2009). Recently, parallel tempering has been used with high-performance computing for seismic full waveform inversion (FWI) problems (Ray et al., 2018), which is amongst one of the most computationally intensive problems in Earth science. Parallel tempering has also been used for Bayesian geophysical inversion to assess uncertainty using a multi-sensor inversion of a three-dimensional structure for mineral prospecting (Scalzo et al., 2019).

Parallel tempering has been implemented in a distributed volunteer computing network via crowd-sourcing for multi-threading and graphic processing units (Karimi et al., 2011). Furthermore, implementation with field-programmable gate array (FPGA) has shown much better performance than multi-core and graphic processing units (GPU) implementations (Mingas et al., 2017).

In a parallel tempering MCMC ensemble, given $M$ replicas defined by multiple temperature levels, the state of the ensemble is specified by $X = x_1, x_2, \ldots, x_M$; where $x_i$ is the $i$th replica, with temperature level $T_i$. The equilibrium distribution of the ensemble $X$ is given by,

$$\Pi(X) = \prod_{i=1}^{M}\frac{\exp(-\frac{1}{T_i}L(x_i))}{Z(T_i)} \tag{6}$$

where $L(x_i)$ is the log-likelihood function for the replica state at each temperature level ($T_i$) and $Z(T_i)$ is an intractable normalization constant. At every iteration of the replica state, the replica can feature two types of transitions that includes a *Metropolis transition* and a *replica transition*.

The *Metropolis transition* enables the replica to perform local Monte Carlo moves defined by the energy function $E(x_i)$. The local replica state $x_i^*$ is sampled using a proposal distribution $q_i(.|x_i)$ and the Metropolis–Hastings acceptance probability $\alpha$ for the local replica $L_{local}$ is given as,

$$\alpha = min\left(1, \exp\left(-\frac{1}{T_i}(L(x_i^*) - L(x_i))\right)\right). \tag{7}$$

The detailed balance condition holds for each replica, and therefore it holds for the ensemble system.

Typically, the Metropolis–Hastings update consists of a single stochastic process that evaluates the energy of the system and accepted is based on the temperature ladder. The selection of the temperature ladder for the replicas is done prior to sampling; we use a geometric spacing methodology (Vousden et al., 2015) as given,

$$T_i = T_{\max}^{\frac{(i-1)}{(M-1)}} \tag{8}$$

where $i = 1, \ldots, M$ and $T_{\max}$ is the maximum temperature which is user-defined.

The *Replica transition* features the exchange of two neighbouring replica states ( $x_i \leftrightarrow x_{i+1}$) defined by their temperature level, $i$ and $i+1$. The replica exchange is accepted by the Metropolis–Hastings criterion with replica exchange probability $\beta$ by,

$$\beta = min\left(1, \exp\left(\left(\frac{1}{T_{i+1}} - \frac{1}{T_i}\right)\left(L(x_{i+1}) - L(x_i)\right)\right)\right). \tag{9}$$

Based on the Metropolis criterion, the configuration (position in the replica) of neighbouring replicas at different temperatures are exchanged. This results in a robust ensemble which can sample both low and high energy configurations.

The replica-exchange enables a replica that could be stuck at a local minimum with low-temperature level to exchange configuration with a higher neighbouring temperature level and hence improve exploration. In this way, the replica-exchange can shorten the sampling time required for convergence. The frequency of determining the exchange and the temperature level is user-defined. For further details about the derivation for the equations in this section, see Sambridge (2014) and Earl and Deem (2005).

### 2.3. Surrogate-assisted optimization

Surrogate assistant optimization refers to the use of machine learning or statistical learning models to develop approximately computationally inexpensive simulation of the actual model (Jin, 2011). The

major advantage is that the surrogate model provides computationally efficiency when compared to the exact model used for evolutionary algorithms and related optimization methods (Ong et al., 2003; Zhou et al., 2007). In the optimization literature, such approaches are also known as response surface methodologies (Montgomery and Vernon M. Bettencourt, 1977; Letsinger et al., 1996) which have been applicable for a wide range of engineering problems such as reliability analysis of laterally loaded piles (Tandjiria et al., 2000).

In the case of evolutionary computation methods, Ong et al. (2003) presented parallel evolutionary optimization for solving computationally expensive functions with application to aerodynamic wing design where surrogate models used radial basis functions. Zhou et al. (2007) accelerated evolutionary optimization with global and local surrogate models while Lim et al. (2010) presented a generalized method that accounted for uncertainty in estimation to unify diverse surrogate models. Jin (2011) reviewed surrogate-assisted evolutionary computation that covered single and multi-objective, dynamic, constrained, and multimodal optimization problems. Furthermore, surrogate models have been widely used in Earth sciences such as modelling water resources (Razavi et al., 2012). Moreover, Díaz-Manríquez et al. (2016) presented a review of surrogate assisted multi-objective evolutionary algorithms that showed that the method has been successful in a wide range of application problems.

The search for the right surrogate model is a significant challenge given different types of likelihood or fitness landscape given by the actual model. Giunta and Watson (0000) presented a comparison of quadratic polynomial models with the least-square method and interpolation models that featured Gaussian process regression (kriging). They discovered that the quadratic polynomial models were more accurate in terms of errors for estimation for the optimization problems. Jin et al. (2001) presented another study that compared several surrogate models that include polynomial regression, multivariate adaptive regression splines, radial-basis functions, and kriging based on multiple performance criteria using different classes of problems. The authors reported radial basis functions as one of the best for scalability and robustness, given different types of problems. They also reported kriging to be computationally expensive.

## 3. Surrogate-assisted multi-core parallel tempering

Surrogate models primarily learn to mimic actual or true models using their behaviour, i.e. how the true model responds to a set of input parameters. A surrogate model captures the relationship between the input and output given by the true model. The input is the set of proposals in parallel tempering MCMC that features the weights and biases of the neural network model. Hence, we utilize the surrogate model to approximate the likelihood of the true model. We define the approximation of the likelihood by the surrogate as *pseudo-likelihood*. We train the surrogate model on the data that is composed of the history of proposals for weights and biases of the neural network with the corresponding true likelihood. We do not estimate the output of the neural network by the surrogate (since in some problems there are many outputs); hence to limit the number of variables, we directly estimate the likelihood using the surrogate.

We implement the neighbouring replica transition or exchange at regular intervals. The cost of inter-process communication must be limited to avoid computational overhead, given that we execute each replica on a separate processing core. The *swap interval* defines the time (number of iterations or samples) after which each replica pauses and awaits for neighbouring replica exchange. After the exchange, the replica manager process enables local Metropolis transition and the process repeats. We note that although we use a neural network model, the framework is general and we can use other models; which could include those from other domains such as Earth science models that are computationally expensive (Chandra et al., 2019b; Sambridge, 2013).

Bayesian neural learning employs parallel tempering MCMC for inference; this can be viewed as a training procedure for the neural network model. The goal of the surrogate model is to save computational time taken for evaluation of the true likelihood function associated with computationally expensive models.

Given that the true model is represented as $L = f(x)$, the surrogate model provides an approximation in the form $\hat{l} = \hat{f}(x)$, such that $L = \hat{l} + e$; where $e$ represents the difference or error. The surrogate model provides an estimate by the *pseudo-likelihood* for replacing *true-likelihood* when needed. The surrogate model is constructed by training from experience which is given by the set of input $\mathbf{x}_{i,s}$ with corresponding true-likelihoods $L_{i,s}$; where $s$ represents the sample and $i$ represents the replica. Hence, input features ($\Phi$) for the surrogate is developed by combining $\mathbf{x}_{i,s}$ using samples generated ($\theta$) across all the replicas for a given surrogate interval ($\psi$). The surrogate interval defines the batch size of the training data for the surrogate model which goes through incremental learning. All the respective replicas sample until the surrogate interval is reached and then the manager process collects the sampled data to create a training data batch for the surrogate model (see Fig. 1). This can be formulated as follows,

$$\begin{aligned} \Phi &= ([\mathbf{x}_{1,s}, \dots, \mathbf{x}_{1,s+\psi}], \dots, [\mathbf{x}_{M,s}, \dots, \mathbf{x}_{M,s+\psi}]) \\ \lambda &= ([L_{1,s}, \dots, L_{1,s+\psi}], \dots, [L_{M,s}, \dots, L_{M,s+\psi}]) \\ \Theta &= [\Phi, \lambda] \end{aligned} \tag{10}$$

where $\mathbf{x}_{i,s}$ represents the set of parameters proposed and $s$, $y_{i,s}$ is the output from the multinomial likelihood, and $M$ is the total number of replicas.

The training surrogate dataset ($\Theta = [\Phi, \lambda]$) consists of input features ($\Phi$) and response ($\lambda$) for the span of each surrogate interval ($s + \psi$). Hence, we denote the pseudo-likelihood $\hat{y}$ by $\hat{y} = \hat{f}(\Theta)$; where $\hat{f}$ is the surrogate model. We amend the likelihood in training data for the temperature level since it has been changed by taking $L_{local}/T_i$ for given replica $i$. The likelihood is amended to reflect the true likelihood rather than that represented by taking into account the temperature level. All the respective replica $\Theta_i$ data is combined, $Y = [\Theta_1, \Theta_2, \dots \Theta_N]$ and trained using the neural network model given in Eq. (1).

Algorithm 1 presents surrogate-assisted multi-core parallel tempering for Bayesian neural learning. We implement the algorithm using (multi-core) parallel processing where a manager process takes care of the ensemble of replicas that run in separate processing cores. Given the parallel processing nature, it is tricky to implement when to terminate sampling distributed amongst parallel processing replicas; hence, our termination condition waits for all the replica processes to end. We monitor the number of *alive replica process* in the master process by setting the number of alive replicas in the ensemble ($alive = M$). We note that the highlighted region of Algorithm 1 shows different processing cores as given in Fig. 2. We highlight the manager process in blue, and in pink we highlight the ensemble of replica processes running in parallel. We initially assign the replicas that sample $\theta_n$ with values using the Gaussian prior distribution with user-defined variance ($\sigma^2 = 25$) centred at the mean of 0. We define the temperature level by geometric ladder (Eq. (8)), and other key parameters which includes the number of replica samples ($R_{max}$), swap-interval which defines after how many samples to check for replica swap ($R_{swap}$), surrogate interval ($\psi$), and surrogate probability ($S_{prob}$). The main purpose of the surrogate interval is to collect enough data for the surrogate model during sampling. This also can be seen as batch-based training where we update the model after we collect the data at regular intervals.

Fig. 2 shows how we use the manager processing unit for the respective replicas running in parallel for the given surrogate interval. The manager process waits for all the replicas to reach the surrogate interval. Then we calculate the replica transition probability for the possibility of swapping the neighbouring replicas. Fig. 2 further highlights the information flow between the master process and the replica

process via inter-process communication.[1] The information flows from the replica process to master process using *signal()* given by the replica process as shown in Stage 2.2 and 5.0 of Algorithm 1.

The surrogate model is re-trained for remaining surrogate interval blocks until the maximum iteration ($R_{max}$) is reached to enable better estimation for the pseudo-likelihood. The surrogate model is trained only in the manger process, and we pass a copy of the surrogate model with the trained parameters to the ensemble of replica processes for estimating the pseudo-likelihood when needed. Note that only the samples associated with the true-likelihood become part of the surrogate training dataset. The surrogate training can consume a significant portion of time which is dependent on the size of the problem in terms of the number of parameters and the type of surrogate model used. We evaluate the trade-off between quality of estimation by pseudo-likelihood and overall cost of computation for the true likelihood function for different types of problems.

In Algorithm 1, Stage 1.4 predicts the pseudo-likelihood ($L_{surrogate}$) with given proposal $\theta_s^*$. Stage 1.5 calculates the likelihood moving average of past three likelihood values, $L_{past} = \text{mean}(L_{s-1}, L_{s-1}, L_{s-2})$. The motivation for doing this is to combine univariate time series prediction approach (moving average) with multivariate regression approach (surrogate model) for robust estimation, where we take both current and past information into account. In Stage 1.6, we combine the likelihood moving average with the pseudo-likelihood to give a prediction that considers the present replica proposal and the past behaviour, $L_{local} = (0.5 * L_{surrogate}) + 0.5 * L_{past}$. Note that although we use the past three values for the moving average, this number can change for different types of problems. Once the swap interval ($\phi$) is reached, Stage 2.0 prepares the replica transition in the manager process (highlighted in blue). Stage 3.0 executes once we reach the surrogate interval where we use data collected from Stage 1.8 for creating surrogate training set batch $\Theta$ as shown in Eq. (10). Stage 4 shows how we execute the global surrogate training in the Manager process with combined surrogate data using the neural network model in Eq. (1). We use the trained knowledge from the global surrogate model ($\Psi_{global}$) in the respective replicas local surrogate model ($\Psi_i$) as shown in Stage 1.3 and Fig. 2. Once we reach the maximum number of samples for the given replica ($R_{max}$, Stage 5.0 signals the manager process to decrement the number of replicas alive for executing the termination criterion.

Finally, we execute Stage 6 in the manager process where we combine the respective replica predictions and proposals (weights and biases) from the ensemble by concatenating the history of the samples to create the posterior distributions. This features the accepted samples and copies of the accepted samples in cases of rejected samples, as shown in Stage 1.9 of Algorithm 1.

Furthermore, the framework features parallel tempering in the first stage of sampling that transforms into a local mode or exploitation in the second stage where the temperature ladder is changed such that $T_i = 1$, for all replicas, $i = 1, 2, \ldots, N$ as previously done in Chandra et al. (2019a,b). We emphasize on exploration in the first phase and emphasize on exploitation in the second phase, as shown in Stage 1.9.1 of Algorithm 1. The duration of each phase is problem dependent, which we determine from trial experiments.

We validate the quality of the surrogate model prediction using the root mean squared error (RMSE)

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(L_i - \hat{L}_i)^2}$$

where $L_i$ and $\hat{L}_i$ are the true likelihood and the pseudo-likelihood values, respectively. $N$ is the number of cases we employ the surrogate during sampling.

[1] Python multiprocessing library for implementation: https://docs.python.org/2/library/multiprocessing.html.

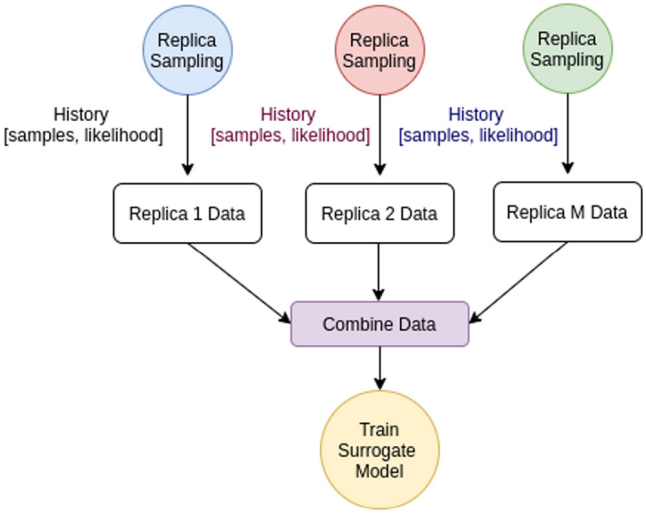


**Fig. 1.** Data collection for training surrogate model.

Note that instead of swapping entire replica configuration, an alternative is to swap the respective temperatures. This could save the amount of information exchanged during inter-process communication as the temperature is only a double-precision number, implemented by Ray and Myer (2019).

### 3.1. Langevin gradient-based proposal distribution

Apart from random-walk proposals, we utilize *stochastic gradient Langevin dynamics* (Welling and Teh, 2011) for the proposal distribution. It features additional noise with stochastic gradients to optimize a differentiable objective function which has been very promising for neural networks (Chandra et al., 2019a). The proposal distribution is constructed as follows.

$$\begin{aligned} \theta^p &\sim \mathcal{N}(\bar{\theta}^{[k]}, \Sigma_\theta), \text{ where} \\ \bar{\theta}^{[k]} &= \theta^{[k]} + r \times \nabla E_{\mathbf{y}_{\mathcal{A}_{D,T}}}[\theta^{[k]}], \\ E_{\mathbf{y}_{\mathcal{A}_{D,T}}}[\theta^{[k]}] &= \sum_{t \in \mathcal{A}_{D,T}} (y_t - f(\mathbf{x}_t)^{[k]})^2, \\ \nabla E_{\mathbf{y}_{\mathcal{A}_{D,T}}}[\theta^{[k]}] &= \left(\frac{\partial E}{\partial \theta_1}, \ldots, \frac{\partial E}{\partial \theta_L}\right) \end{aligned} \tag{11}$$

$r$ is the learning rate, $\Sigma_\theta = \sigma_\theta^2 I_L$ and $I_L$ is the $L \times L$ identity matrix. So that the newly proposed value of $\theta^p$, consists of 2 parts:

1. An gradient descent based weight update given by Eq. (11).
2. Add an amount of noise, from $\mathcal{N}(0, \Sigma_\theta)$.

### 3.2. Surrogate model

The choice of the surrogate model needs to consider the computational resources taken for training the model during the sampling process. We note that Gaussian process models, neural networks, and radial basis function models (Broomhead and Lowe, 1988) have been popular choices for surrogates in the literature.

In our case, we consider the inference problem that features hundreds to thousands of parameters; hence, the model needs to be efficiently trained without taking lots of computational resources. Moreover, the flexibility of the model to have incremental training is also needed. Therefore, we rule out Gaussian process models since they have imitations given large training datasets (Rasmussen, 2004). In our case, tens of thousands of samples could make the training data. Therefore, we use neural networks as the choice of the surrogate model. The training data and neural network model can be formulated as follows.

**Data**: Classification Dataset
**Result**: Posterior distributions for neural network weights
* Set the number of replicas ($M$) in ensemble as $alive$; $alive = M$
* Assign: replica tempertaure level using geometric() temperature ladder, number of replica processes ($M$), surrogate interval ($\psi$), replica swap interval ($R_{swap}$), and maximum number of samples for each replica ($R_{max}$).
**while** $(alive \neq 0$ **do**
  Stage 0: Prepare manager process to execute each replica in parallel cores
  **for** *each* $i$ *until* $M$ **do**
    $s = 0$
    **first-phase:** $T_i$ = geometric()
    **while** $(s < R_{max})$ **do**
      Stage 1.0: Metropolis Transition
      **for** *each* $v$ *until* $\psi$ **do**
        **for** *each* $k$ *until* $R_{swap}$ **do**
          1.1 Random-walk, $\theta_s^* = \theta_s + \epsilon$
          1.2 $L_{local}$ calculate:
          Draw $\kappa$ from a Uniform distribution [0,1]
          **if** $\kappa < S_{prob}$ *and* $s > \psi$ **then**
            Estimate $L_{local}$ from local surrogate's prediction, $L_{surrogate}$
            1.3 Copy global surrogate knowledge to local surrogate, $\Psi_i \leftarrow \Psi_{global}$
            1.4 Predict $L_{surrogate}$ value with the proposed $\theta_i^*$
            1.5 $L_{past}$ = mean($L_{s-1}, L_{s-1}, L_{s-2}$)
            1.6 Assign $L_{local}$ = (0.5 * $L_{surrogate}$) + 0.5 * $L_{past}$
          **else**
            1.7 $L_{local}$ = true-likelihood, given by Likelihood function in Eq. (3))
            1.8 Save $L_s = L_{local}$ (Eq. (10))
          **end**
          1.9 Calculate acceptance probability $\alpha$ and draw $u$ from uniform distribution
          **if** $u \leq \alpha$ **then**
            Accept replica state, $\theta_s \leftarrow \theta_s^*$
          **end**
          **else**
            Reject and retain previous state: $\theta_s \leftarrow \theta_{s-1}^*$
          **end**
          1.9.1 **second-phase**: switch sampling style by updating replica temperature
          **if** *second-phase is true* **then**
            Update temperature, $T_i = 1$
          **end**
          Increment $s$
        **end**
        Stage 2.0: Replica Transition:
        2.1 Calculate acceptance probability $\beta$ and draw $b$ from a Uniform distribution [0,1]
        **if** $b \leq \beta$ **then**
          2.2 Signal() manager process
          2.3 Exchange neighbouring Replica, $\theta_i \leftrightarrow \theta_{s+1}$
        **end**
      **end**
      Stage 3.0: Set $\Theta_i$ which features history of proposals $\Phi$ ($\theta$) and response $\lambda$ ( $L_{local}$ ). Use data collected from Stage 1.8
      Stage 4.0: Global Surrogate Training
      **for** *each replica* **do**
        4.1 Get replica surrogate data $\Theta_i$ from Stage 3.0
      **end**
      4.2 Train global surrogate model with combined surrogate data, $Y = [\Theta_1, \Theta_2, \ldots \Theta_N]$ using neural network model in Eq. (1)
      4.3 Save global surrogate model parameters, $\Psi_{global}$
    **end**
    Stage 5.0: Signal() manager process
    5.1 decrement number of replica processes $alive$
  **end**
**end**
Stage 6: Combine predictions and posterior from respective replicas in the ensemble, using second-phase MCMC samples.

**Algorithm 1:** Surrogate-assisted parallel tempering for Bayesian neural networks. The highlighted regions of the algorithm shows different processing cores. The manager process is highlighted in blue while the ensemble of replica processes running in parallel is highlighted in pink.

The data given to the surrogate model is $\Phi$ and $\lambda$ as in (10), where $\Phi$ is the input and $\lambda$ is the desired output of the model. The prediction of the model is denoted by $\hat{\lambda}$. We explain the surrogate models used in the paper as follows.

We note that stochastic gradient descent maintains a single learning rate for all weight updates and typically the learning rate does not change during training. Adam (adaptive moment estimation) learning algorithm (Kingma and Ba, 2014) differs from classical stochastic gradient descent, as the learning rate is maintained for each network

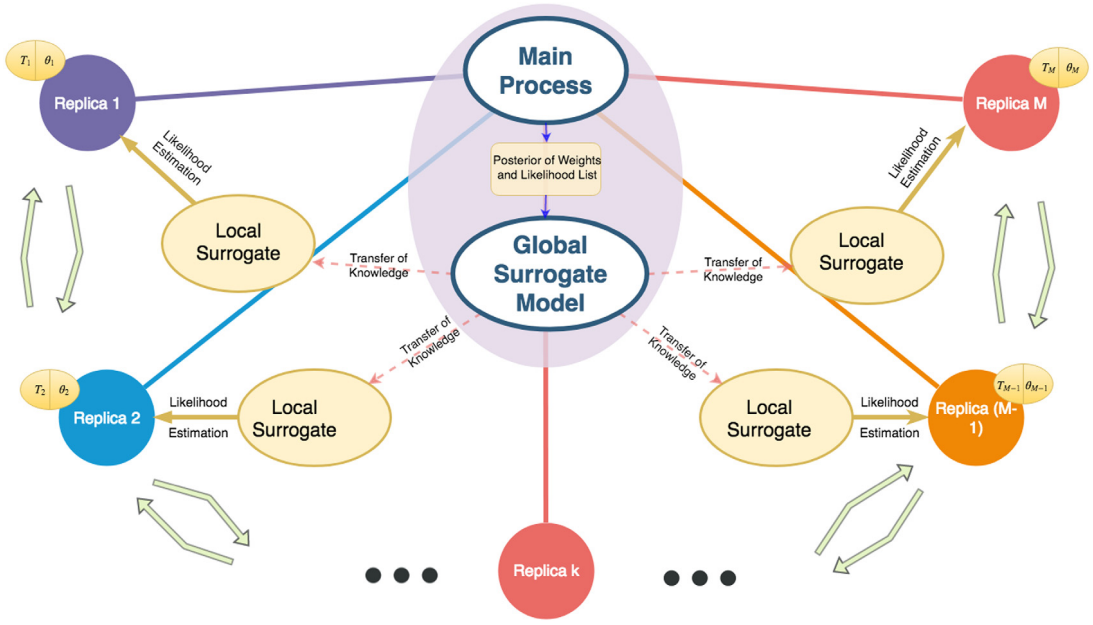


**Fig. 2.** Surrogate-assisted multi-core parallel tempering features surrogates to estimate the likelihood function at times rather than evaluating it.

weight and separately adapted as learning unfolds. Adam computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients. Adam features the strengths of *root mean square propagation* (RMSProp) (Hinton et al., 2012), and *adaptive gradient algorithm* (AdaGrad) (Duchi et al., 2011). Adam has shown better results when compared to stochastic gradient descent, RMSprop and AdaGrad. Hence, we consider Adam as the designated algorithm for the neural network-based surrogate model.

We note that the surrogate model predictions do not provide uncertainty quantification since they are not probabilistic models. The proposed framework is general and hence different surrogate types of models can be used in future. Different types of surrogate models would consume different computational time and have certain strengths and weaknesses (Jin et al., 2001).

## 4. Experiments and results

In this section, we present an experimental analysis of surrogate-assisted parallel tempering (SAPT) for Bayesian neural learning. The experiments consider a wide range of issues that test the accuracy of pseudo-likelihood by the surrogate, the quality in decision making given by the classification performance, and the amount of computational time saved.

### 4.1. Experimental design

We select six benchmark pattern classification problems from the *University of California Irvine* machine learning repository (Dua and Graff, 2017). The problems feature different levels of computational complexity and learning difficultly; in terms of the number of instances, the number of attributes, and the number of classes, as shown in Table 1. We use the multinomial likelihood given in Eq. (3) the selected classification problems. Moreover, we show the performance using Langevin-gradient proposals which takes more computational time due to cost of computing gradients when compared to random-walk proposals; however, it gives better prediction accuracy given the same number of samples as reported in our previous work (Chandra et al., 2019a). The experimental design follows the following strategy in evaluating the performance of the selected parameters from Algorithm 1.

**Table 1**
Dataset description (Dua and Graff, 2017).

| Dataset | Instances | Attributes | Classes | Hidden units |
|---|---|---|---|---|
| Iris | 150 | 4 | 3 | 12 |
| Ionosphere | 351 | 34 | 2 | 50 |
| Cancer | 569 | 9 | 2 | 12 |
| Bank | 11 162 | 20 | 2 | 50 |
| Pen-Digit | 10 992 | 16 | 10 | 30 |
| Chess | 28 056 | 6 | 18 | 25 |

- Evaluate the effect of the surrogate probability ($S_{prob}$ on the computational time and classification performance.
- Evaluate the effect of the surrogate interval ($\psi$ on the computational time and classification performance.
- Evaluate the effect of Langevin-gradients for proposals in parallel tempering MCMC.

We provide the parameter setting for the respective experiments as follows. A *burn-in* time $R_{burn} = 0.50$ (50%) of the samples for the respective replica. The burn-in strategy is standard practice for MCMC sampling which ensures that the chain enters a high probability region, where the states of the Markov chain are more representative of the posterior distribution. Although MCMC methods more commonly use burn-in time of 10%–20% in the literature, we use 50% for getting prediction performance of preferred accuracy. The maximum sampling time, $F_{max} = 50{,}000$ for all the respective problems, and number of replicas, $M = 10$ which run on parallel processing cores. The other key parameters include replica swap interval ($R_{swap} = 50$), surrogate interval ($\psi = 50$), replica sampling time ($R_{max} = F_{max}/M$), surrogate probability ($S_{prob} = 0.25$, $S_{prob} = 0.50$) and maximum temperature ($T_{\max} = 5$). We determined the given values for the parameters in trial experiments. We selected the maximum temperature in trial experiments by taking into account the performance accuracy given with a fixed number of samples. We provide details for the pattern classification datasets with details of Bayesian neural network topology (number of hidden units) in Table 1.

In the case of random-walk proposal distribution, we add a Gaussian noise to the weights and biases of the network from a normal distribution with mean of 0 and standard deviation of 0.025. The user defined constant for the prior (see Eq. (2)) is set as $\sigma^2 = 25$.

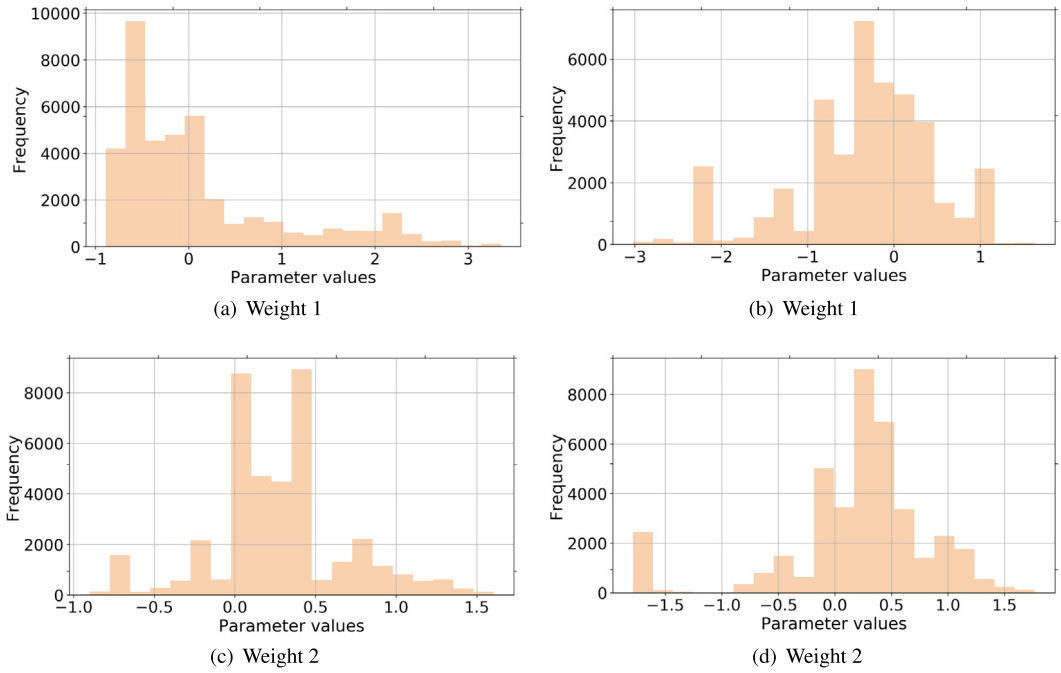


**Fig. 3.** Posterior distribution and trace-plot comparing surrogate-assisted parallel tempering (Panel (a) and (c)) with parallel tempering (Panel (b) and (d)) MCMC for selected weights (1 and 2) for the Cancer problem.

In the respective experiments, we compare SAPT with parallel tempering featuring random-walk proposals (PTRW) and Langevin-gradients (PTLG) taken from the literature (Chandra et al., 2019a). Surrogate-assisted parallel tempering also features Langevin-gradients (SAPT-LG) given in Eq. (11). We use a learning rate of 0.5 for computing the weight update via the gradients. Furthermore, we apply Langevin-gradient with a probability $L_{prob} = 0.5$, and random-walk proposal, otherwise. Note that the respective methods feature parallel tempering for the first 50 per cent of the samples that make the burn-in period. Afterwards, we use canonical MCMC where temperature $T = 1$ using parallel computing environment featuring replica-exchange via interprocess communication.

### 4.2. Implementation

We employ multi-core parallel tempering for neural networks[2] to implement surrogate assisted parallel tempering. We use one hidden layer in the Bayesian neural network for classification problems.

We use the *Keras* machine learning library for implementing the surrogate[3] with Adam learning algorithm (Kingma and Ba, 2014). The surrogate neural network model architecture consists of $[i, h_1, h_2, o]$; where $i$ refers to the number of inputs that consists of the total number of weights and biases used in the Bayesian neural network for the given problem. $h_1$ and $h_2$ refers to the first and second hidden layers, and $o$ represents the output that predicts the likelihood. In our experiments, we use hidden units $h_1 = 64, h_2 = 16$, for the Iris and Cancer problems. In the Ionosphere and Bank problems, we use hidden units $h_1 = 120, h_2 = 40$, and for Pen-Digit and Chess problems, we use hidden units $h_1 = 200, h_2 = 50$. All problems used one output unit for the surrogate model, $o = 1$.

[2] Multi-core parallel tempering: https://github.com/sydney-machine-learning/parallel-tempering-neural-net.

[3] Keras: https://keras.io/.

### 4.3. Results

We first present the results in terms of classification accuracy of posterior samples for SAPT-RW with random-walk proposal distribution (Table 2), where we evaluate different combinations of selected values of surrogate interval and $\psi$ surrogate probability $S_{prob}$. Looking at the elapsed time, we find that SAPT-RW is more costly for the Iris and Cancer problems when compared to PT-RW. These are smaller problems when compared to rest given the size of the dataset shown in Table 1. The Ionosphere problem saved computation time for both instances of SAPT-RW. A larger dataset with a Bayesian neural network model implies that there is more chance for the surrogate to save time which is visible in the Pen-Digit and Chess problems. The Bank problem does not save much time but retains the accuracy in classification performance. Furthermore, we find that instances of SAPT improve the classification accuracy of Iris, Ionosphere, Pen-Digit and Chess problems. In Cancer and Bank problems, the performance is similar.

We provide the results for Langevin-gradient proposals (SAPT-LG) in Table 3. In general, the classification performance improves when compared to random-walk proposal distribution (SAPT-RW) in Table 2. By incorporating the gradient into the proposal, the results show faster convergence with better accuracy given the same number of samples. In a comparison of both forms of parallel tempering (PT-RW and PT-LG) with the surrogate-assisted framework (SAPT-RW and SAPT-LG), we observe that the elapsed time has not improved for Iris, Ionosphere and Cancer problems; however, it has improved in the rest of the problems.

We show the accuracy of the surrogate in predicting the likelihood in Table 4 for smaller problems that include Ionosphere, Cancer and Iris. We notice that the RMSE for surrogate prediction is lower for the Iris problem when compared to the others; however, as shown in Figs. 4–6, this is relative to the range of log-likelihood. We observe that the log-likelihood prediction by the surrogate model is much better for the smaller problems (Iris and Cancer) when compared to the larger problems (Pen-Digit and Chess).

Fig. 3 shows the comparison of the posterior distribution of two selected weights from input to the hidden layer of the Bayesian neural network for the Cancer problem using SAPT-RW and PT-RW. We notice

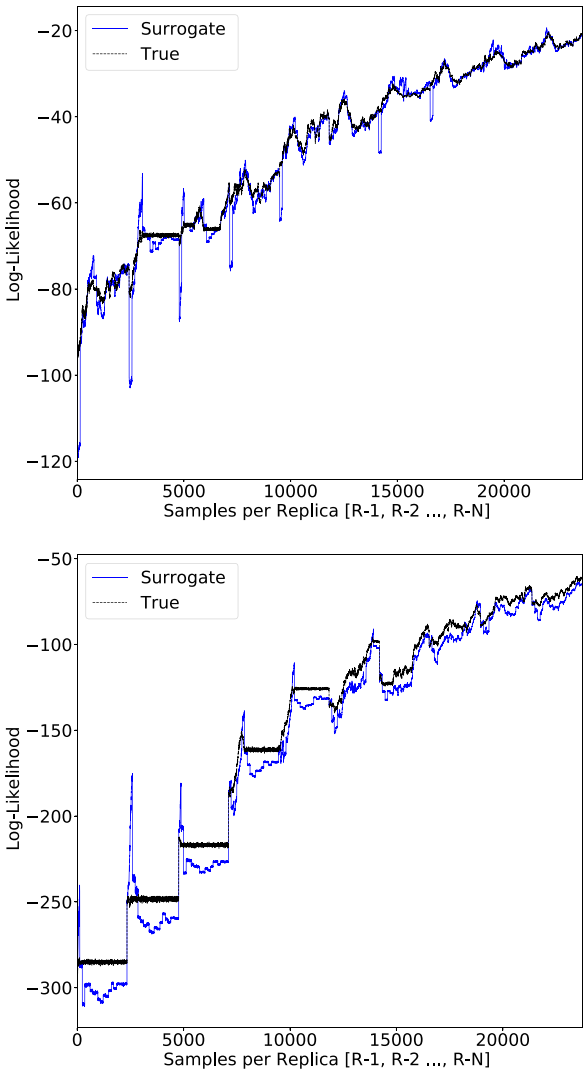


**Fig. 4.** The Iris (top) and Cancer (bottom) surrogate accuracy. The dashed line denotes the real likelihood function while the blue line gives the surrogate likelihood estimation.

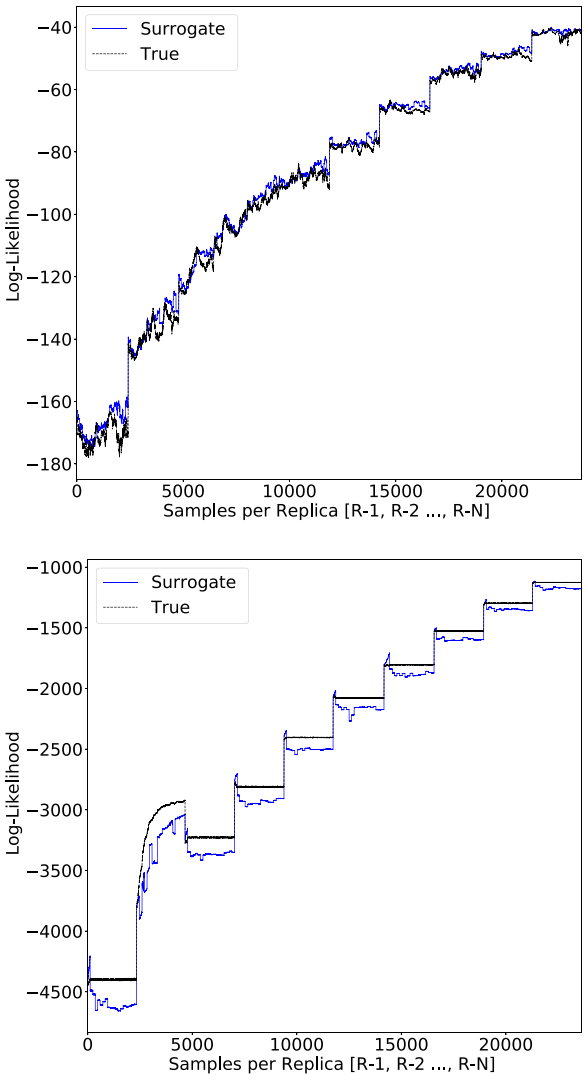


**Fig. 5.** The Ionosphere (top) and Bank (bottom) surrogate accuracy. The dashed line denotes the real likelihood function while the blue line gives the surrogate likelihood estimation.

that although the chains converged to different sub-optimal models in the multimodal distributions, SAPT-RW does not depreciate in terms of exploration.

Table 5 provides an evaluation of the number of replicas (cores) on the computational time (minutes) using SAPT-RW for selected problems. We observe that in general, the computational time reduces as the number of replica increases by taking advantage of parallel processing.

## 5. Discussion

The results, in general, have shown that surrogate-assisted parallel tempering can be beneficial for larger datasets and models, demonstrated by Bayesian neural network architecture for Pen-Digit and Chess classification problems. This implies that the method would be very useful for large scale models where computational time can be lowered while maintaining performance in decision making such as classification accuracy. We observed that in general, the Langevin-gradients improves the accuracy of the results. Although we used Bayesian neural networks to demonstrate the challenge in using computationally expensive models with large datasets, surrogate-assisted parallel tempering can be used for a wide range of models across different domains. We note that Langevin-gradients are limited to models where gradient information is available. In large and computationally expensive geo-scientific models such as modelling landscape (Chandra et al., 2019b) and reef evolution (Pall et al., 2020), it is difficult to obtain gradients from the models; hence, random-walk and related meta-heuristics are used for proposal distributions.

The proposed method employs transfer learning for the surrogate model, where we transfer the knowledge and refine the surrogate model in the forthcoming surrogate intervals with new data. Since each replica of parallel tempering is executed on a separate processing core, inter-process communication is used for exchanging information between the different replicas. Inter-process communication is also used for collecting the history of information in terms of proposals and associated likelihood for creating training datasets for the surrogate model. The proposed method could be seen as a case for online learning that considers a sequence of predictions from previous tasks and currently available information (Shalev-Shwartz et al., 2012). This is because the surrogate is trained at every surrogate interval and the surrogate gives an estimation of the likelihood until the next interval for retraining based on accumulated data of proposal and true-likelihood for the previous interval.

We observe that there is systematic under-estimation of the true likelihood for some of the cases (Figs. 4–6). One reason could be

**Table 2**
Classification results (random-walk proposal distribution).

| Dataset | Method SAPT($S_{prob}$) | Train accuracy [mean, std, best] | Test accuracy [mean, std, best] | Elapsed time (minutes) |
|---|---|---|---|---|
| Iris | PT-RW | 51.39 15.02 91.43 | 50.18 41.78 100.00 | 1.26 |
| | SAPT-RW (0.25) | 69.31 2.05 80.00 | 65.24 4.04 78.38 | 1.93 |
| | SAPT-RW (0.50) | 44.11 11.93 78.89 | 50.01 5.65 70.30 | 1.81 |
| Ionosphere | PT-RW | 68.92 16.53 91.84 | 51.29 30.73 91.74 | 3.50 |
| | SAPT-RW (0.25) | 76.60 4.42 86.94 | 61.23 8.05 86.24 | 2.93 |
| | SAPT-RW (0.50) | 70.34 6.15 83.27 | 73.42 14.06 95.41 | 2.43 |
| Cancer | PT-RW | 83.78 20.79 97.14 | 83.55 27.85 99.52 | 2.78 |
| | SAPT-RW(0.25) | 89.75 6.91 96.32 | 92.80 4.57 99.05 | 3.41 |
| | SAPT-RW(0.50) | 91.17 6.29 96.52 | 97.64 3.17 99.52 | 2.84 |
| Bank | PT-RW | 78.39 1.34 80.11 | 77.49 0.90 79.45 | 27.71 |
| | SAPT-RW(0.25) | 78.44 0.67 79.69 | 77.79 0.63 79.60 | 28.67 |
| | SAPT(0.50) | 77.82 1.05 79.69 | 77.16 0.91 78.80 | 27.38 |
| Pen Digit | PT-RW | 76.67 17.44 95.24 | 71.93 16.59 90.62 | 57.13 |
| | SAPT-RW(0.25) | 88.74 1.94 92.87 | 83.60 2.14 88.74 | 49.25 |
| | SAPT-RW(0.50) | 80.85 1.28 82.87 | 77.66 1.08 80.02 | 36.05 |
| Chess | PT-RW | 89.48 17.46 100.00 | 90.06 15.93 100.00 | 252.56 |
| | SAPT-RW(0.25) | 97.17 8.35 100.00 | 97.66 6.83 100.00 | 197.61 |
| | SAPT-RW(0.50) | 90.87 13.35 100.00 | 90.71 13.31 100.00 | 143.75 |

**Table 3**
Classification results (Langevin-gradient proposal distribution).

| Dataset | Method | Train accuracy [mean, std, best] | Test accuracy [mean, std, best] | Elapsed time (minutes) |
|---|---|---|---|---|
| Iris | PT-LG | 97.32 0.92 99.05 | 96.76 0.96 99.10 | 2.09 |
| | SAPT-LG (0.25) | 98.91 0.16 100.00 | 99.93 0.39 100.00 | 2.85 |
| | SAPT-LG (0.50) | 99.09 0.43 100.00 | 98.63 1.57 100.00 | 2.49 |
| Ionosphere | PT-LG | 98.55 0.55 99.59 | 92.19 2.92 98.17 | 5.07 |
| | SAPT-LG (0.25 ) | 100.00 0.02 100.00 | 90.82 2.43 96.33 | 6.17 |
| | SAPT-LG (0.50 ) | 99.51 0.60 100.00 | 91.24 1.82 97.25 | 4.76 |
| Cancer | PT-LG | 97.00 0.29 97.75 | 98.77 0.32 99.52 | 5.09 |
| | SAPT-LG(0.25) | 99.36 0.11 99.39 | 98.00 0.76 99.52 | 8.18 |
| | SAPT-LG(0.50) | 99.37 0.12 99.59 | 98.61 0.65 100.00 | 6.64 |
| Bank | PT-LG | 80.75 1.45 85.41 | 79.96 0.81 82.61 | 86.94 |
| | SAPT-LG(0.25) | 79.86 0.15 80.30 | 80.53 0.28 79.22 | 75.96 |
| | SAPT-LG(0.50) | 80.86 0.15 80.30 | 81.53 0.28 79.25 | 65.11 |
| Pen Digit | PT-LG | 84.98 7.42 96.02 | 81.24 6.82 91.25 | 86.62 |
| | SAPT-LG(0.25) | 82.12 7.42 94.02 | 82.24 6.82 93.25 | 66.62 |
| | SAPT-LG(0.50) | 83.98 7.42 95.02 | 83.14 6.82 92.25 | 56.62 |
| Chess | PT-LG | 100.00 0.00 100.00 | 100.00 0.00 100.00 | 323.10 |
| | SAPT-LG(0.25) | 100.00 0.00 100.00 | 100.00 0.00 100.00 | 223.70 |
| | SAPT-LG(0.50) | 100.00 0.00 100.00 | 100.00 0.00 100.00 | 173.10 |

**Table 4**
Surrogate accuracy.

| Dataset | Method | Surrogate prediction RMSE | Surrogate training RMSE [mean, std] |
|---|---|---|---|
| Iris | SAPT-RW | 3.55 | 3.84e−05 8.40e−05 |
| Ionosphere | SAPT-RW | 2.63 | 7.20e−05 1.62e−04 |
| Cancer | SAPT-RW | 11.08 | 6.12e−05 1.58e−04 |
| Bank | SAPT-RW | 131.26 | 3.09e−05 1.53e−04 |
| Pen-Digit | SAPT-RW | 1246.60 | 3.93e−06 1.47e−05 |
| Chess | SAPT-RW | 4026.44 | 1.34e−06 3.58e−06 |

**Table 5**
Effect of number of replica on the computational time (minutes).

| Num. Replica | Cancer | Bank | Pen-digit | Chess |
|---|---|---|---|---|
| 4 | 45.66 | 79.25 | 102.28 | 253.98 |
| 6 | 21.16 | 43.48 | 61.38 | 188.79 |
| 8 | 13.25 | 32.57 | 51.92 | 185.52 |
| 10 | 9.17 | 29.75 | 44.7 | 130.57 |

that the variance is overestimated and the other is due to the global surrogate model, where training is done by one model in the *manager process* and the knowledge is used in all the replicas. The surrogate accuracy depends on the nature of the problem, and the neural network model, in terms of the number of parameters in the model and the size of the dataset. We observed that smaller Bayesian neural network models had good accuracy in surrogate estimation when compared to others. In future work, we can consider training surrogate model in the local replicas which could further improve the estimation. The global surrogate model has the advantage of combining information across the different replicas, it faces challenges of dealing with large surrogate training dataset which accumulates over sampling time. We also need to lower the time taken for the exchange of knowledge needed for decision making by the local surrogate model and the master replica. We found that bigger problems (such as Pen-Digit and Chess) give further challenges for surrogate estimation. In such cases, it would be worthwhile to take a time series approach, where the history of the likelihood is treated as time series. Hence, a local surrogate could learn from the past tend of the likelihood, rather than the history of past proposals. This could help in addressing computational challenges given a large number of model parameters to be considered for surrogate training.

Although we ruled out Gaussian process models as the choice of the surrogate model due to computational complexity in training large surrogate data, we need to consider that Gaussian process models naturally account for uncertainty quantification in decision making. Recent

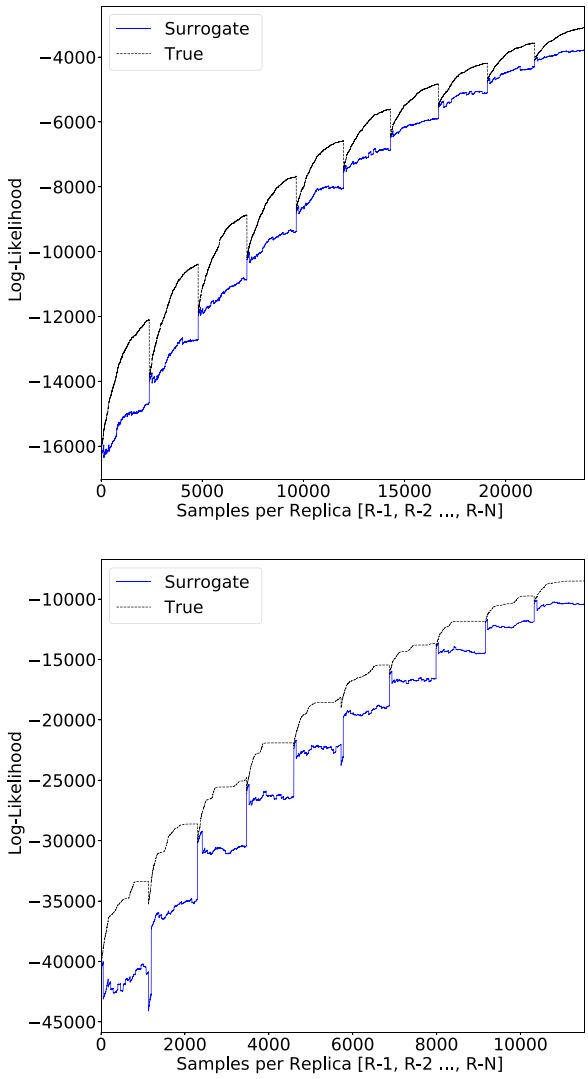


**Fig. 6.** The Pen-Digit (top) and Chess (bottom) surrogate accuracy. The dashed line denotes the real likelihood function while the blue line gives the surrogate likelihood estimation.

techniques to address the issue of training Gaussian process models for large datasets could be a way ahead in future studies (Moore et al., 2016). Moreover, we note that there has not been much work done in the literature that employs surrogate assisted machine learning. Most of the literature considered surrogate assisted optimization, whereas we considered inference for machine learning problems. The results open the road to use surrogate models for machine learning. Surrogates would be helpful in case of big data problems and cases where there are inconsistencies or noisy data. Furthermore, other optimization methods could be used in conjunction with surrogates for big data problems rather than parallel tempering.

Finally, we use the surrogate model with a certain probability in all replicas, hence the resulting distribution is only an approximation of the true posterior distribution. An interesting question would then be whether there is a way to use surrogates while not introducing an approximation error. This is trivially possible for sequential approaches; however, in parallel cases, this could be an issue since multiple replicas are used which all contribute to different levels of uncertainties. This would need to be addressed in future work in uncertainty quantification for the surrogate likelihood prediction, perhaps with the use of Gaussian process surrogate models or by introducing an error term for each surrogate model and integrating it out via MCMC sampling. Potentially, multi-fidelity modelling where the synergy between low-fidelity and how fidelity data and models could be employed for the surrogate model (Peherstorfer et al., 2018).

## 6. Conclusions and future work

We presented surrogate-assisted parallel tempering for implementing Bayesian inference for computationally expensive problems that harness the advantage of parallel processing. We used a Bayesian neural network model to demonstrate the effectiveness of the framework for computationally expensive problems. The results from the experiments reveal that the method gives a promising performance where computational time is reduced for larger problems.

The surrogate-based framework is flexible and can incorporate different surrogate models and be applied to problems across various domains that feature computationally expensive models which require parameter estimation and uncertainty quantification. In future work, we envision strategies to further improve the surrogate estimation for large models and parameters. A way ahead is to utilize local surrogate via time series prediction which could help in alleviating the limitations. Furthermore, the framework could be applied to problems in different domains such as computationally expensive geoscientific models used for landscape evolution.

## Software and data

We provide an open-source implementation of the proposed algorithm in Python along with data and sample results.[4]

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Acknowledgements

The authors would like to thank Prof. Dietmar Muller and Danial Azam for discussions and support during the course of this research project. We sincerely thank the editors and anonymous reviewers for their valuable comments.

## References

Auld, T., Moore, A.W., Gull, S.F., 2007. Bayesian neural networks for Internet traffic classification. IEEE Trans. Neural Netw. 18 (1), 223–239.
Banerjee, S., Carlin, B.P., Gelfand, A.E., 2014. Hierarchical Modeling and Analysis for Spatial Data. CRC Press.
Bishop, C., Bishop, C.M., et al., 1995. Neural Networks for Pattern Recognition. Oxford University Press.
Bittner, E., Nußbaumer, A., Janke, W., 2008. Make life simple: Unleash the full power of the parallel tempering algorithm. Phys. Rev. Lett. 101 (13), 130603.
Blei, D.M., Kucukelbir, A., McAuliffe, J.D., 2017. Variational inference: A review for statisticians. J. Amer. Statist. Assoc. 112 (518), 859–877.
Blundell, C., Cornebise, J., Kavukcuoglu, K., Wierstra, D., 2015. Weight uncertainty in neural network. In: Proceedings of the 32nd International Conference on Machine Learning. pp. 1613–1622.
Brakel, P., Dieleman, S., Schrauwen, B., 2012. Training restricted Boltzmann machines with multi-tempering: harnessing parallelization. In: International Conference on Artificial Neural Networks. Springer, pp. 92–99.
Broomhead, D.S., Lowe, D., 1988. Radial Basis Functions, Multi-Variable Functional Interpolation and Adaptive Networks. Tech. Rep., Royal Signals and Radar Establishment Malvern, United Kingdom.
Calvo, F., 2005. All-exchanges parallel tempering. J. Chem. Phys. 123 (12), 124106.
Chandra, R., Azizi, L., Cripps, S., 2017. Bayesian neural learning via Langevin dynamics for chaotic time series prediction. In: International Conference on Neural Information Processing. Springer, pp. 564–573.

[4] Surrogate-assisted multi-core parallel tempering: https://github.com/Sydney-machine-learning/surrogate-assisted-parallel-tempering.

Chandra, R., Jain, K., Deo, R.V., Cripps, S., 2019a. Langevin-gradient parallel tempering for Bayesian neural learning. Neurocomputing 359, 315–326.

Chandra, R., Kapoor, A., 2020. Bayesian neural multi-source transfer learning. Neurocomputing 378, 54–64.

Chandra, R., Müller, R.D., Azam, D., Deo, R., Butterworth, N., Salles, T., Cripps, S., 2019b. Multicore parallel tempering Bayeslands for basin and landscape evolution. Geochemistry, Geophysics, Geosystems 20 (11), 5082–5104.

Cho, K., Ilin, A., Raiko, T., 2011. Improved learning of Gaussian-Bernoulli restricted Boltzmann machines. In: International Conference on Artificial Neural Networks. Springer, pp. 10–17.

Cho, K., Raiko, T., Ilin, A., 2010. Parallel tempering is efficient for learning restricted Boltzmann machines. In: Neural Networks (IJCNN), the 2010 International Joint Conference on. IEEE, pp. 1–8.

Damianou, A.C., Titsias, M.K., Lawrence, N.D., 2016. Variational inference for latent variables and uncertain inputs in Gaussian processes. J. Mach. Learn. Res. 17 (1), 1425–1486.

Desjardins, G., Courville, A., Bengio, Y., 2010b. Adaptive parallel tempering for stochastic maximum likelihood learning of RBMs. arXiv preprint arXiv:1012.3476.

Desjardins, G., Courville, A., Bengio, Y., Vincent, P., Delalleau, O., 2010. Tempered Markov chain Monte Carlo for training of restricted Boltzmann machines. In: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics. pp. 145–152.

Desjardins, G., Luo, H., Courville, A., Bengio, Y., 2014. Deep tempering. arXiv preprint arXiv:1410.0123.

Díaz-Manríquez, A., Toscano, G., Barron-Zambrano, J.H., Tello-Leal, E., 2016. A review of surrogate assisted multiobjective evolutionary algorithms. Comput. Intell. Neurosci. 2016.

Dua, D., Graff, C., 2017. UCI Machine Learning Repository. University of California, Irvine, School of Information and Computer Sciences, URL http://archive.ics.uci.edu/ml.

Duchi, J., Hazan, E., Singer, Y., 2011. Adaptive subgradient methods for online learning and stochastic optimization. J. Mach. Learn. Res. 12 (Jul), 2121–2159.

Earl, D.J., Deem, M.W., 2005. Parallel tempering: Theory, applications, and new perspectives. Phys. Chem. Chem. Phys. 7 (23), 3910–3916.

Fischer, A., Igel, C., 2015. A bound for the convergence rate of parallel tempering for sampling restricted Boltzmann machines. Theoret. Comput. Sci. 598, 102–117.

Freedman, D.A., 1963. On the asymptotic behavior of Bayes' estimates in the discrete case. Ann. Math. Stat. 1386–1403.

Gal, Y., Ghahramani, Z., 2016. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In: International Conference on Machine Learning. pp. 1050–1059.

Geyer, C.J., Thompson, E.A., 1995. Annealing Markov chain Monte Carlo with applications to ancestral inference. J. Amer. Statist. Assoc. 90 (431), 909–920.

Giunta, A., Watson, L., 0000. A comparison of approximation modeling techniques-Polynomial versus interpolating models. In: 7th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization. p. 4758.

Hansmann, U.H., 1997. Parallel tempering algorithm for conformational studies of biological molecules. Chem. Phys. Lett. 281 (1–3), 140–150.

Hastings, W.K., 1970. Monte Carlo sampling methods using Markov chains and their applications. Biometrika 57 (1), 97–109.

Hicks, R.M., Henne, P.A., 1978. Wing design by numerical optimization. J. Aircr. 15 (7), 407–412.

Hinton, G.E., Osindero, S., Teh, Y.-W., 2006. A fast learning algorithm for deep belief nets. Neural Comput. 18 (7), 1527–1554.

Hinton, G., Srivastava, N., Swersky, K., 2012. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. http://www.cs.toronto.edu/~hinton/coursera/lecture6/lec6.pdf.

Hukushima, K., Nemoto, K., 1996. Exchange Monte Carlo method and application to spin glass simulations. J. Phys. Soc. Japan 65 (6), 1604–1608.

Jeong, S., Murayama, M., Yamamoto, K., 2005. Efficient optimization design method using kriging model. J. Aircr. 42 (2), 413–420.

Jin, Y., 2011. Surrogate-assisted evolutionary computation: Recent advances and future challenges. Swarm Evol. Comput. 1 (2), 61–70.

Jin, R., Chen, W., Simpson, T.W., 2001. Comparative studies of metamodelling techniques under multiple modelling criteria. Struct. Multidiscip. Optim. 23 (1), 1–13.

Karimi, K., Dickson, N., Hamze, F., 2011. High-performance physics simulations using multi-core cpus and gpgpus in a volunteer computing context. Int. J. High Perform. Comput. Appl. 25 (1), 61–69.

Katzgraber, H.G., Trebst, S., Huse, D.A., Troyer, M., 2006. Feedback-optimized parallel tempering Monte Carlo. J. Stat. Mech. Theory Exp. 2006 (03), P03018.

Kendall, A., Gal, Y., 2017. What uncertainties do we need in bayesian deep learning for computer vision? In: Advances in Neural Information Processing Systems. pp. 5580–5590.

Kingma, D.P., Ba, J., 2014. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

Krogh, A., Hertz, J.A., 1992. A simple weight decay can improve generalization. In: Advances in Neural Information Processing Systems. pp. 950–957.

Lamport, L., 1986. On interprocess communication. Distrib. Comput. 1 (2), 86–101.

Letsinger, J.D., Myers, R.H., Lentner, M., 1996. Response surface methods for bi-randomization structures. J. Qual. Technol. 28 (4), 381–397.

Li, Y., Mascagni, M., Gorin, A., 2009. A decentralized parallel implementation for parallel tempering algorithm. Parallel Comput. 35 (5), 269–283.

Lim, D., Jin, Y., Ong, Y.-S., Sendhoff, B., 2010. Generalizing surrogate-assisted evolutionary computation. IEEE Trans. Evol. Comput. 14 (3), 329–355.

MacKay, D.J., 1995. Probable networks and plausible predictions — a review of practical Bayesian methods for supervised neural networks. Network: Comput. Neural Syst. 6 (3), 469–505.

MacKay, D.J., 1996. Hyperparameters: Optimize, or integrate out? In: Maximum Entropy and Bayesian Methods. Springer, pp. 43–59.

Maraschini, M., Foti, S., 2010. A Monte Carlo multimodal inversion of surface waves. Geophys. J. Int. 182 (3), 1557–1566.

Marinari, E., Parisi, G., 1992. Simulated tempering: a new Monte Carlo scheme. Europhys. Lett. 19 (6), 451.

Metropolis, N., Rosenbluth, A.W., Rosenbluth, M.N., Teller, A.H., Teller, E., 1953. Equation of state calculations by fast computing machines. J. Chem. Phys. 21 (6), 1087–1092.

Mingas, G., Bottolo, L., Bouganis, C.-S., 2017. Particle MCMC algorithms and architectures for accelerating inference in state-space models. Internat. J. Approx. Reason. 83, 413–433.

Montgomery, D.C., Vernon M. Bettencourt, J., 1977. Multiple response surface methods in computer simulation. Simulation 29 (4), 113–121.

Moore, C.J., Chua, A.J., Berry, C.P., Gair, J.R., 2016. Fast methods for training Gaussian processes on large datasets. R. Soc. Open Sci. 3 (5), 160125.

Mosegaard, K., Tarantola, A., 1995. Monte Carlo sampling of solutions to inverse problems. J. Geophys. Res. 100 (B7), 12431–12447.

Neal, R.M., 2012. Bayesian Learning for Neural Networks, Vol. 118. Springer Science & Business Media.

Neal, R.M., et al., 2011. MCMC using hamiltonian dynamics. CRC Press New York, NY.

Ong, Y.S., Nair, P.B., Keane, A.J., 2003. Evolutionary optimization of computationally expensive problems via surrogate modeling. AIAA J. 41 (4), 687–696.

Ong, Y.S., Nair, P., Keane, A., Wong, K., 2005. Surrogate-assisted evolutionary optimization frameworks for high-fidelity engineering design problems. In: Knowledge Incorporation in Evolutionary Computation. Springer, pp. 307–331.

Pall, J., Chandra, R., Azam, D., Salles, T., Webster, J.M., Scalzo, R., Cripps, S., 2020. Bayesreef: A Bayesian inference framework for modelling reef growth in response to environmental change and biological dynamics. Environ. Model. Softw. 104610.

Patriksson, A., van der Spoel, D., 2008. A temperature predictor for parallel tempering simulations. Phys. Chem. Chem. Phys. 10 (15), 2073–2077.

Peherstorfer, B., Willcox, K., Gunzburger, M., 2018. Survey of multifidelity methods in uncertainty propagation, inference, and optimization. SIAM Rev. 60 (3), 550–591.

Raftery, A.E., Lewis, S.M., 1996. Implementing mcmc. In: Markov Chain Monte Carlo in Practice. pp. 115–130.

Rasmussen, C.E., 2004. Gaussian processes in machine learning. In: Advanced Lectures on Machine Learning. Springer, pp. 63–71.

Rathore, N., Chopra, M., de Pablo, J.J., 2005. Optimal allocation of replicas in parallel tempering simulations. J. Chem. Phys. 122 (2), 024111.

van Ravenzwaaij, D., Cassey, P., Brown, S.D., 2016. A simple introduction to Markov Chain Monte–Carlo sampling. Psychon. Bull. Rev. 1–12.

Ray, A., Alumbaugh, D.L., Hoversten, G.M., Key, K., 2013. Robust and accelerated Bayesian inversion of marine controlled-source electromagnetic data using parallel tempering. Geophysics 78 (6), E271–E280.

Ray, A., Kaplan, S., Washbourne, J., Albertin, U., 2018. Low frequency full waveform seismic inversion within a tree based Bayesian framework. Geophys. J. Int. 212 (1), 522–542.

Ray, A., Myer, D., 2019. Bayesian geophysical inversion with trans-dimensional Gaussian process machine learning. Geophys. J. Int. 217 (3), 1706–1726.

Ray, A., Sekar, A., Hoversten, G.M., Albertin, U., 2016. Frequency domain full waveform elastic inversion of marine seismic data from the alba field using a Bayesian trans-dimensional algorithm. Geophys. J. Int. 205 (2), 915–937.

Razavi, S., Tolson, B.A., Burn, D.H., 2012. Review of surrogate modeling in water resources. Water Resour. Res. 48 (7).

Richard, M.D., Lippmann, R.P., 1991. Neural network classifiers estimate Bayesian a posteriori probabilities. Neural Comput. 3 (4), 461–483.

Robert, C., 2014. Machine Learning, A Probabilistic Perspective. Taylor & Francis.

Robert, C.P., Elvira, V., Tawn, N., Wu, C., 2018. Accelerating MCMC algorithms. Wiley Interdiscip. Rev. Comput. Stat. 10 (5), e1435.

Salakhutdinov, R., Mnih, A., Hinton, G., 2007. Restricted Boltzmann machines for collaborative filtering. In: Proceedings of the 24th International Conference on Machine Learning. ACM, pp. 791–798.

Samad, A., Kim, K.-Y., Goel, T., Haftka, R.T., Shyy, W., 2008. Multiple surrogate modeling for axial compressor blade shape optimization. J. Propul. Power 24 (2), 301–310.

Sambridge, M., 2013. A parallel tempering algorithm for probabilistic sampling and multimodal optimization. Geophys. J. Int. 196 (1), 357–374.

Sambridge, M., 2014. A parallel tempering algorithm for probabilistic sampling and multimodal optimization. Geophys. J. Int. 196 (1), 357–374.

Scalzo, R., Kohn, D., Olierook, H., Houseman, G., Chandra, R., Girolami, M., Cripps, S., 2019. Efficiency and robustness in Monte Carlo sampling for 3-D geophysical inversions with Obsidian v0.1.2: setting up for success. Geoscientific Model Development 12 (7), 2941–2960.

Schmidhuber, J., 2015. Deep learning in neural networks: An overview. Neural Netw. 61, 85–117.

Sen, M.K., Stoffa, P.L., 1996. Bayesian inference, Gibbs' sampler and uncertainty estimation in geophysical inversion. Geophys. Prospect. 44 (2), 313–350.

Shalev-Shwartz, S., et al., 2012. Online learning and online convex optimization. Found. Trends® Mach. Learn. 4 (2), 107–194.

Swendsen, R.H., Wang, J.-S., 1986. Replica Monte Carlo simulation of spin-glasses. Phys. Rev. Lett. 57 (21), 2607.

Swendsen, R.H., Wang, J.-S., 1987. Nonuniversal critical dynamics in Monte Carlo simulations. Phys. Rev. Lett. 58 (2), 86.

Tandjiria, V., Teh, C.I., Low, B.K., 2000. Reliability analysis of laterally loaded piles using response surface methods. Struct. Saf. 22 (4), 335–355.

Tarantola, A., Valette, B., et al., 1982. Inverse problems = quest for information. J. Geophys. 50 (1), 159–170.

Vousden, W., Farr, W.M., Mandel, I., 2015. Dynamic temperature selection for parallel tempering in Markov chain Monte Carlo simulations. Mon. Not. R. Astron. Soc. 455 (2), 1919–1937.

Wang, H., Zeng, Y., Yu, X., Li, G., Li, E., 2016. Surrogate-assisted Bayesian inference inverse material identification method and application to advanced high strength steel. Inverse Probl. Sci. Eng. 24 (7), 1133–1161.

Welling, M., Teh, Y.W., 2011. Bayesian learning via stochastic gradient Langevin dynamics. In: Proceedings of the 28th International Conference on Machine Learning, ICML-11. pp. 681–688.

White, H., 1982. Maximum likelihood estimation of misspecified models. Econometrica 1–25.

Zhou, Z., Ong, Y.S., Nair, P.B., Keane, A.J., Lum, K.Y., 2007. Combining global and local surrogate models to accelerate evolutionary optimization. IEEE Trans. Syst. Man Cybern. C 37 (1), 66–76.