Bayesian deep learning via MCMC sampling with application to Robot path planning

Dr Rohitash Chandra

School of Mathematics and Statistics
UNSW Sydney

Seminar: Queensland University of Technology

UNSW THE UNIVERSITY OF NEW SOUTH WALES

1/64

Overview

- Bayesian inference for deep learning with Markov Chain Monte Carlo (MCMC) methods.
- Advanced proposal schemes in MCMC sampling (Langevin gradients).
- Bayesian deep autoencoders.
- Bayesian neuroevolution with MCMC
- Coevolutionary multi-task learning
- Revesible Jump MCMC for Bayesian neural networks
- Robot path planning with MCMC

UNSW THE UNIVERSITY OF NEW SOUTH WALES

Bayesian inference

- Bayesian inference provides a principled approach towards uncertainty quantification of free or unknown parameters.
- The use of MCMC methods has been popular in several fields, such as Earth sciences, environmental sciences, health and medicine, astronomy, and machine learning.
- Bayesian inference through variational inference has been popular for deep learning, particularly variational autoencoders.
- There has been slow progress in using MCMC methods for deep learning, however, recent progress in Hamiltonian MCMC and Langevin MCMC opens up the road for faster progress. These methods enable use of gradients for proposal distribution.

UNSW THE UNIVERSITY OF NEW SOUTH WALES

Bayesian inference

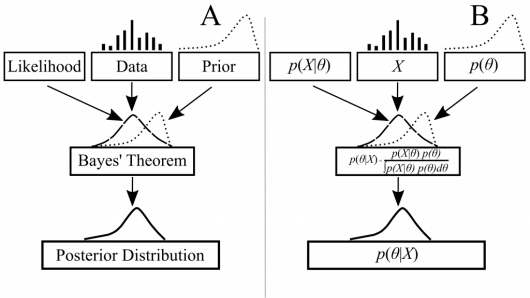


Figure: Bayesian inference overview [1]

Markov Chain Monte Carlo sampling methods (MCMC) implement Bayesian inference that sample from a probability distribution.

[1] Source: Google images

Bayesian inference

The Theory: Bayesian inference

- Methodology of mathematical inference:
  - Choosing between several possible models
  - Extracting parameters for these models
- Bayes' Theorem:

- Remove nuisance parameters by marginalisation
- Interesting ones remain

Rev Thomas Bayes 1702 - 1761

Figure: Bayesian inference overview [2]

Bayesian inference

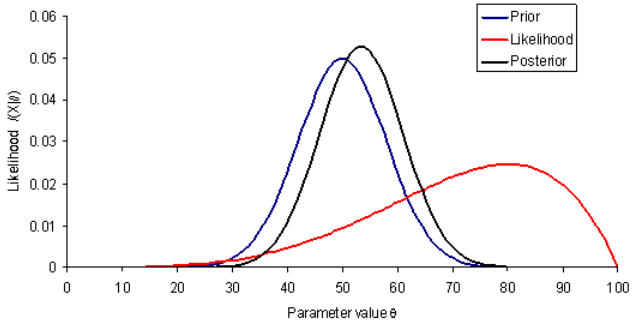


Figure: Bayesian inference overview [3]

[3] Source: Google images

Bayesian neural network and MCMC sampling

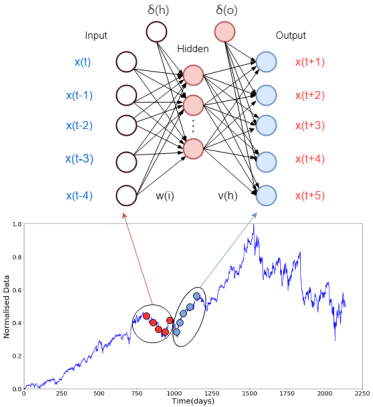


Figure: The time series (shown in red circles) is used as input for the neural network which predicts 5 steps-ahead in time (shown by blue circles). A sliding window approach is used to reconstruct the dataset in this way using Taken's theorem.

Simple neural network for univariate time series prediction

Our reconstructed vector by state-space embedding is denoted by $[\bar{\mathbf{x}}, \mathbf{y}]$ over the time series that map as input and output to the given neural network models. Hence, for the first instance, we have

$$\bar{\mathbf{x}}_{\mathbf{1}} = [x_1, x_2, x_3, x_4, \ldots, x_m]$$

$$\mathbf{y}_{\mathbf{1}} = [x_{m+1}, x_{m+2}, x_{m+3}, \ldots, x_{m+n}]$$

In the same way, we can obtain the rest of the instances for the entire time series as given below.

$$\bar{\mathbf{x}}_{\mathbf{t}} = [x_{1+(t-1)T}, x_{2+(t-1)T}, x_{3+(t-1)T}, x_{4+(t-1)T}, \ldots, x_{m+(t-1)T}]$$

$$\mathbf{y}_{\mathbf{t}} = [x_{m+(t-1)T+1}, x_{m+(t-1)T+2}, x_{m+(t-1)T+3}, \ldots, x_{m+(t-1)T+n}]$$

UNSW THE UNIVERSITY OF NEW SOUTH WALES

8/64

Simple neural network for univariate time series prediction

. The expected value of y_t given $\mathbf{x_t} = (\mathbf{y_1}, \ldots, \mathbf{y_{t-1}})$ is given by:

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

where δ_o and δ_h are the biases for the output and hidden h layers, respectively. I and H are the number of input and hidden neurons, respectively. v_h is the weight which maps the hidden layer h to the output, w_{ih} is the weight which maps $x_{t,i}$ to the hidden layer h and g is the activation function (sigmoid, tanh, linear, ReLu) for the hidden and output layer units of the neural network.

Bayesian neural network: Likelihood

Hence, for model output $f(\bar{\mathbf{x}}_\mathbf{t})$ with given input features $\bar{\mathbf{x}}_\mathbf{t}$, we have

$$p(\mathbf{y_S}|\boldsymbol{\theta}) = -\frac{1}{(2\pi\tau^2)^{S/2}} \times \exp\left(-\frac{1}{2\tau^2}\sum_{t\in S}(\mathbf{y_t} - f(\bar{\mathbf{x}}_\mathbf{t}))^2\right) \quad (2)$$

which satisfies the multivariate probability density function.

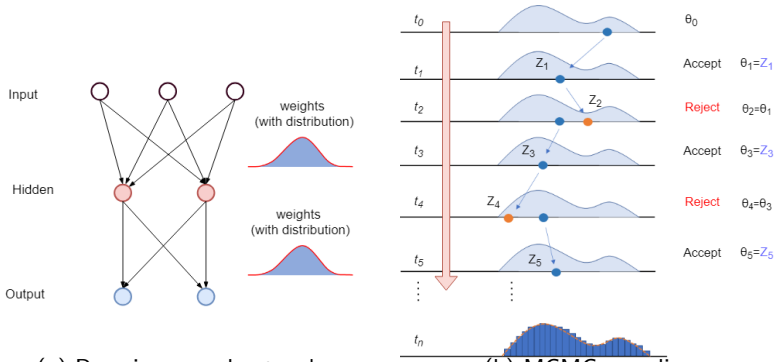
Bayesian neural network: Priors

Our prior is based on normal (weights) and inverse Gamma distribution (noise parameter) and given as

$$p(\boldsymbol{\theta}) \propto \frac{1}{(2\pi\sigma^2)^{L/2}} \times \exp\left\{ -\frac{1}{2\sigma^2}\left(\sum_{i=1}^{M}\theta\right)\right\} \times \tau^{2(1+\nu_1)} \exp\left(\frac{-\nu_2}{\tau^2}\right) \quad (3)$$

where, σ is determined by exploring variance in weights and biases of trained neural networks for similar applications, and ν is a user defined constant.

Bayesian neural network and MCMC sampling



(a) Bayesian neural network (b) MCMC sampling

Figure: Bayesian neural network and MCMC sampling. Note that the posterior distribution is shown that represents weights in Panel (a)

Proposal distribution: Langevin gradients

We use Langevin-gradient proposal distribution that essentially features a one-step gradient over Gaussian noise. At a given chain position k, our new proposal θ^p is given as follows

- A one-step gradient descent based weight update known as Langevin-gradient (LG) proposal distribution (Equation 8),
- A random-walk (RW) proposal distribution where Gaussian noise from distribution centered at mean of 0 and variance, $\mathcal{N}(0, \Sigma_\theta)$.

Single-chain Langevin-gradient MCMC

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 $\mathbf{y}_{A_{D,T}}$ 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

Figure: Langevin-gradient MCMC [4]

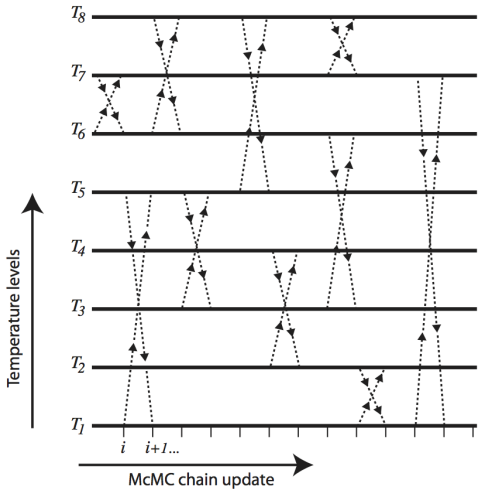
Acceptance probability

Lets consider parallel tempering MCMC with an emsemble of R replicas.
For each replica m in the ensemble, we propose θ_m^p using Langevin-gradient conditional on current value θ_m^c, that is $\theta_m^p \sim q(\theta|\theta_m^c)$. The within replica transition determines if the proposed value of θ_m^p remains at its original location θ_m^c or gets updated by a probability as given

$$\alpha = min\left(1, \frac{p(\theta_m^k|D)^{\beta_m} q(\theta_m^k|\theta_m^p)}{p(\theta_m^k|D)^{\beta_m} q(\theta_m^p|\theta_m^k)}\right). \tag{4}$$

essentially
$q(\theta_m^p|\theta_m^c)$ is given by $\theta_m^p \sim \mathcal{N}(\bar{\theta}_m^c, \Sigma_\theta)$
and $q(\theta_m^c|\theta_m^p) \sim N(\bar{\theta}_m^p, \Sigma_\theta)$
The above ensures that the detailed balance condition holds and the sequence $\boldsymbol{\theta}^{[k]}$ converges to the posterior $p(\boldsymbol{\theta}|\mathbf{y})$.

Parallel tempering



Parallel tempering

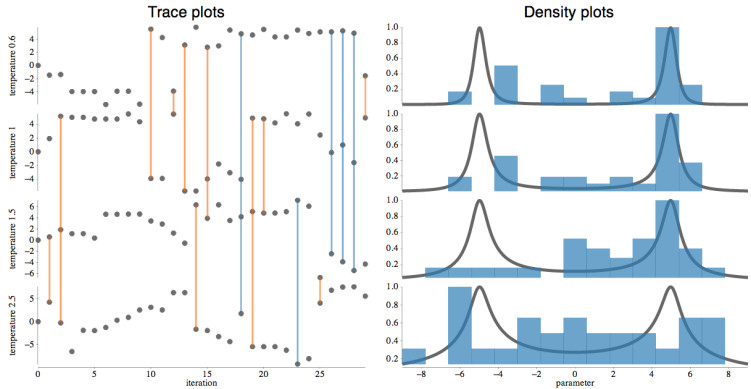
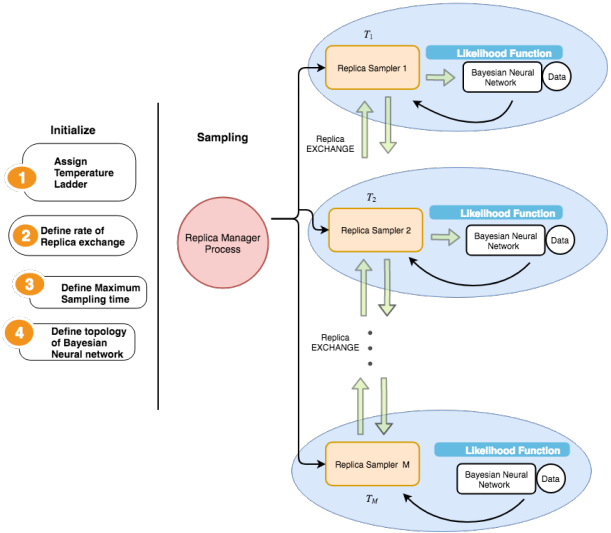


Figure: Parallel tempering - sampling [6]

[6] Source: Google images

Parallel tempering MCMC



Adam optimiser: Adaptive moment estimation

Adaptive moment estimation (Adam) is an effective stochastic optimization method. Adam calculates the individual adaptive learning rates of different parameters from the estimates of first and second moments of the gradients. The Adam update equation can be expressed as follows

$$w_k = w_{k-1} - a_{k-1} \cdot \frac{\sqrt{1-\beta_2^k}}{\sqrt{1-\beta_1^k}} \cdot \frac{u_{k-1}}{\sqrt{n_{k-1}} + \epsilon}$$

where

$$u_{k-1} = \beta_1 u_{k-2} + (1-\beta_1)\nabla f(w_{k-1})$$
$$n_{k-1} = \beta_2 n_{k-2} + (1-\beta_2)\nabla f(w_{k-1})^2$$

where β_1 and β_2 are two hyperparameters, β_1 controls first-order momentum and β_2 controls second-order momentum, and ϵ is a small scalar used to prevent division by

Results: Time series prediction

Results from [7]

Dataset	Method	Train (mean, std, best)	Test (mean, std, best)	Swap per.	Accept per.	Time (min.)
Lazer	PT-RW	0.0640 0.0218 0.0325	0.0565 0.0209 0.0270	42.26	35.31	4.53
	PT-LG (0.10)	0.0383 0.0187 0.0240	0.0353 0.0161 0.0212	48.45	19.91	11.53
	PT-LG (0.01)	0.0446 0.0160 0.0283	0.0414 0.0160 0.0253	51.45	30.97	11.50
	SGD	0.1020 0.0107 0.0832	0.0975 0.0072 0.0764	-	-	1.10
	ADAM	0.0805 0.0156 0.0629	0.0953 0.0150 0.0704	-	-	1.05
Sunspot	PT-RW	0.0242 0.0041 0.0170	0.0239 0.0050 0.0161	44.45	18.30	4.82
	PT-LG(0.10)	0.0199 0.0031 0.0155	0.0192 0.0033 0.0146	48.45	12.57	11.61
	PT-LG (0.01)	0.0215 0.0032 0.0168	0.0204 0.0034 0.0154	46.94	15.16	11.47
	SGD	0.0283 0.0469 0.0207	0.0273 0.0469 0.0216	-	-	1.3
	ADAM	0.0241 0.0018 0.0208	0.0236 0.0010 0.0219	-	-	1.05
Mackey	PT-RW	0.0060 0.0005 0.0051	0.0061 0.0005 0.0051	42.11	8.19	4.59
	PT-LG (0.1)	0.0061 0.0009 0.0047	0.0062 0.0009 0.0048	49.10	5.72	11.68
	PT-LG (0.01)	0.0064 0.0008 0.0052	0.0065 0.0008 0.0053	48.58	8.38	11.43
	SGD	0.0357 0.0166 0.0289	0.0358 0.0167 0.0290	-	-	1.6
	ADAM	0.0313 0.0013 0.0278	0.0314 0.0013 0.0279	-	-	1.05
Lorenz	PT-RW	0.0192 0.0033 0.0113	0.0171 0.0037 0.0094	39.48	14.48	4.45
	PT-LG (0.1)	0.0181 0.0018 0.0117	0.0157 0.0018 0.0094	50.37	9.66	11.48
	PT-LG (0.01)	0.0173 0.0023 0.0123	0.0147 0.0024 0.0095	46.30	11.91	11.42
	SGD	0.0297 0.0019 0.0258	0.0289 0.0019 0.0249	-	-	1.3
	ADAM	0.0322 0.0121 0.0299	0.0323 0.0122 0.0300	-	-	1.2

[7] Chandra, R., Jain, K., Deo, R. V., & Cripps, S. (2019). Langevin-gradient parallel tempering for Bayesian neural learning. Neurocomputing, 359, 315-326.

Results: Pattern classification

Results from [8]

Dataset	Method	Train (mean, std, best)	Test (mean, std, best)	Swap perc.	Accept perc.	Time (min.)
Iris	PT-RW	51.39 15.02 91.43	50.18 41.78 100.00	52.56	95.32	1.26
	PT-LG (0.1)	64.93 21.51 100.00	59.33 39.84 100.00	51.08	51.48	1.81
	PT-LG (0.01)	97.32 0.92 99.05	96.76 0.96 99.10	51.77	97.55	2.09
	SGD	99.11, 0.23, 100	96.92, 1.05, 97.5	-	-	0.6
	Adam	99.61, 0.466, 1.0	96.83, 0.11, 97.5	-	-	0.07
Ionosphere	PT-RW	68.92 16.53 91.84	51.29 30.73 91.74	50.61	89.32	3.50
	PT-LG (0.1)	65.78 10.87 85.71	84.63 9.54 96.33	47.83	45.46	4.70
	PT-LG (0.01)	98.55 0.55 99.59	92.19 2.92 98.17	51.77	92.40	5.07
	SGD	99.14,0.28,100	95.5,1.00,97.5	-	-	0.98
	Adam	100, 0.0, 100	95.17, 0.62, 97.5	-	-	0.04
Cancer	PT-RW	83.78 20.79 97.14	83.55 27.85 99.52	40.18	89.71	2.78
	PT-LG (0.1)	83.87 17.33 97.55	90.59 16.67 99.52	41.71	43.87	5.13
	PT-LG(0.01)	97.00 0.29 97.75	98.77 0.32 99.52	49.25	94.67	5.09
	SGD	99.11,0.23,100	96.92,1.05,97.5	-	-	0.94
	Adam	99.49,0.47,100	96.67,1.18,97.5	-	-	0.17
Bank	PT-RW	78.39 1.34 80.11	77.49 0.90 79.45	49.13	61.59	27.71
	PT-LG(0.1)	77.90 1.92 79.71	77.74 1.27 79.57	46.17	29.61	69.89
	PT-LG(0.01)	80.75 1.45 85.41	79.96 0.81 82.61	50.00	31.50	86.94
	SGD	80.53,0.15,80.84	79.95,1.15,80.20	-	-	1.01
	Adam	80.88, 0.15, 81.16	80.18,0.14,80.51	-	-	0.07

[8] Chandra, R., Jain, K., Deo, R. V., & Cripps, S. (2019). Langevin-gradient parallel tempering for Bayesian neural learning. Neurocomputing, 359, 315-326.

Conclusions

- The experimental results show that proposals by Langevin-gradients significantly improves convergence with better prediction and classification performance when compared to random-walk proposal distributions.
- In the case of classification problems, we observe that both variations of Langevin-gradient parallel tempering significantly improves the classification performance for the majority of the problems.
- In some cases, the classification performance is slightly improved; however, Langevin-gradients have used high computational time in general.

Bayesian Neuroevolution: MCMC with Evolutionary Algorithms

Arpit Kapoor, Eshwar Nukala, Rohitash Chandra, Bayesian neuroevolution using distributed swarm optimization and tempered MCMC, Applied Soft Computing, Vol. 129, 2022, 109528

UNSW THE UNIVERSITY OF NEW SOUTH WALES

Introduction

- 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 optimisation methods to provide an alternative to gradient-based training, where the training algorithm is not constrained to the architecture of the network and has the potential to address local-minima and vanishing gradient problems.
- Parallel tempering MCMC addresses some of these limitations given that they can sample multimodal posterior distributions and utilize high-performance computing.
- We present a synergy of neuroevolution and Bayesian neural networks where the particle swarm optimisation (PSO) are used for creating proposals for tempered MCMC sampling.

UNSW THE UNIVERSITY OF NEW SOUTH WALES

Framework

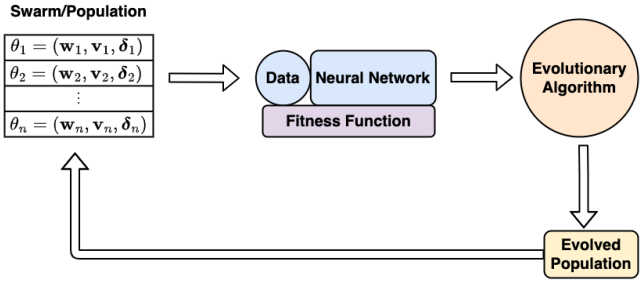


Figure: Neuroevolution

Framework

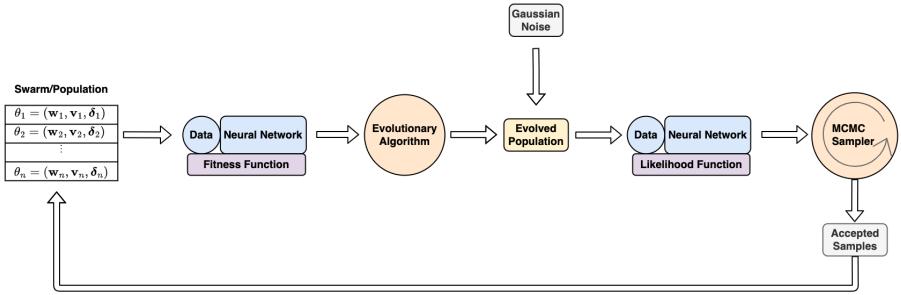


Figure: Bayesian Neuroevolution via MCMC

Parallel implementation

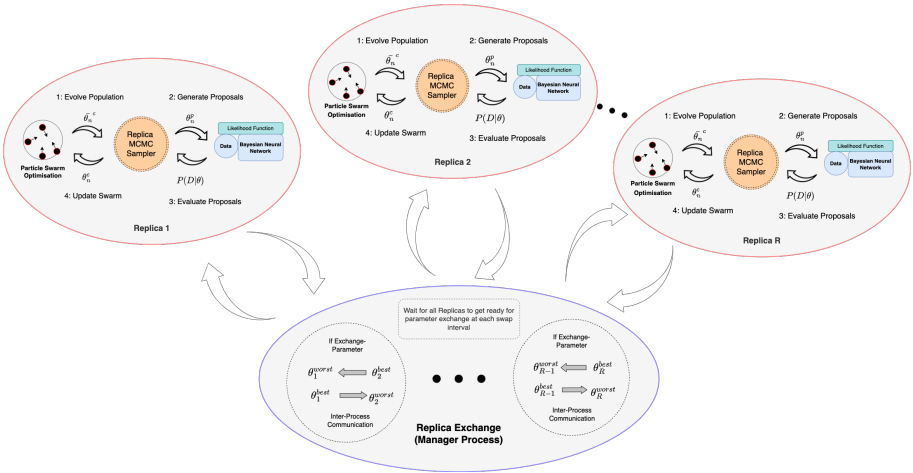


Figure: The architecture of particle-based tempered MCMC featuring a parallel computing environment with inter-process communication.

Results

Table: Performance accuracy on different classification datasets

Problem	Method	Train Accuracy [mean, std, best]	Test Accuracy [mean, std, best]	Swapped(%)	Accepted(%)	Time(s)
Iris	RW-tMCMC	51.39, 15.02, 91.43	50.18, 41.78, 100.0	52.56	95.32	1.26
	LG-tMCMC	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	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	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	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

Conclusions

- The results show that Bayesian neuroevolution with a sufficient number of particles and samples 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.

Bayesian deep learning with MCMC: Autoencoders

R Chandra, M Jain, M Maharana, PN Krivitsky, Revisiting Bayesian Autoencoders with MCMC, IEEE Access 10, 40482 - 40495

Introduction

- Autoencoders have the ability to compress data and provide dimensionality reduction.
- Although prominent deep learning methods have been used to enhance autoencoders, the need to provide robust uncertainty quantification remains a challenge.
- Recent advances with parallel computing and advanced proposal schemes that incorporate gradients have opened routes less travelled for MCMC.
- Next, we present Bayesian autoencoders powered MCMC sampling implemented using parallel computing and Langevin gradient proposal scheme.

Autoencoders

An autoencoder consists of two major parts: an encoder function $f_\phi(\mathbf{x})$ and a decoder function $g_\psi(\mathbf{h})$, where $\mathbf{x}$ is the data that represents a set of features and $\mathbf{h}$ is a set of latent (reduced) features. It is assessed by how well it the decoder is able to reconstruct the data from the encoding:

$$R_{\text{loss}} = \arg\min_{\phi,\psi} |\mathbf{x} - (g_\psi(f_\phi(\mathbf{x}))|^2.$$

In particular, $z(\mathbf{x}, \theta)$ be a feed-forward neural network, with θ being its set of weights and biases. An encoder–decoder pair is then constructed from it as

$$\mathbf{h} = f_\phi(\mathbf{x}) = z(\mathbf{x}, \phi)$$
$$\mathbf{x} = g_\psi(\mathbf{h}) = z(\mathbf{h}, \psi).$$

The encoder extracts the essential features into a reduced representation while the decoder is used to reconstruct the input from the reduced representation.

Autoencoders

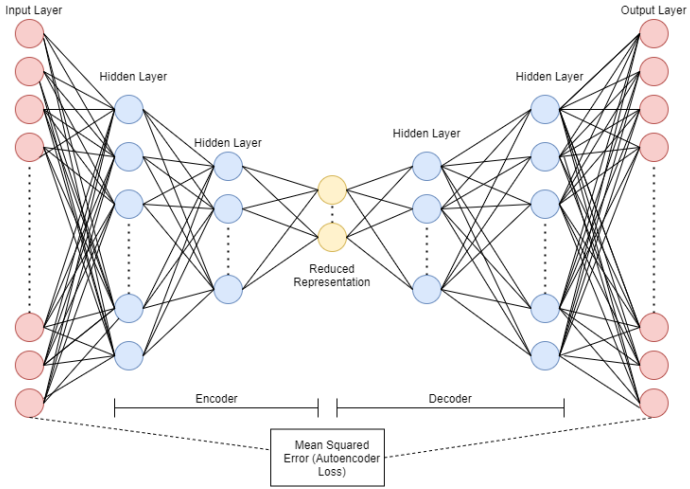


Figure: The autoencoder features an encoder and a decoder, which are highlighted.

Autoencoders

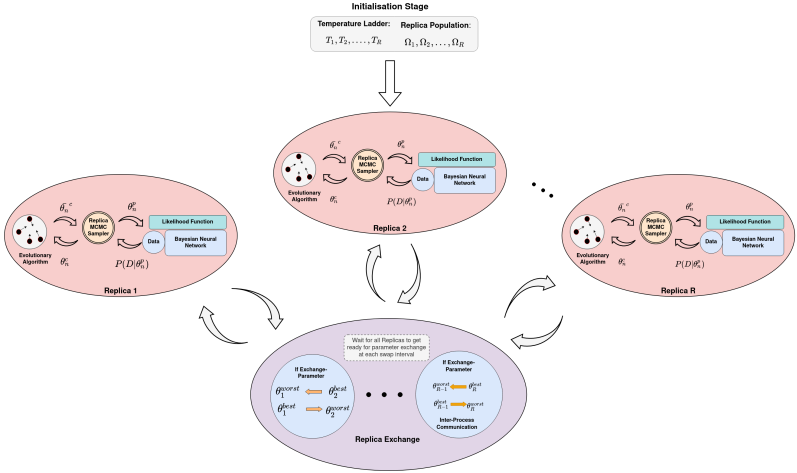


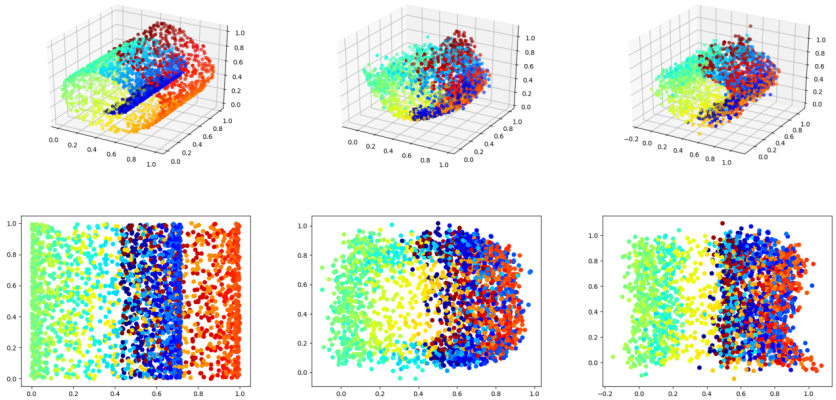
Figure: Bayesian autoencoder framework highlighting tempered MCMC utilising parallel computing and autoencoder neural network.

Autoencoder Config.

Table: Bayesian autoencoder topology showing the total number of parameters (weights and biases).

Data	Input Layer	Hidden Layer 1	Hidden Layer 2	Hidden Layer 3	Hidden Layer 4	Output Layer	Total
Madelon	500–450	450–400	400–300	300–400	400–450	450–500	1053701
Coil 2000	85–70	70–60	60–50	50–60	60–70	70–85	27037
Swiss Roll	3–128	128–64	64–2	2–64	64–128	128–3	21914
Swiss Roll*	3–3	3–2	2–2	2–2	2–3	3–3	50

Results: Swiss Roll



(a) Original (b) Canonical Autoencoder (c) Bayesian Autoencoder

Parameter tuning

Table: Performance (MSE) with different settings of MCMC parameters for the two instances of the Swiss Roll dataset

Method	Topology	MSE	Acceptance Percentage	Step-size	Learning-rate
Canonical Autoencoder	Swiss Roll	0.012	-	-	-
Bayesian Autoencoder	Swiss Roll	0.025	29.25	0.03	0.04
Bayesian Autoencoder	Swiss Roll	**0.011**	**16**	**0.005**	**0.01**
Bayesian Autoencoder	Swiss Roll	0.104	41.63	0.08	0.09
Bayesian Autoencoder	Swiss Roll*	0.078	0.3	0.03	0.04

Table: Autoencoder mean-squared-error(MSE)

Method	Swiss Roll Best (Mean, Std)	Madelon Best (Mean, Std)	Coil 2000 Best (Mean, Std)
Canonical Autoencoder (SGD)	0.084 (0.087, 0.01)	0.0389 (0.042,0.02)	0.044 (0.0469, 0.03)
Canonical Autoencoder (Adam)	0.012 (0.019, 0.005)	0.0192 (0.0199, 0.007)	0.0117 (0.014,0.002)
Bayesian Autoencoder (SGD)	0.185 (0.349,0.15)	0.348 (0.378,0.018)	0.2051 (1.07,0.51)
Bayesian Autoencoder (Adam)	0.011 (0.018,0.003)	0.018 (0.024,0.01)	0.0152 (0.026,0.008)

UNSW THE UNIVERSITY OF NEW SOUTH WALES

Results

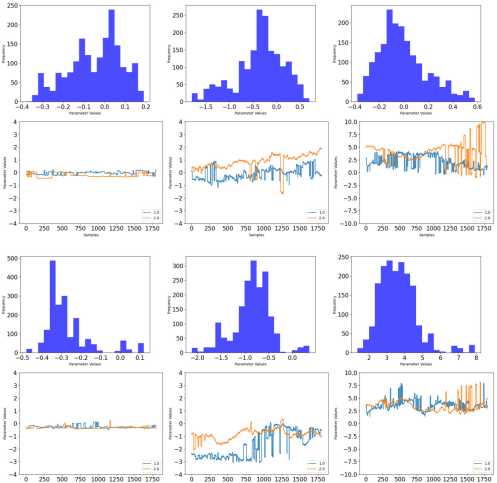
Results from [9]

Table: Bayesian autoencoder accuracy rates comparison of established methods from literature

Method	Algorithm	Madelon Best (Mean, Std)	Coil 2000 Best (Mean, Std)
Autoencoder based classifier (Pulgar et al., 2018)	kNN	0.547 (-,-)	0.929 (-,-)
Ensemble Classification using Dimensionality Reduction (Schclar et al., 2013)	RSE	0.5565 (- , 0.0263)	-
Autoencoder inspired unsupervised feature selection (Han, 2018)	kNN	0.71 (-,-)	-
Multi-objective Evolutionary Approach (Singh et al., 2020)	Naive Bayes	-	0.93 (-, 0.014)
Multi-objective Evolutionary Approach (Singh et al., 2020)	SVM	-	0.948 (-, 0.002)
Bayesian-Autoencoder (SGD)	kNN	0.2 (0.15, 0.02)	0.44 (0.38, 0.05)
Bayesian-Autoencoder (Adam)	kNN	0.556 (0.525, 0.021)	0.94 (0.91, 0.018)
Bayesian-Autoencoder (Adam)	SVM	0.598 (0.556, 0.024)	0.951 (0.93, 0.009)

[9] Chandra, R., Jain, M., Maharana, M., & Krivitsky, P. N. (2021). Revisiting Bayesian Autoencoders with MCMC. arXiv preprint arXiv:2104.05915.

Posterior



(a) Madelon (b) Coil 2000 (c) Swiss Roll

Conclusions

- Our results indicate that the Bayesian framework is as good as related methods from the literature in terms of performance accuracy, with additional feature of robust model uncertainty quantification for generation of reduced datasets.
- The paper motivates further application of the Bayesian framework for other deep learning models that have data generation features, such as the generative adversarial networks.

Coevolutionary Multitask Learning

Chandra R; Cripps S, 2018, 'Coevolutionary multi-task learning for feature-based modular pattern classification', Neurocomputing, vol. 319, pp. 164 - 175
Chandra R; Ong YS; Goh CK, 2018, 'Co-evolutionary multi-task learning for dynamic time series prediction', Applied Soft Computing Journal, vol. 70, pp. 576 - 589
Chandra R; Gupta A; Ong YS; Goh CK, 2018, 'Evolutionary Multi-task Learning for Modular Knowledge Representation in Neural Networks', Neural Processing Letters, vol. 47, pp. 993 - 1009
Chandra R; Ong YS; Goh CK, 2017, 'Co-evolutionary multi-task learning with predictive recurrence for multi-step chaotic time series prediction', Neurocomputing, vol. 243, pp. 21 - 34

Coevolutionary Multitask Learning

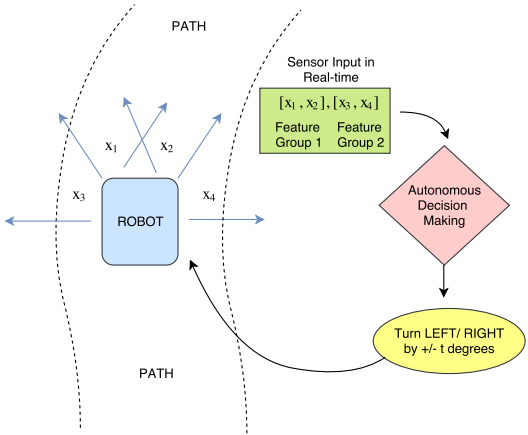


Figure: Robot vehicle in an uncertain environment.

Coevolutionary Multitask Learning

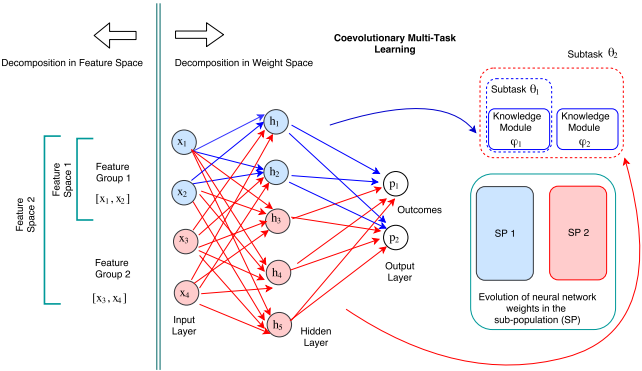


Figure: Coevolutionary Multitask Learning (dynamic inputs)

Coevolutionary Multitask Learning

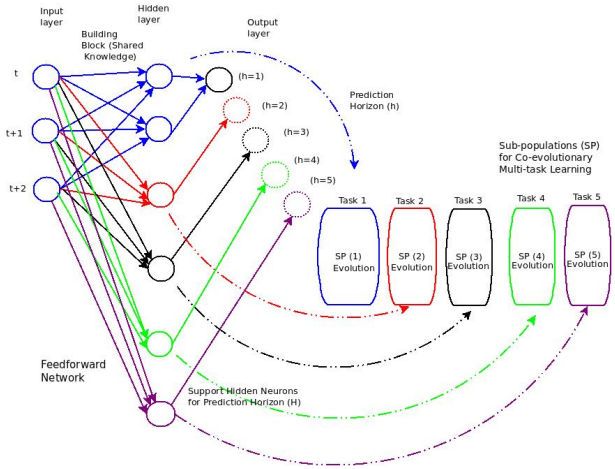


Figure: Coevolutionary Multitask Learning (dynamic outputs)

Coevolutionary Multitask Learning for pattern classification

Table: Comparison with related methods

Prob.	Data	BP	NE	CNE	CMTL-θ_1	CMTL-θ_2	CMTL- θ_3
Wine	Train	97.25 $\pm$ 0.34	66.43 $\pm$ 10.10	75.00 $\pm$ 7.58	75.00 $\pm$ 5.66	87.33 $\pm$ 3.25	91.75 $\pm$ 2.19
	Test	95.25 $\pm$ 2.91	74.92 $\pm$ 9.71	86.17 $\pm$ 3.55	74.88 $\pm$ 5.57	83.96 $\pm$ 4.54	93.99 $\pm$ 2.74
Iris	Train	92.15 $\pm$0.82	69.79 $\pm$ 8.49	91.39 $\pm$ 2.91	54.83 $\pm$ 5.11	76.00 $\pm$ 4.17	88.17 $\pm$ 2.34
	Test	83.67 $\pm$1.79	71.00 $\pm$ 8.08	90.00 $\pm$ 2.27	65.03 $\pm$ 3.88	78.36 $\pm$ 5.66	87.45 $\pm$ 5.02
Ionos.	Train	95.18 $\pm$1.33	95.06 $\pm$ 0.60	90.44 $\pm$ 1.14	93.70 $\pm$ 0.52	94.74 $\pm$ 0.73	95.26 $\pm$ 0.73
	Test	83.33 $\pm$2.53	94.65 $\pm$ 1.04	90.73 $\pm$ 1.80	98.23 $\pm$ 0.24	98.69 $\pm$ 0.23	99.01 $\pm$ 0.19
Zoo	Train	99.72 $\pm$ 0.56	100.00 $\pm$ 0.00	100.00 $\pm$ 0.00	90.63 $\pm$ 0.00	90.31 $\pm$ 0.45	90.00 $\pm$ 0.45
	Test	98.54 $\pm$ 3.30	90.42 $\pm$ 0.50	91.25 $\pm$ 0.80	100.00 $\pm$ 0.00	100.00 $\pm$ 0.00	100.00 $\pm$ 0.00
Cancer	Train	93.46 $\pm$ 0.38	95.45 $\pm$ 0.29	91.60 $\pm$ 0.90	97.52 $\pm$ 0.13	97.83 $\pm$ 0.29	97.81 $\pm$ 0.33
	Test	96.22 $\pm$ 0.64	97.13 $\pm$ 0.51	95.10 $\pm$ 0.63	95.30 $\pm$ 0.12	96.20 $\pm$ 0.23	96.41 $\pm$ 0.23
Lenses	Train	97.50 $\pm$5.25	55.15 $\pm$ 3.62	52.62 $\pm$ 2.87	44.58 $\pm$ 9.37	39.58 $\pm$ 7.89	62.50 $\pm$ 6.43
	Test	85.83 $\pm$5.33	67.67 $\pm$ 1.06	62.75 $\pm$ 2.26	2.92 $\pm$ 2.25	11.67 $\pm$ 4.65	79.17 $\pm$ 8.42

Reversible Jump MCMC

Megan Nguyen, Minh-Ngnoc Tran, Rohitash Chandra, Sequential reversible jump MCMC for training and design of Bayesian neural networks, Neurocomputing (in progress - to be submitted October 2022)
*Collaboration with University of Sydney

Reversible Jump MCMC

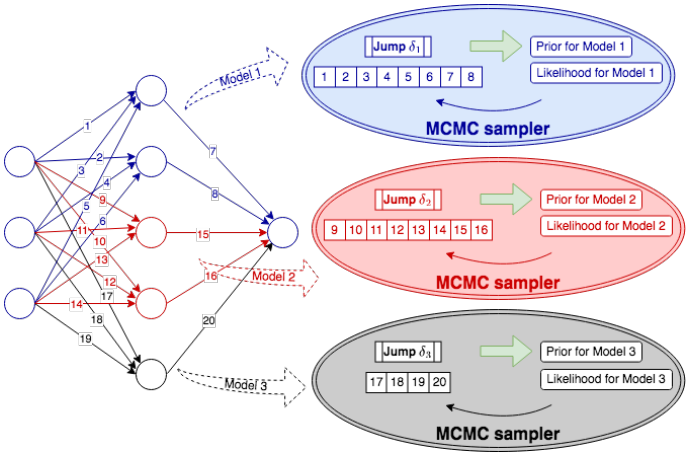


Figure: Reversible jump MCMC framework for dynamic hidden neurons (Jump-H)

Reversible Jump MCMC

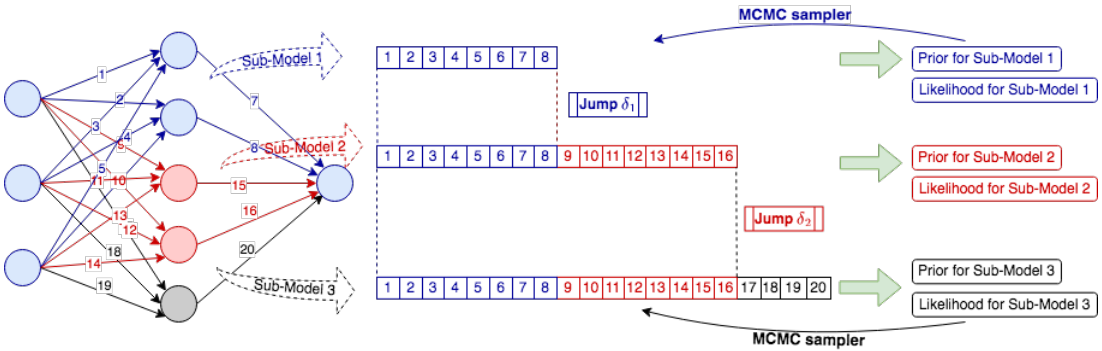


Figure: Reversible jump MCMC framework for dynamic hidden neurons (Jump-H)

Reversible Jump MCMC

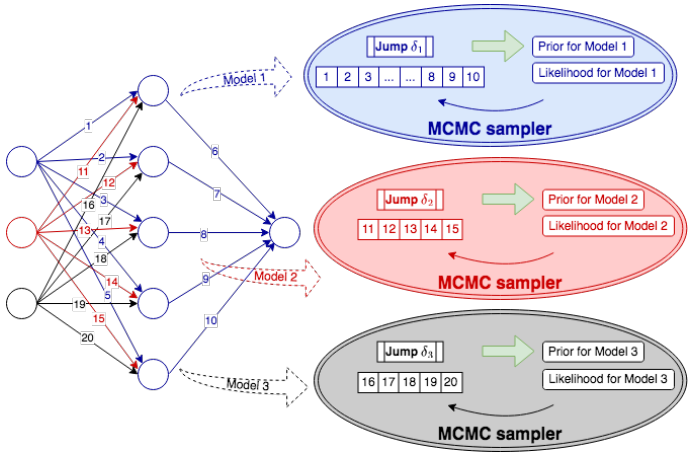


Figure: Reversible jump MCMC framework for dynamic input neurons (Jump-I)

Reversible Jump MCMC

Dataset	Model #	Model Arch.	Train (Mean)	Train (STD)	Test (Mean)	Test (STD)	Accept Perc
Poker	1	[6, 34, 2]	33.71556	0.59347	29.17316	1.18022	
	2	[8, 34, 2]	28.82634	0.59895	32.60676	1.14699	44.00000
	3	[10, 34, 2]	94.41705	3.60629	92.31749	3.84357	
	1	[10, 34, 2]	98.81356	0.00001	98.98649	0.00001	
	2	[10, 38, 2]	98.81356	0.00001	98.98649	0.00001	92.20000
	3	[10, 42, 2]	98.81356	0.00001	98.98649	0.00001	
Abalone	1	[8, 20, 2]	84.98197	0.18678	86.13325	0.44099	
	2	[9, 20, 2]	63.27447	0.00001	63.99522	0.00001	23.20000
	3	[10, 20, 2]	86.71057	0.00001	87.67943	0.00001	
	1	[10, 20, 2]	90.37482	0.13827	90.09145	0.21928	
	2	[10, 24, 2]	90.37911	0.13460	90.09984	0.21690	54.60000
	3	[10, 28, 2]	90.32047	0.16418	90.09577	0.18472	
Shuttle	1	[5, 37, 2]	31.87199	0.00019	31.41596	0.00001	
	2	[8, 37, 2]	34.57091	0.00012	35.14725	0.00001	62.50000
	3	[9, 37, 2]	99.69651	0.00055	99.73102	0.00001	
	1	[9, 37, 2]	99.69654	0.00001	99.73102	0.00001	
	2	[9, 41, 2]	99.69654	0.00001	99.73102	0.00001	80.80000
	3	[9, 45, 2]	99.69654	0.00001	99.73102	0.00001	

Table: Classification Prediction Problems

Conclusions

- Our results demonstrate that Reversible Jump MCMC has the ability to provide uncertainty quantification for Bayesian neural network architectures with dynamic inputs and dynamic hidden neuron settings.
- Furthermore, in Future work, we would like to apply the methodology for Robot contrail problems that emerge in uncertain environments caused by potential accidents or noise in input sensors that can cause accidents.

Lyapunov-based Robot Control and Path Planning

Harsha Patnala, Rohitash Chandra, Avinesh Prasad, Bibhya Sharma , Bayesian robot path planning with Lyapunov-based control via MCMC, (ongoing)
*Collaboration with University of the South Pacific (Fiji)

Lyapunov-based Robot Control and Path Planning

We are considering a 2D space with a single obstacle. A point mass on a plane is required to move to its target under its own instantaneous accelerations. Assume that there is one disk obstacle within the workspace. Derive the acceleration components to enable the point mass to move to its target, where it will stop, whilst successfully avoiding the disk obstacle.

System of ODE: $\dot{x} = \mathrm{v}$; $\dot{y} = \mathrm{w}$; $\dot{v} = \sigma_1$; $\dot{w} = \sigma_2$

v is the velocity in x direction, w is velocity in y direction, σ_1 is acceleration in x direction, σ_2 is acceleration in y direction. $\dot{w}$ is the derivative of σ_2, respectively.

Target Attraction:

The target with centre (τ_1, τ_2) and radius r_1 is described as:

$\mathrm{T} = \{(z_1, z_2) \in \mathbf{R}^2 : (z_1 - \tau_1)^2 + (z_2 - \tau_2)^2 \leq r_1^2\}$

z is the coordinates at any instant

UNSW THE UNIVERSITY OF NEW SOUTH WALES

Lyapunov-based Robot Control and Path Planning

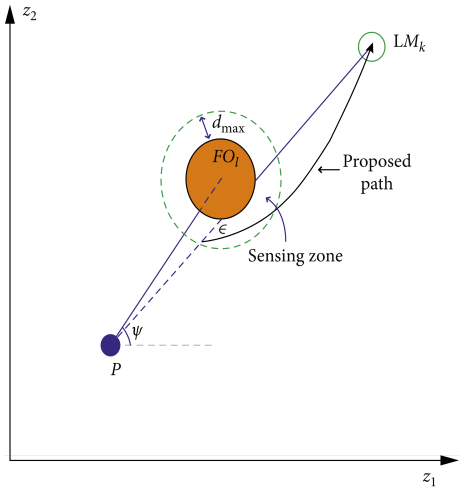


Figure: Robot track with singe obstacle.

Lyapunov-based Robot Control and Path Planning

Avoidance of fixed Obstacle :

A circular obstacle with centre (o_1, o_2) and radius r_o is described as :

$\mathrm{FO} = \{(z_1, z_2) \in \mathbf{r}^2 : (z_1 - o_1)^2 + (z_2 - o_2)^2 \leq r_o^2\}$

For its avoidance, we consider the potential function as:

$\mathrm{W} = \frac{1}{2}(x - o_1)^2 + \frac{1}{2}(y - o_2)^2 - \frac{1}{2}r_o^2$

Obstacle Avoidance and Target Attraction :

To ensure that the Lyapunov function to be proposed is zero at the centre of the target, we introduce another potential function :

$\mathrm{F} = \frac{1}{2}(x - \tau_1)^2 + \frac{1}{2}(y - \tau_2)^2$

where (tau_1, tau_2) is the centre of the target.

Lyapunov Function :

$\mathrm{L} = \mathrm{V} + \beta\frac{F}{W}$

where $\beta > 0$ is the control parameter.

Lyapunov-based Robot Control and Path Planning

Controller Design:

$$\dot{L} = \frac{\partial L}{\partial x}\dot{x} + \frac{\partial L}{\partial y}\dot{y} + \frac{\partial L}{\partial v}\dot{v} + \frac{\partial L}{\partial w}\dot{w}$$
$$= \frac{\partial L}{\partial x}v + \frac{\partial L}{\partial y}w + v\sigma_1 + w\sigma_2$$
$$= \left(\frac{\partial L}{\partial x} + \sigma_1\right)v + \left(\frac{\partial L}{\partial y} + \sigma_2\right)w$$

$\dot{L}$ can be made non-positive by letting

$$-\delta_1 v = \frac{\partial L}{\partial x} + \sigma_1$$
$$-\delta_2 w = \frac{\partial L}{\partial y} + \sigma_2$$

where $\delta_1, \delta_2 > 0$ are called the convergence parameter.

It follows that

$$\sigma_1 = -\delta_1 v - \frac{\partial L}{\partial x}$$
$$\sigma_2 = -\delta_2 w - \frac{\partial L}{\partial y}$$

This model can further be extended to multiple obstacles.

Lyapunov-based Robot Control and Path Planning

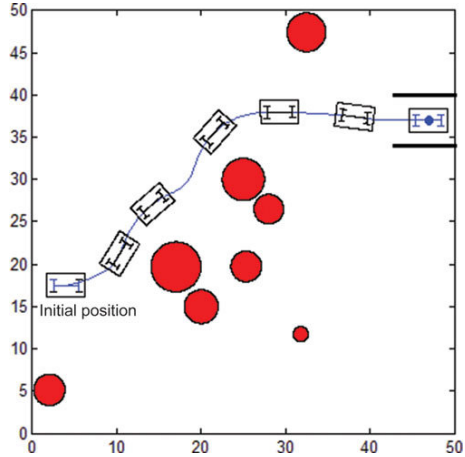


Figure: Robot track with multiple obstacles.

Lyapunov-based Robot Control and Path Planning

- Our objective is to optimize the parameters β, δ_1 and δ_2. We assume δ_1 and δ_2 to be same (δ).
- We employ MCMC sampling process using a Gaussian distribution as a prior for both the parameters.
- The distance travelled by the robot for a particular $\beta and \delta$ will be the error measure and part of my likelihood function.
- As a proof of concept, we will be considering two cases, one is the one obstacle case and the second one is the two obstacles case.

UNSW THE UNIVERSITY OF NEW SOUTH WALES

Lyapunov-based Robot Control and Path Planning

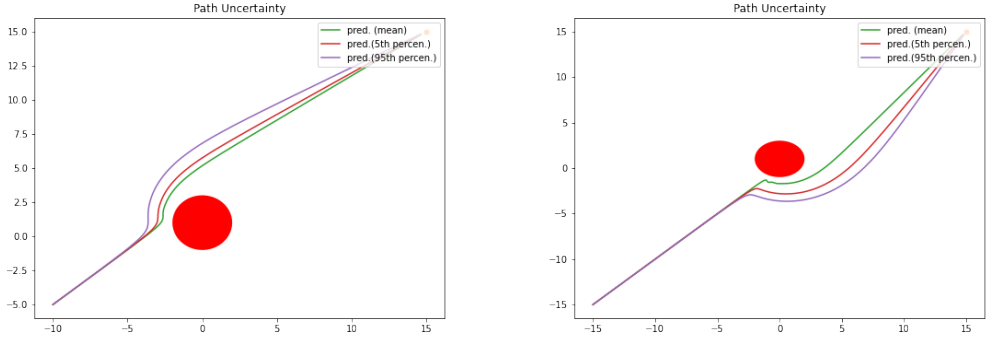


Figure: Single obstacle after MCMC sampling for 500 iterations

Lyapunov-based Robot Control and Path Planning

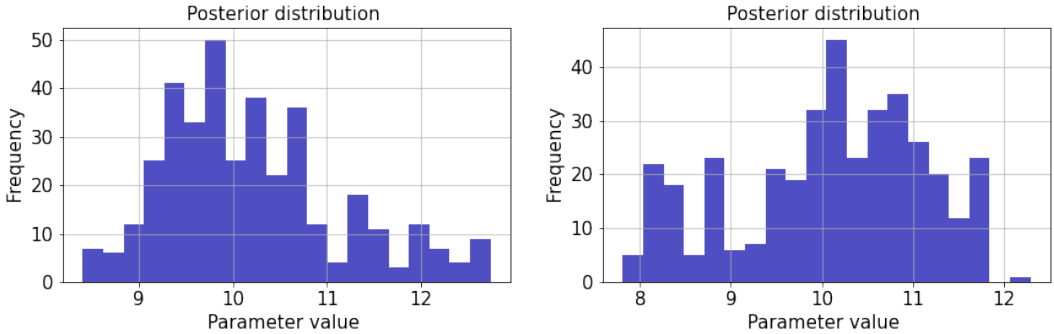


Figure: Posterior distribution of δ for 500 iterations

Lyapunov-based Robot Control and Path Planning

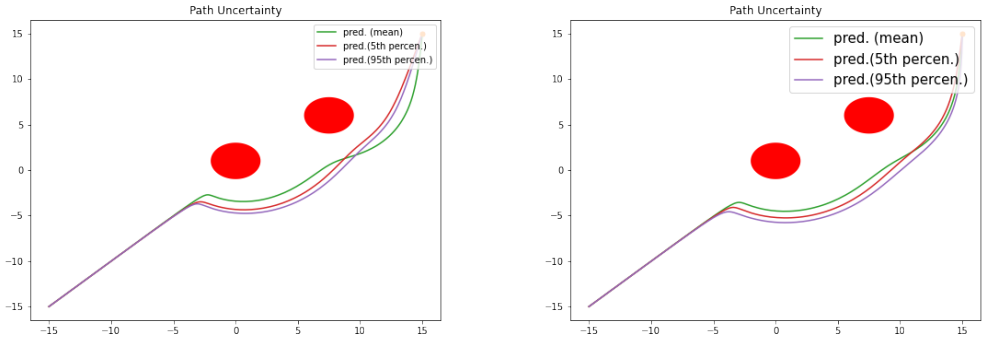


Figure: Two obstacle after MCMC sampling for 500 iterations

Extension

- The Lyapunov controller can be combined with Bayesian neural network, or other machine learning models to cater for uncertain inputs.

Current Future Research Work

- Bayesian Convolutional Neural Networks for multi-step time series prediction (Ongoing)
- Reverse Jump MCMC enabled Bayesian neural networks for robot path control
- Monte-Carlo path planning for wheel chair (Ongoing)
- Drone path planning with Lyapunov-based control via MCMC (Ongoing)
- Pingala: Bayesian deep learning framework in Python (planning stage)

UNSW THE UNIVERSITY OF NEW SOUTH WALES

Thanks, Namaste and Dhanyavaad

- Note all recent papers have software and code via Github repository (with links given in respective papers): `https://github.com/sydney-machine-learning`
- Apart from those already mentioned, I would like to thank all my co-authors in the papers discussed: Sally Cripps, Ratneel Deo, Arpit Kapoor, Konark Jain, Ashray Aman, Manavendra Maharana, Ayush Bhagat, Mahir Jain, and Pavel Krivitsky.
- Youtube link of this presentation will be shared later.

UNSW THE UNIVERSITY OF NEW SOUTH WALES