Neurocomputing 359 (2019) 315–326

Contents lists available at ScienceDirect

Neurocomputing

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

Langevin-gradient parallel tempering for Bayesian neural learning

Rohitash Chandra [a,b,*], Konark Jain [a,c], Ratneel V. Deo [a,d], Sally Cripps [a,e]

[a] Centre for Translational Data Science, The University of Sydney, NSW 2006, Australia
[b] School of Geosciences, The University of Sydney, NSW 2006, Australia
[c] Department of Electronics and Electrical Engineering, Indian Institute of Technology, Guwahati, Assam, India
[d] School of Computing, Information and Mathematical Sciences, University of the South Pacific, Suva, Fiji
[e] School of Mathematics and Statistics, The University of Sydney, NSW 2006, Australia

ARTICLE INFO

Article history:
Received 10 November 2018
Revised 6 February 2019
Accepted 29 May 2019
Available online 4 June 2019

Communicated by Dr. Nianyin Zeng

Keywords:
Bayesian neural networks
Parallel computing
MCMC methods
Parallel tempering

ABSTRACT

Bayesian inference provides a rigorous approach for neural learning with knowledge representation via the posterior distribution that accounts for uncertainty quantification. Markov Chain Monte Carlo (MCMC) methods typically implement Bayesian inference by sampling from the posterior distribution. This not only provides point estimates of the weights, but the ability to propagate and quantify uncertainty in decision making. However, these techniques face challenges in convergence and scalability, particularly in settings with large datasets and neural network architectures. This paper addresses these challenges in two ways. First, parallel tempering MCMC sampling method is used to explore multiple modes of the posterior distribution and implemented in multi-core computing architecture. Second, we make within-chain sampling scheme more efficient by using Langevin gradient information for creating Metropolis–Hastings proposal distributions. We demonstrate the techniques using time series prediction and pattern classification applications. The results show that the method not only improves the computational time, but provides better decision making capabilities when compared to related methods.

1. Introduction

Although backpropagation neural networks have gained immense attention and success for a wide range of problems [1], they face a number of challenges such as finding suitable values of hyper-parameters [2–4] and appropriate network topology [5,6]. These challenges remain when it comes to different neural network architectures, in particular, deep neural networks architectures which have a large number of parameters [7]. Another limitation of current techniques is the lack of uncertainty quantification in decision making or prediction. Bayesian neural networks can address most of these shortfalls. Bayesian methods account for the uncertainty in prediction and decision making via the posterior distribution. Note that the posterior is the conditional probability determined after taking into account the prior distribution and the relevant evidence or data via sampling methods. Bayesian methods can account for the uncertainty in parameters (weights) and topology by marginalisation over the predictive posterior distribution, [8,9]. In other words, as opposed to conventional neural networks, Bayesian neural learning use probability distributions to represent the weights [10,11], rather than featuring single-point estimates by gradient-based learning methods. Markov Chain Monte Carlo (MCMC) methods implement Bayesian inference by sampling from the posterior distribution [12,13]; whereby, a Markov chain is constructed after a number of steps such that the desired distribution becomes the equilibrium distribution [14,15]. In other words, MCMC methods provide numerical approximations of multi-dimensional integrals [16]. Examples of MCMC methods include the Laplace approximation [8], Hamiltonian Monte Carlo [17], expectation propagation [18] and variational inference [19].

Despite the advantages, Bayesian neural networks face a number of challenges that include efficient proposal distributions for convergence, scalability and computational efficiency for larger network architectures and datasets. Hence, a number of attempts have been proposed for enhancing Bayesian neural learning with optimisation strategies to form proposals that feature gradient information [20–23]. Moreover, in the case of deep learning where thousands to millions of parameters (weights) are involved, approximate Bayesian learning methods have emerged. Srivastava et al. [24] presented “dropouts” for deep learning where the key idea was to randomly drop neurons along with their connections during training to prevent over-fitting. Gal et al. used the concept of dropouts in a Bayesian framework for uncertainty quantification

* Corresponding author at: Centre for Translational Data Science, The University of Sydney, NSW 2006, Australia.
E-mail address: rohitash.chandra@sydney.edu.au (R. Chandra).

https://doi.org/10.1016/j.neucom.2019.05.082
0925-2312/

in model parameters that feature weights and network topology for deep learning [25]. This was further extended for computer vision problems [26]. Although promising for uncertainty quantification, it could be argued that the approach does not adequately approximate MCMC based sampling methods typically used for Bayesian inference. Moreover, recently other neural network models have also focused on uncertainty quantification such as delayed chaotic neural networks [27] and Takagi–Sugeno fuzzy delayed neural networks [28].

Parallel tempering [29,30] is an MCMC method that features multiple replica Markov chains that provide global and local exploration which makes them suitable for irregular and multi-modal distributions [31,32]. Parallel tempering carries out an exchange of parameters in neighbouring replicas during sampling that is helpful in escaping local minima. Another feature of parallel tempering is their feasibility of implementation in multi-core or parallel computing architectures. In multi-core implementation, factors such as inter-process communications need to be considered during the exchange between the neighbouring replicas [33]. This needs to be accounted for when designing parallel tempering for neural networks.

In the literature, parallel tempering has been used for restricted Boltzmann machines [34,35]. Desjardins et al. [36] showed that parallel tempering is more effective than Gibbs sampling for restricted Boltzmann machines as they lead to faster and better convergence. Brakel et al. [37] extended the method by featuring efficient exchange of information among the replicas and implementing estimation of gradients by averaging over different replicas. Furthermore, Fischer et al. [38] gave an analysis on the bounds of convergence of parallel tempering for restricted Boltzmann machines. They showed the significance of geometric spacing of temperature values of the replicas against linear spacing.

The adoption of Bayesian techniques in estimating neural networks has been slow mainly because of the challenges in large datasets and the limitations of MCMC methods for large scale inference. Parallel tempering overcomes many of these challenges. The within-chain proposals do not necessarily require gradient information, which avoids limitations of gradient-based learning [39]. Although random-walk proposals are typically used for parallel tempering, it is worthwhile to explore other proposals such as those based on Langevin gradients that are used during sampling [23]. This approach has significantly improved Bayesian neural learning for time series prediction. The synergy of Langevin gradients with parallel tempering can alleviate major weaknesses in MCMC methods. Furthermore, in the past, machine learning methods have taken advantage of distributed and parallel computing [40–42] which motivates their usage in Bayesian neural learning.

In this paper, we present a multi-core parallel tempering approach for Bayesian neural networks that takes advantage of parallel computing for time series prediction and pattern classification problems. We use Gaussian likelihood for prediction and multinomial likelihood for pattern classification problems, respectively. We also compare the posterior distributions and the performance in terms of prediction and classification for the selected problems. Furthermore, we investigate the effect on computational time and convergence given the use of Langevin gradients for proposals in parallel tempering. The major contribution of the paper is in the development of parallel tempering for Bayesian neural learning based on parallel computing.

The rest of the paper is given as follows. Section 2 gives background on parallel tempering and Bayesian neural networks. Section 3 presents the proposed method while Section 4 gives the design of experiments and results. Section 5 provides a discussion of the results with implications, and Section 6 conclusions the paper with directions of future work.

2. Background

2.1. Parallel tempering

Parallel tempering (also known as replica exchange or the Multi-Markov Chain method) [32,43,44] has been motivated by thermodynamics of physical systems [31,43]. Overall, in parallel tempering, multiple MCMC chains (known as replicas) are executed at different *temperature* values defined by the temperature ladder. The temperature ladder is used for altering each replica's likelihood function which enables different level of exploration capabilities. Furthermore, typically the chain at the neighbouring replicas are swapped at certain intervals depending on the Metropolis–Hastings acceptance criterion. Typically, gradient-free proposals within the replica's are used for proposals for exploring multi-modal and discontinuous posteriors [45,46]. Determining the optimal temperature ladder for the replicas has been a challenge that attracted some attention in the literature. Rathore et al. [47] studied the efficiency of parallel tempering in various problems regarding protein simulations and presented an approach for dynamic allocation of the temperatures. Katzgraber et al. [48] proposed systematic optimisation of temperature sets using an adaptive feedback method that minimise the round-trip times between the lowest and highest temperature to increase efficiency. Bittner et al. [49] showed that by adapting the number of sweeps between replica exchanges, the average round-trip time can be significantly decreased to achieve close to 50% swap rate among the replicas. This increases the efficiency of the parallel tempering algorithm. Furthermore, Patriksson and Spoel [31] presented an approach to predict a set of temperature values in the application of molecular biology. All these techniques have been applied to specific settings, none of which use a neural network architecture.

In the canonical implementation, the exchange is limited to neighbouring replicas conditioned by a probability that is determined during sampling. Calvo proposed an alternative technique [50] where the swap probability is determined apriori, and then one swap is proposed. Fielding et al. [51] considered replacing the original target posterior distribution with the Gaussian process approximation which requires less computational requirement. The authors replaced the true target distribution with the approximation in the high-temperature chains while retaining the true target in the lowest temperature chain. Furthermore, Liu et al. proposed an approach to reduce the number of replicas by adapting acceptance probability for exchange for computational efficiency [52]. Although the approaches discussed for adapting temperature and replica exchange are promising, there is not much work that evaluates them on benchmark problems for a fair comparison.

A number of challenges exist when considering multi-core implementations since parallel tempering features exchange or transition between neighbouring replicas. One needs to consider effective strategies that take into account interprocess communication in such systems [53]. In order to address this, Li et al. presented a decentralised implementation that eliminated global synchronisation and reduces the overhead caused by interprocess communication in exchange of solutions between the chains [53]. Karimi et al. implemented parallel tempering in a distributed volunteer computing network where public computers were used with the help of multi-threading and graphics processing units (GPUs) [54]. Furthermore, Mingas and Bouganis presented an implementation with field programmable gate array (FPGA) that showed much better performance than multi-core and GPU implementations [55].

Parallel tempering has been applied to a number of fields of which some are discussed as follows. Musiani and Giorgetti [56] presented a review of computational techniques used for simulations of protein aggregation where parallel tempering was presented as a widely used technique. Xie et al. [57] simulated

lysozome orientations on charged surfaces using an adaption of the parallel tempering. Tharrington and Jordon [58] used parallel tempering to characterise the finite temperature behaviour of H_2O clusters. Littenberg and Neil used parallel tempering for detection problem in gravitational wave astronomy [59]. Moreover, Reid et al. used parallel tempering with multi-core implementation for inversion problem for exploration of Earth's resources [60].

2.2. Bayesian neural networks

In neural network models, the priors can be informative about the distribution of the weights and biases given the network architecture and expert knowledge [61]. For example, it is well known that allowing large values of the weights will put more probability mass on the outcome being either zero or one [62]. For this reason, the weights are often restricted to lie within the range of $[-5, 5]$, which could be implemented using a uniform prior distribution [9,11]. Examples of priors informed by prior knowledge include [10,17,63].

The limitations regarding convergence and scalability of MCMC sampling methods has impeded the use of Bayesian methods in neural networks. A number of techniques have been applied to address this issue by incorporating approaches from the optimisation literature. Gradient-based methods such as Hamiltonian MCMC [64] and Langevin dynamics [65] have significantly improved the rate of convergence of MCMC chains. Furthermore, Chen et al. used simulated annealing to improve stochastic gradient MCMC algorithm for deep neural networks [66].

In the time series prediction literature, Liang et al. presented an MCMC algorithm for neural networks for selected time series problems [67], while Chandra et al. presented Langevin gradient Bayesian neural networks for prediction [23]. For short-term time series forecasting, Bayesian techniques have been used for controlling model complexity and selecting inputs in neural networks [68]. Bayesian recurrent neural networks [69] have been very useful for time series prediction. Evolutionary algorithms have been combined with MCMC methods for Bayesian neural networks for time series forecasting [22].

In classification problems, initial work was done by Wan who provided a Bayesian interpretation for classification with neural networks [70]. Moving on, a number of successful applications of Bayesian neural networks for classification exist, such as Internet traffic classification [63].

Considering other networks architectures, Hinton et al. [35] used complementary priors to derive a fast greedy algorithm for deep belief networks to form an undirected associative memory with application to form a generative model of the joint distribution of handwritten digit images and their labels. Furthermore, parallel tempering has been used for Gaussian Bernoulli Restricted Boltzmann Machine (RBM) [71] and prior to this, Cho et al. [72] demonstrated the efficiency of parallel tempering in RBMs. Furthermore, Desjardins et al. utilised parallel tempering for maximum likelihood training of RBMs [73] and later used it for deep learning using RBMs [74]. Thus, parallel tempering has been vital in the development of one of the fundamental building blocks of deep learning RBMs.

3. Methodology

In this section, we provide the details for using multi-core parallel tempering for time series prediction and pattern classification problems. The multi-core parallel tempering algorithm features two implementations; 1.) random-walk proposals, and 2.) Langevin-gradient proposals. We first present the foundations followed by the details of the implementations.

3.1. Model and priors for time series prediction

Let y_t denote a univariate time series. We assume that y_t is generated from a signal plus noise model; where, the signal is a neural network and the noise is assumed to be Gaussian with variance τ^2, so that

$$y_t = f(\mathbf{x}_t) + \epsilon_t, \text{ for } t = 1, 2, \ldots, n \tag{1}$$

where, $f(\mathbf{x}_t) = E(y_t|\mathbf{x}_t)$ is an unknown function, and $\mathbf{x}_t = (y_{t-1}, \ldots, y_{t-D})$ is a vector of lagged values of y_t, and ϵ_t is the noise with $\epsilon_t \sim \mathcal{N}(0, \tau^2)$ $\forall t$.

We transform the univariate time series into a state-space vector through Taken's theorem [75] which is governed by the embedding dimension (D) and time-lag (T). From Taken's Theorem, we define

$$\mathcal{A}_{D,T} = \{t; t > D, \mod (t - (D+1), T) = 0\}. \tag{2}$$

Let $\mathbf{y}_{\mathcal{A}_{D,T}}$ to be the set of y_t's for which $t \in \mathcal{A}_{D,T}$, then, $\forall t \in \mathcal{A}_{D,T}$. In this representation, the embedding dimension D is equal to the number of inputs in a feedforward neural network, which we denote by I. The expected value of y_t given $\mathbf{x_t} = (\mathbf{y_1}, \ldots, \mathbf{y_{t-1}})$ is given by

$$f(\mathbf{x}_t) = g\Big(\delta_o + \sum_{h=1}^{H} v_h \times g\Big(\delta_h + \sum_{i=1}^{I} w_{ih} x_{t,i}\Big)\Big), \tag{3}$$

where, δ_o and δ_h are the biases for the output and hidden h layers, respectively; v_h is the weight which maps the hidden layer h to the output, and w_{ih} is the weight which maps $x_{t,i}$ to the hidden layer h. g is the activation function which is a sigmoid function for the hidden and output layer units of the neural network. The set of parameters needed to define the likelihood function contains; the variance of the signal, τ^2, given in Eq. 1; the weights of the input to hidden layer $\tilde{\mathbf{w}}$; the weights of the hidden to output layer, $\tilde{\mathbf{v}}$; the bias to the hidden layer $\boldsymbol{\delta}_h$, and the output layer $\boldsymbol{\delta}_o$. There is a total of $L = (I * H + H + (O-1) * H + O - 1) + 1$, parameters; where I is the number of inputs, H is the number of hidden layers, O is the number of classes of the output variable. The likelihood function is the multivariate normal probability density function and is given by

$$p(\mathbf{y}_{\mathcal{A}_{\mathbf{D,T}}}|\boldsymbol{\theta}) = -\frac{1}{(2\pi\tau^2)^{n/2}} \times \exp\left(-\frac{1}{2\tau^2}\sum_{t\in\mathcal{A}_{D,T}}(y_t - E(y_t|\mathbf{x}_t))^2\right) \tag{4}$$

where $E(y_t|\mathbf{x}_t)$ is given by Eq. (3).

We assume that the elements of $\boldsymbol{\theta}$ are independent *apriori*. In addition, we assume apriori that the weights $\mathbf{w}$, $\mathbf{v}$, and biases $\boldsymbol{\delta}$, have a normal distribution with zero mean and variance σ^2. Given nu_1 and nu_2 are user chosen constants, our prior is now

$$p(\boldsymbol{\theta}) \propto \frac{1}{(2\pi\sigma^2)^{L/2}} \times \exp\left\{-\frac{1}{2\sigma^2}\left(\sum_{h=1}^{H}\sum_{d=1}^{D} w_{dh}^2 + \sum_{k=1}^{K}\sum_{h=1}^{H}(\delta_{hk}^2 + v_{hk}^2) + \delta_o^2\right)\right\} \times \tau^{2(1+\nu_1)} \exp\left(\frac{-\nu_2}{\tau^2}\right) \tag{5}$$

3.2. Model and priors for classification problems

When the dataset is discrete such as in a classification problem, it is inappropriate to model the data as Gaussian. Hence for discrete data with K possible classes, we assume that the data $\mathbf{y} = (y_1, \ldots, y_n)$ are generated from a multinomial distribution with parameter vector $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_K)$ where $\sum_{k=1}^{K} \pi_k = 1$. In order to

define the likelihood, we introduce a set of indicator variables $z_{i,k}$ where

$$z_{i,k} = \begin{cases} 1, & \text{if } y_i = k \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

for $i = 1, \ldots, n$ and $k = 1, \ldots, K$. The likelihood function is then

$$p(\mathbf{y}|\boldsymbol{\pi}) = \prod_{i=1}^{n}\prod_{k=1}^{K} \pi_k^{z_{i,k}} \tag{7}$$

for classes $k = 1, \ldots, K$ where π_k, the output of the neural network, is the probability that the data are generated by category k. The dependence between this probability and the input features $\mathbf{x}$ is modelled as a multinomial logit function

$$\pi_k = \frac{\exp(f(x_k))}{\sum_{j=1}^{K} \exp(f(x_j))} \tag{8}$$

where, $f(x)$ is given by Eq. 3 and the priors for the weights and biases are given by Eq. 7.

3.3. Estimation via Metropolis–Hastings parallel tempering

In parallel tempering, each replica corresponds to a predefined temperature ladder which governs the invariant distribution where the higher temperature value gives more chance in accepting weaker proposals [44,76]. Hence, parallel tempering has the feature to sample multi-modal posterior distribution [77]. Given M replicas of an ensemble, defined by multiple temperature levels, the state of the ensemble is specified by $\Theta = (\theta_1, \ldots, \theta_M)$, where θ_i is the replica at temperature level T_i. The samples of θ from the posterior distribution can be obtained by proposing values of θ^p, from some known distribution $q(\theta)$. The chain moves to this proposed value of θ^p with a probability α or remains at its current location θ^c, where α is chosen to ensure that the chain is reversible and has stationary distribution $p(\theta|\mathbf{D})$.

The development of transitions kernels or proposals which efficiently explore posterior distributions is the subject of much research. Random-walk proposals feature a small amount of Gaussian noise to the current value of the chain θ^c. The Markov chains in the parallel replicas have stationary distributions which are equal to (up to a proportionality constant) $p(\theta|\mathbf{D})^{\beta}$; where $\beta \in [0, 1]$, with $\beta = 0$ corresponding to a stationary distribution which is uniform, and $\beta = 1$ corresponding to a stationary distribution which is the posterior. The replicas with smaller values of β can explore a larger region of distribution, while those with higher values of β typically explore local regions. Communication between the parallel replicas is essential for the efficient exploration of the posterior distribution. This can be done by considering the chain and the parameters as part of the space to be explored. Suppose there are R replicas indexed by m with corresponding stationary distributions, $p_m(\theta|\mathbf{D}) = p(\theta|\mathbf{D})^{\beta_r}$, for $m = 1, \ldots, R$, with $\beta_1 = 1$ and $\beta_R < \beta_{R-1} < \ldots, \beta_1$, then the pair (m, θ) are jointly proposed and accepted/rejected according to the Metropolis–Hastings criterion. The stationary distribution of this sampler is proportional to $p(\theta|\mathbf{D})^{\beta_m} p(m)$. The quantity $p(m)$ must be chosen by the user and is referred to as a *pseudoprior* [78].

3.4. Langevin gradient-based proposals

We utilise Langevin-gradients to update the parameters at each iteration rather than only by using random-walk [79]. The gradients are calculated as follows

$$\boldsymbol{\theta}^p \sim \mathcal{N}(\bar{\boldsymbol{\theta}}^{[k]}, \Sigma_\theta), \text{ where} \tag{9}$$

$$\bar{\boldsymbol{\theta}}^{[k]} = \boldsymbol{\theta}^{[k]} + r \times \nabla E_{\mathbf{y}_{\mathcal{A}_{\mathbf{D},\mathbf{T}}}}[\boldsymbol{\theta}^{[k]}], \tag{10}$$

$$E_{\mathbf{y}_{\mathcal{A}_{\mathbf{D},\mathbf{T}}}}[\boldsymbol{\theta}^{[k]}] = \sum_{t \in \mathcal{A}_{D,T}} (y_t - f(\mathbf{x}_t)^{[k]})^2, \tag{11}$$

$$\nabla E_{\mathbf{y}_{\mathcal{A}_{\mathbf{D},\mathbf{T}}}}[\boldsymbol{\theta}^{[k]}] = \left(\frac{\partial E}{\partial \theta_1}, \ldots, \frac{\partial E}{\partial \theta_L} \right) \tag{12}$$

where r is the learning rate, $\Sigma_\theta = \sigma_\theta^2 I_L$, and I_L is the $L \times L$ identity matrix. The newly proposed value of $\boldsymbol{\theta}^p$ consists of 2 parts

1. A gradient descent based weight update given by Eqs. 9–12.
2. Add an amount of noise, from $\mathcal{N}(0, \Sigma_\theta)$.

We note that Langevin-gradient proposal is incorporated in Algorithm 1 as an alternative to random-walk proposal depending on the choice of the user.

3.5. Algorithm

This parallel tempering algorithm for Bayesian neural learning is given in Algorithm 1. At first, the algorithm initialises the replicas by following the temperature ladder which is in a geometric progression [80]. Along with this, the hyper-parameters such as the maximum number of samples, number of replicas, swap interval and the type of proposals (random-walk or Langevin-gradients) is chosen. We note that the algorithm uses parallel tempering for global exploration and transforms to canonical MCMC using parallel computing for local exploration. The major change in the transformation is the temperature ladder; when in local exploration, all the replicas are set to temperature of 1. The local exploration also features swap of neighbouring replica. Hence, one needs to set the percentage of samples for global exploration phase beforehand. The sampling begins by executing each of the replicas in parallel (Step 1). Each replica θ is updated when the respective proposal is accepted using the Metropolis–Hasting acceptance criterion given by Step 1.3 of Algorithm 1. In case the proposal is accepted, the proposal becomes part of the posterior distribution, otherwise the last accepted sample is added to the posterior distribution as shown in Step 1.3. This procedure is repeated until the replica swap interval is reached. When all the replicas have sampled until the swap interval, the algorithm evaluates if the neighbouring replicas need to be swapped by using the Metropolis–Hastings acceptance criterion as done for within replica proposals (Step 2).

The multi-core implementation takes into account operating system concepts such as inter-process communication when considering neighbouring replica exchange [33] as shown in Fig. 1. The main process manages the replicas and enables them to exchange the neighbouring replicas given the swap interval and probability of exchange is satisfied. Each replica is allocated a fixed sampling time controlled by the number of iterations. The main process waits for all replicas to sample until the swap interval which determines replica exchange based on likelihoods of adjacent replicas. The main process notifies the replicas, post exchange, to resume sampling with latest configuration. The process continues until the maximum sampling time is reached.

We utilise multi-processing software development packages for the resources to have efficient inter-process communication [33] taking into account the swapping of replicas at certain intervals in the iterations. We implement our parallel tempering multi-core architecture using the Python multiprocessing library [81][1].

[1] Langevin-gradient parallel tempering for Bayesian neural learning: https://github.com/sydney-machine-learning/parallel-tempering-neural-net.

Algorithm 1: Metropolis-Hastings Parallel Tempering (MHPT).

Result: Draw samples from $p(\theta|\mathbf{D})$
1. Set maximum number of samples ($Max_{samples}$), swap interval ($Swap_{int}$), and number of replicas (R)
2. Initialise $\theta = \theta^{[0]}$, and $m = m^{[0]}$
3. Set the current value of θ, to $\theta^c = \theta^{[0]}$ and m to $m^c = m^{[0]}$
4. Choose type of proposal (Random-walk (RW) or Langevin-gradients (LG)
5. Choose Langevin-gradient frequency (LG-freq)
6. Set percentage of samples for global exploration phase
while $Max_{samples}$ **do**
  for $m = 1, \ldots, R$ **do**
    for $k = 1, \ldots, Swap_{int}$ **do**
      *Local/Global exploration phase
      if *global is true* **then**
        $R_{temp} = 1/Temp_m$
      else
        $R_{temp} = 1$
      end
      Step 1: Replica sampling
      1.1 Propose new sample (solution)
      Draw $l \sim U[0, 1]$
      if *(LG is true) and* $(l < LG - freq)$ **then**
        Conditional on m^c propose a new value of θ using Langevin-gradients, $\theta^p \sim q(\theta|\theta^c, m^c)$, where $q(\theta|\theta^c, m^c)$ is given in Equations 9–12
      else
        Conditional on m^c propose a new value of θ, using Random-Walk
      end
      1.2 Compute acceptance probability

$$\alpha = \min\left(1, \frac{p(\theta^p|\mathbf{D}, m^c)^{\beta_m^c} q(\theta^c|\theta^p, m^c)}{p(\theta^c|\mathbf{D}, m^c)^{\beta_m^c} q(\theta^p|\theta^c, m^c)}\right)$$

      1.3 Acceptance criterionDraw $u \sim U[0, 1]$
      if $u < \alpha$ **then**
        $\theta^{[k]} = \theta^p$
      else
        $\theta^{[k]} = \theta^c$
      end
    end
    Step 2: Replica transition
    2.1 Propose a new value of replica swap transition m^p, from $q(m^p|m^c)$.
    2.2 Compute acceptance probability

$$\alpha = \min\left(1, \frac{p(\theta^{[k]}|\mathbf{D})^{\beta_{m^p}} p(m^p) q(m^c|m^p)}{p(\theta^{[k]}|\mathbf{D})^{\beta_{m^c}} p(m^c) q(m^p|m^c)}\right)$$

    2.3 Neighbouring replica swap
    Draw $u \sim U[0, 1]$
    if $u < \alpha$ **then**
      $m^{[k]} = m^p$
    else
      $m^{[k]} = m^c$
    end
  end
end

Table 1
Classification dataset description.

Dataset	Instances	Attributes	Classes	Hidden units
Iris	150	4	3	12
Breast cancer	569	9	2	12
Ionosphere	351	34	2	50
Bank-Market	11162	20	2	50
Pen-Digit	10992	16	10	30
Chess	28056	6	18	25

4. Experiments and results

This section presents the experimental evaluation of multi-core parallel tempering for Bayesian neural learning given selected time series prediction and classification problems.

4.1. Experimental design

Parallel tempering is executed in the first stage that compromises of 60% of the sampling time. In the second stage, the framework changes to canonical MCMC where the temperature value of the replicas become 1. Moreover, the first 50 percent of samples are discarded as burn-in period which is a standard procedure in sampling methods used for Bayesian inference. Furthermore, the Langevin-gradient proposals are applied with a probability of 0.5 and the effect of the learning rate used for the Langevin-gradient are evaluated in the experiments. The experiments are designed as follows

- Evaluate the effect of maximum temperature for a selected time series problem.
- Evaluate the effect of swap interval for a selected time series problem.
- Evaluate the effect of Langevin-gradient rate for a selected time series problem.
- Compare PT-RW and PT-LG for time series prediction problems.
- Compare PT-RW and PT-LG for pattern classification problems.

We use one hidden layer feedforward neural network with $h = 5$ hidden units for the respective time series prediction problems. In the case of pattern classification, the number of hidden units is given in Table 1. The number of hidden units for the respective problem has been determined in trial experiments. The geometric temperature ladder requires a maximum temperature value which needs to be selected beforehand. We evaluate the effect of the hyper-parameters in parallel tempering, which includes the maximum geometric temperature spacing and swap interval. Note that all experiments used 10 replicas in the multi-core implementation.

In random-walk proposals, we draw and add a Gaussian noise to the weights and biases of the network from a standard normal distribution, with mean of 0 and standard deviation of 0.025. The random-walk step becomes the standard deviation when we draw the weights and η from the normal distribution centered around 0. The parameters of the priors (Eq. 5) were set as $\sigma^2 = 25, \nu_1 = 0$ and $\nu_2 = 0$.

4.1.1. Time series problems

The benchmark problems are Sunspot, Lazer, Mackey-Glass, Lorenz, Henon and Rossler time series. Takens' embedding theorem is applied for data reconstruction with dimension, $D = 4$ and time lag, $T = 2$; these values are also used in previous works [82,83]. All the problems used one thousand data points of the time series from which the first 60% was used for training and remaining for testing. The prediction performance is measured by root mean

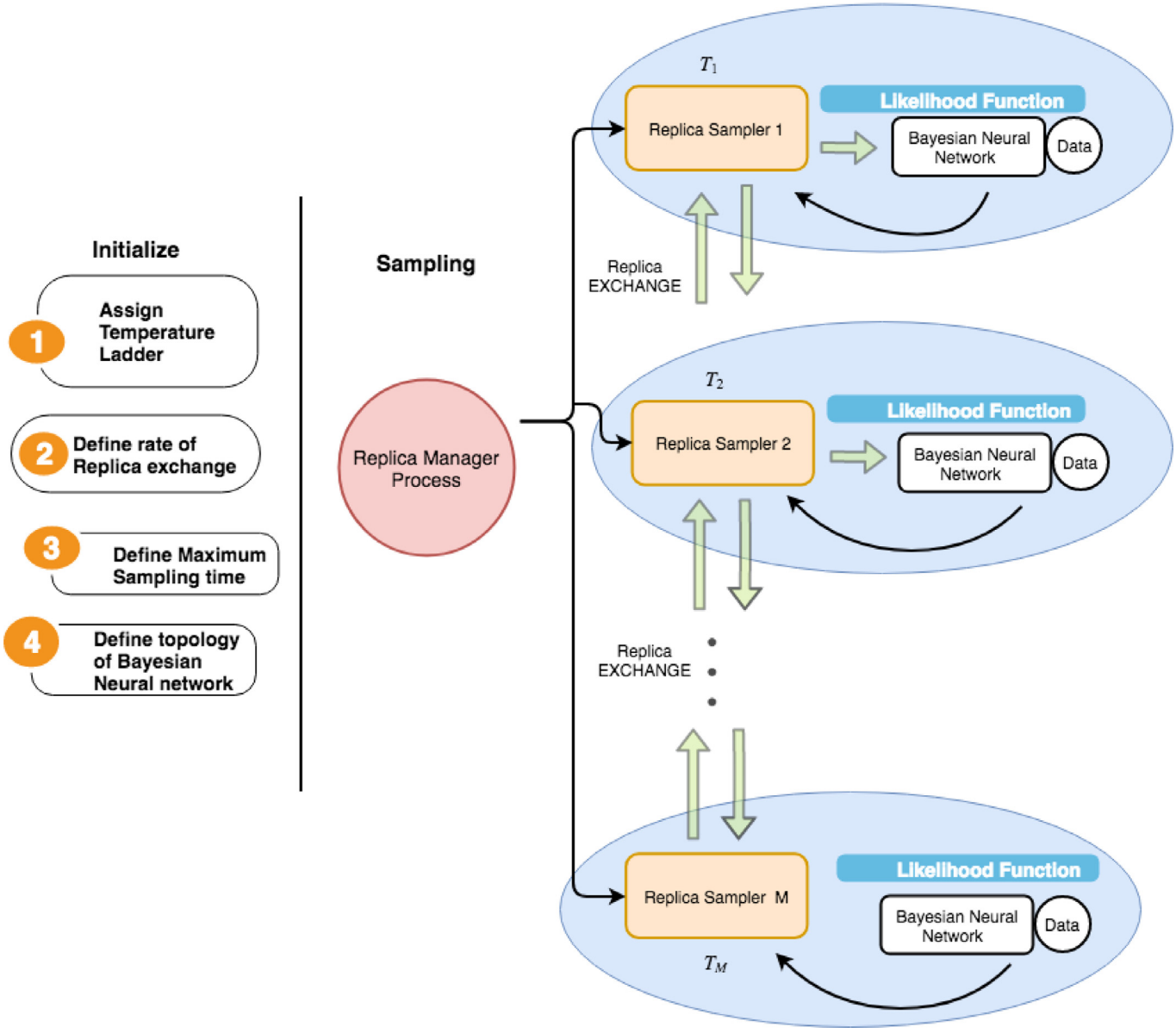


Fig. 1. An overview of the different replicas that are executed on a parallel computing architecture. Note that the *Replica Manager Process* controls the given replicas and enables them to exchange the neighbouring replicas given the swap time and probability of exchange is satisfied. Each replica is executed on a separate processing unit.

squared error as follows

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

where, y_i and $\hat{y}_i$ are the actual value and the predicted value respectively. N is the total length of the data. The maximum sampling time is set to 100,000 samples for the respective time series problems.

4.1.2. Pattern classification problems

We select six pattern classification problems from University of California Irvine machine learning repository [84]. The problems were selected according to their size and complexity in terms of the number of features and instances. We first consider smaller datasets with relatively low number of instances and features (Iris and Cancer). Then we consider dataset with a smaller number of instances but a higher number of features (Ionosphere). Moreover, we use three relatively large datasets with relatively higher number of instances and features. One of these datasets features a lower number of classes (Bank-Market) and others a relatively higher number of classes (Pen-Digit and Chess) as shown in Table 1.

4.2. Results: time series prediction

We first show results for the effect of different parameter values for the selected time series problems. Table 2 shows the performance of the PT-RW method for Lazer problem given different values of the maximum temperature of the geometric temperature ladder given fixed swap interval of 100 samples. We note that the maximum temperature has a direct effect on the swap probability. The results show that temperature value of 4 gives the best results in prediction denoted by lowest RMSE for training and test performance considering both the train and test datasets. It seems that the problem requires low swap rate as it needs to concentrate more on exploiting the local region. Moving on, Table 3 shows the results for the changes in the swap interval that determines

Table 2
Evaluate maximum geometric temperature.

Max. temp.	Train (mean, best, std)	Test (mean, best, std)	Swap per.	Accept per.	Time (min.)
2	0.0639 0.0208 0.0292	0.0543 0.0184 0.0244	31.97	30.26	3.65
4	0.0599 0.0206 0.0299	0.0523 0.0176 0.0248	38.21	33.74	3.94
6	0.0656 0.0220 0.0299	0.0576 0.0210 0.0277	48.59	36.01	4.70
8	0.0692 0.0277 0.0311	0.0584 0.0209 0.0261	44.97	37.72	3.79
10	0.0635 0.0250 0.0293	0.0547 0.0212 0.0251	43.97	38.14	3.73

Table 3
Evaluate swap interval.

Swap interval	Train (mean, best, std)	Test (mean, best, std)	Swap per.	Accept per.	Time (min.)
100	0.0628 0.0233 0.0286	0.0554 0.0213 0.0264	31.33	34.19	3.66
200	0.0608 0.0226 0.0308	0.0521 0.0194 0.0262	40.65	33.41	3.49
300	0.0657 0.0190 0.0328	0.0562 0.0173 0.0263	39.01	34.38	3.47
400	0.0648 0.0238 0.0281	0.0552 0.0214 0.0230	40.11	34.35	3.68
500	0.0676 0.0220 0.0310	0.0577 0.0190 0.0279	50.00	34.82	3.83
600	0.0621 0.0255 0.0266	0.0549 0.0209 0.0247	50.00	32.62	3.54
700	0.0614 0.0197 0.0320	0.0522 0.0169 0.0280	47.36	34.10	3.52
800	0.0739 0.0217 0.0291	0.0666 0.0214 0.0264	31.57	36.08	3.52

Table 4
Evaluate LG rate.

LG-frequency	Train (mean, std, best)	Test (mean, std, best)	Swap per.	Accept per.	Time (min.)
0	0.0628 0.0233 0.0286	0.0554 0.0213 0.0264	31.33	34.19	3.66
0.1	0.0539 0.0224 0.0286	0.0500 0.0238 0.0273	34.20	30.50	6.06
0.2	0.0361 0.0099 0.0261	0.0331 0.0083 0.0239	45.66	26.33	7.33
0.3	0.0346 0.0096 0.0226	0.0318 0.0077 0.0233	39.42	23.84	9.10
0.4	0.0365 0.0113 0.0249	0.0340 0.0107 0.0232	45.57	22.74	10.12
0.5	0.0451 0.0188 0.0266	0.0402 0.0177 0.0248	32.88	21.19	11.6
0.6	0.0464 0.0247 0.0245	0.0433 0.0205 0.0234	52.32	19.65	13.22
0.7	0.0396 0.0232 0.0234	0.0373 0.0214 0.0202	48.65	18.74	15.23
0.8	0.0350 0.0195 0.0230	0.0346 0.0168 0.0231	45.13	16.66	17.63

how often to calculate the swap probability for swapping given the maximum geometric temperature of 4. Note that during swapping, the master process freezes all the replica sampling process and calculates the swap probability to swap between neighbouring replicas. We note that there is not a significant difference in the performance given the range of swap interval considered. Howsoever, the results show that the swap interval of 200 is best for time series problems.

Furthermore, Table 4 shows the effects of the Langevin-gradient frequency (LG-frequency) where the swap interval of 100 samples was used with a learning rate of 0.1. We notice that the higher rate of using gradients review more time. This is due to the computational cost of calculating gradients for the given proposals. The LG-frequency of [0.2–0.4] shows the best results in prediction with a moderate computational load. These values may be slightly different for the different problems.

Considering the hyper-parameters (maximum temperature, swap interval and LG-frequency) from previous results, Table 5 provides results for PT-RW and PT-LG for the selected problems. We present the results of two experimental setting for PT-LG (learning rate) where learning rate of 0.1 and 0.01 are used. The results show that PT-LG (0.1) gives the best performance regarding training and test performance (RMSE) for Lazer, Sunspot, Lorenz and Henon problems. In other problems (Mackey, Rossler), it achieves similar performance when compared to PT-RW and PT-LG (0.01). We note that a highly significant improvement in results is made for the Henon and Lazer problems with PT-LG (0.1). We observe that the learning rate is an important parameter to harness the advantage of Langevin-gradients. Regarding computational time, we notice that PT-LG is far more time consuming than PT-RW, due to the cost of calculating the gradients.

Regarding the percentage of accepted proposals over the entire sampling process, PT-LG (0.1) has the least rate of accepted proposals. On the other hand, we notice that PT-RW has the highest acceptance percentage followed by PT-LG (0.01). Moreover, the swap percentage is generally higher for PT-LG (0.1) when compared to others for most of the problems. We note that having a lower rate of proposals accepted does not necessarily relate to the accuracy of prediction or classification. It is reported to indicate the behaviour of the algorithm given different experimental setting. Howsoever, based on the results, it could be argued that the higher rate of accepted proposals deteriorates the results. This could imply that weaker proposals are accepted more often in this setting; hence there is an emphasis on exploration rather than exploitation. The higher swap percentage provides a higher level of exploration. Furthermore, smaller learning rate (0.01) tends to give the gradient less influence on the proposals, and hence PT-LG (0.01) behaves more like PT-RW. We also provide further comparision with stochastic gradient descent (SGD) and Adam optimiser used for neural networks [85,86]. We observe that PT-RW and PT-LG provide better training and test performance when compared to these methods. We note that SGD and Adam optimiser are faster than the proposed methods, howsoever, they do not feature uncertainty quantification.

4.3. Results: classification

We use the multinomial likelihood given in Eq. 7 for pattern classification problems. We use a maximum sampling time of 50,000 samples with LG-frequency of 0.5, swap interval of 100, and a maximum geometric temperature of 10. The results for the classification performance along with the time taken and acceptance for the respective dataset is shown in Table 6. The results show that

Table 5
Prediction results.

Dataset	Method	Train (mean, std, best)	Test (mean, std, best)	Swap per.	Accept per.	Time (min.)
Lazer	PT-RW	0.0640 0.0218 0.0325	0.0565 0.0209 0.0270	42.26	35.31	4.53
	PT-LG (0.10)	0.0383 0.0187 0.0240	0.0353 0.0161 0.0212	48.45	19.91	11.53
	PT-LG (0.01)	0.0446 0.0160 0.0283	0.0414 0.0160 0.0253	51.45	30.97	11.50
	SGD	0.1020 0.0107 0.0832	0.0975 0.0072 0.0764	–	–	1.10
	ADAM	0.0805 0.0156 0.0629	0.0953 0.0150 0.0704	–	–	1.05
Sunspot	PT-RW	0.0242 0.0041 0.0170	0.0239 0.0050 0.0161	44.45	18.30	4.82
	PT-LG (0.10)	0.0199 0.0031 0.0155	0.0192 0.0033 0.0146	48.45	12.57	11.61
	PT-LG (0.01)	0.0215 0.0032 0.0168	0.0204 0.0034 0.0154	46.94	15.16	11.47
	SGD	0.0283 0.0469 0.0207	0.0273 0.0469 0.0216	–	–	1.3
	ADAM	0.0241 0.0018 0.0208	0.0236 0.0010 0.0219	–	–	1.05
Mackey	PT-RW	0.0060 0.0005 0.0051	0.0061 0.0005 0.0051	42.11	8.19	4.59
	PT-LG (0.1)	0.0061 0.0009 0.0047	0.0062 0.0009 0.0048	49.10	5.72	11.68
	PT-LG (0.01)	0.0064 0.0008 0.0052	0.0065 0.0008 0.0053	48.58	8.38	11.43
	SGD	0.0357 0.0166 0.0289	0.0358 0.0167 0.0290	–	–	1.6
	ADAM	0.0313 0.0013 0.0278	0.0314 0.0013 0.0279	–	–	1.05
Lorenz	PT-RW	0.0192 0.0033 0.0113	0.0171 0.0037 0.0094	39.48	14.48	4.45
	PT-LG (0.1)	0.0181 0.0018 0.0117	0.0157 0.0018 0.0094	50.37	9.66	11.48
	PT-LG (0.01)	0.0173 0.0023 0.0123	0.0147 0.0024 0.0095	46.30	11.91	11.42
	SGD	0.0297 0.0019 0.0258	0.0289 0.0019 0.0249	–	–	1.3
	ADAM	0.0322 0.0121 0.0299	0.0323 0.0122 0.0300	–	–	1.2
Rossler	PT-RW	0.0173 0.0011 0.0144	0.0175 0.0011 0.0148	48.11	12.53	4.22
	PT-LG (0.1)	0.0172 0.0008 0.0154	0.0175 0.0009 0.0155	39.57	8.58	11.60
	PT-LG (0.01)	0.0171 0.0011 0.0136	0.0173 0.0011 0.0135	50.18	10.98	11.36
	SGD	0.0296 0.00335 0.0255	0.0328 0.0039 0.0284	–	–	1.2
	ADAM	0.0264 0.0012 0.0249	0.0291 0.0014 0.0272	–	–	1.2
Henon	PT-RW	0.1230 0.0296 0.0167	0.1198 0.0299 0.0161	48.58	38.08	4.21
	PT-LG (0.1)	0.0201 0.0025 0.0146	0.0190 0.0029 0.0131	47.43	11.39	11.41
	PT-LG (0.01)	0.0992 0.0347 0.0221	0.0963 0.0341 0.0209	36.04	36.27	11.49
	SGD	0.0631 0.0144 0.0339	0.0609 0.0150 0.0316	–	–	1.21
	ADAM	0.0574 0.0109 0.0493	0.0554 0.0111 0.0492	–	–	1.03

Table 6
Classification results.

Dataset	Method	Train (mean, std, best)	Test (mean, std, best)	Swap perc.	Accept perc.	Time (min.)
Iris	PT-RW	51.39 15.02 91.43	50.18 41.78 100.00	52.56	95.32	1.26
	PT-LG (0.1)	64.93 21.51 100.00	59.33 39.84 100.00	51.08	51.48	1.81
	PT-LG (0.01)	97.32 0.92 99.05	96.76 0.96 99.10	51.77	97.55	2.09
	SGD	99.11, 0.23, 100	96.92, 1.05, 97.5	–	–	0.6
	Adam	99.61, 0.466, 1.0	96.83, 0.11, 97.5	–	–	0.07
Ionosphere	PT-RW	68.92 16.53 91.84	51.29 30.73 91.74	50.61	89.32	3.50
	PT-LG (0.1)	65.78 10.87 85.71	84.63 9.54 96.33	47.83	45.46	4.70
	PT-LG (0.01)	98.55 0.55 99.59	92.19 2.92 98.17	51.77	92.40	5.07
	SGD	99.14,0.28,100	95.5,1.00,97.5	–	–	0.98
	Adam	100, 0.0, 100	95.17, 0.62, 97.5	–	–	0.04
Cancer	PT-RW	83.78 20.79 97.14	83.55 27.85 99.52	40.18	89.71	2.78
	PT-LG (0.1)	83.87 17.33 97.55	90.59 16.67 99.52	41.71	43.87	5.13
	PT-LG(0.01)	97.00 0.29 97.75	98.77 0.32 99.52	49.25	94.67	5.09
	SGD	99.11,0.23,100	96.92,1.05,97.5	–	–	0.94
	Adam	99.49,0.47,100	96.67,1.18,97.5	–	–	0.17
Bank	PT-RW	78.39 1.34 80.11	77.49 0.90 79.45	49.13	61.59	27.71
	PT-LG(0.1)	77.90 1.92 79.71	77.74 1.27 79.57	46.17	29.61	69.89
	PT-LG(0.01)	80.75 1.45 85.41	79.96 0.81 82.61	50.00	31.50	86.94
	SGD	80.53,0.15,80.84	79.95,1.15,80.20	–	–	1.01
	Adam	80.88, 0.15, 81.16	80.18,0.14,80.51	–	–	0.07
Pen-Digit	PT-RW	76.67 17.44 95.24	71.93 16.59 90.62	45.60	50.72	57.13
	PT-LG(0.1)	73.91 17.36 91.98	70.09 16.08 85.68	46.89	24.95	87.06
	PT-LG(0.01)	84.98 7.42 96.02	81.24 6.82 91.25	51.08	25.09	86.62
	SGD	80.37,0.15,80.62	79.86,0.13,80.17	–	–	0.6
	Adam	80.64,0.19,80.97	80.00,0.19,80.39	–	–	0.07
Chess	PT-RW	89.48 17.46 100.00	90.06 15.93 100.00	48.09	69.09	252.56
	PT-LG(0.1)	100.00 0.00 100.00	100.00 0.00 100.00	49.88	92.61	327.45
	PT-LG(0.01)	100.00 0.00 100.00	100.00 0.00 100.00	50.12	88.87	323.10
	SGD	99.76,0.96,100	99.76,0.97,100	–	–	8.43
	Adam	100,0.00,100	100,0.00,100	–	–	1.07

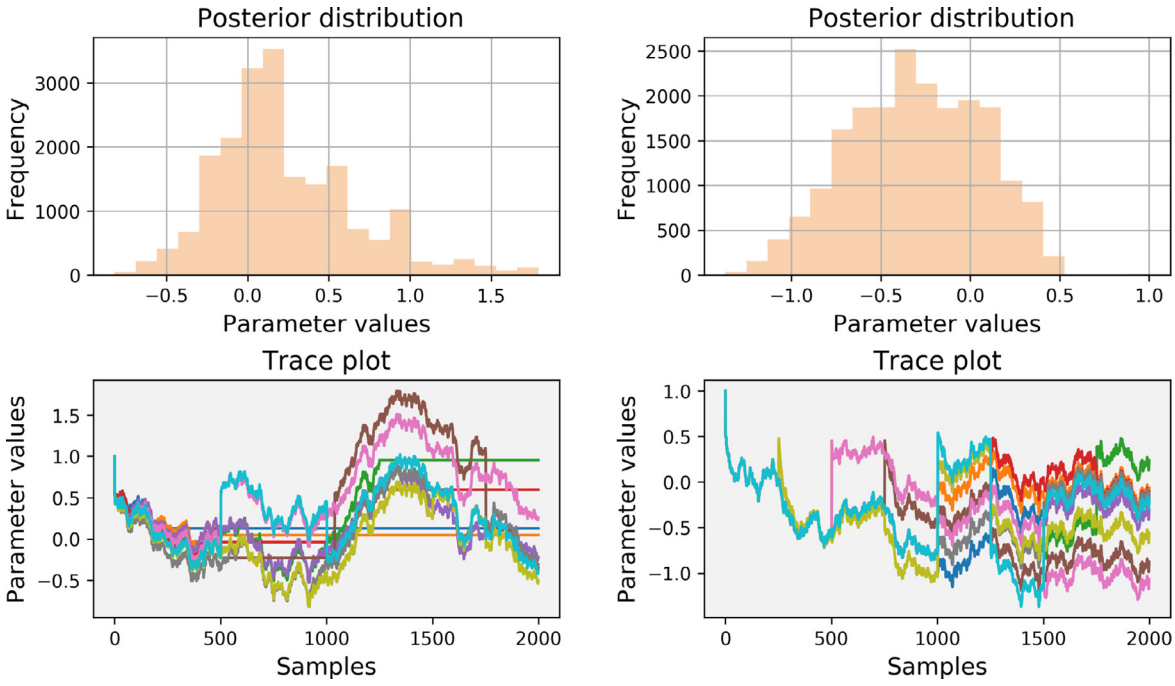


Fig. 2. Posterior distribution with sampling trace-plot of a selected weight in the input-hidden layer for Iris problem given by PT-LG.

PT-LG (0.01) overall achieves the best training and generalisation performance for all the given problems. PT-LG (0.1) has similar training and test performance when compared to PT-RW for specific problems (Bank and Pen-Digit only). In the majority of the cases, PT-LG (0.1) outperforms PT-RW. We notice that the smaller datasets (Iris, Ionosphere and Cancer) have much higher acceptance percentage of proposals when compared to larger datasets. Furthermore, the higher computational cost of applying Langevin-gradients is evident in all the problems. We also provide further comparision with SGD and Adam optimiser. We observe that PT-LG (0.01) provides similar training and test performance when compared to these methods. Figure 2 shows a typical posterior distribution with sampling trace-plot of a selected weight in the input-hidden layer for Iris problem given by PT-LG (0.01).

5. Discussion

The experiments reveal that it is essential to have the right hyper-parameter setting such as swap-interval and maximum temperature for geometric temperature spacing. The replicas with high values of the temperature give more opportunities for exploration while the replicas with lower temperature value focus on exploitation of a local region in the likelihood landscape. Moreover, the learning rate used for Langevin-gradients can also be an issue given the type of problem, which also depends on the nature of the time series; whether it is a synthetic or real-world problem and how much of noise and inconsistencies are present. We notice that the Langevin-gradients improve the results further for real-world time series (Lazer and Sunspot issues). In some instances, a higher level of noise and uncertainty is present in synthetic time series, such as the Henon time series. In these cases, the Langevin-gradient significantly improves performance. One needs to take into account the computational costs of employing the Langevin-gradients as shown in the results. Hence it is essential to limit their usage as much as possible while ensuring that accuracy in prediction is not affected.

In the case of classification problems, we observe that both variations of Langevin-gradient parallel tempering significantly improves the classification performance for the majority of the problems. In some cases, the classification performance is slightly improved; however, Langevin-gradients have used high computational time in general. We also observe that the lower learning rate (0.01) gives the best performance for classification problems. However, it shows to be less effective for the time series prediction problems. Therefore, the sampling is highly sensitive to the Langevin-gradient learning rate which needs to be tailored for the type of the problem. We note that the rate at which the Langevin-gradient is employed adds to the computational cost. We need to highlight further that the parallel tempering is used mostly for exploration in the first phase of sampling and most of these values are discarded given 50% of *burnin* period. The second phase follows canonical MCMC sampling while taking advantage of replica exchange and parallel computing.

We considered small scale to medium scale classification problems that considered a few hundred to thousands of weights in the neural networks. A large number of neural network weights implies large scale inference which has been a challenge of Bayesian neural networks that have mainly been limited to small scale problems. Furthermore, our implementation and experiments were limited to ten replicas, which could be increased for bigger datasets and problems. Multi-core parallel tempering has demonstrated to have the potential to be applied to deep learning that involves recurrent neural networks and convolution networks that typical deal with thousands to millions of weights. Although, millions of weights is a huge task for vision-related problems, given the incorporation of Langevin-gradients, this could be possible. Another possibility is to use stochastic gradient descent implementation for Langevin-gradients with parallel tempering. We only considered the calculation of gradient for entire datasets which could be difficult for big data problems. The deep learning literature has focused on different frameworks for acquiring and applying gradients which can be used to improve parallel tempering methodology in this paper further. This opens up Bayesian inference for new neural network architectures such as long short-term memory recurrent networks (LSTMs) and generative adversarial networks (GANs) ([87,88].

Furthermore, the use of efficient priors via transfer learning methods could also be a direction, where the priors feature knowledge from similar problems. Although uncertainty quantification is not a major requirement in some of the computer vision problems; in specific domains such as medical imaging and security, it is important to feature a Bayesian perspective that can be used to fuse many aspects of the data. For instance, three-dimensional (3D) face recognition could provide data from different sensors and cameras that can be combined given a Bayesian methodology. Furthermore, the framework proposed could be used for other models, such as in Earth science, which have expensive models that require efficient optimisation or parameter estimation techniques [89]. Related work has explored the use of such framework for estimation of solid Earth evolution models. In such problems, due to the complexity of the models, there is no scope for incorporating gradients since they are unavailable. In such cases, the need to develop heuristic methods for acquiring gradients is essential. Hence, meta-heuristic and evolutionary algorithms could be incorporated with parallel tempering in models that do not have gradients. This could also apply to dynamic and complex neural network architectures, which in the past, has been addressed through neuro-evolution [83,90]. Although this paper has not focused on convergence analysis, this would be a focus of future research.

6. Conclusions and future work

We presented a multi-core implementation of parallel tempering with proposal distributions that featured random-walk and Langevin-gradients. We applied the method for time series prediction and pattern classification problems with different levels of complexity. The method takes advantage of parallel computing for large datasets where thousands of samples are required for convergence. The experimental results show that proposals by Langevin-gradients significantly improves convergence with better prediction and classification performance when compared to random-walk proposal distributions. We observed that Langevin-gradients can further add to the computational cost, and in future work, such challenges can be tackled with more efficient gradient proposals.

The results motivate the use of the methodology for large scale problems that face challenges with canonical implementations of Bayesian neural networks. The method can be further adapted for deep neural network architectures and various applications that require uncertainty quantification in prediction or decision-making process.

Disclosure of conflicts of interest

None

Acknowledgment

We would like to thanks Artemis high performance computing support at University of Sydney and Arpit Kapoor for providing technical support.

Supplementary material

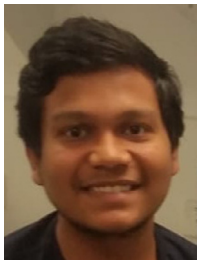
Supplementary material that includes data and code for Langevin-gradient parallel tempering for Bayesian neural learning: https://github.com/sydney-machine-learning/parallel-tempering-neural-net.

References

[1] D.E. Rumelhart, G.E. Hinton, R.J. Williams, Learning representations by back-propagating errors, Nature 323 (6088) (1986) 533.
[2] M.D. Zeiler, ADADELTA: an adaptive learning rate method, 2012, arXiv:1212.5701.
[3] G.C. Cawley, N.L. Talbot, Preventing over-fitting during model selection via Bayesian regularisation of the hyper-parameters, J. Mach. Learn. Res. 8 (Apr) (2007) 841–861.
[4] Y. Bengio, Gradient-based optimization of hyperparameters, Neural Comput. 12 (8) (2000) 1889–1900.
[5] S. Lawrence, C.L. Giles, A.C. Tsoi, What size neural network gives optimal generalization? Convergence properties of backpropagation, Technical Report, Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742, 1996.
[6] D. White, P. Ligomenides, Gannet: a genetic algorithm for optimizing topology and weights in neural network design, in: International Workshop on Artificial Neural Networks, Springer, 1993, pp. 322–327.
[7] J. Schmidhuber, Deep learning in neural networks: an overview, Neural Netw. 61 (2015) 85–117.
[8] D.J. MacKay, A practical Bayesian framework for backpropagation networks, Neural Comput. 4 (3) (1992) 448–472.
[9] D.J. MacKay, Hyperparameters: optimize, or integrate out? in: Maximum Entropy and Bayesian Methods, Springer, 1996, pp. 43–59.
[10] D.J. MacKay, Probable networks and plausible predictions a review of practical Bayesian methods for supervised neural networks, Netw.: Comput. Neural Syst. 6 (3) (1995) 469–505.
[11] C. Robert, in: Machine learning, a probabilistic perspective, 2014.
[12] W.K. Hastings, Monte carlo sampling methods using markov chains and their applications, Biometrika 57 (1) (1970) 97–109.
[13] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, E. Teller, Equation of state calculations by fast computing machines, J. Chem. Phys. 21 (6) (1953) 1087–1092.
[14] A.E. Raftery, S.M. Lewis, Implementing MCMC, Markov Chain Monte Carlo in Practice, 1996, pp. 115–130.
[15] D. van Ravenzwaaij, P. Cassey, S.D. Brown, A simple introduction to markov chain monte–carlo sampling, Psychon. Bull. Rev. (2016) 1–12.
[16] S. Banerjee, B.P. Carlin, A.E. Gelfand, Hierarchical Modeling and Analysis for Spatial Data, Crc Press, 2014.
[17] R.M. Neal, Bayesian Learning for Neural Networks, 118, Springer Science & Business Media, 2012.
[18] P. Jylänki, A. Nummenmaa, A. Vehtari, Expectation propagation for neural networks with sparsity-promoting priors., J. Mach. Learn. Res. 15 (1) (2014) 1849–1901.
[19] G.E. Hinton, D. Van Camp, Keeping the neural networks simple by minimizing the description length of the weights, in: Proceedings of the Sixth Annual Conference on Computational Learning Theory, ACM, 1993, pp. 5–13.
[20] C. Li, C. Chen, D. Carlson, L. Carin, Preconditioned stochastic gradient langevin dynamics for deep neural networks, in: Thirtieth AAAI Conference on Artificial Intelligence, 2016.
[21] F. Liang, Annealing stochastic approximation monte carlo algorithm for neural network training, Mach. Learn. 68 (3) (2007) 201–233.
[22] O. Kocadağlı, B. Aşıkgil, Nonlinear time series forecasting with Bayesian neural networks, Expert Syst. Appl. 41 (15) (2014) 6596–6610.
[23] R. Chandra, L. Azizi, S. Cripps, Bayesian neural learning via langevin dynamics for chaotic time series prediction, in: D. Liu, S. Xie, Y. Li, D. Zhao, E.-S. M. El-Alfy (Eds.), Neural Information Processing, 2017.
[24] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: a simple way to prevent neural networks from overfitting, J. Mach. Learn. Res. 15 (1) (2014) 1929–1958.
[25] Y. Gal, Z. Ghahramani, Dropout as a Bayesian approximation: representing model uncertainty in deep learning, in: International Conference on Machine Learning, 2016, pp. 1050–1059.
[26] A. Kendall, Y. Gal, What uncertainties do we need in Bayesian deep learning for computer vision? in: Advances in Neural Information Processing Systems, 2017, pp. 5580–5590.
[27] J. Wang, K. Shi, Q. Huang, S. Zhong, D. Zhang, Stochastic switched sampled–data control for synchronization of delayed chaotic neural networks with packet dropout, Appl. Math. Comput. 335 (2018) 211–230.
[28] K. Shi, J. Wang, Y. Tang, S. Zhong, Reliable asynchronous sampled-data filtering of ts fuzzy uncertain delayed neural networks with stochastic switched topologies, Fuzzy Sets Syst. (2019) In press.
[29] E. Marinari, G. Parisi, Simulated tempering: a new monte carlo scheme, EPL (Europhys. Lett.) 19 (6) (1992) 451.
[30] C.J. Geyer, E.A. Thompson, Annealing markov chain monte carlo with applications to ancestral inference, J. Am. Stat. Assoc. 90 (431) (1995) 909–920.
[31] A. Patriksson, D. van der Spoel, A temperature predictor for parallel tempering simulations, Phys. Chem. Chem. Phys. 10 (15) (2008) 2073–2077.
[32] K. Hukushima, K. Nemoto, Exchange monte carlo method and application to spin glass simulations, J. Phys. Soc. Jpn. 65 (6) (1996) 1604–1608.
[33] L. Lamport, On interprocess communication, Distrib. Comput. 1 (2) (1986) 86–101.
[34] R. Salakhutdinov, A. Mnih, G. Hinton, Restricted boltzmann machines for collaborative filtering, in: Proceedings of the 24th International Conference on Machine Learning, ACM, 2007, pp. 791–798.
[35] G.E. Hinton, S. Osindero, Y.-W. Teh, A fast learning algorithm for deep belief nets, Neural Comput. 18 (7) (2006) 1527–1554.
[36] G. Desjardins, A. Courville, Y. Bengio, P. Vincent, O. Delalleau, Tempered markov chain monte carlo for training of restricted boltzmann machines, in:

Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, 2010, pp. 145–152.
[37] P. Brakel, S. Dieleman, B. Schrauwen, Training restricted boltzmann machines with multi-tempering: harnessing parallelization, in: International Conference on Artificial Neural Networks, Springer, 2012, pp. 92–99.
[38] A. Fischer, C. Igel, A bound for the convergence rate of parallel tempering for sampling restricted boltzmann machines, Theoret. Comput. Sci. 598 (2015) 102–117, doi:10.1016/j.tcs.2015.05.019. URL: http://www.sciencedirect.com/science/article/pii/S0304397515004235
[39] R. Pascanu, T. Mikolov, Y. Bengio, On the difficulty of training recurrent neural networks, in: International Conference on Machine Learning, 2013, pp. 1310–1318.
[40] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen, et al., Mllib: machine learning in apache spark, J. Mach. Learn. Res. 17 (1) (2016) 1235–1241.
[41] X. Bi, X. Zhao, G. Wang, P. Zhang, C. Wang, Distributed extreme learning machine with kernels based on mapreduce, Neurocomputing 149 (2015) 456–463. Advances in neural networks Advances in Extreme Learning Machines.
[42] Y. Sun, Y. Yuan, G. Wang, An os-elm based distributed ensemble classification framework in p2p networks, Neurocomputing 74 (16) (2011) 2438–2443. Advances in Extreme Learning Machine: Theory and Applications Biological Inspired Systems. Computational and Ambient Intelligence.
[43] R.H. Swendsen, J.-S. Wang, Replica monte carlo simulation of spin-glasses, Phys. Rev. Lett. 57 (21) (1986) 2607.
[44] U.H. Hansmann, Parallel tempering algorithm for conformational studies of biological molecules, Chem. Phys. Lett. 281 (1–3) (1997) 140–150.
[45] M.K. Sen, P.L. Stoffa, Bayesian inference, gibbs' sampler and uncertainty estimation in geophysical inversion, Geophys. Prospect. 44 (2) (1996) 313–350.
[46] M. Maraschini, S. Foti, A monte carlo multimodal inversion of surface waves, Geophys. J. Int. 182 (3) (2010) 1557–1566.
[47] N. Rathore, M. Chopra, J.J. de Pablo, Optimal allocation of replicas in parallel tempering simulations, J. Chem. Phys. 122 (2) (2005) 024111.
[48] H.G. Katzgraber, S. Trebst, D.A. Huse, M. Troyer, Feedback-optimized parallel tempering monte carlo, J. Stat. Mech.: Theory Exp. 2006 (03) (2006) P03018.
[49] E. Bittner, A. Nußbaumer, W. Janke, Make life simple: unleash the full power of the parallel tempering algorithm, Phys. Rev. Lett. 101 (13) (2008) 130603.
[50] F. Calvo, All-exchanges parallel tempering, J. Chem. Phys. 123 (12) (2005) 124106.
[51] M. Fielding, D.J. Nott, S.-Y. Liong, Efficient MCMC schemes for computationally expensive posterior distributions, Technometrics 53 (1) (2011) 16–28.
[52] P. Liu, B. Kim, R.A. Friesner, B.J. Berne, Replica exchange with solute tempering: a method for sampling biological systems in explicit water, Proc. Natl. Acad. Sci. 102 (39) (2005) 13749–13754.
[53] Y. Li, M. Mascagni, A. Gorin, A decentralized parallel implementation for parallel tempering algorithm, Parallel Comput. 35 (5) (2009) 269–283.
[54] K. Karimi, N. Dickson, F. Hamze, High-performance physics simulations using multi-core cpus and gpgpus in a volunteer computing context, Int. J. High Perform. Comput. Appl. 25 (1) (2011) 61–69.
[55] G. Mingas, L. Bottolo, C.-S. Bouganis, Particle MCMC algorithms and architectures for accelerating inference in state-space models, Int. J. Approx. Reason. 83 (2017) 413–433.
[56] F. Musiani, A. Giorgetti, Protein aggregation and molecular crowding: perspectives from multiscale simulations, in: International Review of Cell and Molecular Biology, 329, Elsevier, 2017, pp. 49–77.
[57] Y. Xie, J. Zhou, S. Jiang, Parallel tempering monte carlo simulations of lysozyme orientation on charged surfaces, J. Chem. Phys. 132 (6) (2010) 02B602.
[58] A.N. Tharrington, K.D. Jordan, Parallel-tempering monte carlo study of (h2o) n= 6–9, J. Phys. Chem. A 107 (38) (2003) 7380–7389.
[59] T.B. Littenberg, N.J. Cornish, Bayesian approach to the detection problem in gravitational wave astronomy, Phys. Rev. D 80 (6) (2009) 063007.
[60] A. Reid, E.V. Bonilla, L. McCalman, T. Rawling, F. Ramos, Bayesian joint inversions for the exploration of Earth resources, in: IJCAI, 2013, pp. 2877–2884.
[61] M.D. Richard, R.P. Lippmann, Neural network classifiers estimate Bayesian a posteriori probabilities, Neural Comput. 3 (4) (1991) 461–483.
[62] A. Krogh, J.A. Hertz, A simple weight decay can improve generalization, in: Advances in Neural Information Processing Systems, 1992, pp. 950–957.
[63] T. Auld, A.W. Moore, S.F. Gull, Bayesian neural networks for Internet traffic classification, IEEE Trans. Neural Netw. 18 (1) (2007) 223–239.
[64] R.M. Neal, et al., MCMC using hamiltonian dynamics, Handbook of Markov Chain Monte Carlo 2 (11) (2011).
[65] M. Welling, Y.W. Teh, Bayesian learning via stochastic gradient langevin dynamics, in: Proceedings of the 28th International Conference on Machine Learning (ICML-11), 2011, pp. 681–688.
[66] C. Chen, D. Carlson, Z. Gan, C. Li, L. Carin, Bridging the gap between stochastic gradient MCMC and stochastic optimization, in: Artificial Intelligence and Statistics, 2016, pp. 1051–1060.
[67] F. Liang, Bayesian neural networks for nonlinear time series forecasting, Stat. Comput. 15 (1) (2005) 13–29.
[68] H.S. Hippert, J.W. Taylor, An evaluation of Bayesian techniques for controlling model complexity and selecting inputs in a neural network for short-term load forecasting, Neural Netw. 23 (3) (2010) 386–395.
[69] D.T. Mirikitani, N. Nikolaev, Recursive Bayesian recurrent neural networks for time-series modeling, IEEE Trans. Neural Netw. 21 (2) (2010) 262–274.
[70] E.A. Wan, Neural network classification: a Bayesian interpretation, IEEE Trans. Neural Netw. 1 (4) (1990) 303–305.
[71] K. Cho, A. Ilin, T. Raiko, Improved learning of gaussian-bernoulli restricted boltzmann machines, in: International Conference on Artificial Neural Networks, Springer, 2011, pp. 10–17.
[72] K. Cho, T. Raiko, A. Ilin, Parallel tempering is efficient for learning restricted boltzmann machines, in: The 2010 International Joint Conference on Neural Networks (IJCNN), IEEE, 2010, pp. 1–8.
[73] G. Desjardins, A. Courville, Y. Bengio, Adaptive parallel tempering for stochastic maximum likelihood learning of RBMs, 2010, arXiv:1012.3476.
[74] G. Desjardins, H. Luo, A. Courville, Y. Bengio, Deep tempering, 2014, arXiv:1410.0123.
[75] F. Takens, Detecting strange attractors in turbulence, in: Dynamical Systems and Turbulence, Warwick 1980, Lecture Notes in Mathematics, 1981, pp. 366–381.
[76] D.J. Earl, M.W. Deem, Parallel tempering: theory, applications, and new perspectives, Phys. Chem. Chem. Phys. 7 (23) (2005) 3910–3916.
[77] R.M. Neal, Sampling from multimodal distributions using tempered transitions, Stat. Comput. 6 (4) (1996) 353–366.
[78] L. Ventura, S. Cabras, W. Racugno, Prior distributions from pseudo-likelihoods in the presence of nuisance parameters, J. Am. Stat. Assoc. 104 (486) (2009) 768–774.
[79] R. Chandra, L. Azizi, S. Cripps, Bayesian neural learning via langevin dynamics for chaotic time series prediction, in: Neural Information Processing - 24th International Conference, ICONIP 2017, Guangzhou, China, November 14–18, 2017, Proceedings, Part V, 2017, pp. 564–573, doi:10.1007/978-3-319-70139-4_57.
[80] A. Kone, D.A. Kofke, Selection of temperature intervals for parallel-tempering simulations, J. Chem. Phys. 122 (20) (2005) 206101.
[81] N. Singh, L.-M. Browne, R. Butler, Parallel astronomical data processing with python: recipes for multicore machines, Astron. Comput. 2 (2013) 1–10.
[82] R. Chandra, Y.-S. Ong, C.-K. Goh, Co-evolutionary multi-task learning for dynamic time series prediction, Appl. Soft Comput. 70 (2018) 576–589.
[83] R. Chandra, Y.-S. Ong, C.-K. Goh, Co-evolutionary multi-task learning with predictive recurrence for multi-step chaotic time series prediction, Neurocomputing 243 (2017) 21–34.
[84] A. Asuncion, D. Newman, in: UCI machine learning repository, 2007. http://www.ics.uci.edu/~mlearn/MLRepository.html.
[85] B. Léon, Large-scale machine learning with stochastic gradient descent, in: Proceedings of COMPSTAT'2010, Physica-Verlag HD, 2010, pp. 177–186.
[86] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, 2014, arXiv:1412.6980.
[87] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Comput. 9 (8) (1997) 1735–1780.
[88] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio, Generative adversarial nets, in: Advances in Neural Information Processing Systems, 2014, pp. 2672–2680.
[89] R. Chandra, R.D. Müller, R. Deo, N. Butterworth, T. Salles, S. Cripps, in: Multi-core parallel tempering Bayeslands for basin and landscape evolution, 2018, arXiv:1806.10939.
[90] M.A. Potter, K.A. De Jong, Cooperative coevolution: an architecture for evolving coadapted subcomponents, Evol. Comput. 8 (1) (2000) 1–29, doi:10.1162/106365600568086.

Dr. Rohitash Chandra holds a Ph.D in Computer Science (2012) from Victoria University of Wellington, M.S. in Computer Science (2008) from the University of Fiji, and B. Sc. in Computer Science and Engineering Technology (2006) from the University of the South Pacific. He is currently USyd Research Fellow at the Centre for Translational Data Science and School of Geosciences, University of Sydney, Australia. His research interests are in areas of deep learning, neuro-evolution, Bayesian methods, solid Earth evolution, reef modelling and mineral exploration.



Konark Jain is an undergraduate at Department of Electronics and Electrical Engineering, Indian Institute of Technology Guwahati, Assam, India. His research interests are in areas of deep learning, machine learning and Bayesian methods.

Ratneel Deo is an Assiatnt Lecturer in Computer Science at the School of Computing, Information and Mathematical Sciences, University of the South Pacific, Suva Fiji. His research interests are in areas of recurrent neural networks, parallel tempering MCMC, reinforcement learning, and machine learning applications.

Prof. Sally Cripps is the Co-Director of Centre for Translational Data Science and Professor in Statistics at the School of Mathematics and Statistics, University of Sydney, Australia. Her research interests include, Bayesian inference and computational statistics, with application to solid Earth evolution and reef modelling.