Applied Soft Computing 129 (2022) 109528

Contents lists available at ScienceDirect

Applied Soft Computing

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

Bayesian neuroevolution using distributed swarm optimization and tempered MCMC

Arpit Kapoor [a,b,f,*], Eshwar Nukala [c], Rohitash Chandra [d,e,f]

[a] School of Civil and Environmental Engineering, University of New South Wales, Sydney, Australia
[b] Department of Computer Science, SRM Institute of Science and Technology, Kattankulathur, Tamil Nadu, India
[c] Department of Civil Engineering, Indian Institute of Technology Guwahati, Assam, India
[d] Transitional Artificial Intelligence Research Group, School of Mathematics and Statistics, University of New South Wales, Sydney, Australia
[e] UNSW Data Science Hub, University of New South Wales, Sydney, Australia
[f] Data Analytics for Resources and Environments, Australian Research Council - Industrial Transformation Training Centre (ARC-ITTC), Australia

ARTICLE INFO

Article history:
Received 4 January 2022
Received in revised form 7 August 2022
Accepted 15 August 2022
Available online 20 August 2022

Keywords:
Bayesian neural networks
Markov chain Monte Carlo
Neuroevolution
Parallel tempering
Tempered MCMC

ABSTRACT

The major challenge of Bayesian neural networks has been in developing effective sampling methods that address deep neural networks and big data-related problems. As an alternative to gradient-based training methods, neuro-evolution features evolutionary algorithms that provide a black-box approach to learning in neural networks. Neuroevolution employs evolutionary and swarm optimization methods to provide an alternative where the training algorithm is not constrained to the architecture of the network and has the potential to reduce local-minima and vanishing gradient problems. Bayesian neural networks use variational inference and Markov chain Monte Carlo (MCMC) sampling methods. Tempered MCMC is a powerful MCMC method that can take advantage of a parallel computing environment and efficient proposal distributions. In this paper, we present a synergy of neuroevolution and Bayesian neural networks where operators in particle swarm optimization (PSO) are used for forming efficient proposals in tempered MCMC sampling. The results show that the proposed method provides better prediction accuracy when compared to random-walk proposal distribution in MCMC for both time-series and pattern classification problems. The results also show substantial computational time reduction compared to gradient-based proposals while generating comparable accuracy performance. The Bayesian neuroevolution framework can be further introduced to models that do not have gradient information.

Code metadata

Permanent link to reproducible Capsule: https://doi.org/10.24433/CO.3633421.v1.

1. Introduction

Bayesian neural networks feature probability distributions to represent the trainable parameters (weights and biases) of a neural network [1–4]. The training process is implemented via Bayesian inference, which essentially performs numerical integration using Markov chain Monte Carlo (MCMC) sampling method via the posterior distribution of weights and biases [4,5]. Bayesian neural network offers rigorous uncertainty quantification as opposed to canonical gradient-based methods [6,7]. MCMC sampling methods face limitations due to slow convergence and scalability, and hence there has been slow progress in Bayesian deep learning when large models and datasets are involved. A number of methods have been developed to address these challenges, which include using gradient-based methods for developing efficient proposal distributions within MCMC sampling methods [8–12], combining meta-heuristics such as evolutionary algorithms with MCMC methods [13–16,14], and developing robust MCMC approaches that balance exploration with exploitation such as nested sampling [17] and parallel tempering (tempered MCMC) [18–20].

The search for transition kernels or proposal distributions for MCMC methods to efficiently explore multimodal posterior distributions in various models is an open problem. The random-walk proposal distribution is canonical to MCMC sampling where

The code (and data) in this article has been certified as Reproducible by Code Ocean: (https://codeocean.com/). More information on the Reproducibility Badge Initiative is available at https://www.elsevier.com/physical-sciences-and-engineering/computer-science/journals.

* Corresponding author.
E-mail addresses: arpit.kapoor@unsw.edu.au (A. Kapoor), eshwar@iitg.ac.in (E. Nukala), rohitash.chandra@unsw.edu.au (R. Chandra).
URLs: https://arpit-kapoor.com (A. Kapoor), https://rohitash-chandra.github.io (R. Chandra).

https://doi.org/10.1016/j.asoc.2022.109528
1568-4946/

a small noise, typically Gaussian, is added to the current state. The major advantage of this approach is the gradient-free requirement, and hence it is applicable for a wide range of models and usually where parameters have multiple constraints, such as in our case of reef modelling [21]. On the other hand, given that a model features gradient information, this information can be vital for developing efficient proposals. Such models include Hamiltonian Monte Carlo (HMC) [10] and more advanced Riemannian Manifold HMC that further exploits the geometric properties of the parameter space [8]. The no-u-turn (NUTS) sampler provides additional improvements via adaptation mechanism in HMC [22].

Chandra et al. [12] combined the power of high-performance computing with tempered MCMC for Bayesian neural networks for pattern classification and time series prediction problems. The tempered MCMC used Langevin-gradient proposals which provided better convergence in terms of classification and prediction accuracy when compared to random-walk and gradient-based methods from the literature, such as stochastic gradient-descent [23] and Adam optimization [24]. However, the Langevin-gradient proposals are generally computationally expensive since it requires gradient information. Recently, tempered MCMC with Langevin-gradients has been successfully used for Bayesian graph CNNs [25] and Bayesian autoencoders [26] that features more than a million parameters. Bayesian neural networks via tempered MCMC and Langevin-gradients has also been applied for stock price prediction during COVID-19 [27].

Evolutionary and swarm optimization algorithms are well known to have global optimization features which provide gradient-free optimisation for large scale problems [28–30]. Particle swarm optimization (PSO) was motivated by swarm behaviour and demonstrated effectiveness in handling non-linear, non-differentiable, and multi-modal optimization problems [30–32] Previously hybrid-modelling approaches such as Bayesian networks with PSO [33, 34] have been used for a variety of applications such as fault detection and diagnosis [35,36]. Evolutionary and swarm optimisation have been effective in training neural networks which is known as neuroevolution [37–42]. Neuroevolution has focused on autonomous construction of neural network typologies while training the weights, as well as training the weights in fixed network typologies [40,43–45]. Neuroevolution has also been combined with Bayesian optimisation for training deep neural networks [46]. Neuroevolution is not constrained to the architecture of the neural network and avoids the drawbacks of gradient descent such as concerns with local minima and vanishing gradients, commonly seen in recurrent neural networks with long-term dependencies. In the past decade, neuroevolution has been extended to more complex deep learning architectures including long short-term memory (LSTM) networks [47], convolutional neural networks (CNNs) [48], and has been demonstrated to be a very promising technique for complicated reinforcement learning problems [49–51].

As opposed to the Langevin-gradient proposals that require computationally expensive gradients to be available, PSO can provide gradient-free proposals. We also note that gradient-descent suffers from local minima problems and is not applicable in models where gradient information is not readily available. This is often the case in Earth science applications such as reef modelling and landscape evolution models [52,21]. Additionally, there is a strong need for uncertainty quantification in models for decision making in medical science [53], such as in the case of detecting heart disease in patients. Uncertainty quantification provides a better picture of the risks involved to patients. Hence, the application of Bayesian neural networks with effective MCMC sampling is critical for a wide range of problems.

The strength of neuroevolution motivates its extension to a Bayesian neural network framework to incorporate uncertainty quantification offered by MCMC sampling methods through probabilistic representation of the parameters (weights and biases). Furthermore, parallel computing and tempered MCMC can provide global exploration during sampling while limiting computational time. Although there has been some work combining evolutionary and swarm optimizationalgorithms with MCMC methods [13,15,16,14,54], not much has been done for the case of Bayesian neural networks. Moreover, previous research in this area has not taken advantage of parallel computing that is useful for big data problems and computationally expensive models.

In this paper, we present a framework that combined PSO with tempered MCMC to form efficient proposals for Bayesian neural networks. We focus on learning neural network weights while keeping the neural network architecture fixed. We refer to the framework as Bayesian neuroevolution here-within, which is implemented in a distributed computing environment and tested with various time-series prediction and pattern classification problems. In this framework, we utilize gradient-free proposals offered by evolutionary operators for efficient exploration of the posterior distribution. We also investigate key metrics including accuracy, computational efficiency and convergence of MCMC replicas (chains). Finally, we present a comparison between the posterior distribution of model parameters of Bayesian neuroevolution with Langevin-gradient tempered MCMC.

The rest of the paper is organized as follows. Section 2 provides a background on related methods and Section 3 presents the proposed method. Section 4 presents experiments, Section 5 discussion and Section 6 conclusions with future research.

2. Background and related work

2.1. Bayesian inference with MCMC

In the case of Bayesian neural networks, MCMC sampling has been used to infer the posterior distribution of the neural network parameters (weights and biases) with applications that mainly include pattern classification [3,2] and time-series prediction [55–58,12]. The posterior distribution to estimate the neural network parameters using the Bayes' theorem is given as,

$$p(\theta|\mathbf{y}) = \frac{p(\mathbf{y}|\mathbf{x},\theta)p(\theta)}{p(\mathbf{y})} \propto p(\mathbf{y}|\mathbf{x},\theta)p(\theta) \tag{1}$$

where, $\mathbf{y}$ is the data and $p(\mathbf{y})$ is the marginal distribution which is a normalizing constant. The prior $p(\theta)$ represents prior knowledge about the distribution of the neural network parameters given by θ. The likelihood $p(\mathbf{y}|\mathbf{x},\theta)$ evaluates the model with given proposal (θ) [5,59]. MCMC sampling methods set up a Markov chain whose stationary distribution is the posterior of the parameters in order to draw samples from this distribution.

Tempered MCMC [18–20] was designed for probabilistic sampling of multi-modal density functions by taking inspiration from the field of thermodynamics focusing on fluid dynamics as a system cools down [18,60]. This features an ensemble of parallel MCMC processes known as *replicas* that are defined by a *temperature* ladder. The temperature ladder assigns each replica r with temperature value T which is used to rescale the likelihood as,

$$p(\mathbf{y}|\theta, r) = p(\mathbf{y}|\theta)^{\frac{1}{T}} \tag{2}$$

Essentially, the replicas with higher temperature values have a higher probability of accepting proposals, which provides exploration, while the replicas with lower temperature values provide exploitation. We note that replica proposals can also feature an exchange of state of configuration for neighbouring replicas. Based on different temperature values, with the exchange of

neighbouring replicas, tempered MCMC sampling provides a balance between exploration and exploitation for exploring multimodal and discontinuous posteriors [61,62]. Although random-walk proposal distributions are typically used, Langevin-gradient based proposals within the replicas of tempered MCMC [12,25,26] have shown very promising results.

2.2. Neuroevolution

Neuroevolution inherits some of the same drawbacks as evolutionary methods, such as computational inefficiency when dealing with high-dimensional parameter spaces and big data problems. Parallel computing architectures have been used as a remedy, and they have demonstrated enhanced computational performance when compared to traditional methodologies, particularly in the case of deep reinforcement learning [63]. Given the popularity of deep learning methods and the recent growth in computational power, neuroevolution has been used with a variety of deep learning architectures. Verbancsics et al. [48] proposed a generative neuroevolution approach and demonstrated that although neuroevolution can struggle to perform image classification by itself, it is useful in designing feature extractors, which when coupled with other machine learning models can show impressive results. Poulsen et al. [64] investigated the effectiveness of deep neuroevolution in designing *bots* capable of playing *first person shooter games*. In the case of recurrent neural networks, Rawal et al. [47] evolved deep LSTM networks using an information maximization objective and the results showed that neuroevolution can discover powerful LSTM-based memory solutions that outperform traditional RNNs. Furthermore, the scope of neuroevolution in memory-augmented neural network architectures has been explored in the literature, where various RNN architectures were evolved using state-of-the-art techniques [65,66].

Neural networks are known to be quite sensitive to hyper-parameters and network architectures. Designing the best architecture by trial and error is not an effective solution and hence, neuroevolution has been used in designing neural network architectures. Stanley et al. [67] reviewed how recent advances in computing strength have enabled significant growth in the performance of neuroevolution in training deep learning models with meta-learning applications. Pruning neural networks has become a vital approach with the impact of 'dropouts' regularisation methodology in improving deep learning methods [68,69]. In the case of neuroevolution, there has been a focus on producing compact network architectures during topology evolution. Assunção et al. [70] presented an efficient deep neuroevolution, where they enhanced *deep evolutionary network structured representation* up to 20 times while retaining state-of-the-art results. Deep imitation learning is another area where neuroevolution has proved to be a viable option for learning the design and weights of deep neural networks [71].

2.3. Evolutionary MCMC methods

Evolutionary Markov Chain Monte Carlo (EMCMC) algorithms are a class of MCMC algorithms that use evolutionary algorithms and related meta-heuristics for generating effective proposals [13]. EMCMC algorithms are classified into two categories which are (1) *family-competitive* EMCMC, and (2) *population-driven* EMCMC. Family-competitive EMCMC algorithms consist of multiple chains, one for each population member. These chains interact with each other to exchange information via the proposal mechanism or acceptance rule. The proposal distribution in family-competitive algorithms is further enhanced via mutation operation and crossover-based genetic operators. In contrast to family-competitive EMCMC, population-driven EMCMC maintains a single MCMC chain at the population level.

Differential evolution (DE) which is a prominent evolutionary algorithm has been commonly used in EMCMC approaches, particularly in the case of optimization in continuous spaces [15,16]. Ter et al. [15] demonstrated the success of combining differential evolution with MCMC (DE-MC) showing up to 500% improvement in performance over random-walk proposal distribution in MCMC. The authors also provided a proof of symmetric proposal distribution for DE-MC. The computational efficiency of DE has motivated its use in *approximate Bayesian computation* (ABC) framework which needs an extensive number of samples to be drawn, thus requiring more computation time [72]. Due to the robustness and accuracy of swarm-based optimization, PSO algorithms have been successfully applied for the learning structure of Bayesian networks [73]. Moreover, PSO has demonstrated to be an effective algorithm for visual tracking problems when used along with MCMC [54]. The results showed that the PSO-MCMC based tracker outperformed some of the state-of-the-art algorithms. Therefore, due to its simplicity and effectiveness, we use PSO as the evolutionary optimization algorithm in our Bayesian neuroevolution framework given in the next section.

3. Bayesian neuroevolution framework

3.1. Model, prior and likelihood function

We consider neural network model parameters $\boldsymbol{\theta}$ and data consisting of input features $X = \{\mathbf{x_1}, \ldots, \mathbf{x_M}\}$ and target labels $Y = \{y_1, \ldots, y_M\}$. The model prediction is given by the objective function as $f(\mathbf{x_i}, \boldsymbol{\theta})$. During the training process, the model tries to learn the objective function for pair $(\mathbf{x_i}, y_i)$, where $\mathbf{x_i} \in X$ and $y_i \in Y$. From a probabilistic viewpoint, $f(\mathbf{x_i}, \boldsymbol{\theta})$ can be written as $P(y_i|\mathbf{x_i}, \boldsymbol{\theta})$.

In time-series prediction and regression problems, we use a Gaussian likelihood as shown in Eq. (3). The likelihood is the multivariate normal probability density function and is given by

$$p(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta}) = -\frac{1}{(2\pi\tau^2)^{M/2}} \times \exp\left(-\frac{1}{2\tau^2}\sum_{i=1}^{M}(y_i - E(y_i|\mathbf{x}_i, \boldsymbol{\theta}))^2\right) \quad (3)$$

where $E(y_i|\mathbf{x}_i, \boldsymbol{\theta})$ is given by the output of the neural network.

We assume that the elements of $\boldsymbol{\theta}$ are independent *apriori* and also assume that the weights ($\mathbf{w}$, $\mathbf{v}$), and biases ($\boldsymbol{\beta}$) have a normal distribution (with mean $\mu = 0$ and a user-defined variance σ^2). We model the prior distribution for τ^2 as an inverse-Gamma distribution. We consider a simple feedforward neural network with one hidden layer where the number the number of parameters can be given as $L = ((D + 1)H + (H + 1)O)$, for input D, hidden H and output layer O, respectively. The prior probability distribution for the neural network parameters θ with mean $\mu = 0$ is given by,

$$p(\boldsymbol{\theta}) \propto \frac{1}{(2\pi\sigma^2)^{L/2}} \times \exp\left\{-\frac{1}{2\sigma^2}\sum_{l=1}^{L}\theta_l^2\right\} \times p(\boldsymbol{\tau^2}) \quad (4)$$

Given that ν_1 and ν_2 are user defined quantities, the prior probability $p(\tau^2)$ can be formulated as,

$$p(\boldsymbol{\tau^2}) \propto \tau^{2(1+\nu_1)} \times \exp\left(\frac{-\nu_2}{\tau^2}\right) \quad (5)$$

The likelihood function for regression problems is defined using parameters $\boldsymbol{\theta} = (\mathbf{w}, \mathbf{v}, \boldsymbol{\beta}, \boldsymbol{\tau}^2)$, with $\boldsymbol{\beta} = (\beta_o, \beta_h)$; where β_o and β_h are the biases for the output layer o and hidden layer h, respectively. Symbol $\mathbf{w}$ represents the weights between input and hidden layers indexed by d and h, respectively. $\mathbf{v}$ represents the weights between the hidden and output layers indexed by h and o, respectively.

In the case of discrete outcomes emerging from pattern classification problems, we adopt a multinomial distribution to model the likelihood. Given K classes in data, we assume that the data, $\mathbf{y} = (y_1, \ldots, y_M)$ are generated from a multinomial distribution with parameter vector $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_K)$, where $\sum_{k=1}^{K} \pi_k = 1$. The likelihood is given using 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 $i = 1, \ldots, M$ and $k = 1, \ldots, K$. The likelihood function is then

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

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 the probability (π_k) and the input features $\mathbf{x}$ is modelled as a multinomial logit function,

$$\pi_k = \frac{\exp(f_k(\mathbf{x}, \boldsymbol{\theta}))}{\sum_{j=1}^{K} \exp(f_j(\mathbf{x}, \boldsymbol{\theta}))} \tag{8}$$

where, $f_j(\mathbf{x}, \boldsymbol{\theta})$ is the jth output of the neural network.

Note that in the multinomial likelihood, τ^2 is not present, $\boldsymbol{\theta} = (\mathbf{w}, \mathbf{v}, \boldsymbol{\beta})$. Thus, the prior density function is given as

$$p(\boldsymbol{\theta}) \propto \frac{1}{(2\pi\sigma^2)^{L/2}} \times \exp\left\{-\frac{1}{2\sigma^2} \sum_{l=1}^{L} \theta_l^2\right\} \tag{9}$$

We note that we use log-likelihood in our implementation to ensure we do not have numerical instabilities as we will deal with a large number of model parameters.

3.1.1. Particle-based MCMC proposals

A swarm of N particles in PSO, given as $\Omega = (\theta_1, \ldots, \theta_N)$, is initialized randomly by drawing samples from a known distribution with each particle θ_n having its own velocity. During optimization, these particles improve using specialised operators and fitness function and move towards the optimal solution. The movement of each particle is guided by two values, i.e. the best position achieved so far by the particle, and the best position achieved overall by any particle in the swarm. The new position for n^{th} particle is proposed using the current position of the particle θ_n^c, and proposal velocity $\bar{v}_n^p$ which are computed as follows:

$$\bar{v}_n^p = \phi \times v_n^c + \left(z_1 \times r_1 \times \left(\hat{\theta}_n - \theta_n^{c)}\right)\right) + \left(z_2 \times r_2 \times \left(\hat{\theta} - \theta_n^c\right)\right) \tag{10}$$

$$\bar{\theta}_n^p = \theta_n^c + \bar{v}_n^p \tag{11}$$

In the above equations, $\hat{\theta}_n$ is the best position for nth particle observed so far, while $\hat{\theta}$ is the best position for any particle in the swarm. In Eq. (10), ϕ, c_1, and c_2 are constants and assigned empirically; whereas, $r_1 \in (0, 1)$ and $r_2 \in (0, 1)$ are random values drawn from a uniform distribution.

We note that PSO is an optimization method while MCMC is a sampling method and neural network training can be approached either via sampling or optimization. Neuroevolution essentially takes an optimization viewpoint and does not cater for uncertainty quantification. MCMC sampling provides uncertainty quantification by representing the parameters (weights) as probability (posterior) distributions rather than single-point estimates by optimization methods. We can use PSO directly for neuroevolution [74,41] and an MCMC algorithm for sampling neural network weights [10,12]. Our proposed Bayesian neuroevolution features the synergy of two worlds, where neuroevolution (using PSO) is leveraged by MCMC sampling so that the neural weights and biases are represented as posterior distributions. We choose PSO as its intuitively easier to mathematically describe and has good convergence properties [75]. However, any evolutionary optimization algorithm can be used in principle for Bayesian neuroevolution.

Conventionally in MCMC sampling, we can generate the proposal for the parameters in the n^{th} position in the chain from a multivariate normal distribution as shown in Eq. (12)

$$\theta_n^p \sim \mathcal{N}(\bar{\theta_n}^p, \Sigma_\theta) \tag{12}$$

where, $\Sigma_\theta = \sigma_\theta^2 I_L$ is the covariance matrix and I_L is the $L \times L$ identity matrix. L represents the number of parameters in the neural network.

In our proposed Bayesian neuroevolution, we essentially accept/reject proposals created by the particles in swarm (PSO) using the Metropolis–Hastings criterion. The Metropolis–Hastings acceptance ratio of the proposal for nth particle is generated for single-chain MCMC can be given as

$$\alpha = \min\left\{1, \frac{p(\mathbf{y}|\mathbf{x}, \theta_n^p)p(\theta_n^p)q(\theta_n^c|\theta_n^p)}{p(\mathbf{y}|\mathbf{x}, \theta_n^c)p(\theta_n^c)q(\theta_n^p|\theta_n^c)}\right\} \tag{13}$$

where, $p(\mathbf{y}|\mathbf{x}, \theta_n^p)$ and $p(\mathbf{y}|\mathbf{x}, \theta_n^c)$ are the likelihood (Eqs. (3) or (7)) of the proposal and the last accepted sample, respectively. We calculate the prior densities $p(\theta_n^p)$ and $p(\theta_n^c)$ using Eqs. (4) or (9). The probability of reversible jump is computed using the q-ratio given by the ratio of $q(\theta_n^c|\theta_n^p)$, and $q(\theta_n^p|\theta_n^c)$ with details shown in Eq. (12).

In the case of random-walk proposals, the q-ratio cancels out since we assume the detailed balance condition holds when proposals are symmetric. In the case of Langevin-gradients and particle-based proposals from PSO, the particle updates (proposals) are not symmetric and we need to ensure that the detailed balance condition holds so the q-ratio is computed and becomes part of the Metropolis–Hastings acceptance ratio.

3.2. Particle-based proposals in tempered MCMC

We first revisit the literature of evolutionary MCMC sampling and canonical neuroevolution in order to develop the foundations of Bayesian neuroevolution. Fig. 2(b) shows a general architecture to leverage evolutionary MCMC algorithms to evolve neural network weights and biases. This enhances the neuroevolution architecture, shown in Fig. 2(a), by introducing an MCMC sampling step. Our implementation of Bayesian neuroevolution is based on the family-competitive evolutionary MCMC architecture. As a result, proposals are generated at the member level. In this case, Gaussian noise is added to individual members of the evolved population which are then accepted/rejected using a Metropolis–Hastings acceptance ratio. Therefore, the MCMC chains are maintained at the member-level which interact with each other in order to generate evolutionary proposals. At the end of sampling, the samples from the individual chains are combined to get the posterior distribution.

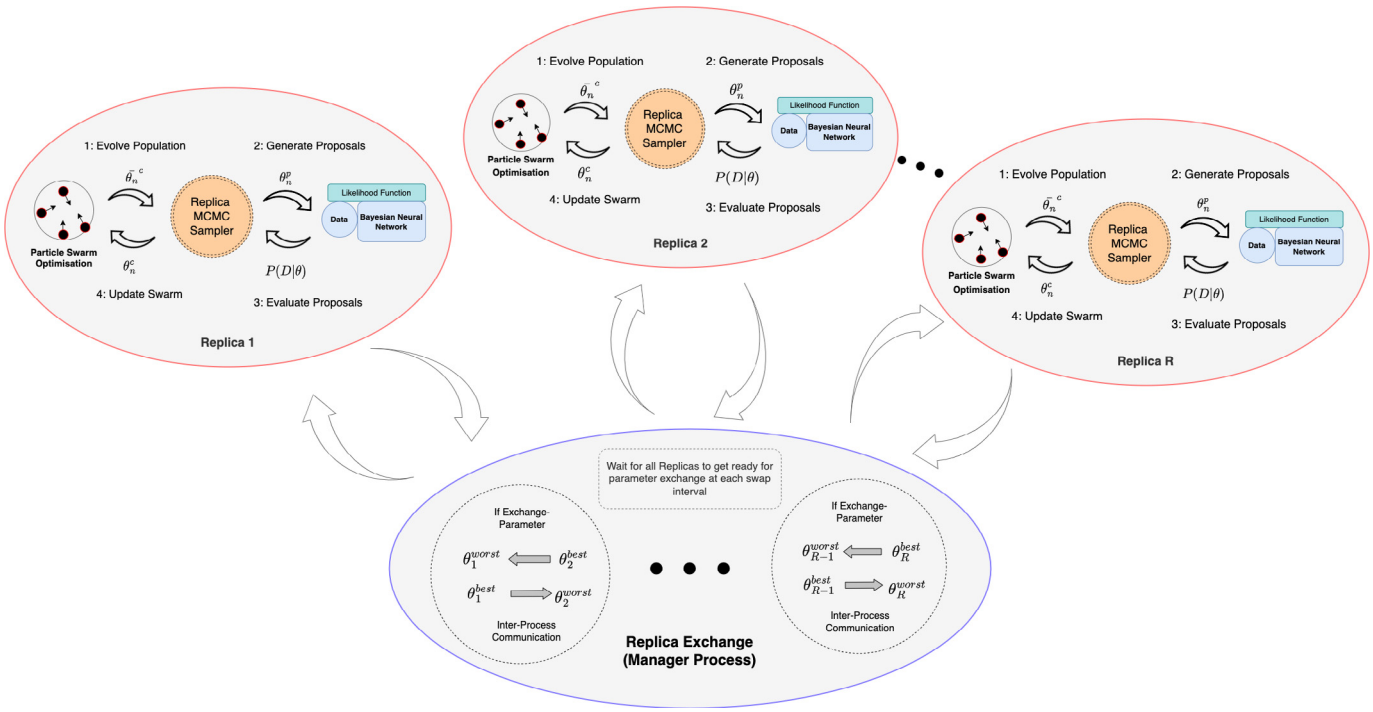

Fig. 1. The architecture of particle-based tempered MCMC featuring a parallel computing environment with inter-process communication. Given available cores, the algorithm tries to run each replica on a separate processing unit that communicates with swarms within the same unit.

Our Bayesian neuroevolution framework features particle-based proposals in tempered MCMC (Particle-tMCMC) implemented in a parallel computing environment with inter-process communication. In this implementation, we use PSO to generate efficient proposals via particles in swarm rather than random-walk proposal distribution in canonical MCMC.

The Bayesian neuroevolution framework consists of an ensemble of tempered MCMC replicas (R), where each replica correspond to a predefined temperature level (T) from a user-defined geometric ladder. The state of the ensemble is represented by $\boldsymbol{\Omega} = (\Omega_1, \ldots, \Omega_R)$, where $\Omega_i = (\theta_1, \ldots, \theta_N)$ represents a swarm of N particles. Each particle in the swarm corresponds to a vector of neural network parameters sampled via MCMC.

We represent the particle-based proposals in the swarm by θ_n^p, where $n \in \{1, \ldots, N\}$. This proposal is drawn from a known proposal distribution $q(\theta)$ and accepted/rejected with probability α via Metropolis–Hastings condition. Proposal distribution $q(\theta)$ is a multivariate normal with mean as the evolved particle state. We ensure that the chain is reversible and has a stationary distribution, $p(\theta|\mathbf{y})$. We note that a separate instance of the swarm is executed in each processing core with the given replica sampler. There is communication between the MCMC replica sampler and the swarm to create new particle-based proposals. Fig. 1 shows the framework highlighting the population of solutions used by the evolutionary algorithm for creating proposals for the respective replica state configuration in tempered MCMC.

We provide further details in Algorithm 1 which begins by constructing the geometric temperature ladder to define temperature levels for respective replicas. We first assign the hyper-parameters, i.e. the *number of samples and replicas, swap interval, and the swarm size*. We then initialize the swarm of particles for each replica by drawing particles from a known prior distribution. We see that initially the tempered MCMC is used for global exploration which transforms to canonical but parallel MCMC with replica temperature of 1. The local exploration also features the exchange of neighbouring replicas in the same way as done in global exploration.

Algorithm 1 begins sampling by executing each of the replicas in parallel (Step 1) where, a new proposal θ_n^p is generated using either random-walk or particle-based proposal distribution. In particle-based proposals, PSO is used to compute the updated particle velocity and position, as shown in Eqs. (10) and (11). Then we draw and add noise to the proposal using multivariate normal distribution as shown in Eq. (12). This is done for all the particles in each replica which have their own swarms to create particle-based proposals. We update the particles within a replica when the respective proposal is accepted using the Metropolis–Hastings acceptance criterion given as

$$\alpha = \min\left(1, \frac{p(\theta_n^p|\mathbf{y})^{\frac{1}{T_r}}\, q(\theta_n^c|\theta_n^p, r)}{p(\theta_n^c|\mathbf{y})^{\frac{1}{T_r}}\, q(\theta_n^p|\theta_n^c, r)}\right) \tag{14}$$

Eq. (14) is a variation of Eq. (13) with the addition of parameters r and T_r, representing the replica and the temperature value of the replica, respectively. Here, the proposal is conditional on replica r while the posterior is scaled using the temperature T_r. If the proposal is accepted, it 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 neighbour exchange interval (swap interval) is reached; then the algorithm evaluates if the neighbouring replicas need to be exchanged using the Metropolis–Hastings acceptance criterion as done for within replica proposals (Step 2). This neighbourhood exchange is also referred to as *replica transition*.

During the replica transition, we replace the worst particle in the population with the best particle from the adjacent chain. This differs from the typical neighbourhood exchange seen in previous tempered MCMC implementation [12]. The replacement is done with probability γ in Step 2 which is given by

$$\gamma = \min\left(1, \frac{p(\theta_{best}^p|\mathbf{y}, r^p)q(r^c|r^p)}{p(\theta_{worst}^c|\mathbf{y}, r^c)q(r^p|r^c)}\right) \tag{15}$$

Here, γ is the probability of transitioning from worst particle of replica r^c to the best particle of neighbouring replica r^p, represented by θ_{worst}^c and θ_{best}^p, respectively. We propose the exchange

Result: Draw samples from $p(\theta|\mathbf{y})$
Note: The manager process is highlighted in blue while the ensemble of replica processes running on parallel processing cores are highlighted in red.
i. Set number of replicas (R), number of samples for each replica (S_{max}), swarm size for each replica (N), and swap interval (I)
ii. Initialise R replicas, with particle swarms $[\Omega_1, \ldots, \Omega_R]$ with temperatures $[T_1, \ldots, T_R]$
iii. Set the current value of swarm, $\Omega^c = \Omega^{[0]}$
iv. Set number of warm-up samples Γ
v. Set percentage of samples for global exploration phase using parallel tempering
while S_{max} **do**
  for $r = 1, 2, \ldots, R$ **do**
    if *global is true* **then**
      $T_{r^c} = T_r$;
    else
      $T_{r^c} = 1$;
    end
    for $s = 1, 2, \ldots, S_{max}$ **do**
      Step 1: Evolutionary Sampling
      for $n = 1, 2, \ldots, N$ **do**
        1.1: Propose new positions for particles
        if $s <= \Gamma$ **then**
          Propose new particle positions θ_n^p for replica r, using Random-Walk proposals;
        else
          Propose new particle position values θ_n^p for replica r:
          - Update velocity of particle, $\bar{v}_n^p$, as given in Eq 10.
          - Update position of particle, $\bar{\theta}_n^p$, from the velocity, as given in Eq 11.
          - Propose θ_n^p by adding Gaussian noise to $\bar{\theta}_n^p$, as given in Eq 12.
        end
        1.2: Calculate the acceptance probability α as shown in Equation 14
        1.3 Accept/Reject particle proposals
        Draw from Uniform $u \sim \mathcal{U}[0, 1]$:
        if $u < \alpha$ **then**
          Set $\theta_n^{[s]} = \theta_n^p$
        else
          Set $\theta_n^{[s]} = \theta_n^c$
        end
      end
      Step 2: Neighbourhood Exchange
      if *s is a multiple of I* **then**
        Replace the worst particle of the current chain with the best particle from the adjacent chain.
        2.1 Compute swarm particle exchange acceptance probability in Equation 15;
        2.2 Draw $u \sim U[0, 1]$
        if $u < \gamma$ **then**
          Accept the best particle from replica r^p
          $\theta_{worst}^{[s]} \leftarrow \theta_{best}^p$;
        else
          Retain particle from the current replica r^c:
          $\theta_{worst}^{[s]} \leftarrow \theta_{worst}^c$;
        end
      end
    end
  end
end
Step 3: Combine the posterior from all the replicas in an ensemble

Algorithm 1: Bayesian neuroevolution via participle-based proposals in tempered MCMC.

between neighbouring replicas such that when $r^c = r$, then $r^p = r + 1$, and vice-versa.

The architecture enables each replica to be executed on a separate processing unit which communicates with the population of the evolutionary algorithm within the same unit. The main process manages the replicas and provides the mechanism for the neighbourhood exchange given the swap interval and acceptance criterion of exchange is satisfied. The main process waits for all replicas to sample until the swap interval is reached which determines neighbouring replica-exchange based on the Metropolis–Hastings criterion. The main process notifies the replicas after the exchange of replicas to resume sampling with the latest configuration. The process continues until the maximum sampling time is reached. We use Python multiprocessing library [76] and provide the software code online.[1]

4. Experiments and results

4.1. Experimental setup

We use a feedforward neural network model with single hidden layer ($h = 5$ neurons) for time-series prediction problems as done in previous work [12]; further details are presented in Table 1. We use 6 replicas in the tempered MCMC implementations which is also motivated by previous work [12] where 6 replicas were empirically found to show best performance in parallel computing environment. The random-walk (RW) proposal

[1] https://github.com/sydney-machine-learning/pt-bayesian-neuroevolution

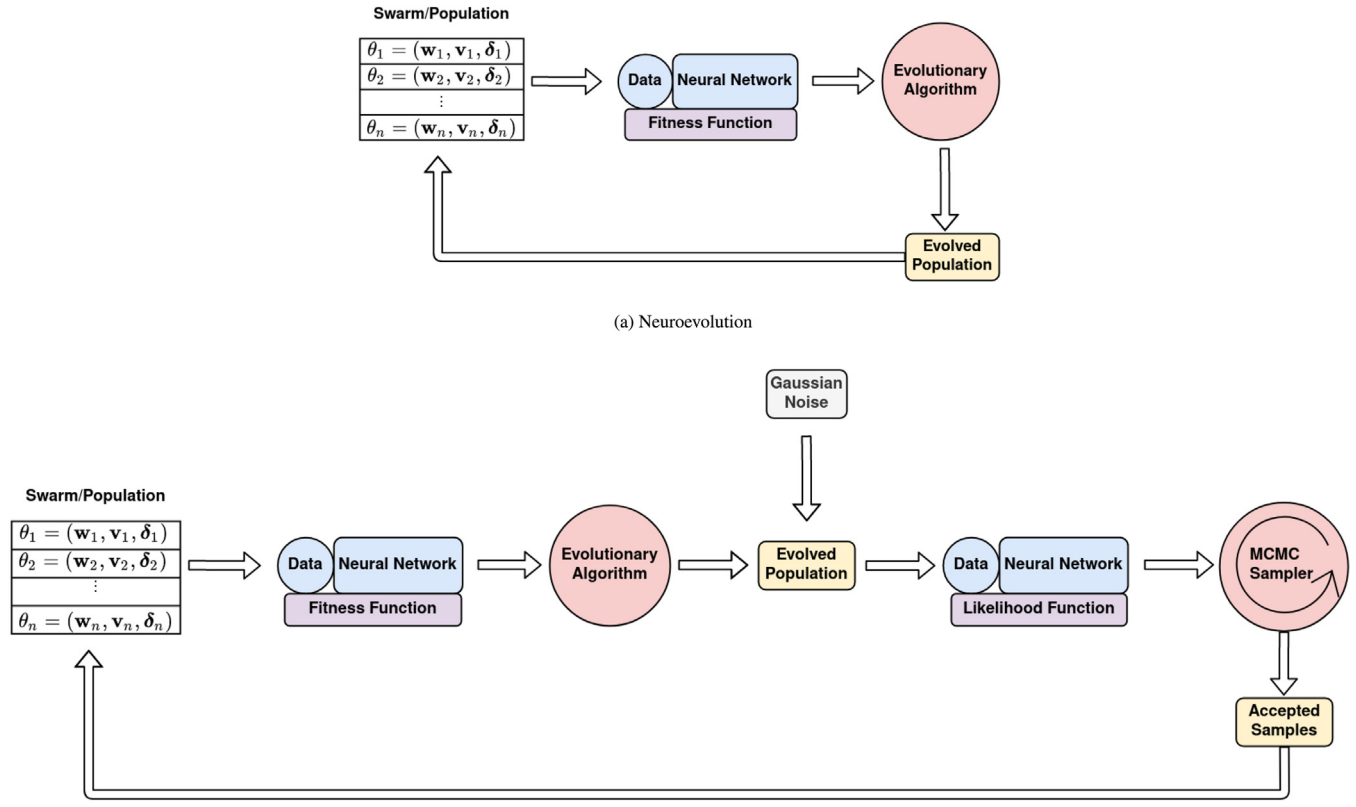

Fig. 2. Canonical neuroevolution (a) vs Bayesian neuroevolution via MCMC (b) where a likelihood function is used in conjunction with the fitness function.

Table 1
Classification dataset description.

Dataset	Instances	Attributes	Classes	Hidden
Iris	150	4	3	12
Cancer	569	9	2	12
Bank-Market	11162	20	2	50
Chess	28056	6	18	25

distribution uses a Gaussian noise drawn from a normal distribution $\mathcal{N}(0, 0.05)$. We note that values of the aforementioned parameters were assigned empirically. In addition to the neural network parameters (weights and biases), tempered MCMC is used to sample η in log-space, that denotes the variance τ^2 of the likelihood density, shown in Eq. (3). The user–defined parameters of the Gaussian and inverse-Gamma priors (see Eqs. (4) and (9)) were set as $\sigma^2 = 50$, $\nu_1 = 0$ and $\nu_2 = 0$.

We execute tempered MCMC in the first 50% of the samples which are discarded as burn-in, as done in our previous work [12]. We select a replica swap interval as the swarm size, which means that after every generation of PSO, we consider a neighbour exchange. We execute tempered MCMC-PSO with a swarm size of 20 while the geometric temperature ladder is defined in the range [1, 2] and a warm-up period at the start of sampling is used for optimising the swarm of particles. The warm-up period represented by Γ, consists of a small number of initial samples where we use random-walk proposals in each replica. In our case, we set the value of Γ to 100. We note that this is not standard in MCMC literature; however, our trial experiments revealed that warm-up samples improves the sampling efficiency and performance of Particle-tMCMC.

We consider four pattern classification problems from the University of California Irvine machine learning repository [77]. To test the proposed method on problems of varying size and complexity, the datasets were selected based on the number of features and instances in order to represent accordingly, problems of varying size and complexity as shown in Table 1. The maximum MCMC sampling time is set to 50,000 samples for these problems.

The benchmark problems used for performance analysis on time-series prediction are Sunspot, Lazer, Mackey-Glass, Lorenz, Henon and Rossler time-series. We apply Takens' embedding theorem for data reconstruction (windowing) with dimension, $D = 4$ and time lag, $T = 2$; these values are also used in previous work [12]. These problems used the first 1000 data points from which the first 60% was used for training and the remaining for testing. The prediction performance is measured by root mean squared error. The maximum MCMC sampling time is set to 100,000 samples for the respective time-series problems which is in line with previous research [12].

We take the following steps to ensure that the performance of our proposed Bayesian neuroevolution framework is properly evaluated.

- Evaluate the effect of particle-based proposals for selected problems;
- Evaluate the effect of population size of the swarm for selected problems;
- Compare accuracy performance of random-walk proposal distribution in tempered MCMC (RW-tMCMC) with Particle-tMCMC for selected problems;
- Provide convergence diagnosis for a selected problems;
- Compare time performance with Langevin gradient-based (LG-tMCMC) given by Chandra et al. [12].

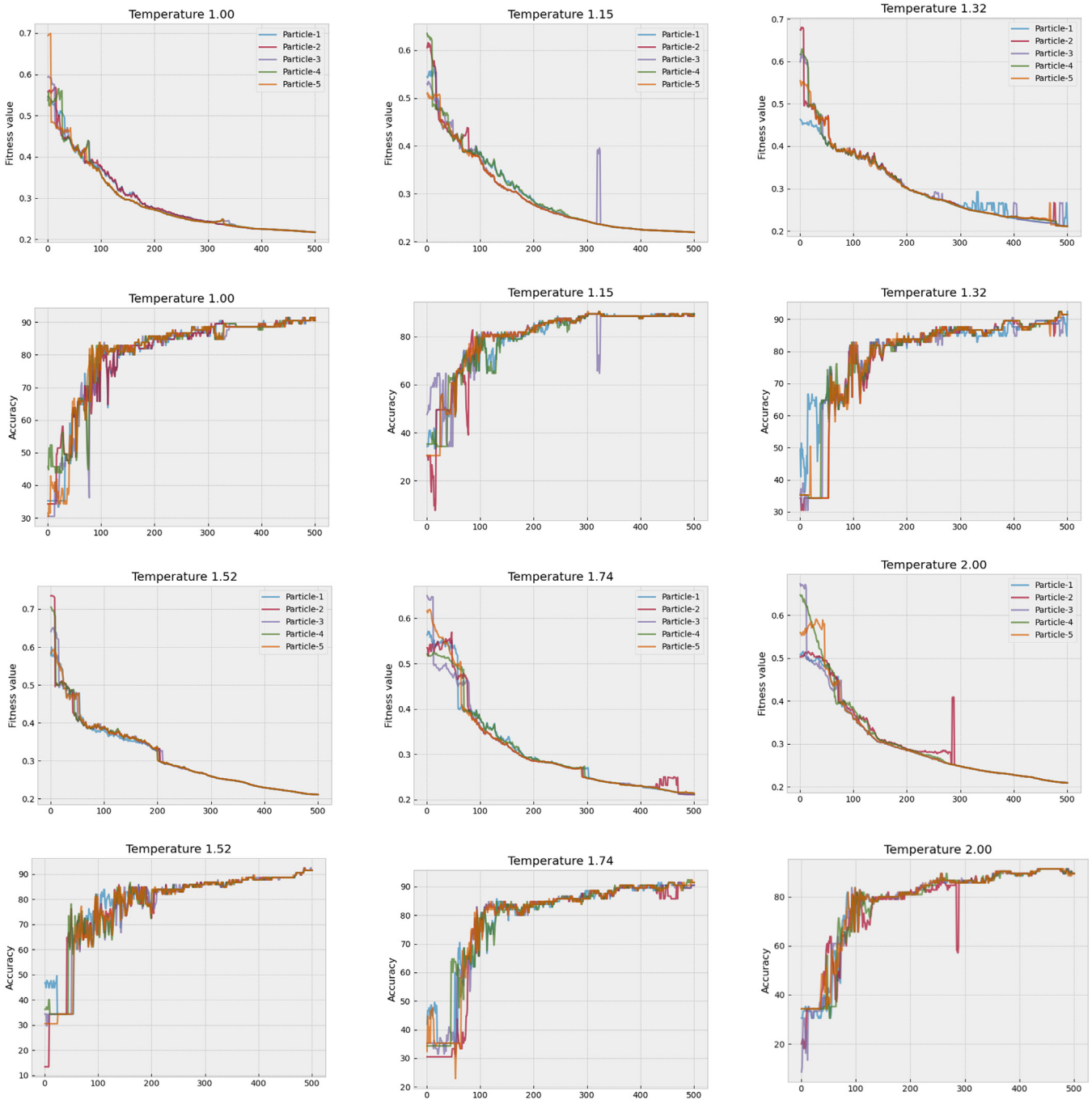

Fig. 3. Convergence plot showing the Fitness and accuracy values for Top 5 particles in the swarm of each replica for Iris problem.

4.2. Preliminary investigation

The influence of swarm size on the accuracy for Sunspot time-series prediction and Cancer classification problems is shown in Tables 2 and 3. We notice that in both cases, a modest swarm size (10 or 20 particles) delivers the highest performance and that as the swarm size grows, the performance deteriorates. A larger swarm size generally implies increased global search features. Since we are dealing with convex optimization with small-scale problems, more emphasis is required on local exploitation which is offered by smaller swarms.

Table 2
Effect of swarm size on RMSE performance for Sunspot time-series.

Swarm size	Train RMSE	Test RMSE
10	0.0176, 0.0050, 0.0140	0.0157, 0.0040, 0.0116
20	0.0184, 0.0113, 0.0138	0.0166, 0.0087, 0.0126
30	0.0214, 0.0059, 0.0168	0.0194, 0.0060, 0.0145
40	0.0262, 0.0186, 0.0153	0.0247, 0.0178, 0.0140
50	0.0235, 0.0066, 0.0157	0.0213, 0.0058, 0.0144

Table 3
Effect of swarm size on accuracy performance for Cancer problem.

Swarm size	Train accuracy	Test accuracy
10	95.70, 8.18, 98.10	91.43, 6.26, 94.27
20	95.71, 8.15, 98.10	91.42, 5.96, 94.48
30	89.49, 18.72, 98.57	86.85, 13.95, 94.48
40	76.99, 29.98, 98.10	77.04, 21.86, 94.27
50	55.88, 29.83, 98.57	61.45, 20.90, 94.27

4.3. Results

We provide an experimental evaluation of tempered MCMC methods such as LG-tMCMC, RW-tMCMC, and Bayesian neuroevolution via Particle-tMCMC for selected problems. We also present an accuracy comparison of these methods with the Adam optimizer and stochastic gradient descent (SGD).

Table 4 shows the performance evaluation results for the time-series prediction problems. We observe that for real-world problems (Lazer and Sunspot), Particle-tMCMC results were comparable to LG-tMCMC and were significantly better than RW-tMCMC. In the case of some of the simulated problems such as Mackey and Rossler, we find that LG-tMCMC and RW-tMCMC produce slightly better performance when compared to Particle-tMCMC. Note that these problems feature no noise in the dataset. We next consider a more complex problem (Henon) and find that Particle-tMCMC significantly improved accuracy performance over RW-tMCMC and LG-tMCMC. Overall, Particle-tMCMC produces a similar level of accuracy performance with uncertainty quantification to LG-tMCMC and RW-tMCMC, and performed considerably better compared to SGD and Adam optimizers.

Table 5 shows the performance of Particle-tMCMC on selected classification problems. In smaller problems such as Iris and Cancer, the minimum and maximum values θ is set to $[-5, 5]$. In larger and more complex problems such Bank and Chess, we found that a smaller limit of $[-2, 2]$ for θ gave better results. In the case of Iris and Cancer, Particle-tMCMC performed considerably better than RW-tMCMC and LG-tMCMC, but not as good as Adam and SGD. In the Bank problem, Particle-tMCMC outperformed the rest, while giving sub-optimal results for Chess. In general, Particle-tMCMC shows a significant boost to the computational time over LG-tMCMC for both pattern classification and time-series prediction problems. This is because Particle-tMCMC eliminates the time needed to perform gradient computation.

We show the convergence plots of fitness and classification accuracy values of top 5 particles in each replica using the train of Iris problem in Fig. 3. The plots show that the particles have converged in terms of training accuracy. Furthermore, Fig. 4 shows the trace-plot of mean swarm RMSE [(a), (d)] and mean of weights in the input-hidden layer for each replica [(b), (e)], along with the typical posterior distribution [(c), (f)] for Iris and Lazer problem using Particle-tMCMC. Figs. 4(b) and 4(e) compare the posterior distribution of Particle-tMCMC with LG-tMCMC. While both the posteriors are multi-modal in nature, the varying position of the peeks suggest that Particle-tMCMC and LG-tMCMC converge to slightly different posterior distributions, while achieving similar performance (RMSE).

4.4. Convergence diagnosis

Getting better precision accuracy does not necessarily mean that the MCMC method has converged [78–80] and given the hybrid nature of Particle-tMCMC, convergence diagnosis is essential. The Gelman-Rubin diagnostic [81] evaluates MCMC convergence by analyzing the behaviour of multiple Markov chains. We can implement assessment by comparing the estimated between-chains and within-chain variances for each parameter, given multiple chains from different experimental runs. The large differences between the variances indicate non-convergence. We provide convergence diagnosis for the proposal distribution by analysing multiple replica chains for a single experimental run for selected problems shown in Table 6. We calculate the potential scale reduction factor (PSRF) which gives the ratio of the current variance in the posterior variance for each parameter compared to that being sampled. Table 6 shows the PSRF values for the first three weights of the neural network (θ_1, θ_2, and θ_3) along with the mean and standard deviation PSRF value overall weights ($\boldsymbol{\theta}$). The values for the PSRF near 1 indicates convergence and we find that all the respective problems have converged for Particle-tMCMC. In comparison with LG-tMCMC, we observe that Particle-tMCMC results in lower PRSF values for neural network weights in 3 out of 4 problems; indicating better overall convergence by Particle-tMCMC.

5. Discussion

In Bayesian neuroevolution via Particle-tMCMC, it is essential that each particle of the population is given a sufficient number of sampling steps to converge to an optimal solution. Our preliminary investigation revealed that with an increase of the population size while keeping the number of replica samples constant, the accuracy decreases for both classification and time-series problems. Selecting the right set of hyper-parameters such as the population size, the maximum number of samples and the number of parallel replicas is essential for optimal results.

Particle-tMCMC provides an enhanced form of exploration where exploration via multiple temperature levels in parallel tempering is leveraged with a swarm of particles within each replica. Particle-tMCMC provided better results when compared to random walk-proposal distribution for most of the problems. When compared to Langevin-gradients (LG-tMCMC), the results from the Bayesian neuroevolution were quite close in terms of accuracy while consuming significantly less amount of time, with an average difference of approximately 9 minutes. Thus, the results show that the addition of evolutionary proposals is computationally less expensive than Langevin-gradients while generating optimal results for most problems. We further note that we have given a computation time of 1 experimental run for Adam and SGD; however, proper uncertainty quantification takes 30 experimental runs. On the other hand, our Bayesian neuroevolution method requires one run since uncertainty quantification is incorporated automatically via Bayesian inference.

The detailed balance condition is a critical factor in the design of novel MCMC methods. When considering heuristic-based proposal distributions, it is essential to ensure the proposal distribution satisfies the detailed balance condition [82]. We note that in the literature, evolutionary MCMC methods such as differential evolution MCMC and others [15,16] assumed that the q-ratio (Eq. (13)) in the acceptance criterion cancels out. We based our implementation on the Langevin-gradient based proposal distribution where the q-ratio takes into account gradient-based proposals which are not symmetric when compared to random-walk proposals. The same issue arises when we are using particle-based proposals, and hence we used the same framework as shown in Eq. (13). Further work can analyse convergence issues given by the different tempered MCMC samplers used in this study.

We also note that it is essential that the MCMC chain is ergodic which states that an arbitrarily selected portion of samples from a chain must represent the average statistical properties of the entire chain. Hence, convergence diagnosis is essential [80]; we provided Gelman-Rubin convergence diagnosis and auto-correlation time could provide further analysis. We note that given that the

Table 4
Performance accuracy on different time-series datasets.

Problem	Method	Train accuracy [mean, std, best]	Test accuracy [mean, std, best]	Swapped (%)	Accepted (%)	Time (min)
Lazer	RW-tMCMC	0.0640, 0.0218, 0.0325	0.0565, 0.0209, 0.0270	42.26	35.31	4.53
	LG-tMCMC [12]	0.0383, 0.0187, 0.0240	0.0353, 0.0161, 0.0212	48.45	19.91	11.53
	Particle-tMCMC	0.0422, 0.0151, 0.0301	0.0390, 0.0113, 0.0305	10.93	20.00	2.54
	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
Henon	RW-tMCMC	0.1230, 0.0296, 0.0167	0.1198, 0.0299, 0.0161	48.58	38.08	4.21
	LG-tMCMC [12]	0.0201, 0.0025, 0.0146	0.0190, 0.0029, 0.0131	47.43	11.39	11.41
	Particle-tMCMC	0.0667, 0.0363, 0.0216	0.0645, 0.0371, 0.0197	16.58	6.11	2.58
	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
Lorenz	RW-tMCMC	0.0192, 0.0033, 0.0113	0.0171, 0.0037, 0.0094	39.48	14.48	4.45
	LG-tMCMC [12]	0.0181, 0.0018, 0.0117	0.0157, 0.0018, 0.0094	50.37	9.66	11.48
	Particle-tMCMC	0.0163, 0.0070, 0.0088	0.0143, 0.0070, 0.0074	18.93	8.88	2.53
	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	RW-tMCMC	0.0173, 0.0011, 0.0144	0.0175, 0.0011, 0.0148	48.11	12.53	4.22
	LG-tMCMC [12]	0.0172, 0.0008, 0.0154	0.0175, 0.0009, 0.0155	39.57	8.58	11.60
	Particle-tMCMC	0.0230, 0.0057, 0.0166	0.0246, 0.0071, 0.0163	23.91	13.30	2.56
	SGD	0.0296, 0.0033, 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
Mackey	RW-tMCMC	0.0060, 0.0005, 0.0051	0.0061, 0.0005, 0.0051	42.11	8.19	4.59
	LG-tMCMC [12]	0.0061, 0.0009, 0.0047	0.0062, 0.0009, 0.0048	49.10	5.72	11.68
	PSO-tMCMC	0.0083, 0.0040, 0.0049	0.0083, 0.0039, 0.0050	35.52	8.66	2.55
	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
Sunspot	RW-tMCMC	0.0242, 0.0041, 0.0170	0.0239, 0.0050, 0.0161	44.45	18.30	4.82
	LG-tMCMC [12]	0.0199, 0.0031, 0.0155	0.0239, 0.0050, 0.0161	48.45	12.57	11.61
	Particle-tMCMC	0.0195, 0.0059, 0.0152	0.0184, 0.0056, 0.0138	28.45	9.13	2.64
	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

Table 5
Performance accuracy on different classification datasets.

Problem	Method	Train accuracy	Test accuracy	Swapped (%)	Accepted (%)	Time (s)
		[mean, std, best]	[mean, std, best]			
Iris	RW-tMCMC	51.39, 15.02, 91.43	50.18, 41.78, 100.0	52.56	95.32	1.26
	LG-tMCMC [12]	97.32, 00.92, 99.05	96.76, 0.96, 99.10	51.77	97.55	2.09
	Particle-tMCMC	88.00, 2.93, 92.38	91.06, 2.34, 93.33	24.45	16.72	2.89
	SGD	99.11, 00.23, 100.0	96.92, 1.05, 97.50			0.60
	Adam	99.61, 0.46, 1.00	96.83, 0.11, 97.50			0.07
Cancer	RW-tMCMC	83.78, 20.79, 97.14	83.55, 27.85, 99.52	40.18	89.71	2.78
	LG-tMCMC [12]	97.00, 0.29, 97.75	98.77, 0.32, 99.52	49.25	94.67	5.09
	Particle-tMCMC	96.20, 8.37, 98.57	91.98, 6.24, 94.27	26.77	15.13	2.21
	SGD	99.11, 0.23, 100.0	96.92, 1.05, 97.5			0.94
	Adam	99.49, 0.47, 100.0	96.67, 1.18, 97.5			0.17
Bank Marketing	RW-tMCMC	78.39, 1.34, 80.11	77.49, 0.90, 79.45	49.13	61.59	27.71
	LG-tMCMC [12]	80.75, 1.45, 85.41	79.96, 0.81, 82.61	50.00	31.50	86.94
	Particle-tMCMC	81.35, 1.16, 83.44	80.20, 0.89, 81.91	28.17	27.07	52.03
	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
Chess	RW-tMCMC	89.48, 17.46, 100.0	90.06, 15.93, 100.0	48.09	69.09	252.46
	LG-tMCMC [12]	100.0, 0.00, 100.0	100.0, 0.0, 100.0	50.12	88.87	323.10
	Particle-tMCMC	84.87, 0.54, 84.87	85.04, 0.54, 85.04	9.85	14.36	125.54
	SGD	99.76, 0.96, 100.0	99.76, 0.97, 100.0			8.43
	Adam	100.0, 0.00, 100.0	100.0, 0.00, 100.0			1.07

proposed MCMC method uses tempered replica-sampling with particle-based proposals, there could be bias and convergence analysis of past MCMC methods may not be applicable [83], which opens directions for future research in convergence analysis given multiple replicas and a swarm of particles (proposals).

We also note that besides particle-based proposals, generative adversarial network (GANs) [84,85] could also be used for generating proposals in tempered MCMC. GANs have been used in a variety of problems such as anomaly detection, active learning, and adversarial attacks. This motivates future work of exploring GAN-based proposals in tempered MCMC frameworks to further improve sampling performance in Bayesian neural networks.

6. Conclusion and future work

In this paper, we presented a Bayesian neuroevolution framework that featured a parallel computing implementation of tempered MCMC with proposals created using particle swarm optimization. The effectiveness of this approach was evaluated on time-series prediction and classification problems. We presented a Bayesian neuroevolution framework to ensure that detailed balance condition holds in the Metropolis–Hastings acceptance criterion of MCMC sampling. The results show that Bayesian neuroevolution with a sufficient number of particles and samples

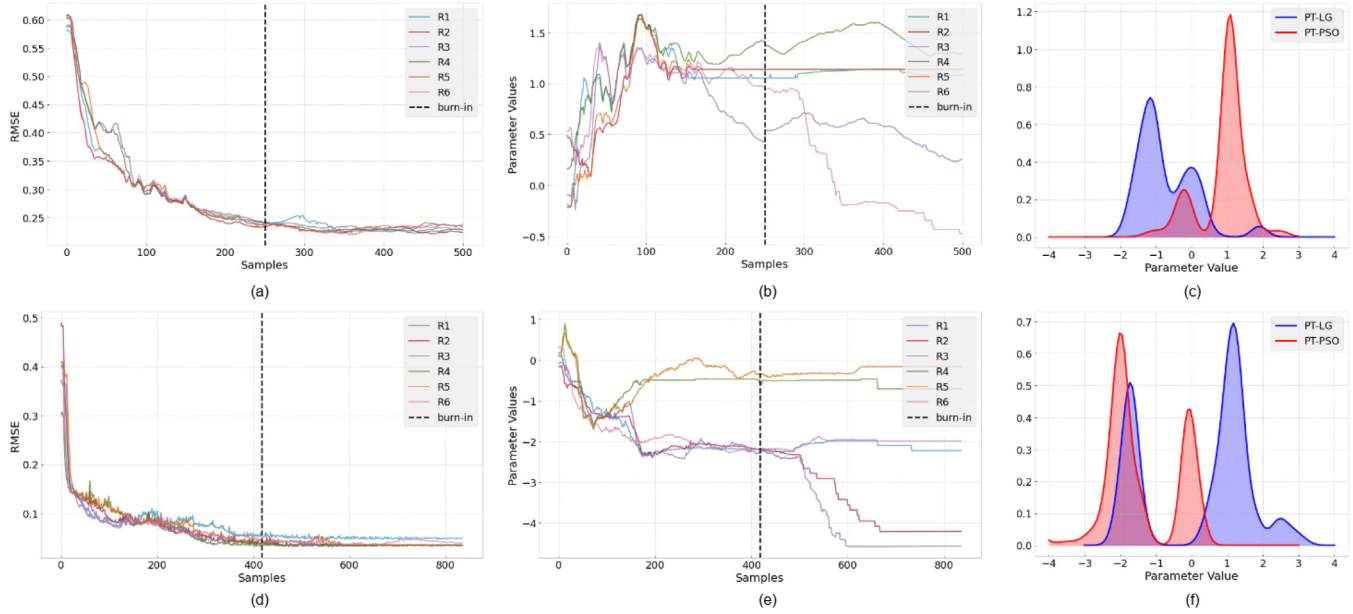

Fig. 4. Sampling trace-plot for mean swarm RMSE and weight value from the input-hidden layer of the neural network along with its posterior distribution for the Iris [(a), (b), (c)] and Lazer [(d), (e), (f)] problems.

Table 6
PSRF values for the first three weights along with the overall mean and standard deviation observed in selected classification and time-series problems.

Problem	Method	θ_1	θ_2	θ_3	Mean	Std
Lazer	Particle-tMCMC	1.017	1.031	1.036	1.189	0.190
	LG-tMCMC	1.081	1.149	1.165	2.024	1.526
Sunspot	Particle-tMCMC	1.008	1.047	1.064	1.178	0.121
	LG-tMCMC	1.158	1.230	1.238	1.934	0.969
Iris	Particle-tMCMC	1.011	1.014	1.017	1.153	0.181
	LG-tMCMC	1.017	1.024	1.028	1.250	0.199
Cancer	Particle-tMCMC	1.009	1.010	1.011	1.150	0.190
	LG-tMCMC	1.007	1.009	1.009	1.084	0.085

improve performance accuracy over random-walk, while remaining computationally less expensive when compared to our previous method known as Langevin-gradient MCMC. The proposed method achieves a significant improvement in terms of computation time with results that are comparable to Langevin-gradients in terms of accuracy. Bayesian neuroevolution framework could be seen as a remedy to offer uncertainty quantification in various neuroevolution paradigms.

Large-scale problems can face challenges when with the canonical implementation of MCMC sampling in Bayesian neural networks where random-walk proposal distribution struggles as neural network parameters increase. Langevin-gradients provide an improvement in the results by incorporating gradient-based proposals, this approach can be computationally expensive and our results show that the proposed method can be used as an alternative to Langevin-gradients especially in models where gradient information is not available. Our Bayesian neuroevolution framework is more applicable to the evolution of various neural network topologies where it is difficult to attain gradient information. Moreover, the approach can be useful for complex models, such as those in Earth and environmental science where gradient information is difficult to attain.

CRediT authorship contribution statement

Arpit Kapoor: Methodology, Software, Visualization, Writing – original draft. **Eshwar Nukala:** Software, Visualization. **Rohitash Chandra:** Supervision, Conceptualization, Writing – review & editing.

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.

Data availability

The code and data has been provided in the code ocean capsule attached along with this submission.

References

[1] D.F. Specht, Probabilistic neural networks, Neural Netw. 3 (1) (1990) 109–118.
[2] M.D. Richard, R.P. Lippmann, Neural network classifiers estimate Bayesian a posteriori probabilities, Neural Comput. 3 (4) (1991) 461–483.
[3] E.A. Wan, Neural network classification: A Bayesian interpretation, IEEE Trans. Neural Netw. 1 (4) (1990) 303–305.
[4] D.J.C. MacKay, Probable networks and plausible predictions—A review of practical Bayesian methods for supervised neural networks, Network: Comput. Neural Syst. 6 (3) (1995) 469–505.
[5] R.M. Neal, Bayesian Learning for Neural Networks, vol. 118, Springer Science & Business Media, 2012.
[6] P.J. Werbos, Backpropagation through time: What it does and how to do it, Proc. IEEE 78 (10) (1990) 1550–1560.
[7] D.E. Rumelhart, G.E. Hinton, R.J. Williams, Learning representations by back-propagating errors, Cogn. Modeling 5 (3) 1.
[8] M. Girolami, B. Calderhead, Riemann manifold Langevin and Hamiltonian Monte Carlo methods, J. R. Stat. Soc. Ser. B Stat. Methodol. 73 (2) (2011) 123–214.
[9] G.O. Roberts, J.S. Rosenthal, Optimal scaling of discrete approximations to Langevin diffusions, J. R. Stat. Soc. Ser. B Stat. Methodol. 60 (1) (1998) 255–268.
[10] R.M. Neal, et al., MCMC using Hamiltonian dynamics, in: Handbook of Markov Chain Monte Carlo, vol. 2, (11) CRC Press New York, NY, 2011.

[11] 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.
[12] R. Chandra, K. Jain, R.V. Deo, S. Cripps, Langevin-gradient parallel tempering for Bayesian neural learning, Neurocomputing (2019).
[13] M.M. Drugan, D. Thierens, Evolutionary markov chain Monte Carlo, in: International Conference on Artificial Evolution (Evolution Artificielle), Springer, 2003, pp. 63–76.
[14] M. Strens, Evolutionary MCMC sampling and optimization in discrete spaces, in: Proceedings of the 20th International Conference on Machine Learning, ICML-03, 2003, pp. 736–743.
[15] C.J.F. Ter Braak, A Markov chain Monte Carlo version of the genetic algorithm differential evolution: Easy Bayesian computing for real parameter spaces, Stat. Comput. 16 (3) (2006) 239–249.
[16] C.J.F. ter Braak, J.A. Vrugt, Differential evolution Markov chain with snooker updater and fewer chains, Stat. Comput. 18 (4) (2008) 435–446.
[17] J. Skilling, et al., Nested sampling for general Bayesian computation, Bayesian Anal. 1 (4) (2006) 833–859.
[18] R.H. Swendsen, J.-S. Wang, Replica Monte Carlo simulation of spin-glasses, Phys. Rev. Lett. 57 (21) (1986) 2607.
[19] K. Hukushima, K. Nemoto, Exchange Monte Carlo method and application to spin glass simulations, J. Phys. Soc. Japan 65 (6) (1996) 1604–1608.
[20] U.H.E. Hansmann, Parallel tempering algorithm for conformational studies of biological molecules, Chem. Phys. Lett. 281 (1–3) (1997) 140–150.
[21] J. Pall, R. Chandra, D. Azam, T. Salles, J.M. Webster, R. Scalzo, S. Cripps, Bayesreef: a Bayesian inference framework for modelling reef growth in response to environmental change and biological dynamics, Environmental Modelling & Software 125 (2020) 104610.
[22] M.D. Hoffman, A. Gelman, The no-U-turn sampler: Adaptively setting path lengths in Hamiltonian Monte Carlo, J. Mach. Learn. Res. 15 (1) (2014) 1593–1623.
[23] L. Bottou, Large-scale machine learning with stochastic gradient descent, in: Proceedings of COMPSTAT'2010, Springer, 2010, pp. 177–186.
[24] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, in: 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings, 2015.
[25] R. Chandra, A. Bhagat, M. Maharana, P.N. Krivitsky, Bayesian graph convolutional neural networks via tempered MCMC, IEEE Access 9 (2021) 130353–130365.
[26] R. Chandra, M. Jain, M. Maharana, P.N. Krivitsky, Revisiting Bayesian autoencoders with MCMC, IEEE Access 10 (2022) 40482–40495.
[27] R. Chandra, Y. He, Bayesian neural networks for stock price forecasting before and during covid-19 pandemic, PloS One 16 (7) (2021) e0253217.
[28] K. Deb, A. Anand, D. Joshi, A computationally efficient evolutionary algorithm for real-parameter optimization, Evol. Comput. 10 (4) (2002) 371–395.
[29] M. Potter, K. De Jong, A cooperative coevolutionary approach to function optimization, in: Y. Davidor, H.-P. Schwefel, R. Männer (Eds.), Parallel Problem Solving from Nature — PPSN III, in: Lecture Notes in Computer Science, vol. 866, Springer Berlin Heidelberg, 1994, pp. 249–257.
[30] J. Kennedy, R. Eberhart, Particle swarm optimization, in: Proceedings of ICNN'95-International Conference on Neural Networks, vol. 4, IEEE, 1995, pp. 1942–1948.
[31] X. Zhang, W. Hu, S. Maybank, X. Li, M. Zhu, Sequential particle swarm optimization for visual tracking, in: 2008 IEEE Conference on Computer Vision and Pattern Recognition, IEEE, 2008, pp. 1–8.
[32] D. Parrott, Xiaodong Li, Locating and tracking multiple dynamic optima by a particle swarm model using speciation, IEEE Trans. Evol. Comput. 10 (4) (2006) 440–458.
[33] S. Gheisari, M.R. Meybodi, BNC-PSO: Structure learning of Bayesian networks by particle swarm optimization, Inform. Sci. 348 (2016) 272–289.
[34] T. Du, S.S. Zhang, Z. Wang, Efficient learning Bayesian networks using PSO, in: International Conference on Computational and Information Science, Springer, 2005, pp. 151–156.
[35] F. Sahin, M.Ç. Yavuz, Z. Arnavut, O. Uluyol, Fault diagnosis for airplane engines using Bayesian networks and distributed particle swarm optimization, Parallel Comput. 33 (2) (2007) 124–143.
[36] N.M. Nor, C.R.C. Hassan, M.A. Hussain, A review of data-driven fault detection and diagnosis methods: Applications in chemical process systems, Rev. Chem. Eng. 36 (4) (2020) 513–553.
[37] P.J. Angeline, G.M. Saunders, J.B. Pollack, An evolutionary algorithm that constructs recurrent neural networks, IEEE Trans. Neural Netw. 5 (1) (1994) 54–65.
[38] F. Gomez, J. Schmidhuber, R. Miikkulainen, Accelerated neural evolution through cooperatively coevolved synapses, J. Mach. Learn. Res. 9 (2008) 937–965.
[39] V. Heidrich-Meisner, C. Igel, Neuroevolution strategies for episodic reinforcement learning, J. Algorithms 64 (4) (2009) 152–168, Special Issue: Reinforcement Learning. URL http://www.sciencedirect.com/science/article/B6WH3-4W7RY8J-3/2/22f7075bc25dab10a8ff3714e2fee303.
[40] X. Yao, Evolving artificial neural networks, Proc. IEEE 87 (9) (1999) 1423–1447.
[41] 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.
[42] R. Chandra, S. Cripps, Coevolutionary multi-task learning for feature-based modular pattern classification, Neurocomputing 319 (2018) 164–175.
[43] M.A. Potter, K.A. De Jong, Cooperative coevolution: An architecture for evolving coadapted subcomponents, Evol. Comput. 8 (2000) 1–29.
[44] N. García-Pedrajas, C. Hervas-Martinez, J. Munoz-Perez, Multi-objective cooperative coevolution of artificial neural networks (multi-objective cooperative networks), Neural Netw. 15 (2002) 1259–1278.
[45] R. Chandra, Competition and collaboration in cooperative coevolution of elman recurrent neural networks for time-series prediction, IEEE Trans. Neural Netw. Learn. Syst. 26 (2015) 3123–3136.
[46] R. Chandra, A. Tiwari, Distributed Bayesian optimisation framework for deep neuroevolution, Neurocomputing 470 (2022) 51–65.
[47] A. Rawal, R. Miikkulainen, Evolving deep LSTM-based memory networks using an information maximization objective, in: Proceedings of the Genetic and Evolutionary Computation Conference 2016, ACM, 2016, pp. 501–508.
[48] P. Verbancsics, J. Harguess, Image classification using generative neuro evolution for deep learning, in: 2015 IEEE Winter Conference on Applications of Computer Vision, IEEE, 2015, pp. 488–493.
[49] K.O. Stanley, R. Miikkulainen, Evolving neural networks through augmenting topologies, Evol. Comput. 10 (2) (2002) 99–127.
[50] S. Risi, J. Togelius, Neuroevolution in games: State of the art and open challenges, IEEE Trans. Comput. Intell. AI Games 9 (1) (2017) 25–41.
[51] F.P. Such, V. Madhavan, E. Conti, J. Lehman, K.O. Stanley, J. Clune, Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning, 2017, arXiv preprint arXiv:1712.06567.
[52] R. Chandra, D. Azam, R.D. Müller, T. Salles, S. Cripps, Bayeslands: a Bayesian inference approach for parameter uncertainty quantification in Badlands, Computers & Geosciences 131 (2019) 89–101.
[53] E. Begoli, T. Bhattacharya, D. Kusnezov, The need for uncertainty quantification in machine-assisted medical decision making, Nat. Mach. Intell. 1 (1) (2019) 20–23.
[54] J. Kwon, Particle swarm optimization–Markov chain Monte Carlo for accurate visual tracking with adaptive template update, Appl. Soft Comput. (2019) 105443.
[55] F. Liang, Bayesian neural networks for nonlinear time series forecasting, Stat. Comput. 15 (1) (2005) 13–29.
[56] O. Kocadağlı, B. Aşıkgil, Nonlinear time series forecasting with Bayesian neural networks, Expert Syst. Appl. 41 (15) (2014) 6596–6610.
[57] D.T. Mirikitani, N. Nikolaev, Recursive bayesian recurrent neural networks for time-series modeling, IEEE Trans. Neural Netw. 21 (2) (2010) 262–274.
[58] 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.
[59] B. Cheng, D.M. Titterington, Neural networks: A review from a statistical perspective, Statist. Sci. (1994) 2–30.
[60] A. Patriksson, D. van der Spoel, A temperature predictor for parallel tempering simulations, Phys. Chem. Chem. Phys. 10 (15) (2008) 2073–2077.
[61] M.K. Sen, P.L. Stoffa, Bayesian inference, Gibbs' sampler and uncertainty estimation in geophysical inversion, Geophys. Prospect. 44 (2) (1996) 313–350.
[62] M. Maraschini, S. Foti, A Monte Carlo multimodal inversion of surface waves, Geophys. J. Int. 182 (3) (2010) 1557–1566.
[63] F.P. Such, V. Madhavan, E. Conti, J. Lehman, K.O. Stanley, J. Clune, Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning, 2017, arXiv preprint arXiv:1712.06567.
[64] A. Poulsen, M. Thorhauge, M.H. Funch, S. Risi, Dlne: a hybridization of deep learning and neuroevolution for visual control, in: 2017 IEEE Conference on Computational Intelligence and Games (CIG), 2017, pp. 256–263.
[65] A. Ororbia, A. ElSaid, T. Desell, Investigating recurrent neural network memory structures using neuro-evolution, in: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '19, Association for Computing Machinery, New York, NY, USA, 2019, pp. 446–455, URL https://doi.org/10.1145/3321707.3321795.
[66] S. Khadka, J.J. Chung, K. Tumer, Neuroevolution of a modular memory-augmented neural network for deep memory problems, Evol. Comput. 08 (11) (2018) 1–26.
[67] K.O. Stanley, J. Clune, J. Lehman, R. Miikkulainen, Designing neural networks through neuroevolution, Nat. Mach. Intell. 1 (1) (2019) 24–35.
[68] 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.

[69] 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.
[70] F. Assunção, N. Lourenço, P. Machado, B. Ribeiro, Fast denser: Efficient deep neuroevolution, in: European Conference on Genetic Programming, Springer, 2019, pp. 197–212.
[71] S.M.J. Jalali, P.M. Kebria, A. Khosravi, K. Saleh, D. Nahavandi, S. Nahavandi, Optimal autonomous driving through deep imitation learning and neuroevolution, in: 2019 IEEE International Conference on Systems, Man and Cybernetics (SMC), 2019, pp. 1215–1220.
[72] B.M. Turner, P.B. Sederberg, Approximate Bayesian computation with differential evolution, J. Math. Psych. 56 (5) (2012) 375–385.
[73] J. Ji, C. Yang, J. Liu, J. Liu, B. Yin, A comparative study on swarm intelligence for structure learning of Bayesian networks, Soft Comput. 21 (22) (2017) 6713–6738.
[74] M.A. Potter, K.A. De Jong, Cooperative coevolution: An architecture for evolving coadapted subcomponents, Evol. Comput. 8 (1) (2000) 1–29.
[75] J. Kennedy, R. Eberhart, Particle swarm optimization, in: Proc. of the IEEE Int. Conf. on Neural Networks, Piscataway, NJ, 1995, 1942–1948.
[76] N. Singh, L.-M. Browne, R. Butler, Parallel astronomical data processing with Python: Recipes for multicore machines, Astron. Comput. 2 (2013) 1–10.
[77] A. Asuncion, D.J. Newman, UCI machine learning repository, 2007, URL http://archive.ics.uci.edu/ml/datasets.html.
[78] M.K. Cowles, B.P. Carlin, Markov chain Monte Carlo convergence diagnostics: A comparative review, J. Amer. Statist. Assoc. 91 (434) (1996) 883–904.
[79] N. Toft, G.T. Innocent, G. Gettinby, S.W. Reid, Assessing the convergence of Markov chain Monte Carlo methods: An example from evaluation of diagnostic tests in absence of a gold standard, Prevent. Vet. Med. 79 (2–4) (2007) 244–256.
[80] K.L. Mengersen, C.P. Robert, C. Guihenneuc-Jouyaux, MCMC convergence diagnostics: A review, Bayesian Stat. 6 (1999) 415–440.
[81] A. Gelman, D.B. Rubin, et al., Inference from iterative simulation using multiple sequences, Statist. Sci. 7 (4) (1992) 457–472.
[82] S. Chib, E. Greenberg, Understanding the metropolis-hastings algorithm, Amer. Statist. 49 (4) (1995) 327–335.
[83] M.K. Cowles, G.O. Roberts, J.S. Rosenthal, Possible biases induced by MCMC convergence diagnostics, J. Stat. Comput. Simul. 64 (1) (1999) 87–104.
[84] K.-C. Wang, P. Vicol, J. Lucas, L. Gu, R. Grosse, R. Zemel, Adversarial distillation of bayesian neural network posteriors, in: International Conference on Machine Learning, PMLR, 2018, pp. 5190–5199.
[85] J. Song, S. Zhao, S. Ermon, A-NICE-MC: Adversarial training for MCMC, Adv. Neural Inf. Process. Syst. 30 (2017).