Revisiting Bayesian deep learning with advancements in MCMC

Rohitash Chandra

School of Mathematics and Statistics, UNSW Sydney

28th, April 2021

Overview

- The application of Bayesian inference for deep learning remains limited due to the computational requirements of the Markov Chain Monte Carlo (MCMC) methods.
- Recent advances in parallel computing and advanced proposal schemes in sampling, such as incorporating gradients has provided the potential for Bayesian deep learning methods to be implemented.
- This seminar provides a roadmap from the inception, implementation to the execution of Bayesian deep learning methods that begin with simple neural networks and extends to deep learning models that feature autoencoders and graph-based convolutional neural works.
- It will also feature a discussion on Bayesian recurrent neural networks, including long-short-term memory (LSTM) networks and implementation for Bayesian Transformer networks for language modelling.

Overview: Papers presented

- Chandra, R., Jain, K., Deo, R. V., & Cripps, S. (2019). Langevin-gradient parallel tempering for Bayesian neural learning. Neurocomputing, 359, 315-326.
- Chandra, R., Jain, K., Kapoor, A., & Aman, A. (2020). Surrogate-assisted parallel tempering for Bayesian neural learning. Engineering Applications of Artificial Intelligence, 94, 103700.
- Chandra, R., Jain, M., Maharana, M., & Krivitsky, P. N. (2021). Revisiting Bayesian Autoencoders with MCMC. arXiv preprint arXiv:2104.05915.
- R Chandra, A Bhagat, M Maharana, P. N. Krivitsky (2021). Bayesian graph convolutional neural networks via tempered MCMC. arXiv preprint arXiv:2104.08438

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.

Bayesian inference

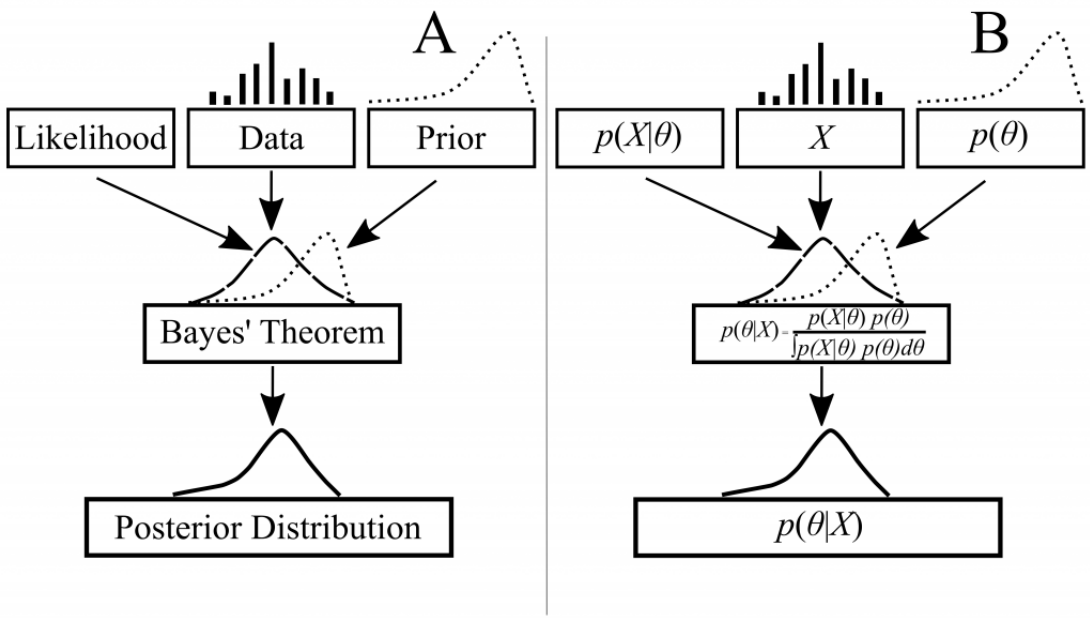


Figure: Bayesian inference overview [1]

Markov Chain Monte Carlo sampling methods (MCMC) implement Bayesian inference that sample from a probability distribution. This is based on constructing a Markov chain after a number of steps that has the desired distribution as its equilibrium distribution.

[1] Source: Google images

Bayesian inference

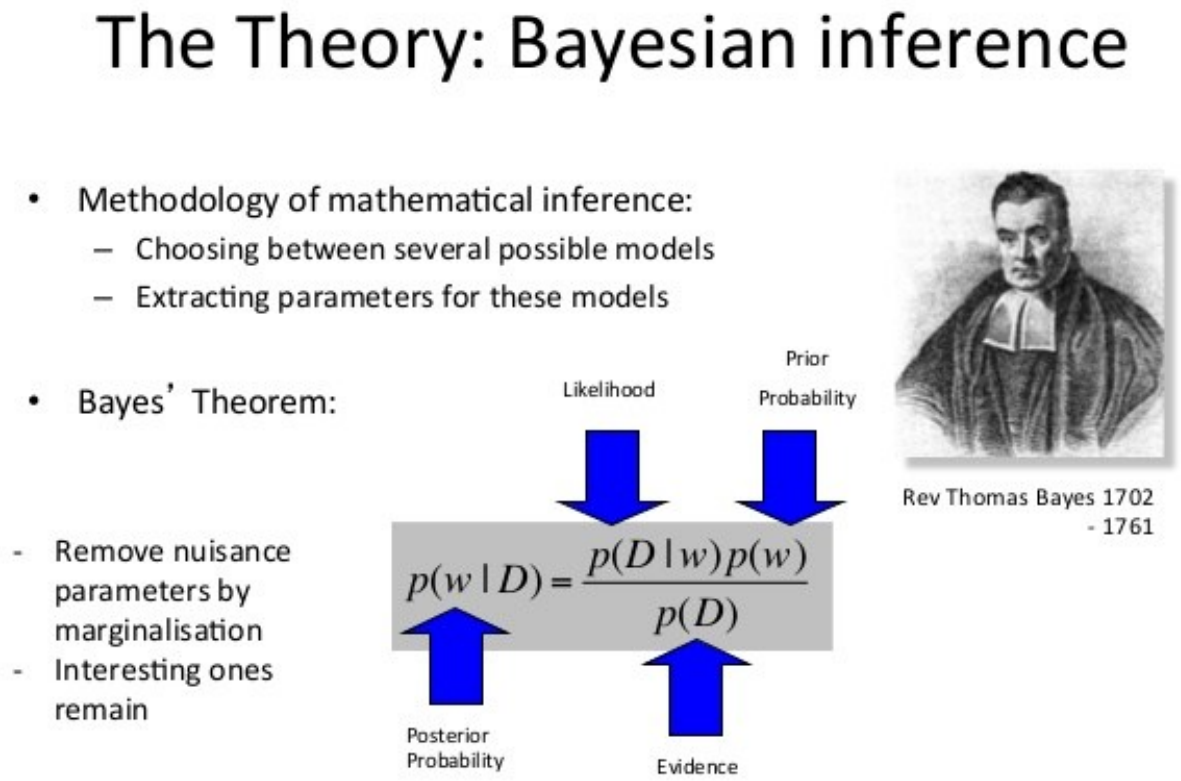


Figure: Bayesian inference overview [2]

[2] Source: Google images

Bayesian inference

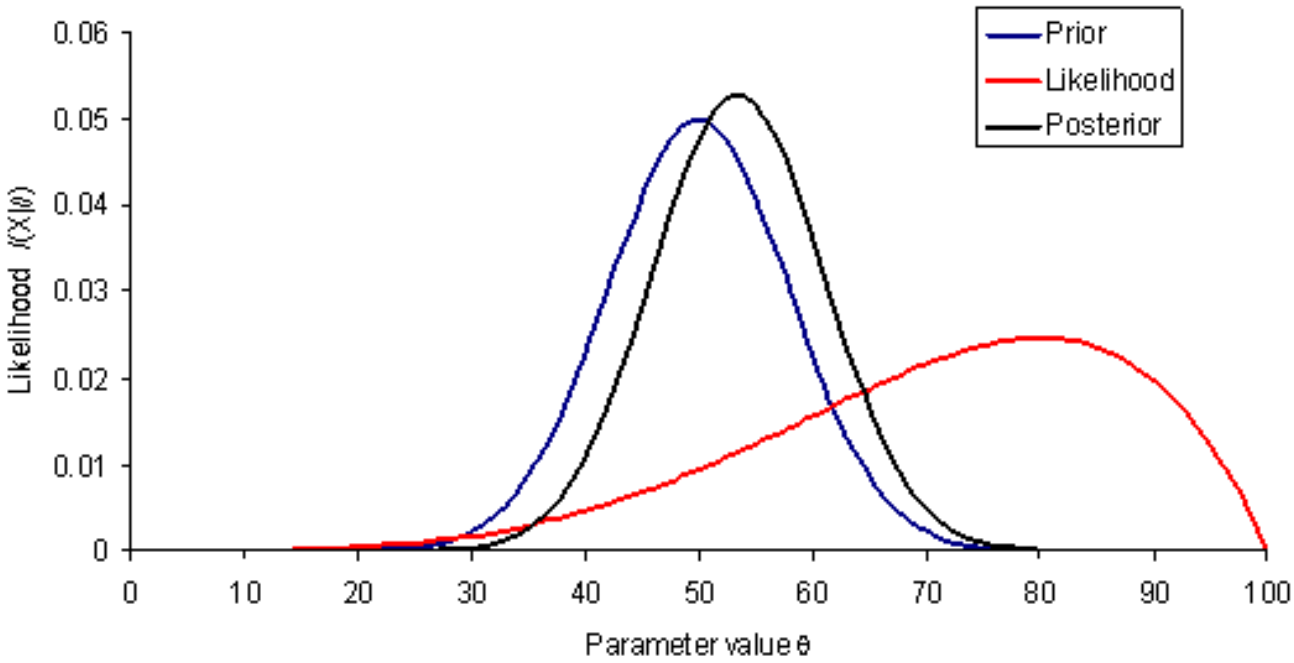


Figure: Bayesian inference overview [3]

[3] Source: Google images

Bayesian neural network and MCMC sampling

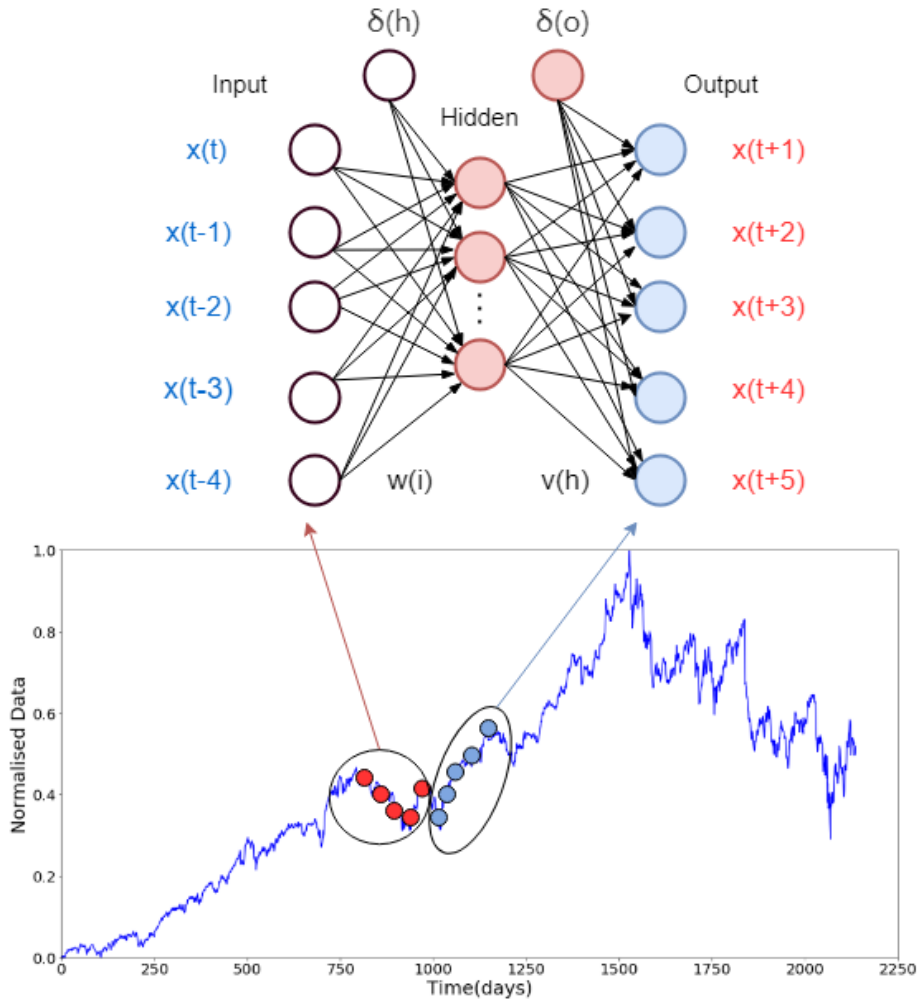


Figure: The time series (shown in red circles) is used as input for the neural network which predicts 5 steps-ahead in time (shown by blue circles). A sliding window approach is used to reconstruct the dataset in this way using Taken's theorem.

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.

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

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

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: Log-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

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

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.

Pattern classification: Multinomial Likelihood

In the case of discrete outcomes with K possible classes, we assume that the data $\mathbf{y} = (y_1, \ldots, y_n)$ is generated from a multinomial distribution with parameter vector $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_K)$ where $\sum_{k=1}^{K} \pi_k = 1$. In order to define the likelihood, we introduce a set of indicator variables $z_{i,k}$ where

$$z_{i,k} = \begin{cases} 1, & \text{if } y_i = k \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

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

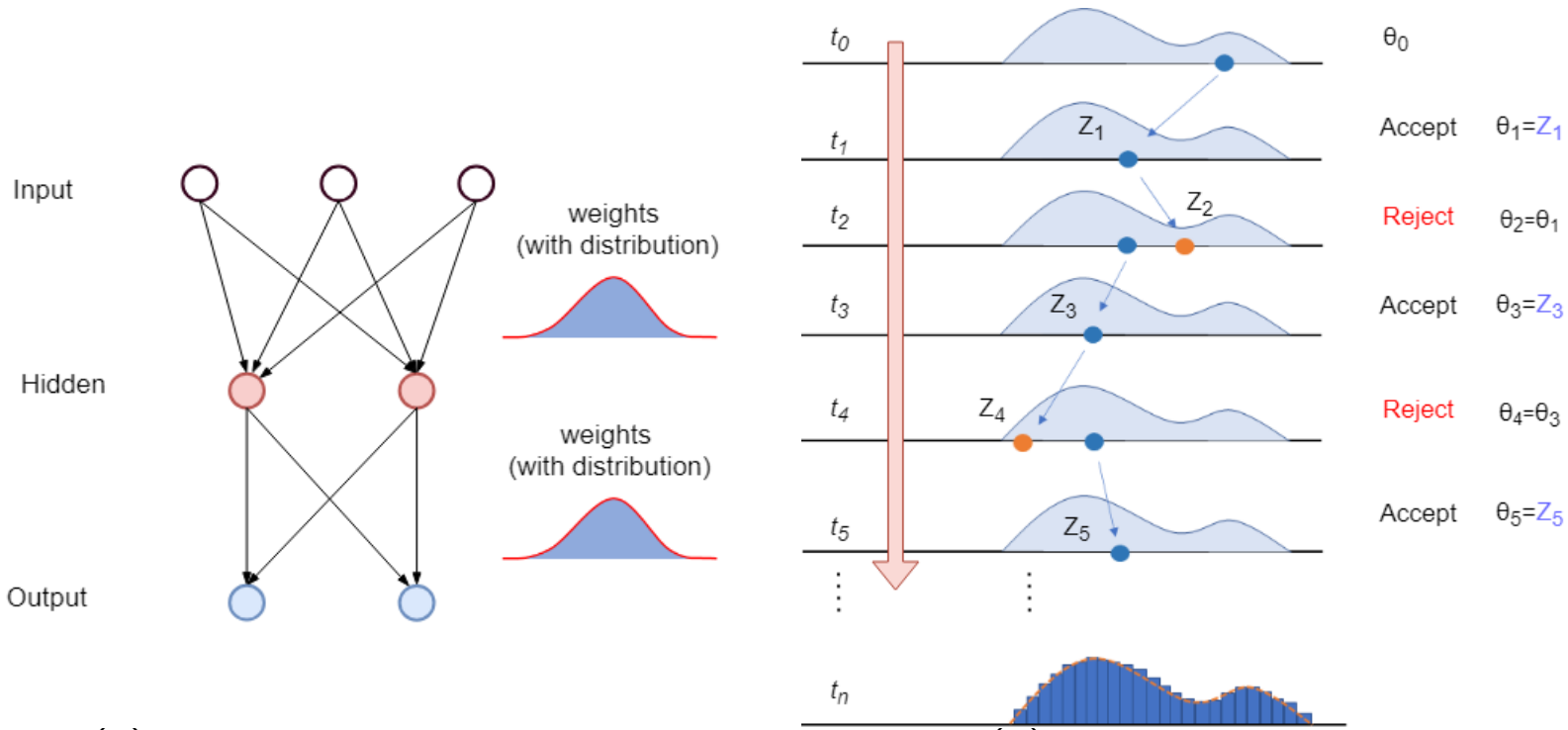
$$p(\mathbf{y}_| \boldsymbol{\pi}) \quad = \quad \prod_{i=1}^{n} \prod_{k=1}^{K} \pi_k^{z_{i,k}} \tag{5}$$

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

$$\pi_k = \frac{\exp(f(x_k))}{\sum_{j=1}^{K} \exp(f(x_j))} \tag{6}$$

where, $f(x)$ is the output given by the neural neural network.

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

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

$$\bar{\boldsymbol{\theta}}^k = \boldsymbol{\theta}^k + r \times \nabla E(\boldsymbol{\theta}^k), \tag{8}$$

$$E = \sum_{t \in S} (x_t - f(\mathbf{x_t}))^2 \tag{9}$$

$$\nabla E(\boldsymbol{\theta}^k) = \left(\frac{\partial E}{\partial \theta_1}, \dots \frac{\partial E}{\partial \theta_L} \right) \tag{10}$$

where r is the learning rate, $\Sigma_\theta = \sigma_\theta^2 I$ and I is the $L \times L$ identity matrix and L refers to the number of model parameters (weights and biases). Based on a user defined probability ϕ, the proposal $\boldsymbol{\theta}^p$ can be either

- 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 $\mathbf{y}_{\mathcal{A}_{D,T}}$ 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

Figure: Langevin-gradient MCMC [4]

[4] Chandra, R., Azizi, L., & Cripps, S. (2017, November). Bayesian neural learning via langevin dynamics for chaotic time series prediction. In International Conference on Neural Information Processing (pp. 564-573). Springer, Cham.

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

$$\alpha = min\left(1, \frac{p(\theta_m^k|D)^{\beta_m} q(\theta_m^k|\theta_m^p)}{p(\theta_m^k|D)^{\beta_m} q(\theta_m^p|\theta_m^k)}\right). \tag{11}$$

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

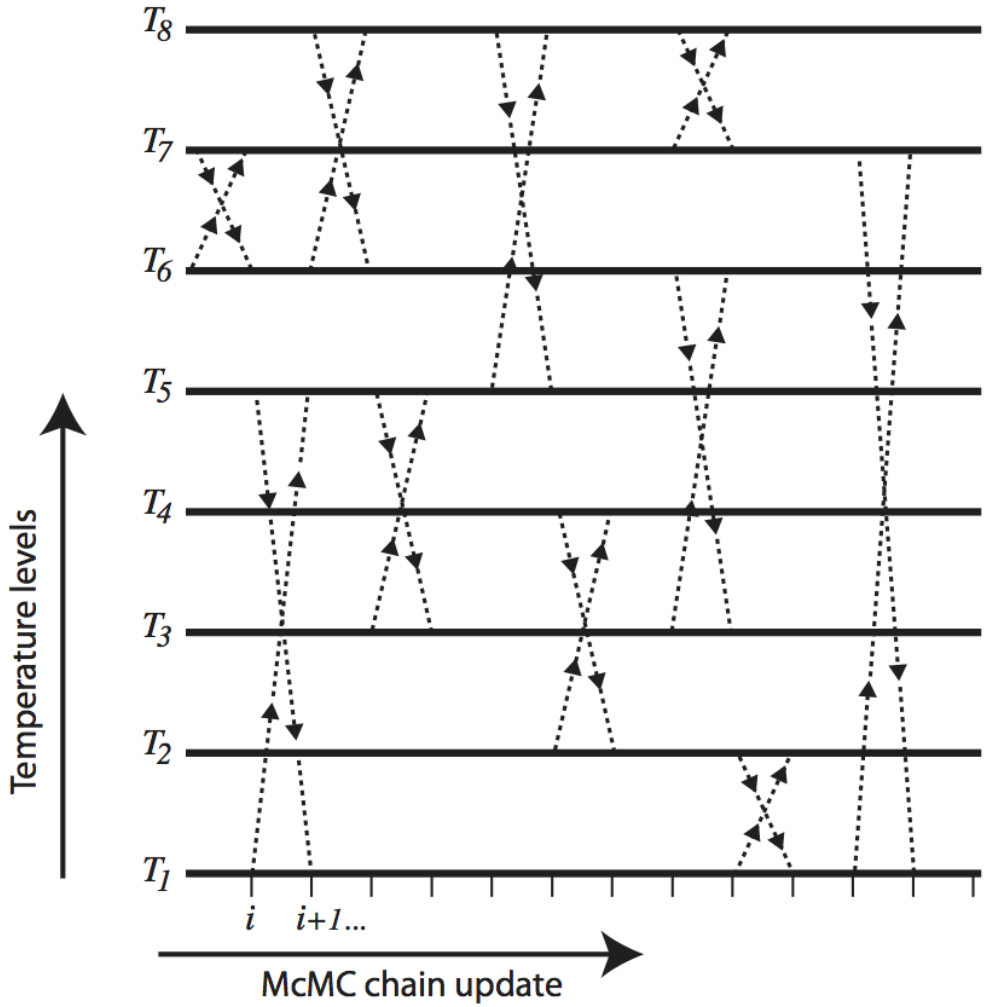


Figure: Replica swapping [5]

Parallel tempering

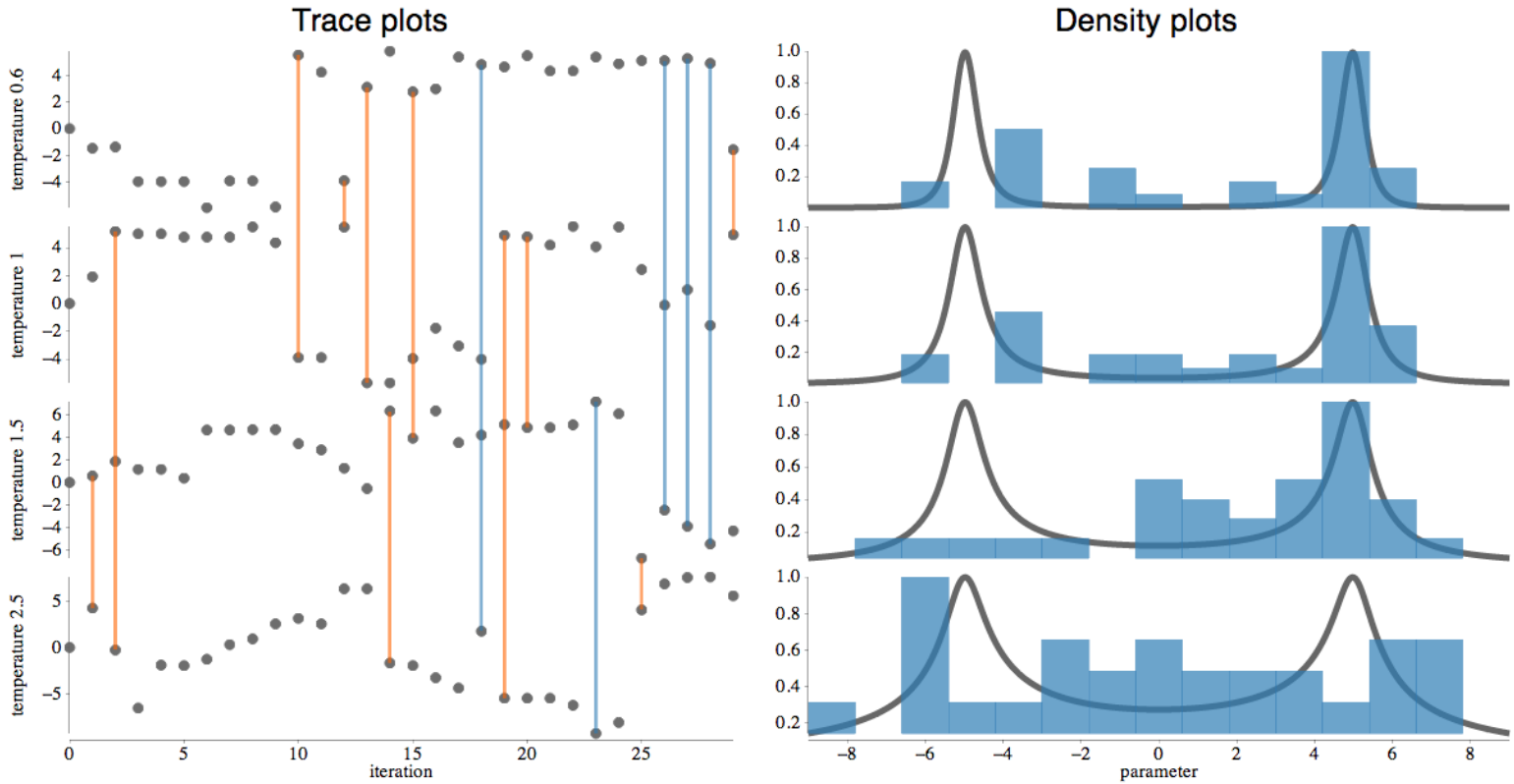


Figure: Parallel tempering - sampling [6]

[6] Source: Google images

Parallel tempering MCMC

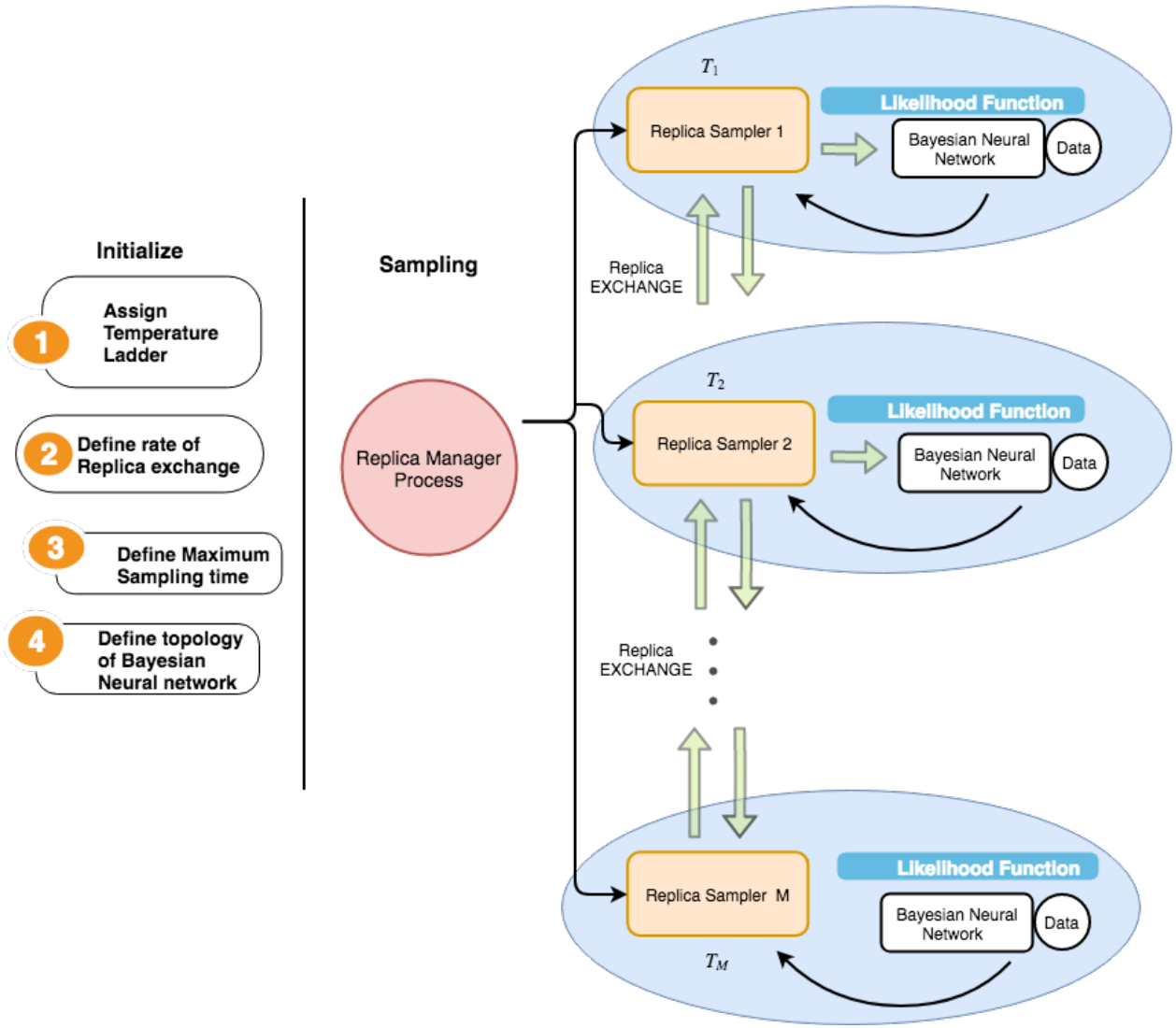


Figure: An overview of the different replicas that are executed on a multi-processing architecture. The man process controls the replicas and enables them to exchange when needed.

Adam optimiser: Adaptive moment estimation

Adaptive moment estimation (Adam) is an effective stochastic optimization method that only requires first-order gradients with a small amount of memory requirement focused in adapting learning rate [7]. 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 0.

[7] Kingma, Diederik P., and Jimmy Ba. "Adam: A method for stochastic optimization." arXiv preprint arXiv:1412.6980 (2014).

Results: Time series prediction

Results from [8]

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

[8] 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 [9]

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

[9] 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.
- We also observe that the lower learning rate (0.01) gives the best performance for classification problems. However, it shows to be less effective for the time series prediction problems.
- The sampling is highly sensitive to the Langevin-gradient learning rate which needs to be tailored for the type of the problem.

Surrogate-based MCMC parallel tempering framework

Introduction

- Bayesian inference faces several challenges given a large number of parameters, complex and multimodal posterior distributions, and computational complexity of large neural network models.
- Parallel tempering MCMC addresses some of these limitations given that they can sample multimodal posterior distributions and utilize high-performance computing.
- However, certain challenges remain given large neural network models and big data.
- Surrogate-assisted optimization features the estimation of an objective function for models which are computationally expensive.
- Next, we address the inefficiency of parallel tempering MCMC for large-scale problems by combining parallel computing features with surrogate assisted likelihood estimation that describes the plausibility of a model parameter value, given specific observed data.

Framework

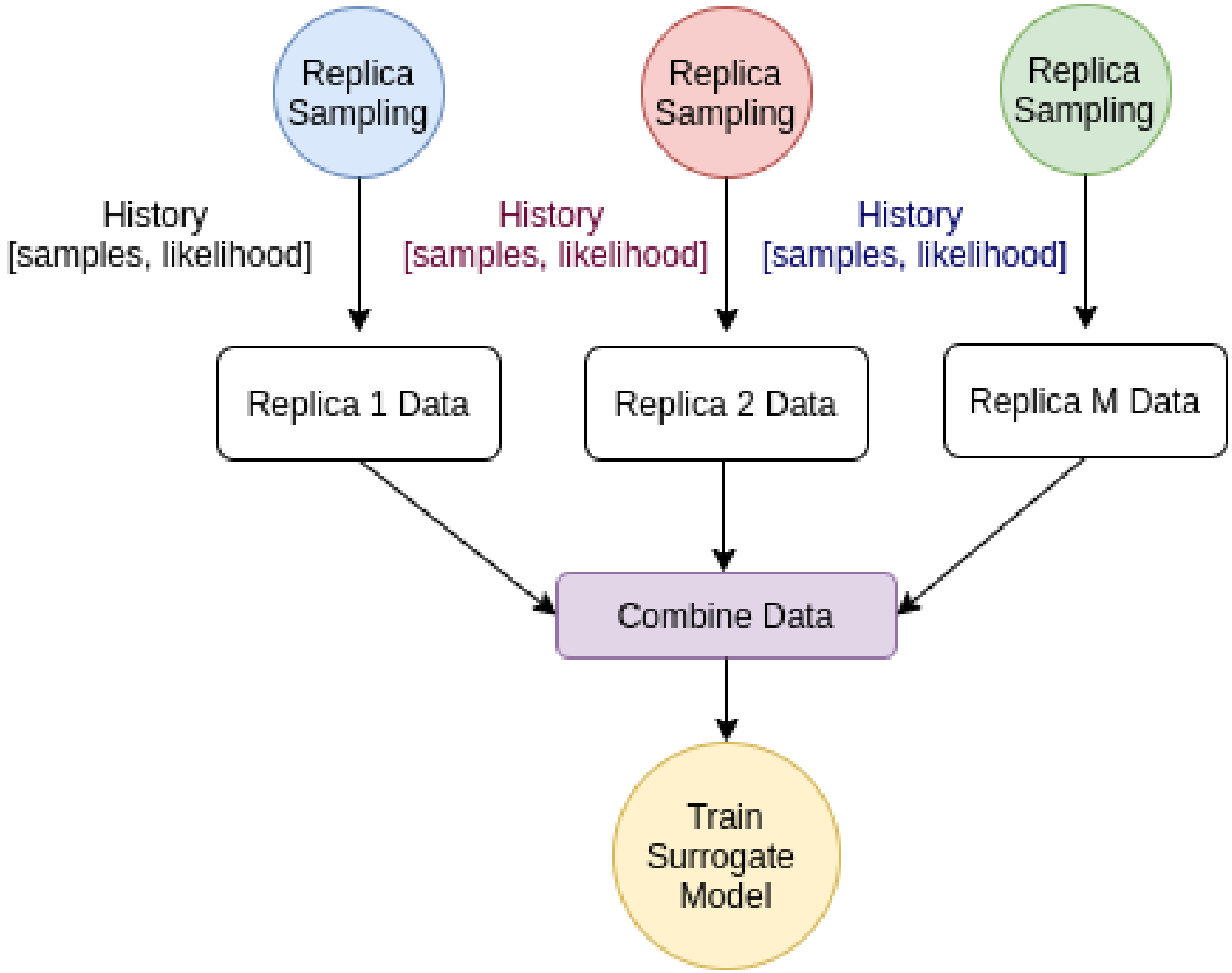


Figure: Data collection for training surrogate model

Framework

The surrogate model is constructed by training from experience which is given by the set of input $\mathbf{x_{i,s}}$ with corresponding true-likelihoods $L_{i,s}$; where s represents the sample and i represents the replica.
Hence, input features (Φ) for the surrogate is developed by combining $\mathbf{x_{i,s}}$ using samples generated (θ) across all the replicas for a given surrogate interval (ψ). The surrogate interval defines the batch size of the training data for the surrogate model which goes through incremental learning. All the respective replicas sample until the surrogate interval is reached and then the manager process collects the sampled data to create a training data batch for the surrogate model. This can be formulated as follows,

$$\begin{aligned} \Phi &= ([\mathbf{x_{1,s}}, \ldots, \mathbf{x_{1,s+\psi}}], \ldots, [\mathbf{x_{M,s}}, \ldots, \mathbf{x_{M,s+\psi}}]) \\ \lambda &= ([L_{1,s}, \ldots, L_{1,s+\psi}], \ldots, [L_{M,s}, \ldots, L_{M,s+\psi}]) \\ \Theta &= [\Phi, \lambda] \end{aligned} \tag{12}$$

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

Surrogate model

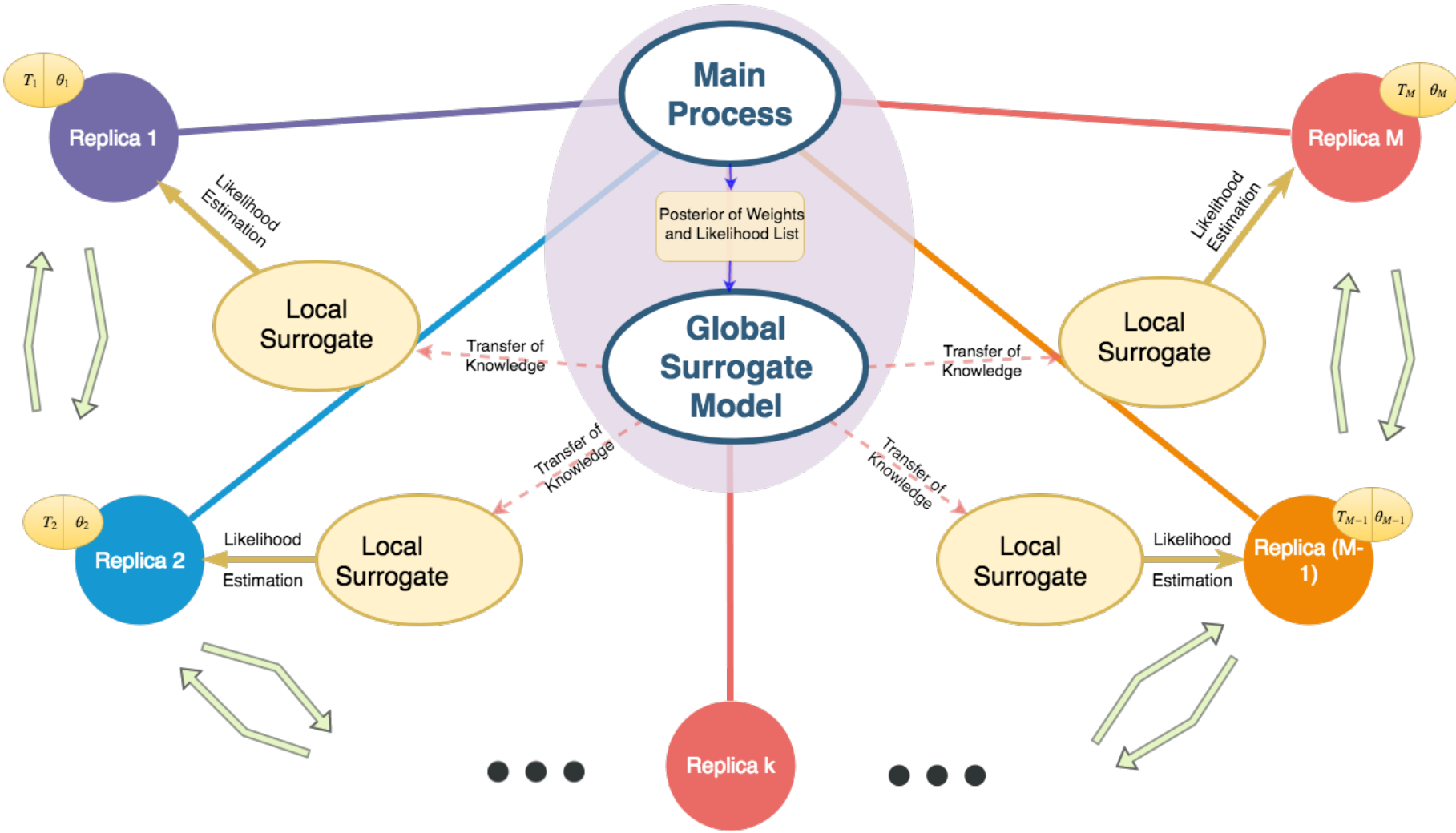


Figure: Surrogate-assisted multi-core parallel tempering features surrogates to estimate the likelihood function at times rather than evaluating it.

Results: Likelihood estimation by surrogate model

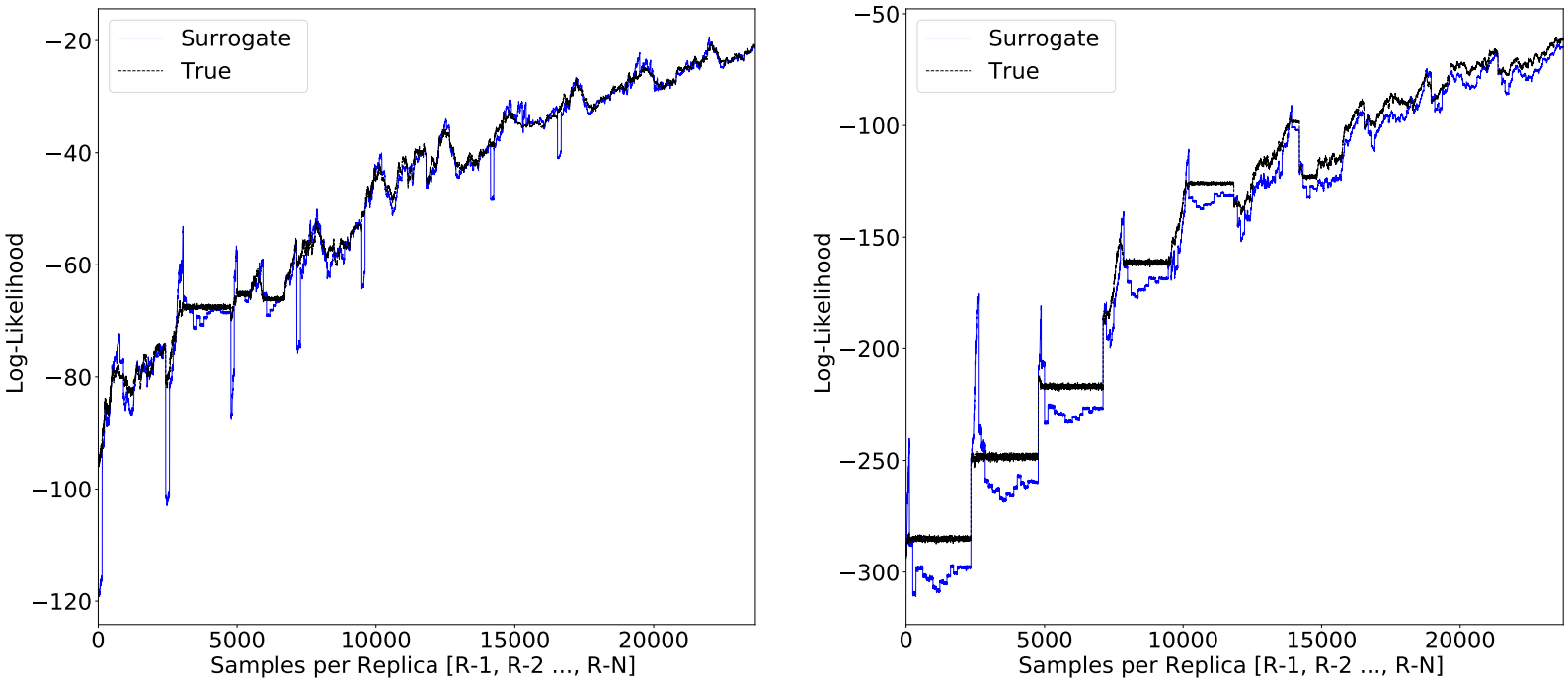


Figure: The Iris (top) and Cancer (bottom) surrogate accuracy. The dashed line denotes the real likelihood function while the blue line gives the surrogate likelihood estimation.

Results: classification problems

Results from [10]

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

[10] Chandra, R., Jain, K., Kapoor, A., & Aman, A. (2020). Surrogate-assisted parallel tempering for Bayesian neural learning. Engineering Applications of Artificial Intelligence, 94, 103700.

Conclusions

- The results have shown that surrogate-assisted parallel tempering can be beneficial for larger datasets and models, demonstrated by Bayesian neural network architecture for Pen-Digit and Chess classification problems.
- This implies that the method would be very useful for large scale models where computational time can be lowered while maintaining performance in decision making such as classification accuracy.
- We observed that in general, the Langevin-gradients improves the accuracy of the results.
- In future work, we envision strategies to further improve the surrogate estimation for large models and parameters. A way ahead is to utilize local surrogate via time series prediction could help in alleviating the challenges.
- Future work can be in area of geoscientifc models [11] and deep learning models

[11] Chandra, R., Azam, D., Kapoor, A., & Mller, R. D. (2020). Surrogate-assisted Bayesian inversion for landscape and basin evolution models. Geoscientific Model Development, 13(7), 2959-2979.

Bayesian deep learning with MCMC: Autoencoders

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.
The challenge is in determining the optimal $\theta = (\phi, \psi)$ and typically gradient-based learning algorithms are used to minimize R_{loss}.

Autoencoders

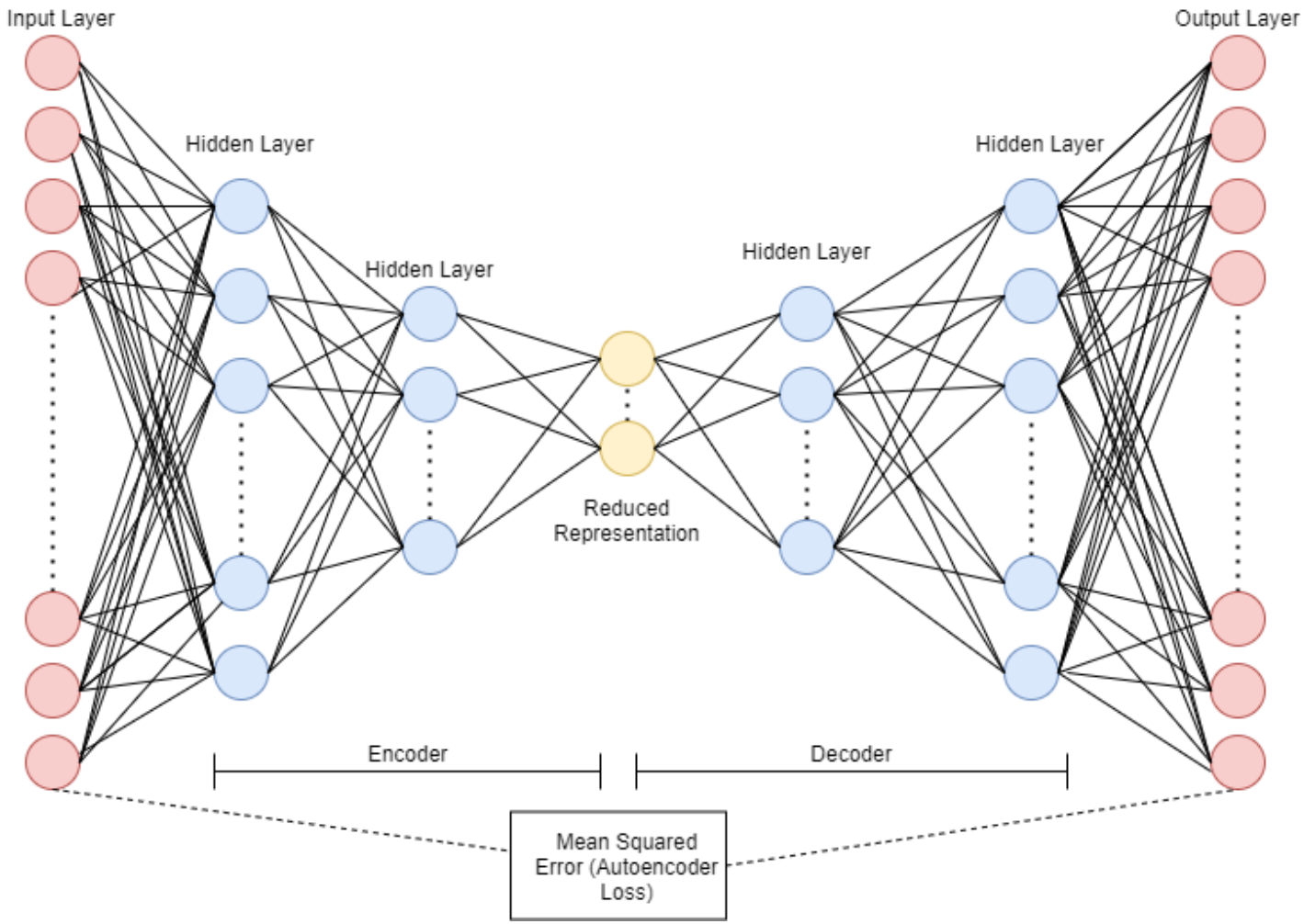


Figure: The autoencoder features an encoder and a decoder, which are highlighted.

Autoencoders

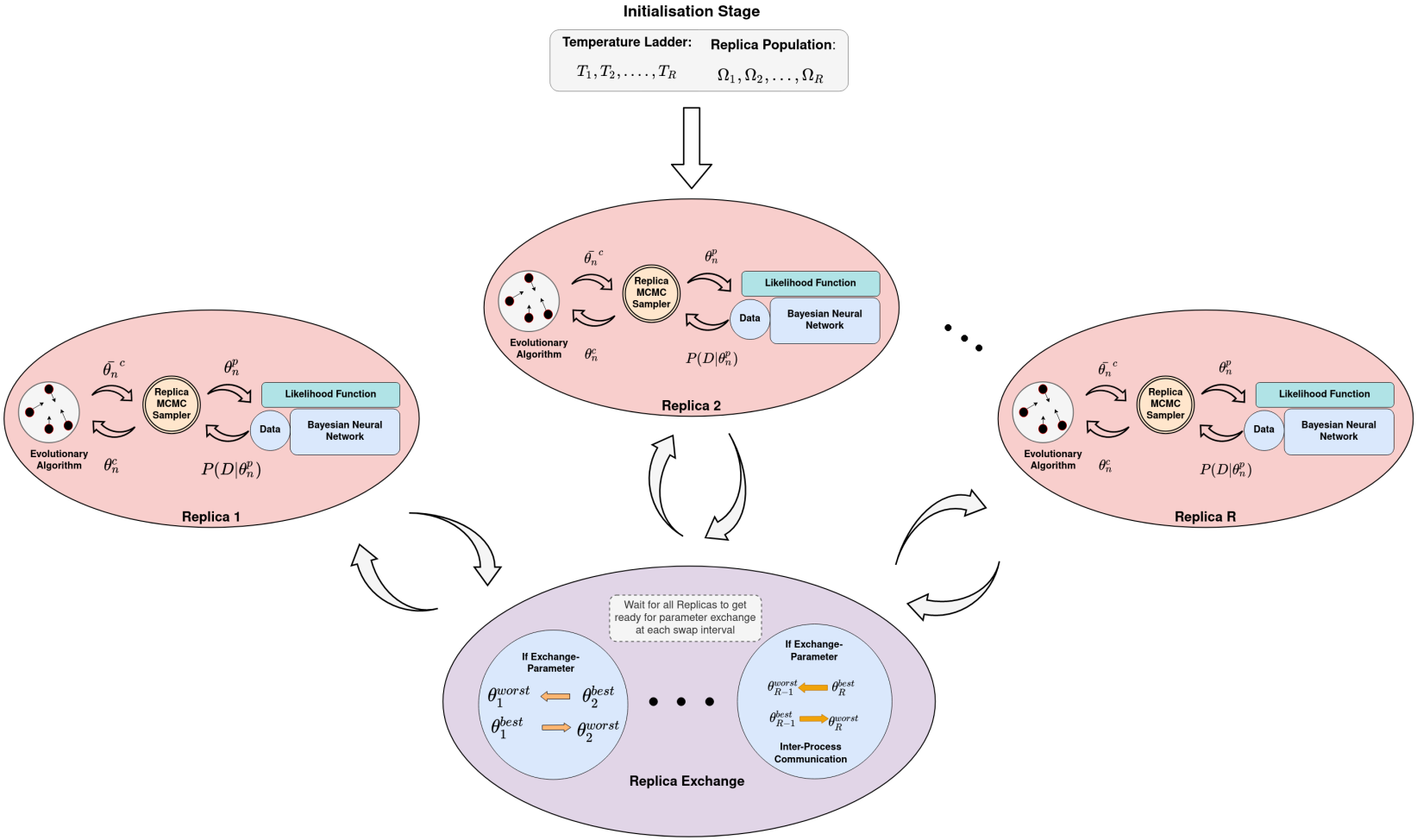


Figure: Bayesian autoencoder framework highlighting tempered MCMC utilising parallel computing and autoencoder neural network.

Algorithm

Stage 0: initialisation:
* Define autoencoder topology, i.e number of input, hidden layer, and neurons for encoder and decoder.
* Define geometric temperature ladder using maximum temperature ($T_{\max}$), replica swap interval (R_{swap}), and maximum number of samples for each replica ($R_{\max}$).
* Set the number of replicas (M) in ensemble as *alive*; $alive = M$
* Set the number-samples (R_{switch}) for first-phase (parallel tempering MCMC) and second-phase (canonical MCMC)
* Set g_{prob} which determines how often to apply Langevin-gradients.

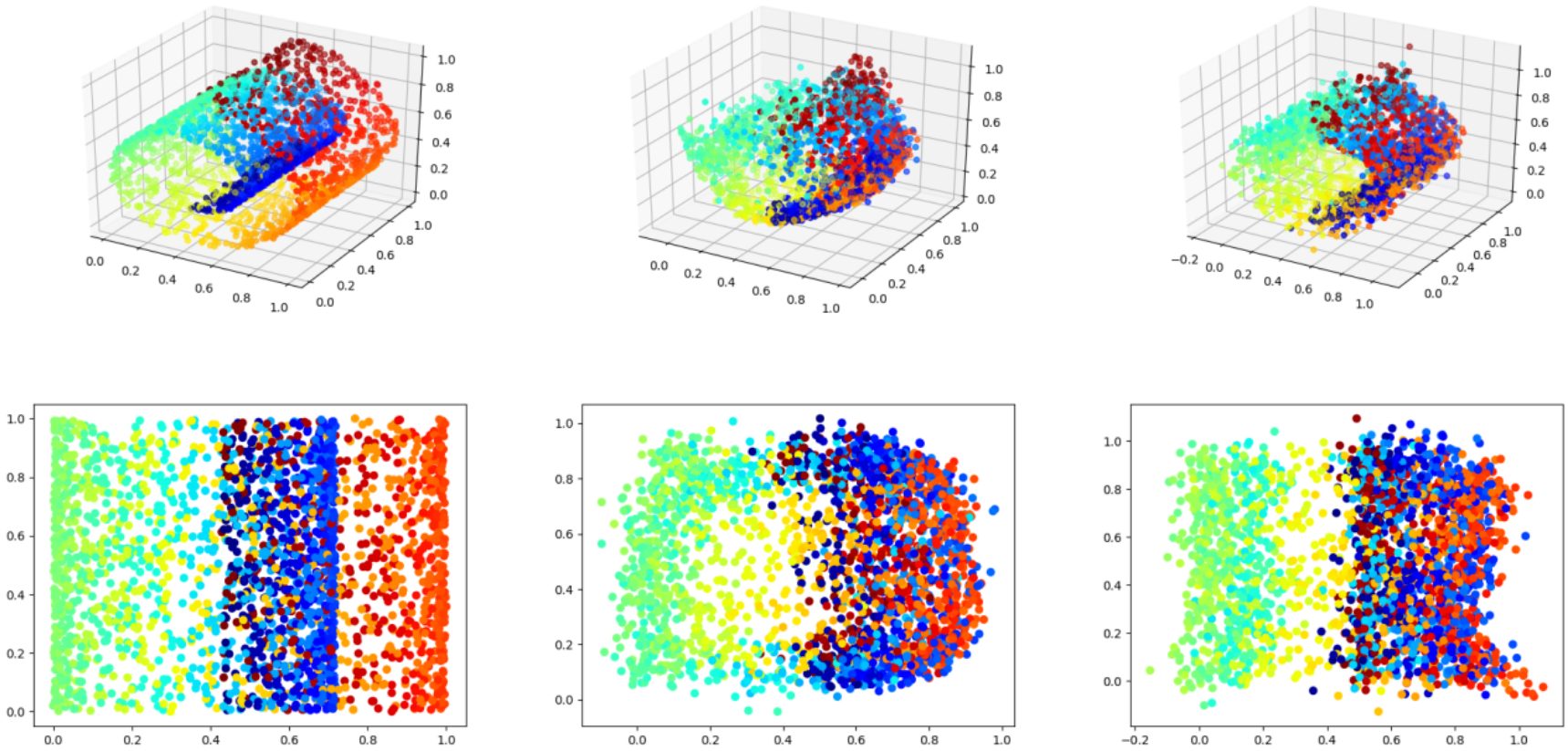
while $alive \neq 0$ **do**
  Stage 1.0: Execute each replica via manager process
  for $i = 1$ **to** M **do**
    first-phase: T_i = geometric()
    for $s = 1$ **to** R_{max} **do**
      Stage 1.1: Within Replica Transition
      for $k = 1$ **to** R_{swap} **do**
        1.2 **if** Unif$(0, 1) \leq g_{prob}$ **then**
          Create Langevin-gradient proposal using (3)
        else
          Create Random-walk proposal
        end
        1.3 Evaluate the respective likelihoods compute probability α (4)
        1.4 **if** Unif$(0, 1) \leq \alpha$ **then**
          Accept replica position, $\Theta_s \leftarrow \Theta^*_s$
        else
          Reject to retain previous state: $\Theta_s \leftarrow \Theta^*_{s-1}$
        end
        1.5 **if** *second-phase is true* **then**
          Update temperature, $T_i = 1$
        end
      end
      Stage 2.0: Between Replica Transition:
      2.1 Compute exchange replica acceptance probability β (5)
      2.2 **if** Unif$(0, 1) \leq \beta$ **then**
        Signal() manager process Swap selected neighboring Replica, $\Theta_i \leftrightarrow \Theta_{s+1}$
      end
    end
  end
  Stage 3.0: Signal() manager process
  3.1 Decrement number of replica processes *alive*
end
Stage 4: Combine posterior using second-phase MCMC samples

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)

Results

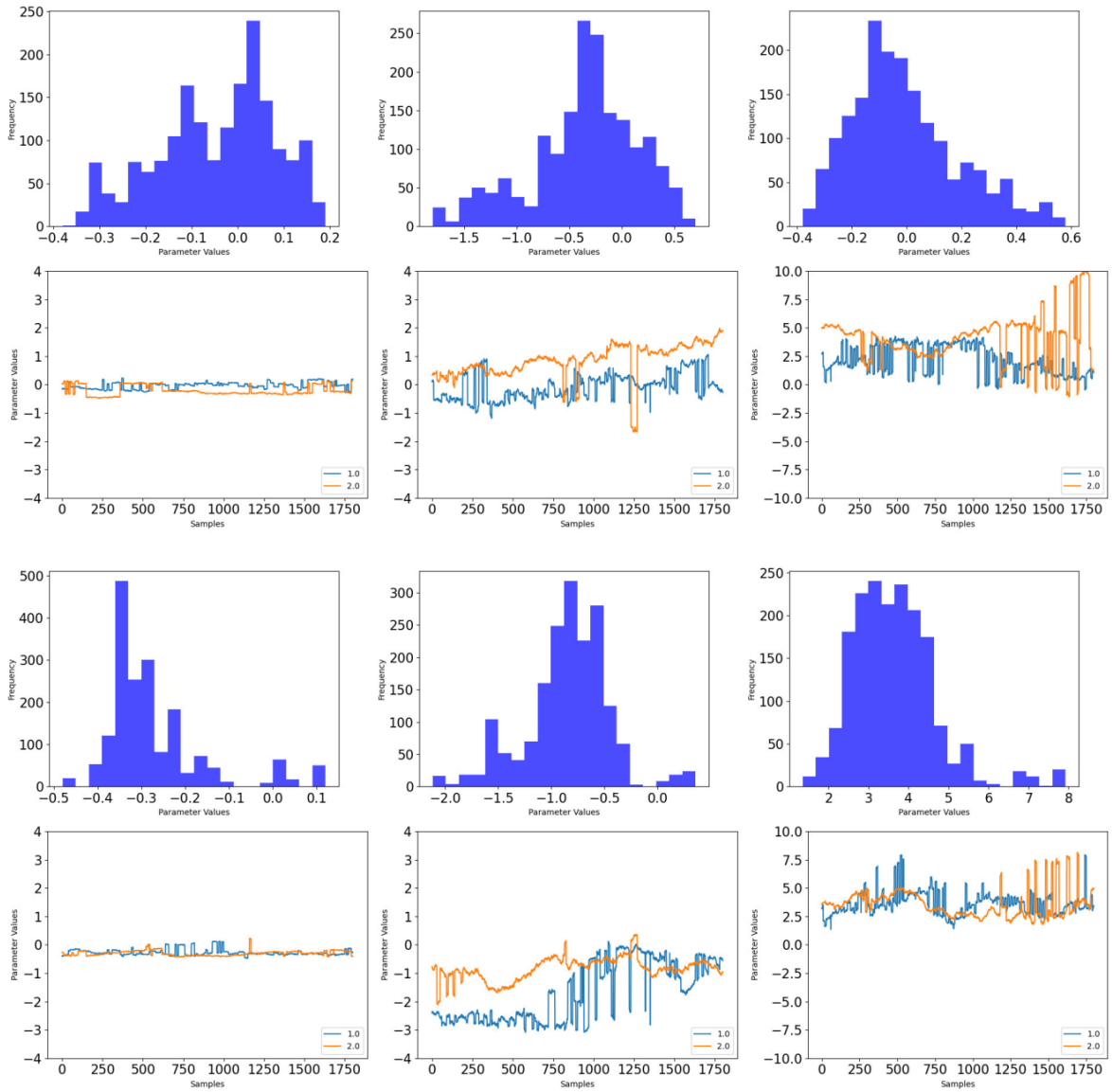
Results from [12]

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)

[12] 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.

Bayesian deep learning with MCMC: Graph CNNs

Introduction

- More recently, there has been more attention to unstructured data that can be represented via graphs. These types of data are often found in health and medicine, social networks, and research data repositories.
- Graph convolutional neural networks have recently gained attention in the field of deep learning that takes advantage of graph-based data representation with automatic feature extraction via convolutions.
- Recent advances in parallel computing and advanced proposal schemes in sampling, such as incorporating gradients has allowed Bayesian deep learning methods to be implemented.
- Next, we present Bayesian graph convolutional neural networks (Bayes-GCNN) that employ state-of-art methods such as tempered MCMC sampling and advanced proposal schemes.

Graph Data Example

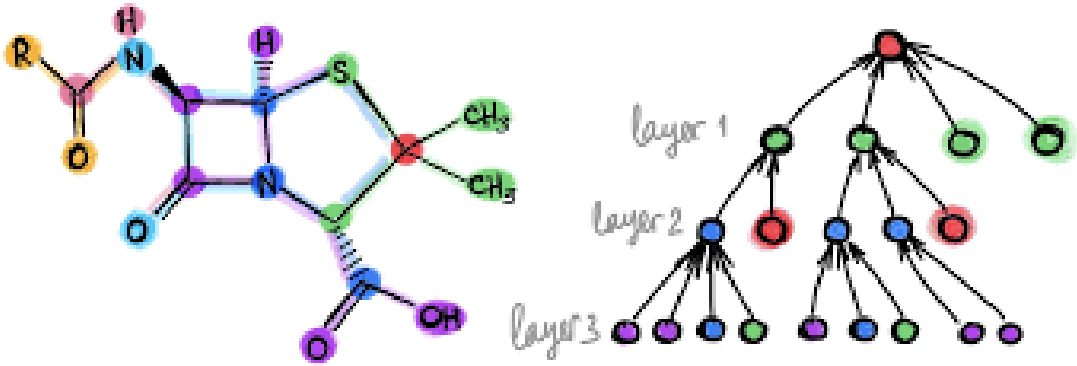


Figure: Graph Data: Molecule intreactions [13]

[13] Source: Google images

Graph neural networks

A graph G data structure consists of a set of vertices (nodes) V and edges E, which can be either directed or undirected. Each node represents a data element, and the edges denote the relationships between the data elements. Each node has its own graph embedding via a feature vector, which summarises the properties of that particular data element. The nodes send their graph embedding to their immediate neighbours in the form of messages.
The message received by node v at time t, m_v^t is constructed by aggregating over the set of neighbours v, $N(v)$, the results of a message function M_t which takes three arguments: the feature vector of v itself (h_v^t), the feature vector of neighbour w (h_w^t), and the features of the edge between v and w (e_{vw}):

$$m_v^t = \sum_{w \in N(v)} M_t(h_v^t, h_w^t, e_{vw}). \tag{13}$$

Based on this message m_v^t and the previous value h_v^t, the latter is updated via a function U_t:

$$h_v^{t+1} = U_t\left(h_v^t, m_v^t\right). \tag{14}$$

Graph Neural Network

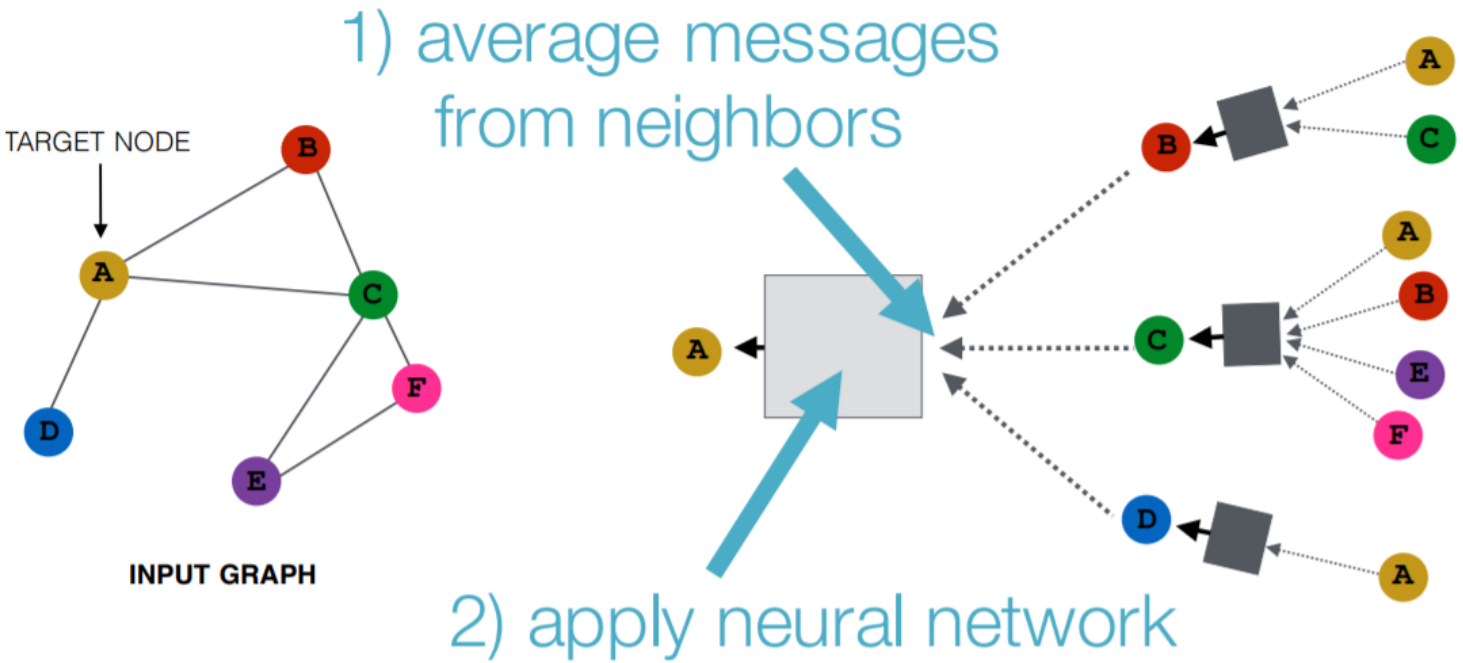
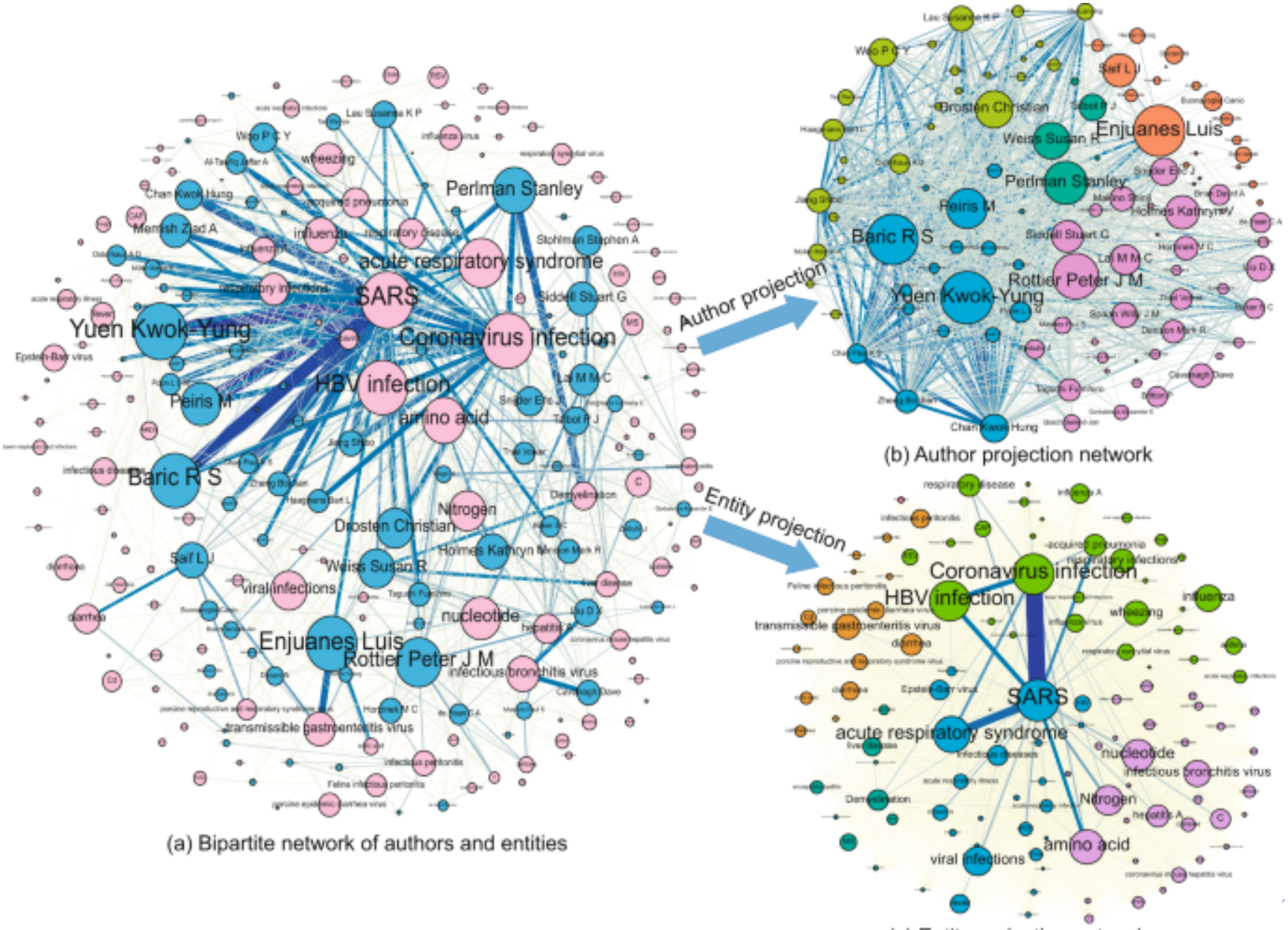


Figure: Graph Neural Network [14]

[14] Source: Google images

PubMed citation graph dataset



(a) Bipartite network of authors and entities

(b) Author projection network

Graph problems

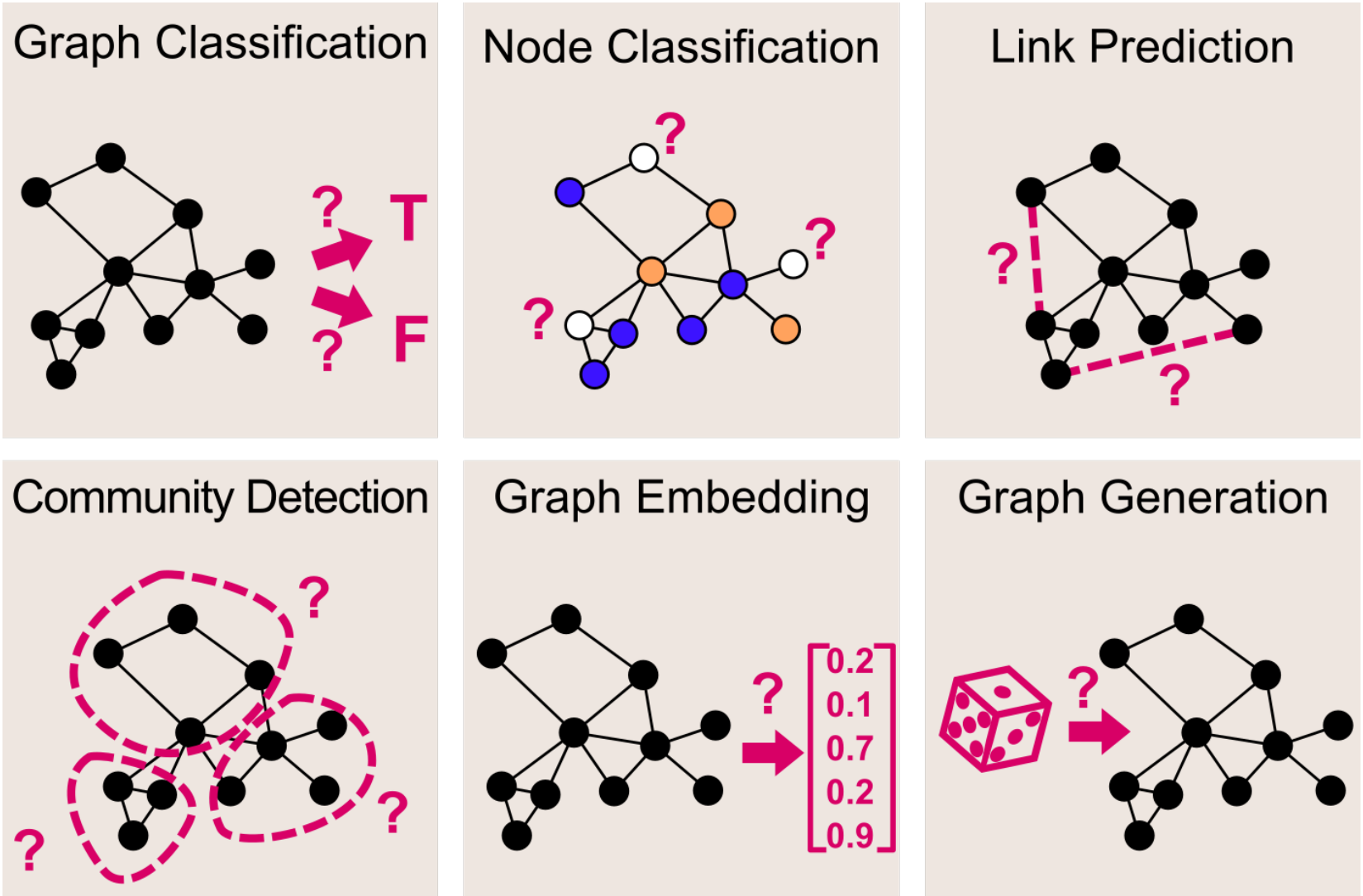


Figure: Graph problems [16]

[16] Source: Google images

Graph CNN

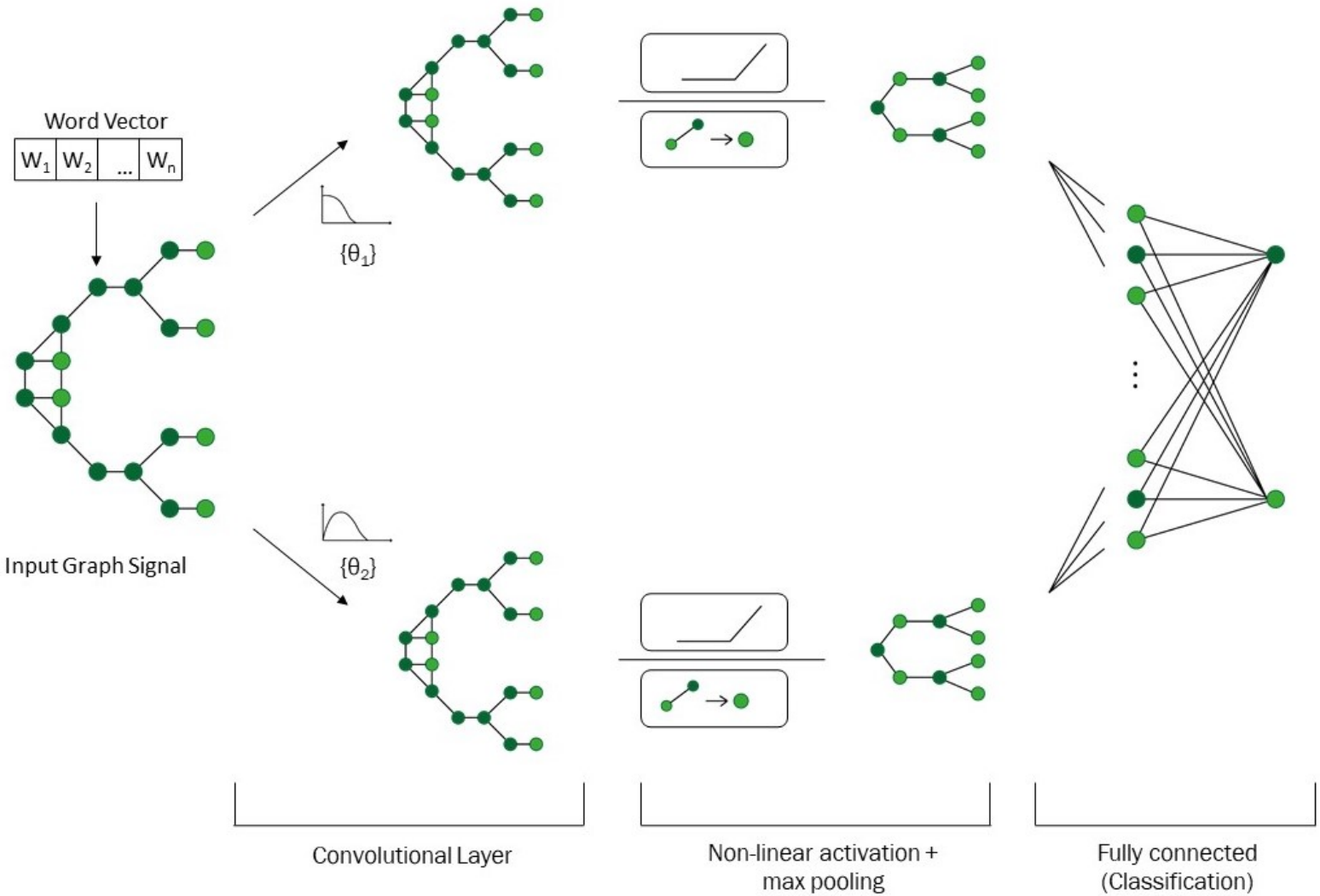


Figure: Graph CNN

Graph CNN

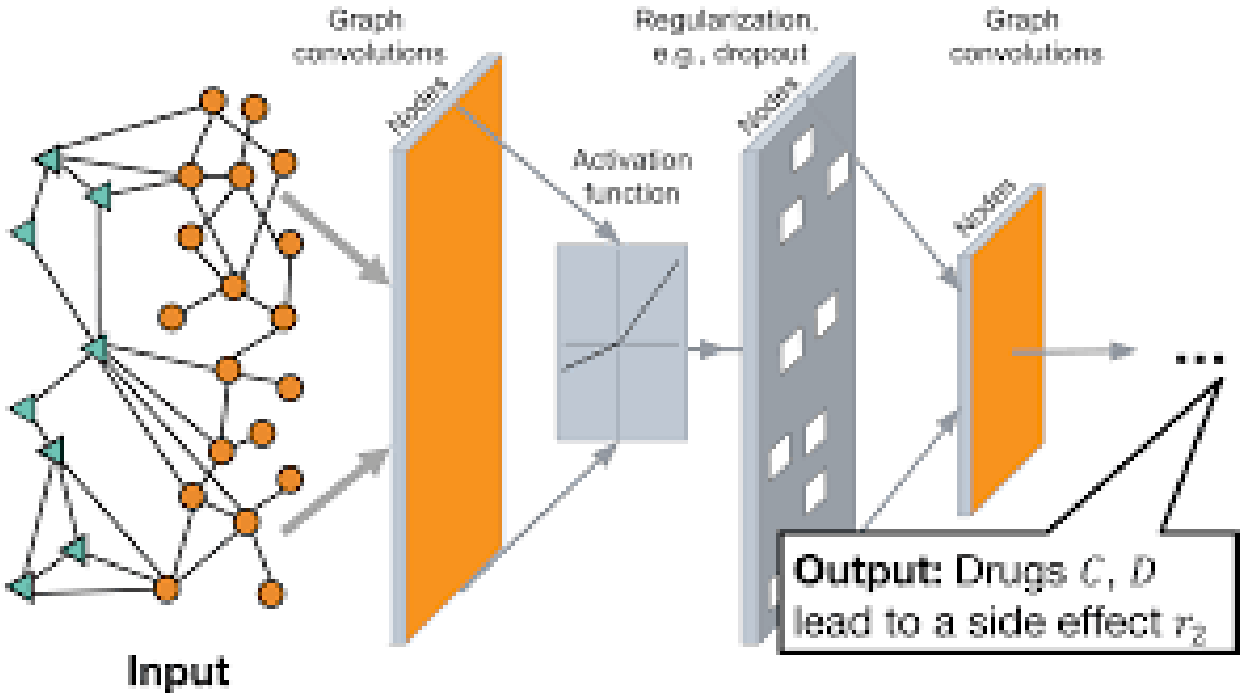


Figure: Graph CNN [17]

[17] Source: Google images

Bayesian Graph-CNN Framework

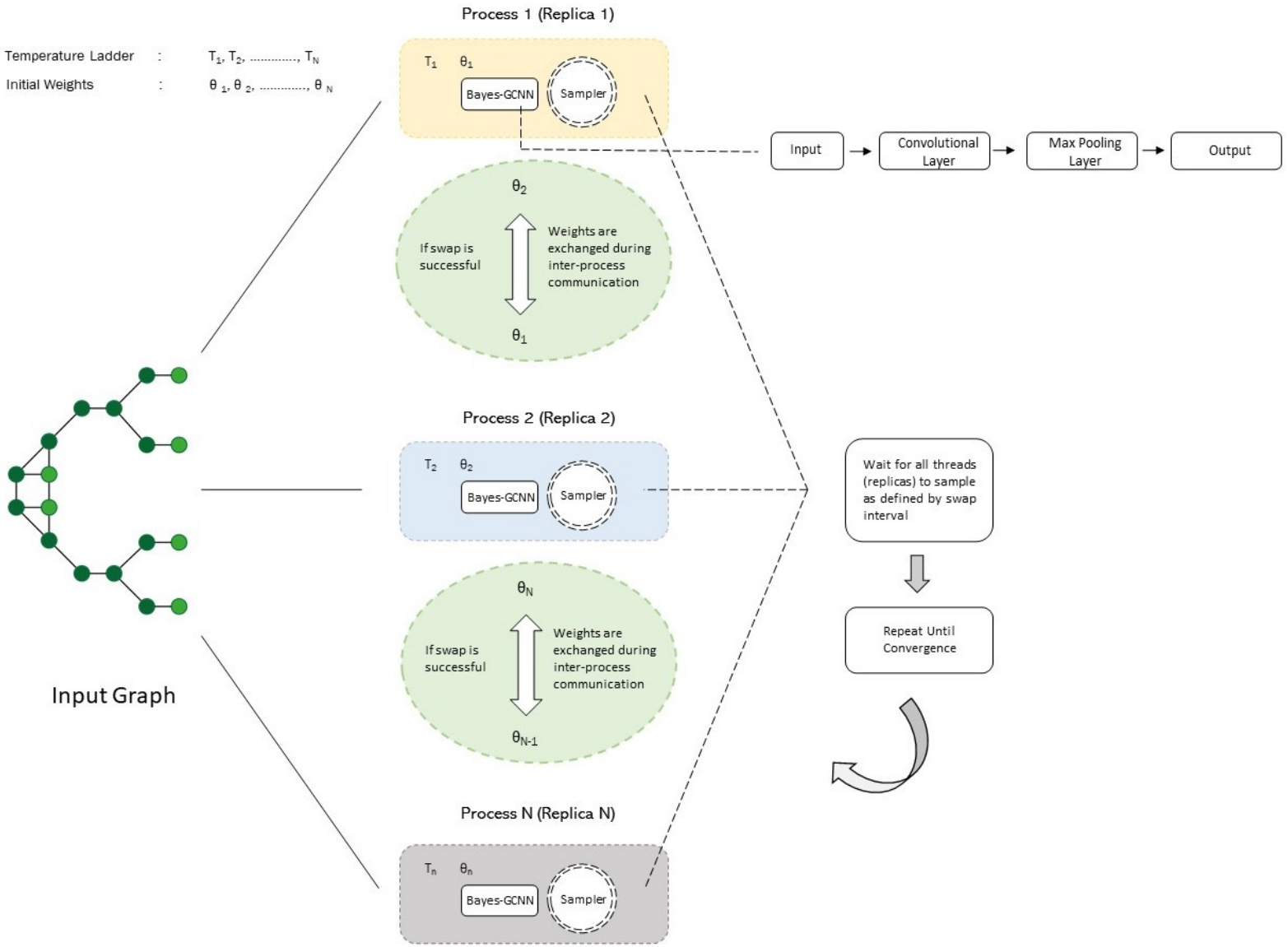


Figure: Bayesian CNN framework highlighting tempered MCMC utilising parallel computing and CNN.

Cora scientific publication citation dataset

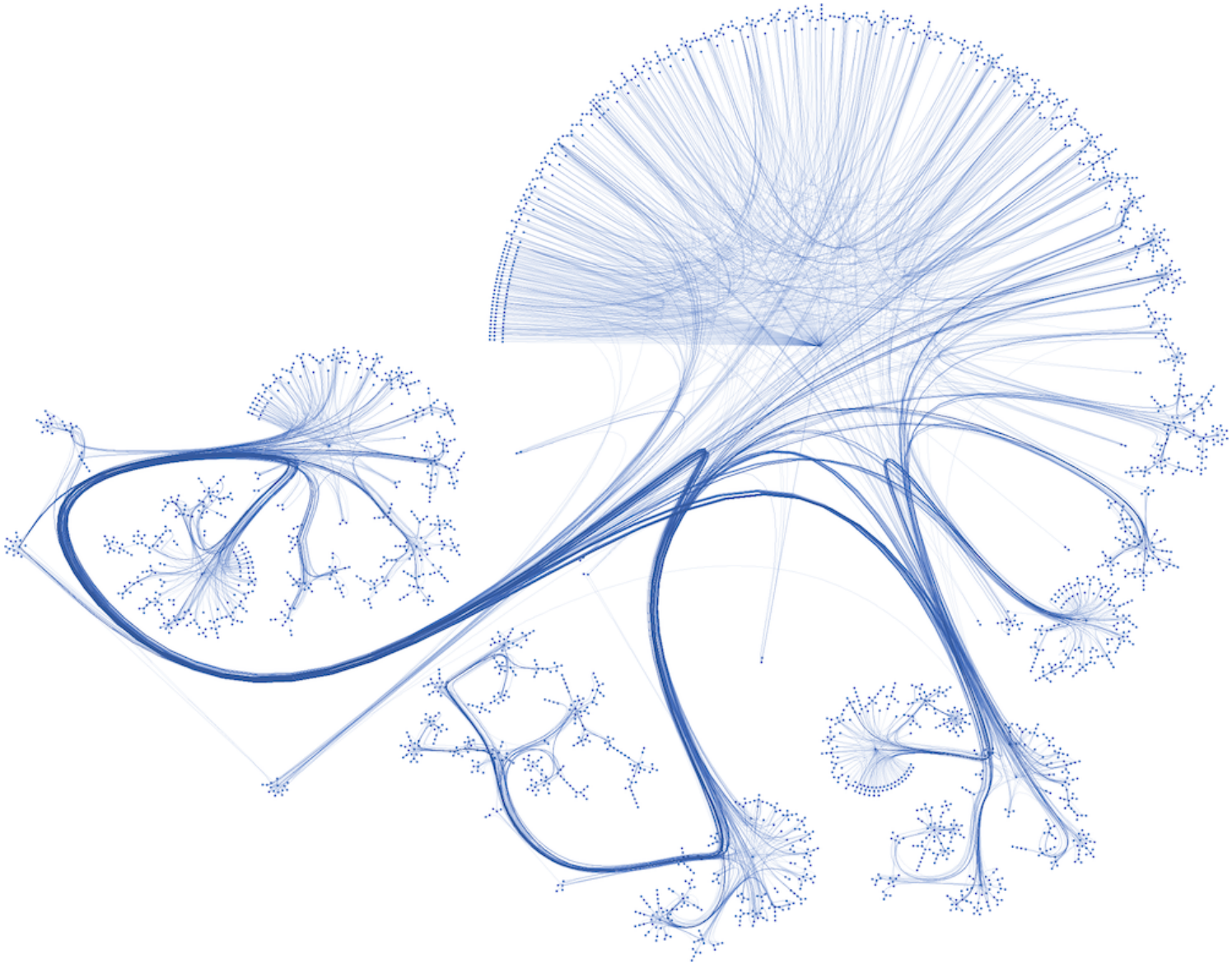


Figure: Cora dataset [18]

Algorithm

Data: Graph-based Data
Result: Bayes-GCNN posterior distribution

Stage 0: initialisation:
* Define Baye-GCNN architecture (number of input, convolution and pooling layers, number of outputs, and respective activation functions). * Provide user-defined parameters: maximum temperature ($T_{\max}$), swap-interval (R_{swap}), and maximum replica samples ($R_{\max}$).
* Termination condition: Set $active = M$
* Define (R_{switch}) for parallel tempering MCMC to canonical MCMC
* Define l_{rate} for applying Langevin-gradients.

while $(alive \neq 0$ **do**
  Stage 1.0: Execute each replica via manager process
  for $i = 1$ **to** M **do**
    $s = 0$
    phase-one: T_i = geometric()
    for $s = 1$ **to** R_{max} **do**
      Stage 1.1: Local Replica Sampling
      for $k = 1$ **to** R_{swap} **do**
        1.2 **if** Unif$(0, 1) \leq g_{prob}$ **then**
          Langevin-gradient proposal
        else
          Random-walk proposal
        end
        1.3 Get log-likelihood and computer Metropolis-Hastings probability α
        1.4 **if** Unif$(0, 1) \leq \alpha$ **then**
          Accept proposal, $\boldsymbol{\theta}_s \leftarrow \boldsymbol{\theta}^*_s$
        else
          Reject proposed sample, retain current sample: $\boldsymbol{\theta}_s \leftarrow \boldsymbol{\theta}^*_{s-1}$
        end
        1.5 **if** *phase-two is true* **then**
          Change replica temperature, $T_i = 1$
        end
      end
      Stage 2.0: Neighbouring replica exchange:
      2.1 Get neighbouring replica Metropolis-Hastings acceptance probability β
      2.2 **if** Unif$(0, 1) \leq \beta$ **then**
        Give signal() to the manager process Exchange replicas, $\boldsymbol{\theta}_i \leftrightarrow \boldsymbol{\theta}_{s+1}$
      end
    end
  end
  Stage 3.0: Signal() manager process
  3.1 Decrement number of replica processes *alive*
end
Stage 4: Combine posterior using second-phase MCMC samples

Experiment configuration

Table: Overview of datasets using graph representation showing number (num.) of classes with training and test instances.

Dataset	Nodes	Edges	Num. classes	Num. training	Num. testing
Cora	2708	5429	7	140	1000
CiteSeer	3327	4732	6	120	1000
PubMed	19717	44338	3	60	1000

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

Dataset	Input Neurons	Output Neurons	Hidden Layers	Total parameters
Cora	1433	7	16	23063
CiteSeer	3703	6	16	59366
PubMed	500	3	16	8067

Results

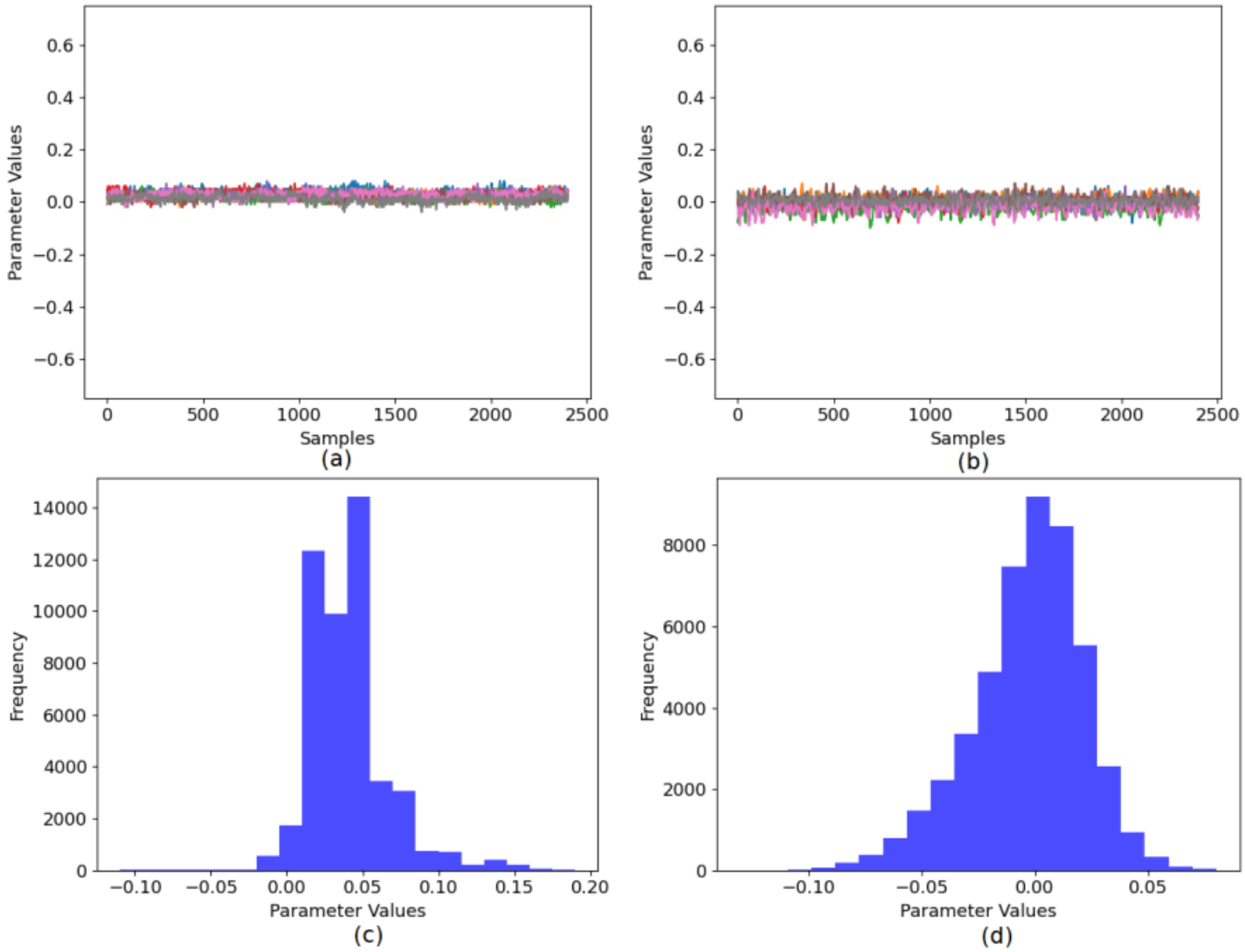
Results from [19]

Table: Bayesian GCNN results comparison of established methods from literature. (Unavailable values are left blank.)

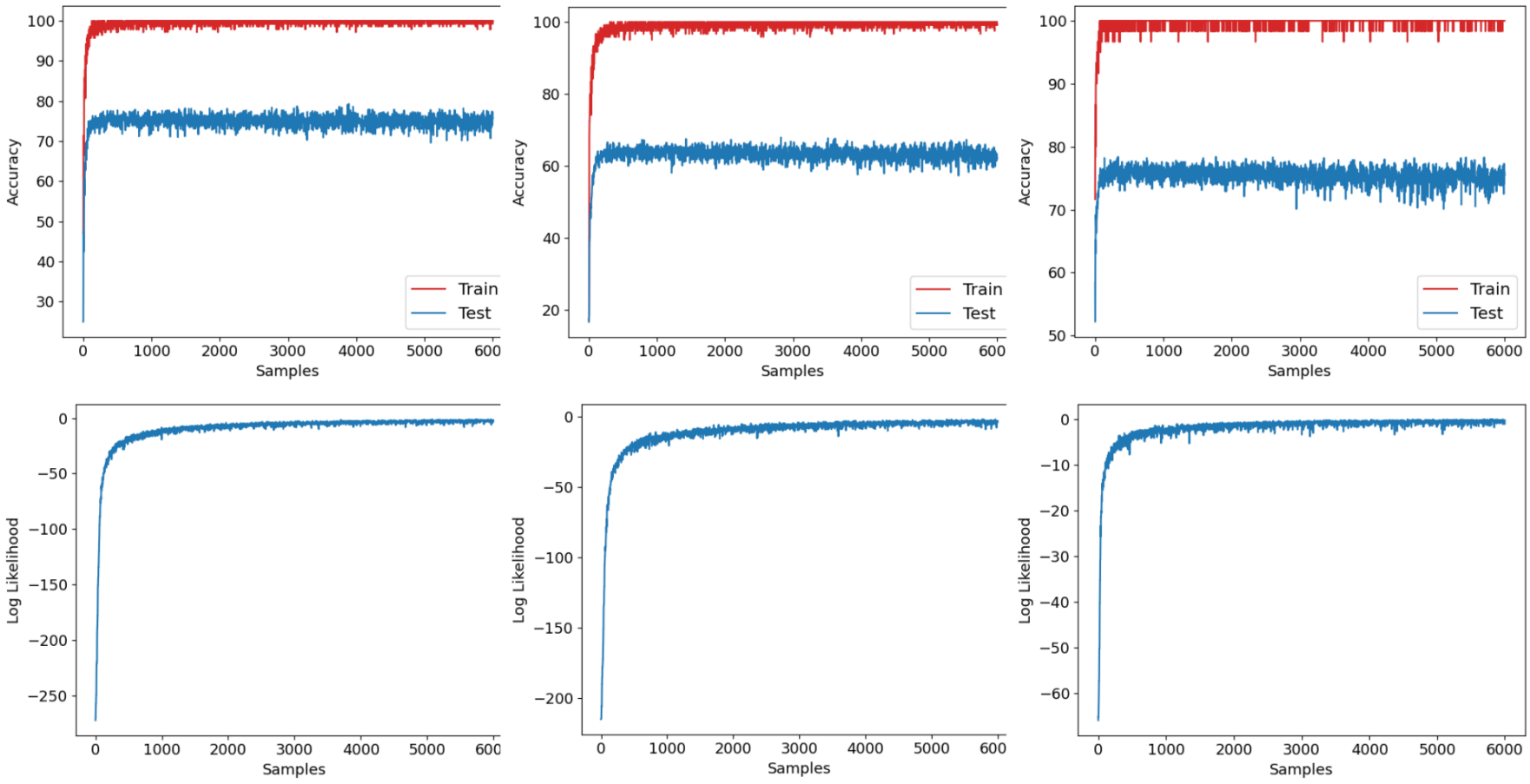
Method	Cora best (mean, std)	CiteSeer best (mean, std)	PubMed best (mean, std)
GCNN (SGD) (Yang et. al, 2016)	75.70 (,)	64.70 (,)	77.20 (,)
GCNN (Adam) (Kipf et. al, 2017)	(81.50,)	(70.30,)	(79.00,)
GCNN* (SGD)	23.70 (16.78, 4.18)	25.60 (20.36, 2.74)	45.60 (38.13, 8.55)
Bayes-GCNN (SGD)	32.40 (15.42, 5.02)	25.20 (16.49, 2.78)	50.90 (37.75, 5.58)
GCNN* (Adam)	81.70 (80.81, 0.65)	72.00 (70.51, 0.85)	79.50 (79.00, 0.60)
Bayes-GCNN (Adam)	79.00 (74.71, 2.42)	68.90 (63.26, 2.74)	78.70 (74.94, 1.63)

[19] R Chandra, A Bhagat, M Maharana, PN Krivitsky (2021). Bayesian graph convolutional neural networks via tempered MCMC. arXiv preprint arXiv:2104.08438

Posterior


Parameter Values
Samples
(a)
Parameter Values
Samples
(b)
Frequency
Parameter Values
(c)
Frequency
Parameter Values
(d)

Convergence



(a) Cora

(b) CiteSeer

(c) PubMed

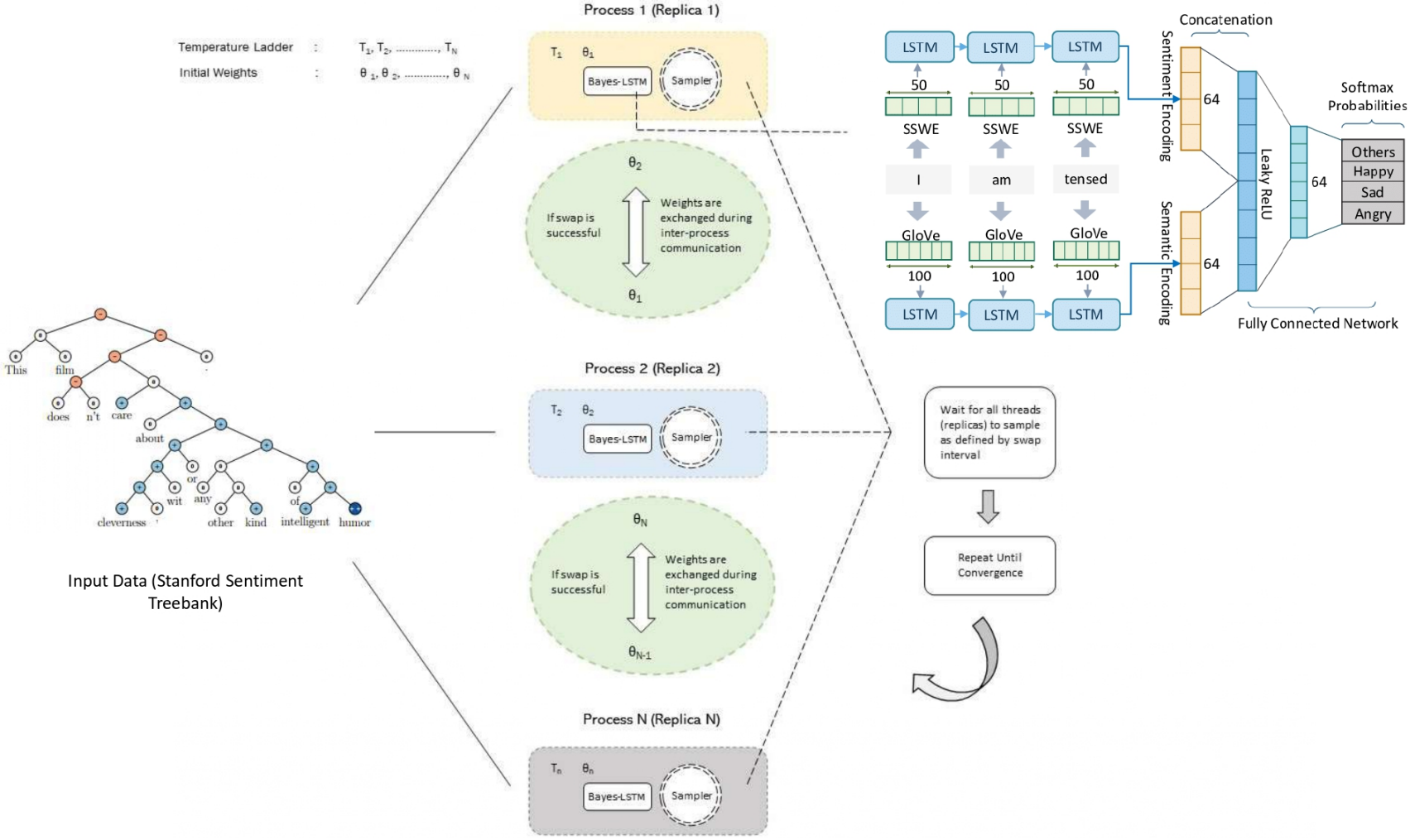
Conclusions

- Our results indicate that the method provides an alternative form of training that features a principled way to quantify uncertainty in model parameters.
- Bayes-GCNN would eliminate the need to run repetitive experiments with probabilistic representation of weights and biases.
- While the mean accuracy of the Bayes-GCNN was around 4-5% lower than canonical GCNN, the maximum accuracy is on par with the accuracy of canonical GCNN.
- In addition, canonical methods do not offer any uncertainty quantification which motivates implementation of Bayes-GCNN framework for real-world applications.

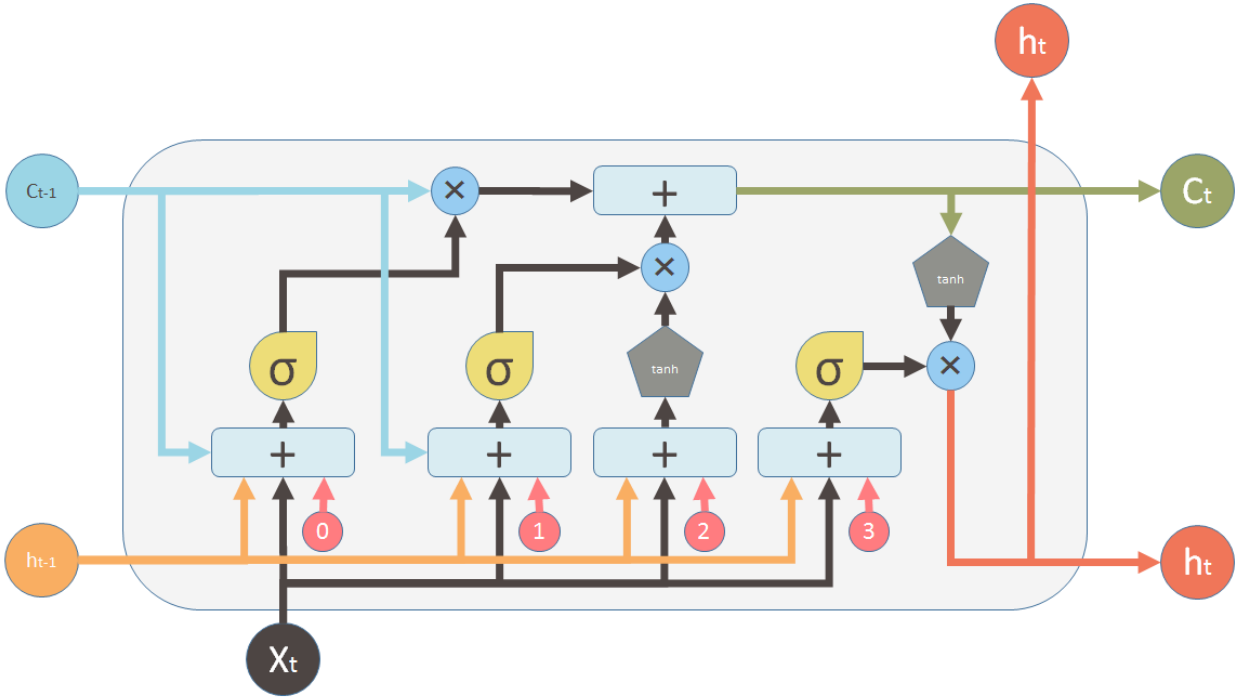
Future Research Directions

- Bayesian deep learning for language models: Bayesian Transformers via MCMC (with Scott Sisson and Ayush Bhagat)
- Bayesian deep learning for few-shot learning via Convolutional Neural Networks: incomplete data (with Manavendra Maharana)
- Bayesian LSTM models for multi-step time series prediction (with Scott Sisson and Eshwar Nukala)
- Pingala: Bayesian deep learning framework in Python (planning stage)

Bayes LSTM and Transformers for Language Modelling



LSTM model



Inputs:

- X_t Input vector
- C_{t-1} Memory from previous block
- h_{t-1} Output of previous block

outputs:

- C_t Memory from current block
- h_t Output of current block

Nonlinearities:

- σ Sigmoid
- tanh Hyperbolic tangent

Vector operations:

- $\times$ Element-wise multiplication
- $+$ Element-wise Summation / Concatenation

Bias: 0

Thanks, Namaste and Dhanyavaad

- Reseach papers, software and data: webpage R Chandra, Google Scholar, Github Repo: [20] [21]
- Note all recent papers have software and code via Github repository (with links given in respective papers): `https://github.com/sydney-machine-learning`
- 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.

[20] `https://scholar.google.com.au/citations?user=pVPvRLoAAAAJ`
[21] `https://research.unsw.edu.au/people/dr-rohitash-chandra`