Bayesian neural learning via Langevin dynamics for chaotic time series prediction

Rohitash Chandra[1,2], Lamiae Azizi[2,1], Sally Cripps[1,2]

[1] Centre for Translational Data Science
[2] School of Mathematics and Statistics
The University of Sydney, NSW 2006, Australia.
rohitash.chandra@sydney.edu.au

Abstract. Although neural networks have been very promising tools for chaotic time series prediction, they lack methodology for uncertainty quantification. Bayesian inference using Markov Chain Mont-Carlo (MCMC) algorithms have been popular for uncertainty quantification for linear and non-linear models. Langevin dynamics refer to a class of MCMC algorithms that incorporate gradients with Gaussian noise in parameter updates. In the case of neural networks, the parameter updates refer to the weights of the network. We apply Langevin dynamics in neural networks for chaotic time series prediction. The results show that the proposed method improves the MCMC random-walk algorithm for majority of the problems considered. In particular, it gave much better performance for the real-world problems that featured noise.

Keywords:
Backpropagation, gradient descent, MCMC algorithms, chaotic time series, neural networks

1 Introduction

The success of neural networks is partially due to the ability to train them on massive data sets with variants of the backpropagation algorithm [14,1]. Despite the successes, there are some challenges in training neural networks with backpropagation. Firstly, with large data sets, finding the optimal values of hyperparameters (eg. learning and momentum rate) and weights take a large amount of time [1]. Secondly, through canonical backpropagation, we can only obtain point estimates of the weights in the network. As a result, these networks make predictions that do not account for uncertainty in the parameters. However, in many cases, these weights may be poorly specified and it is desirable to produce uncertainty estimates along with predictions. A Bayesian approach to neural networks can potentially avoid some of the pitfalls of canonical backpropagation [9]. Bayesian techniques, in principle, can automatically infer hyperparameter values by marginalizing them out of the posterior distribution [10]. Furthermore, Bayesian methods naturally account for uncertainty in parameter estimates and can propagate this uncertainty into predictions. Finally, they are often more robust to overfitting, since they average over parameter values instead of choosing a single point estimate. Bayesian neural networks (BNNs) use Markov Chain Monti-Carlo (MCMC) methods such as those based on e.g the Laplace approxi-

mation [9], Hamiltonian Monte Carlo [12], expectation propagation [4] and variational inference [2]. However, these approaches have not seen widespread adoption due to their lack of scalability in both network architecture and data size. A number of attempts have been proposed combining MCMC techniques with the gradient optimization algorithms [6], the simulated annealing method [8], and evolutionary algorithms [5].

In the context of the non-linear time series prediction, there are a number of studies in the literature. Liang et. al presented an MCMC algorithm for neural networks for selected time series problems [7]. Bayesian techniques have also been used for controlling model complexity and selecting inputs in neural networks for the short term time series forecasting [3]. Moreover, recursive Bayesian recurrent neural networks [11] and evolutionary MCMC Baysiean neural networks have been used for time series forecasting [5]. Although these methods have been used, there has not been much emphasis on experimental evaluation of the method on chaotic time series benchmark problems. Langevin dynamics refer to a class of MCMC algorithms that incorporate gradients with Gaussian noise in parameter updates [13]. In the case of neural networks, the parameter updates refer to the weights of the network.

In this paper, we provide the synergy of MCMC random-walk algorithm and gradient descent via backpropagation neural network. This is referred as Langevin dynamics MCMC for training neural networks. We employ six benchmark chaotic time series problems to demonstrate the effectiveness of the proposed method. Furthermore, comparison is done with standalone gradient descent and the MCMC random-walk algorithm.

The rest of the paper is organised as follows. Section 2 presents the proposed method and Section 3 presents experiments and results. Section 4 concludes the paper with a discussion of future work.

2 Langevin Dynamics for Neural Networks

2.1 Model and Priors

Let y_t denote a univariate time series modelled by:

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

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

In order to use neural networks for time series prediction, the original dataset is constructed into a state-space vector through Taken's theorem [15] which is governed by the embedding dimension (D) and time-lag (T).
Define

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

Let $\mathbf{y}_{\mathcal{A}_{\mathcal{D},\mathcal{T}}}$ to be the collection of y_t's for which $t \in \mathcal{A}_{\mathcal{D},\mathcal{T}}$, then, $\forall t \in A_{D,T}$, we compute the $f(\mathbf{x}_t)$ by a feedforward neural network with one hidden layer defined by the function

$$f(\mathbf{x}_t) = g\bigg(\delta_o + \sum_{h=1}^{H} v_j g\bigg(\delta_h + \sum_{d=1}^{D} w_{dh} y_{t-d})\bigg)\bigg) \tag{3}$$

where δ_o and δ_h are the bias weights for the output o and hidden h layer, respectively. V_j is the weight which maps the hidden layer h to the output layer. w_{dh} is the weight which maps y_{t-d} to the hidden layer h and g is the activation function, which we assume to be a sigmoid function for the hidden and output layer units.

Let $\boldsymbol{\theta} = (\tilde{\mathbf{w}}, \mathbf{v}, \boldsymbol{\delta}, \tau^2)$, with $\boldsymbol{\delta} = (\delta_o, \delta_h)$, denote $L = (DH + (2 * H) + O + 1)$ vector of parameters that includes weights and biases, with O number of neurons in output layer. H is the number of hidden neurons required to evaluate the likelihood for the model given by (1), with $\tilde{\mathbf{w}} = (\mathbf{w_1}_{\cdot}', \ldots, \mathbf{w}_{D\cdot}')'$, and $\mathbf{w_d}_{\cdot} = (w_{d1}, \ldots, w_{dH})'$, for $d = 1, \ldots, D$.

To conduct a Bayesian analysis, we need to specify prior distributions for the elements of θ which we choose to be

$$\begin{aligned}
v_h &\sim \mathcal{N}(0, \sigma^2) \text{ for } h = 1, \ldots, H, \\
\delta_0 &\sim N(0, \sigma^2) \\
\delta_h &\sim N(0, \sigma^2) \\
w_{dj} &\sim \mathcal{N}(0, \sigma^2) \text{ for } h = 1, \ldots, H \text{ and } d = 1, \ldots, D, \\
\tau^2 &\sim \mathcal{IG}(\nu_1, \nu_2)
\end{aligned} \tag{4}$$

where H is the number of hidden neurons. In general the log posterior is

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

In our particular model the log likelihood is

$$\log\left(p(\mathbf{y}_{\mathcal{A}_{\mathbf{D},\mathbf{T}}}|\boldsymbol{\theta})\right) = -\frac{n-1}{2}\log(\tau^2) - \frac{1}{2\tau^2}\sum_{t \in \mathcal{A}_{\mathcal{D},\mathcal{T}}} \left(y_t - E(y_t|\mathbf{x}_t)\right)^2 \tag{5}$$

where $E(y_t|\mathbf{x}_t)$ is given by (3). We further assume that the elements of θ are independent *apriori* so that the log of the prior distributions is

$$\begin{aligned}
\log\left(p(\boldsymbol{\theta})\right) = &-\frac{HD + H + 2}{2}\log(\sigma^2) - \frac{1}{2\sigma^2}\left(\sum_{h=1}^{H}\sum_{d=1}^{D} w_{dh}^2 + \sum_{h=1}^{H}(\delta_h^2 + v_h^2) + \delta_o^2\right) \\
&-(1+\nu_1)\log(\tau^2) - \frac{\nu_2}{\tau^2}
\end{aligned} \tag{6}$$

2.2 Algorithm

As discussed earlier, principled way to incorporate uncertainty during learning is to use Bayesian inference [9,12]. This requires obtaining samples from the

posterior distribution $p(\boldsymbol{\theta}|\mathbf{y}_{\mathcal{A}_{\mathbf{D},\mathbf{T}}})$. The proposed MCMC algorithm consists of a single Metropolis Hasting step with proposals formed using Langevin dynamics that employs gradients. In particular, we propose a new value of $\boldsymbol{\theta}$ from

$$\boldsymbol{\theta}^p \sim \mathcal{N}(\bar{\boldsymbol{\theta}}^{[k]}, \Sigma_\theta), \text{ where} \tag{7}$$

$$\bar{\boldsymbol{\theta}}^{[k]} = \boldsymbol{\theta}^{[k]} + r \times \nabla E_{\mathbf{y}_{\mathcal{A}_{\mathbf{D},\mathbf{T}}}}[\boldsymbol{\theta}^{[k]}], \tag{8}$$

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

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

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

1. An gradient descent based weight update given by Equation (8)
2. Add an amount of noise, from $\mathcal{N}(0, \Sigma_\theta)$.

Hereafter, we refer to the proposed Langevin dynamics for neural networks as LD-MCMC. This combined update is used as a proposal in a Metropolis-Hastings step, which accepts the proposed value of $\boldsymbol{\theta}^p$ with the usual probability α, where

$$\alpha = \min \left\{ 1, \frac{p(\boldsymbol{\theta}^p|\mathbf{y}_{\mathcal{A}_{D,T}}) q(\boldsymbol{\theta}^{[k]}|\boldsymbol{\theta}^p)}{p(\boldsymbol{\theta}^{[k]}|\mathbf{y}_{\mathcal{A}_{D,T}}) q(\boldsymbol{\theta}^p|\boldsymbol{\theta}^{[k]})} \right\} \tag{9}$$

where $p(\boldsymbol{\theta}^p|\mathbf{y}_{\mathcal{A}_{D,T}})$ and $p(\boldsymbol{\theta}^{[k]}|\mathbf{y}_{\mathcal{A}_{D,T}})$ can be computed using Equation (5) and Equation (6). $q(\boldsymbol{\theta}^p|\boldsymbol{\theta}^{[k]})$, is given by Equation (7) and $q(\boldsymbol{\theta}^{[k]}|\boldsymbol{\theta}^p) \sim N(\bar{\boldsymbol{\theta}}^p, \Sigma_{\boldsymbol{\theta}})$, with $\bar{\boldsymbol{\theta}}^p = \boldsymbol{\theta}^p + r \times \nabla E_{\mathbf{y}_{\mathcal{A}_{\mathbf{D},\mathbf{T}}}}[\boldsymbol{\theta}^p]$, thus ensuring that the detailed balance condition holds and the sequence $\boldsymbol{\theta}^{[k]}$ converges to draws from the posterior $p(\boldsymbol{\theta}|\mathbf{y})$.

For testing, given a test input $\tilde{\mathbf{x}}$ (with missing label $\tilde{y}$), the uncertainty learned in training is transferred to prediction, yielding the following posterior distribution:

$$p(\tilde{y}|\tilde{x}, \mathbf{y}_{\mathcal{A}_{D,T}}) = \mathbb{E}_{p(\boldsymbol{\theta}|\mathbf{y}_{\mathcal{A}_{D,T}})}[p(\tilde{y}|\tilde{x}, \theta)] = \int_\theta p(\tilde{y}|\tilde{x}, \boldsymbol{\theta}) p(\boldsymbol{\theta}|\mathbf{y}_{\mathcal{A}_{D,T}}) d\boldsymbol{\theta} \tag{10}$$

The predicted distribution of $\tilde{y}$ can be viewed in terms of model averaging across parameters, based on the learned $p(\boldsymbol{\theta}|\mathbf{y}_{\mathcal{A}_{D,T}})$; this is contrasted with learning a single point estimate of $\boldsymbol{\theta}$ based on $\mathbf{y}_{\mathcal{A}_{D,T}}$.

Each iteration of the LD-MCMC algorithm requires refinement of the proposal drawn through gradient descent on the entire dataset for one iteration via backpropagation. The LD-MCMC algorithm for feedforward neural networks (FNNs) is given in Algorithm 1. The major difference of this algorithm from MCMC random-walk lies in the additional step where gradient descent is used to update the location of the proposal distribution. This is helpful as gradients help in better proposals when compared to a random-walk MCMC algorithm. Langevin dynamics through gradient descent is possible for the selected time

series problems given that the size of the datasets are relatively low when compared to big data problems. The implementation of Algorithm 1 in Python is also given online [3]. The MCMC random-walk for neural networks is also given online [4].

Alg. 1 Langevin Dynamics for neural networks

Data: Univariate time series $\mathbf{y}$
Result: Posterior of weights and biases $p(\boldsymbol{\theta}|\mathbf{y})$
Step 1: State-space reconstruction $\mathbf{y}_{\mathcal{A}_{D,T}}$ by Equation 2
Step 2: Define feedforward network as given in Equation 3
Step 3: Define $\boldsymbol{\theta}$ as the set of all weights and biases
Step 4: Set parameters σ^2 ,ν_1, ν_2 for prior given in Equation 6

for *each k until max-samples* **do**
    1.Compute gradient $\Delta\boldsymbol{\theta}^{[k]}$given by Equation 8
    2. Draw $\boldsymbol{\eta}$ from $\mathcal{N}(0, \Sigma_\eta)$
    3. Propose $\boldsymbol{\theta}^* = \theta^{[k]} + \Delta\boldsymbol{\theta}^{[k]} + \boldsymbol{\eta}$
    4. Draw from uniform distribution $u \sim \mathcal{U}[0, 1]$
    5. Obtain acceptance probability α given by Equation 9
    if $u < \alpha$ **then**
        $\boldsymbol{\theta}^{[k+1]} = \boldsymbol{\theta}^*$
    end
    else
        $\boldsymbol{\theta}^{[k+1]} = \boldsymbol{\theta}^{[k]}$
    end
end

3 Experiments and Results

This section presents experimental evaluation of LD-MCMC algorithm (Algorithm 1) for six benchmark one-step-ahead chaotic time series prediction problems. The comparison is done with MCMC random-walk algorithm and gradient descent (backpropagation).

3.1 Problem Description

The benchmark problems employed are Mackey-Glass, Lorenz, Sunspot, and Laser, Henon, and Rossler time series. Takens' embedding theorem[16] is applied with selected values as follows. $D = 4$ and $T = 2$ is used for reconstruction of the respective time series into state-space vector. All the problems used first 1000 data points of the time series from which 60 % was used for training and 40 %

[3] https://github.com/rohitash-chandra/LDMCMC_timeseries
[4] https://github.com/rohitash-chandra/MCMC_fnn_timeseries

for testing the method. The prediction performance is measured using the root mean squared error (RMSE)

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2} \tag{11}$$

where $y_i, \hat{y}_i$ are the observed data and predicted data, respectively. N is the length of the observed data. These results are also compared with related methods from the literature.

3.2 Experimental Design

We provide details of the experimental design followed by experimentation and results. The FNN employs one hidden layer and sigmoid activation function for all the respective problems. The first 10% was discarded for "burn-in" which is standard procedure for MCMC based algorithms. In the LD-MCMC algorithm, each sample drawn was further refined using gradient descent for $n = 1$ iterations. Note that the FNN used 4 neurons in the input and 5 neurons in the hidden layer for all the respective problems. In the MCMC methods, the random-walk for the respective weights and biases θ was drawn using a normal distribution with mean $\mu = 0$ and standard deviation $\sigma = 0.02$. A normal distribution was also used to draw samples that define η with mean $\mu = 0$ and standard deviation $\sigma = 0.2$. These values gave acceptable results in trial runs and hence were used in all the problems. In the case of GD, fixed learning rate of 0.1 and momentum rate of 0.01 was used for all the problems. The parameters for the prior given in Equation 6 was set: $\sigma^2 = 25$, $\nu_1 = 0$, $\nu_2 = 0$.

3.3 Results

The results for the three methods for the six benchmark problems are given in Table 1. Note that (*) represents early convergence or termination criteria which is 2000 epochs for GD*. On the other hand, GD convergence is defined by 80 000 iterations (epochs) in order to compare with MCMC random-walk algorithm which draws 80 000 samples. LD-MCMC draws 40 000 samples and each of them are refined by GD for 1 iteration which makes the computation time similar in terms of number of fitness evaluation given by the loss function. Note that GD method employed 30 independence experimental runs, while MCMC and LD-MCMC results are for a single experimental run. The mean and standard deviation (std) of the results are given.

We evaluate the performance achieved by the respective algorithms for six benchmark chaotic time series problems which include Lazer, Sunspot, Mackey-Glass, Lorenz, Rossler and Henon. We note that Lazer and Sunspot are real-world problems that contain noise while the rest are simulated time series. We find that backpropagation via GD outperforms the rest of the the methods for all the problems. Moving on, we observe that LD-MCMC outperforms MCMC random-walk algorithm for all the problems except for the Mackey-Glass problem. Generally, LD-MCMC accepted more solutions when compared to MCMC which could be due to the refinement of samples through gradient descent. In the

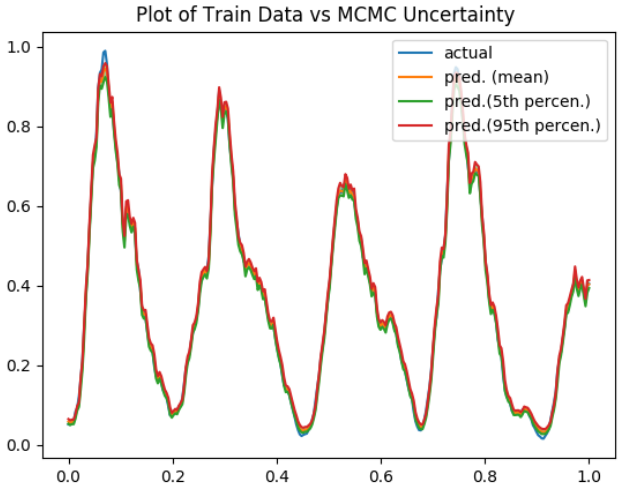
case of Lorenz problem, they achieve similar results. The MCMC methods generally gave better performance when compared to early convergence instance of gradient descent (GD*). The results for early convergence was shown taking into consideration typical training time used for related problems in the literature.

Table 1: Results for the respective methods

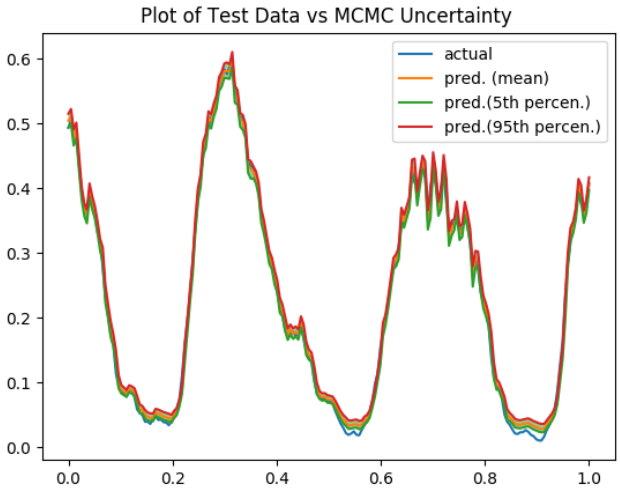
Problem	Method	Train (mean)	Train (std)	Test (mean)	Test(std)	% Accepted
Lazer	GD*	0.02191	0.00129	0.02675	0.00085	-
	GD	0.01035	0.00162	0.01732	0.00142	-
	MCMC	0.02549	0.00837	0.02643	0.00718	9.70
	LD-MCMC	0.01658	0.00211	0.02280	0.00351	57.86
Sunspot	GD*	0.02117	0.00160	0.02359	0.00209	-
	GD	0.00775	0.00026	0.00975	0.00031	-
	MCMC	0.01466	0.00174	0.01402	0.00178	4.55
	LD-MCMC	0.01155	0.00167	0.01090	0.00146	45.89
Mackey	GD*	0.00590	0.00026	0.00669	0.00029	-
	GD	0.00286	0.00025	0.0033	0.00026	-
	MCMC	0.00511	0.00058	0.00520	0.00058	1.74
	LD-MCMC	0.00615	0.00091	0.00627	0.00091	30.35
Lorenz	GD*	0.01570	0.00056	0.01608	0.00052	-
	GD	0.00400	0.00024	0.00460	0.00026	-
	MCMC	0.00813	0.00174	0.00713	0.00150	2.74
	LD-MCMC	0.00890	0.00211	0.00821	0.00207	26.95
Rossler	GD*	0.01570	0.00056	0.01608	0.00052	-
	GD	0.00281	0.00029	0.00462	0.00045	-
	MCMC	0.01371	0.00291	0.01355	0.00297	3.69
	LD-MCMC	0.00722	0.00121	0.00692	0.00120	35.53
Henon	GD*	0.01366	0.00033	0.01778	0.00025	-
	GD	0.00555	0.00029	0.00604	0.00025	-
	MCMC	0.03256	0.03920	0.03127	0.03850	8.71
	LD-MCMC	0.00948	0.00112	0.00912	0.00114	27.17

The results for a selected problem (Sunspot) is given in Figure 1 which shows uncertainty quantification using LD-MCMC. The results show the uncertainty (through 5th and 95th percentile) and prediction on the training and the test datasets. The posterior distributions for the set of weights and biases are given as box-plot in Figure 2 that could be further visualised as probability distributions.

The results in general have shown that LD-MCMC further improves the performance of MCMC random-walk for training FNNs for majority of the problems. This improvement has been through the incorporation of gradients in weight updates via Langevin dynamics. This has been beneficial for improving the proposals drawn by MCMC random-walk algorithm. In the case for larger datasets, stochastic gradient version of Langevin dynamics can be implemented [17].



(a) Prediction and uncertainty quantification over training data



(b) Prediction and uncertainty quantification over test data

Fig. 1: Results for Sunspot time series using LD-MCMC algorithm. Note that the x-axis represents the time in year while y-axis gives the Sunspot index.

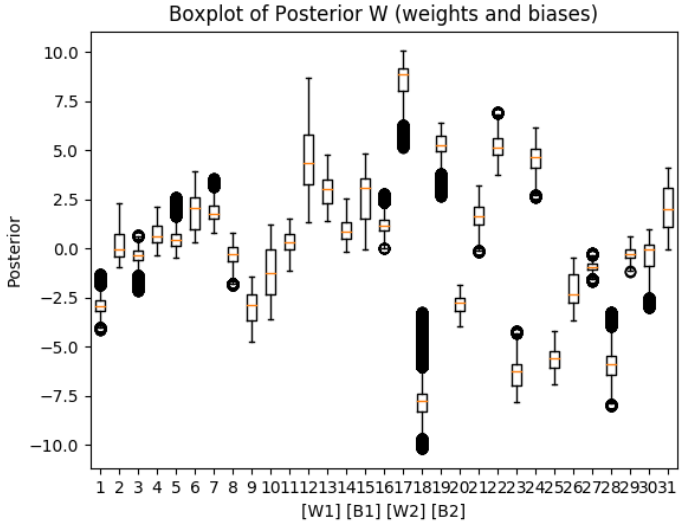


Fig. 2: Posterior of weights and biases. W1 refers to the set of weights from input-hidden layer, W2 refers to the set of weights from hidden-output layer, B1 refers to the set of biases of the hidden layer and B2 refers to the bias of the output layer.

4 Conclusions and Future Work

We applied Langevin dynamics for Bayesian learning in neural networks that featured synergy of MCMC with backpropagation. This provides further advantage to conventional neural network training algorithms through uncertainty quantification in predictions. The results showed that the gradient descent weight updates though Langevin dynamics improved MCMC random-walk algorithm for majority of the problems. In particular, it gave much better performance for real-world problems such as Lazer and Sunspot time series that featured noise. A limitation is that the proposed method cannot get the same performance when compared to gradient decent alone when given a large number of training epochs that match with number of samples used by the MCMC algorithms. However, gradient decent on its own has limitations as it produces a one-point solution which does not feature uncertainty quantification.

Future work would focus on Langevin dynamics based Bayesian learning for other neural network architectures. The incorporation of Langevin dynamics with hyper-parameters such as momentum and learning rate could be further explored. In addition, uncertainty quantification could also be provided in terms of number of hidden neurons and layers of the network. Moreover, the method can be directly used for real-world time series applications that includes climate change problems such as rainfall, storms and cyclones.

References

1. Bottou, L.: Large-scale machine learning with stochastic gradient descent. In: Proceedings of COMPSTAT'2010, pp. 177–186. Springer (2010)
2. Hinton, G.E., Van Camp, D.: Keeping the neural networks simple by minimizing the description length of the weights. In: Proceedings of the sixth annual conference on Computational learning theory. pp. 5–13. ACM (1993)
3. Hippert, H.S., Taylor, J.W.: An evaluation of bayesian techniques for controlling model complexity and selecting inputs in a neural network for short-term load forecasting. Neural networks 23(3), 386–395 (2010)
4. Jylänki, P., Nummenmaa, A., Vehtari, A.: Expectation propagation for neural networks with sparsity-promoting priors. Journal of Machine Learning Research 15(1), 1849–1901 (2014)
5. Kocadağlı, O., Aşıkgil, B.: Nonlinear time series forecasting with bayesian neural networks. Expert Systems with Applications 41(15), 6596–6610 (2014)
6. Li, C., Chen, C., Carlson, D., Carin, L.: Preconditioned stochastic gradient langevin dynamics for deep neural networks. In: Thirtieth AAAI Conference on Artificial Intelligence (2016)
7. Liang, F.: Bayesian neural networks for nonlinear time series forecasting. Statistics and computing 15(1), 13–29 (2005)
8. Liang, F.: Annealing stochastic approximation monte carlo algorithm for neural network training. Machine Learning 68(3), 201–233 (2007)
9. MacKay, D.J.: A practical bayesian framework for backpropagation networks. Neural computation 4(3), 448–472 (1992)
10. MacKay, D.J.: Hyperparameters: Optimize, or integrate out? In: Maximum entropy and Bayesian methods, pp. 43–59. Springer (1996)
11. Mirikitani, D.T., Nikolaev, N.: Recursive bayesian recurrent neural networks for time-series modeling. IEEE Transactions on Neural Networks 21(2), 262–274 (2010)
12. Neal, R.M.: Bayesian learning for neural networks, vol. 118. Springer Science & Business Media (2012)
13. Neal, R.M., et al.: Mcmc using hamiltonian dynamics. Handbook of Markov Chain Monte Carlo 2(11) (2011)
14. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning representations by back-propagating errors. Cognitive modeling 5(3), 1
15. Takens, F.: Detecting strange attractors in turbulence. In: Dynamical Systems and Turbulence, Warwick 1980, pp. 366–381. Lecture Notes in Mathematics (1981)
16. Takens, F.: On the numerical determination of the dimension of an attractor. Springer (1985)
17. Welling, M., Teh, Y.W.: Bayesian learning via stochastic gradient langevin dynamics. In: Proceedings of the 28th International Conference on Machine Learning (ICML-11). pp. 681–688 (2011)