Revisiting Bayesian Autoencoders with MCMC

Rohitash Chandra 1,∗∗, Mahir Jain 2,∗∗, Manavendra Maharana 2, Pavel N. Krivitsky 1

Abstract

Autoencoders gained popularity in the deep learning revolution given their 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. This has been addressed with variational autoencoders so far. Bayesian inference via Markov Chain Monte Carlo (MCMC) sampling methods have faced limitations; however, recent advances in parallel computing and advanced proposal schemes that incorporate gradients have opened routes less travelled. In this paper, we present Bayesian autoencoders powered by MCMC sampling implemented using parallel computing and Langevin-gradient proposal distribution. The proposed Bayesian autoencoder provides similar performance accuracy when compared to related methods from the literature, with the additional feature of robust uncertainty quantification while obtaining a reduced or compressed data representation. This motivates further application of the Bayesian autoencoder framework for other deep learning models.

Keywords: Bayesian deep learning, MCMC, Langevin Dynamics, autoencoders, parallel tempering, deep learning

1. Introduction

Autoencoders are a family of unsupervised learning methods that use neural network architectures and learning algorithms to learn a lower-dimensional representation (encoding) of the data, which can then be used to reconstruct a representation close to the original input. They thus facilitate dimensionality reduction for prediction and classification [1,2], and have been successfully applied to image classification [3,4], face recognition [5,6], geoscience and remote sensing [7], speech-based emotion recognition [8], and data generation [9]. Autoencoders have been prominent in deep neural network architectures and also for transfer learning tasks [10,11]. Recent developments in this area include regularized autoencoders [12], variational autoencoders [13,14,15,16], adversarial autoencoders [17,18,19,20], variational graph autoencoders [21,22], and convolutional autoencoders [10].

Bayesian neural learning is an alternative learning method in which parameters (weights and biases) are (jointly) represented by a probability distribution rather than a point estimate that is usually returned by maximum likelihood estimation [23]. The posterior probability distribution naturally accounts for uncertainty in parameter estimates, which is further propagated into the decision making process [24]. Bayes’ theorem is used as foundation for inference in Bayesian neural networks. Markov chain Monte Carlo (MCMC) sampling methods [25] are used for estimating the posterior distribution. Variational inference [26] provides another way to approximate the posterior distribution, which approximates an intractable posterior distribution by a tractable one. This makes it particularly suited to large data sets and models, and so it has been popular for autoencoders and neural networks [13,27]. Variational autoencoders [13] employ variational inference [26] as a form of regularisation of the parameters or weights during the training, in order to ensure that it has good properties for prediction [14,15,16]. Burda et. al [28] described an importance weighted autoencoder that extended the variational autoencoder to use multiple samples to approximate more complex posteriors.

Deep learning model can feature tens of thousands to millions of parameters that are typically optimised by gradient-based algorithms. The major challenge of MCMC methods for deep learning is addressing the computational time required for constructing a good approximation of the posterior distribution, incurring heavy computational costs. However, progress in incorporating gradient-based proposals into MCMC sampling has been inspired by earlier works [29,30] for the development of Hamiltonian and Langevin-based MCMC methods [24,31]. Gradient-based proposal distributions have been demonstrated to be effective for Bayesian neural learning [32,33]. Combining them with parallel computing resulted in further improvements [32], perhaps allowing them to be used for deep learning methods such as autoencoders, and in particular deep autoencoders that feature thousands of weights or parameters.

1 School of Mathematics and Statistics, UNSW Sydney, Kensington, NSW 2052, Australia
2 Manipal Institute of Technology, Manipal, India

∗∗Corresponding author
∗∗Authors contributed equally

Email addresses: rohitash.chandra@unsw.edu.au (Rohitash Chandra ), jainmahir35@gmail.com (Mahir Jain ), manav.maharana7@gmail.com (Manavendra Maharana ), p.krivitsky@unsw.edu.au (Pavel N. Krivitsky )

URL: https://research.unsw.edu.au/people/dr-rohitash-chandra (Rohitash Chandra ), https://www.krivitsky.net/research (Pavel N. Krivitsky )

Preprint submitted to Elsevier
In this paper, we present Bayesian autoencoders powered by tempered MCMC sampling that incorporate parallel computing and Langevin-gradient proposal distribution. We demonstrate the effectiveness of Bayesian autoencoders with benchmark datasets that involve small-scale to large-scale models featuring around a million parameters. We also provide an investigation into the effect of two prominent gradient-based methods and develop an adaptive Langevin-gradient proposal distribution. We finally compare our results with the literature where gradient-based methods have been used for the same datasets.

The rest of the paper is organised as follows. In Section 2 we present a background and literature review of related methods. Section 3 presents the proposed methodology, followed by experiments and results in Section 4. Section 5 provides a discussion and Section 6 concludes the paper with directions of future work.

2. Related work

2.1. Autoencoders

Autoencoders compress data by providing a reduced feature set that may be used for supervised learning. Autoencoders have been used with supervised learning methods such as simple neural networks [34], recurrent neural networks [35], long short term memory (LSTM) networks [36], and graph neural networks [37]. Depending on the dataset, the features from other deep learning methods are used to augment the canonical autoencoder. For instance, max-pooling and convolutional layers can be used to augment an autoencoder for unsupervised learning [38]. The applications of autoencoders in machine learning problems include data classification, dimensionality reduction, pattern recognition, image denoising, anomaly detection, and recommender systems [39] [40] [41] [42] [43] [44]. Autoencoders have been successfully applied to a number of real-world applications such as cyber-security [45], medical imaging [46], and bio-metrics [47]. Furthermore, a review of autoencoders has been done for geoscience and remote sensing with focus to hyper-spectral images [48].

Next, we review some of the key autoencoder architectures from the literature [49]. Sparsity autoencoders [48] are used to capture the latent, rather than redundant, representations of the data using a loss function that penalises activation of the hidden layer; hence, enforcing the sparsity constraint. Denoising autoencoders [49] involve adding artificial noise to the data, after which the autoencoder attempts to reconstruct the original data from noisy data. This prevents the autoencoder from capturing an overly-complete representation of the data which is common to many conventional autoencoders. Contractive autoencoders [50] makes the reduced representation less sensitive to small variations in the input samples; this is achieved by adding a regularizer to the loss function that the algorithm is trying to minimize. This then reduces the encoded representation’s sensitivity to the input. Convolutional autoencoders [51] [52] create a reduced representation of image data which is helpful for supervised machine learning.

2.2. Bayesian deep learning

Among the difficulties of conventional autoencoders are the lack of uncertainty quantification in the estimation of model parameters and the need to tune regularization hyper-parameters. Bayesian neural networks address these by estimating the parameters of the model (weights and biases) as random variables, in contrast to single point estimates by backpropagation learning that employ gradient-based methods [53] [54]. The progress in neural networks has led to revolution in deep learning models; however, progress in Bayesian neural networks and Bayesian deep learning has been relatively slow due to certain challenges [53] [54]. Bayesian inference implemented with canonical MCMC methods handles the large numbers of parameters involved with difficulty. However, variational inference provides an alternative Bayesian inference approach, with variational autoencoders [55], variational autoencoders and generative adversarial networks (GANs) [56], variational convolutional neural networks (CNNs) [57] [58] [59] [60], variational recurrent neural networks via long short-term memory (LSTM) networks [61], and variational graph neural networks [62] [63]. Further details about their applications are given in [53]. We note that variational autoencoders also use a probabilistic representation of model parameters, as opposed to a single value representation in conventional autoencoders, facilitating uncertainty quantification [64] [65] [66].

The limitations of MCMC sampling has been addressed with better computational resources and advanced proposal distributions, incorporating gradients [67] [68]. Hamiltonian MCMC sampling methods have been used for Bayesian neural networks with enhanced computation strategies [69]. Langevin-based MCMC methods have been used in implementation of Bayesian neural networks for pattern classification and time series prediction problems [70]. A major challenge in MCMC sampling has been in addressing big data and computationally expensive models; hence, surrogate-assisted estimation, in which a low cost surrogate model provides approximation of the likelihood, has been used to address this issue [71]. Langevin MCMC methods have been used in Bayesian neural networks for transfer learning given multiple sources of data [72]. The challenge has been in developing an efficient MCMC sampling method for large number of model parameters. In previous research, simple Bayesian neural networks have been used that at most had several thousand model parameters [71] [72]; hence, the challenge lies in adapting them for autoencoder-based deep learning models that can feature up to a million parameters.

3. Methodology

3.1. Model and Priors

An autoencoder consists of two major parts: an encoder function \( f_\phi(x) \) and a decoder function \( f_\theta(h) \), where \( x \) is the data that represents a set of features and \( h \) is a set of latent (reduced) features. The autoencoder is assessed by how well the decoder can reconstruct the data from the encoding:

\[
R_{\text{loss}} = \arg \min_{\phi, \theta} |x - (f_\phi(f_\theta(x)))|^2.
\]
In particular, \( z(x, \theta) \) represents a feedforward neural network, with \( \theta \) being its set of weights and biases. An encoder-decoder pair is then constructed from it as

\[
\begin{align*}
h &= f_\theta(x) = z(x, \phi) \\
x &= f_\theta(h) = z(h, \psi)
\end{align*}
\]

where \( \phi \) and \( \psi \) represent the parameters (weights and biases) of the encoder and decoder, respectively.

The encoder extracts the essential features into a reduced representation while the decoder is used to reconstruct the input from the reduced representation as shown in Figure 1.

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

\[
R_{\text{loss}}
\]

In case when the data are labelled, another way to measure the performance of the autoencoder is by training another neural network using the features from the reduced representation \( h \) and comparing with performance when trained with original data. In contrast to principal component analysis, autoencoders are capable of nonlinear data reduction and can detect repetitive structures in the data [73].

Our likelihood function compares the original data \( x \) with decoder output \( x' = z(x, \theta) = z(z(x, \phi), \psi) \). We incorporate the \( \tau^2 \) parameter in the model which denotes the variance of data and also estimated; hence, our combined set of parameters becomes \( \Theta = (\theta, \tau^2) \). We only consider continuous data for the Bayesian autoencoder, where mean-squared-error (MSE) has been used as a measure for evaluation [74]. We hence assume the data to have Gaussian distribution and use a Gaussian likelihood given by

\[
P(x|\theta) = \frac{1}{(2\pi \tau)^{N/2}} \exp \left( -\frac{1}{2\tau^2} \sum_{i=1}^{N} (y_i - E(x_i|\theta'))^2 \right), \tag{1}
\]

where \( E(x_i|\theta') \) is the output of the neural network model for \( x_i \) input features denoted by \( t \) instances in the training data. \( N \) is the total number of instances in the training data. The priors are based on Gaussian distribution for \( \theta \) and inverse gamma distribution for \( \tau^2 \), respectively. Hence, the autoencoder weights and biases \( \theta \) are independent \textit{a priori} with a normal distribution with zero mean and variance \( \sigma^2 \). If \( \theta \) has \( L \) parameters in total, its joint prior with \( \tau^2 \) is

\[
\hat{P}(\Theta) \propto \frac{1}{(2\pi \tau^2)^{L/2}} \exp \left( -\frac{1}{2\tau^2} \sum_{t=1}^{L} \theta_t^2 \right)
\]

\[
\times \tau^{2(1+\nu_1)} \exp \left( -\frac{\nu_2}{\tau^2} \right), \tag{2}
\]

where \( \nu_1 \) and \( \nu_2 \) are user chosen constants. We note that \( \tau^2 \) is updated via random-walk proposal distribution.

### 3.2. Langevin gradient Metropolis–Hastings

We now present the MCMC algorithm for sampling from the Bayesian Autoencoder posterior corresponding to the prior and the likelihood given in previous section. We use a combination of methods: 1) efficient proposal distribution that uses Langevin-gradients, 2) parallel computing, 3) efficient multimodal sampling via tempered MCMC (parallel tempering).

Langevin-gradient proposal distribution that incorporate Gaussian noise with gradients for a single iteration (epoch). Recently, Langevin-gradient proposal distribution in MCMC sampling has been utilised for novel Bayesian neural learning methods [32, 75, 76, 77].

The Langevin-gradient (LG) proposal distribution at a given step \( n \) proposes to update the vector of weights and biases \( \theta_n \) from a biased multivariate normal distribution

\[
\theta_n \sim \mathcal{N}(\theta_n + \nu_3 \times \nabla E(\theta, x), \nu^2 \bar{I}_L) \tag{3}
\]

where \( \nu_3 \) (learning rate) and \( \nu_2 \) are user-defined tuning parameters. \( I_L \) is an \( L \times L \) identity matrix used to generate the stochastic noise, and the gradient component is given by

\[
E(\theta, x) = \sum_{t \in \mathcal{S}} (x_t - f(x_t, \theta))^2 \]

\[
\nabla E(\theta, x) = \left( \frac{\partial E}{\partial \theta_1}, \ldots, \frac{\partial E}{\partial \theta_L} \right)
\]

where \( \nabla E(\theta, x) \) is the gradient, and \( E(\theta, x) \) is the error or loss function for the autoencoder that features data \( x \).

The proposal attempts to “explore” the posterior density, given our prior \( \hat{P}(\Theta) \) and likelihood \( P(x|\Theta) \) defined in Equations (2) and (1). Furthermore, with \( \Theta^*_n = (\theta^*_n, \tau^2_n) \), we either accept or reject the proposal using the standard Metropolis-Hastings criterion

\[
\alpha = \min \left\{ 1, \frac{P(x|\Theta^*_n)\hat{P}(\Theta^*_n)Q(\Theta,\Theta^*_n)}{P(x|\Theta)\hat{P}(\Theta)Q(\Theta,\Theta^*_n)} \right\} \tag{4}
\]

with \( Q(\Theta^*_n|\Theta_n) = P(\Theta^*_n|\Theta_n) \), the conditional proposal density and vice versa.

In general, \( \nabla_{\theta} \log(\hat{P}(\Theta)P(x|\Theta)) \neq \nabla_{\theta} \log(\hat{P}(\Theta^*_n)P(x|\Theta^*_n)) \), and so \( Q(\Theta^*_n|\Theta_n) = Q(\Theta_n|\Theta^*_n) \) – an asymmetric proposal, and thus they do not cancel in Equation (4).

#### 3.3. Adaptive Langevin-gradient proposal distribution

In practice, in our trial experiments, we found that using the LG directly produces relatively slow mixing leading to inferior results for the case or large neural networks such as autoencoders. We therefore formulate a different proposals, borrowing ideas from the popular Adam optimiser [78] which for the case of optimisation is given as follows.

The Adam-based weight update in conventional learning paradigm is expressed as follows

\[
\theta_t = \theta_{t-1} - \alpha \times \hat{g}
\]

\[
\hat{g} = \frac{1}{\sqrt{1 - \beta^2_1} + \epsilon} \cdot \frac{\eta \cdot \nabla_{\theta} \log(\hat{P}(\Theta)P(x|\Theta))}{\sqrt{1 - \beta^2_2} + \epsilon} + \epsilon
\]

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3163270, IEEE Access

3.4. Parallel tempering MCMC Framework

The development of parallel tempering MCMC (also called tempered and replica exchange MCMC) has been motivated by the cooling behaviour of certain materials in the field of thermodynamics [79, 80, 81]. An ensemble of $M$ MCMC replicas $\Omega = \{R_1, R_2, \ldots, R_M\}$ is run each with a different temperature $T = [1, \ldots, T_{\text{max}}]$ where $T_{\text{max}}$ set by the user and defines the extent of exploration. A replica with temperature $t$ samples from an attenuated posterior distribution $P_t(\theta_t|\mathbf{x}) \propto P(\mathbf{x}|\theta_t)^{1/\beta} \mathbb{P}(\theta_t)$. This serves to “flatten” the distribution, giving replicates with high $t$ a generally higher probability of accepting proposals. The ensemble of replicas facilitates enhanced exploration capabilities, in particular, it enables a given replica to escape from a local maximum, allowing multi-modal and discontinuous posteriors to be explored [82, 83].

In previous work [77], Adam-based gradients known as adapt-LG proposal distributions have been used in MCMC framework for convolutional graph neural networks- which produced better results when compared to LG proposal distribution. We note that the learning rate is a user chosen hyper-parameter that is fixed in the case of LG and LG proposal distribution. We use an enhanced form of gradients $\hat{E}(\theta, \mathbf{x})$ to obtain the adapt-LG proposal distribution as follows

$$\hat{\nu}_{k} = \beta_{1} \nu_{k-1} + (1 - \beta_{1}) \times \nabla E(\theta, \mathbf{x})$$

$$\mu_{k} = \beta_{2} \mu_{k-1} + (1 - \beta_{2}) \times \nabla E(\theta, \mathbf{x})$$

where $\beta_{1}$ and $\beta_{2}$ are first and second moment estimates, respectively. $\epsilon$ is a small scalar used to prevent division by 0. $\alpha$ is the user defined learning rate and typically, the default values are $\beta_{1} = 0.99$, $\beta_{2} = 0.999$, $\alpha = 10^{-3}$, and $\epsilon = 10^{-8}$ (implementation in PyTorch).

In previous work [77], Adam-based gradients known as adaptive Langevin gradient (adapt-LG) proposal distributions have been used in MCMC framework for convolutional graph neural networks- which produced better results when compared to LG proposal distribution. We note that the learning rate is a user chosen hyper-parameter that is fixed in the case of LG and LG proposal distribution. We use an enhanced form of gradients $\hat{E}(\theta, \mathbf{x})$ to obtain the adapt-LG proposal distribution as follows

$$\hat{\nu}_{k} = \beta_{1} \nu_{k-1} + (1 - \beta_{1}) \times \nabla E(\theta, \mathbf{x})$$

$$\mu_{k} = \beta_{2} \mu_{k-1} + (1 - \beta_{2}) \times \nabla E(\theta, \mathbf{x})$$

where $\beta_{1}$ and $\beta_{2}$ are first and second moment estimates, respectively. $\epsilon$ is a small scalar used to prevent division by 0. $\alpha$ is the user defined learning rate and typically, the default values are $\beta_{1} = 0.99$, $\beta_{2} = 0.999$, $\alpha = 10^{-3}$, and $\epsilon = 10^{-8}$ (implementation in PyTorch).

In previous work [77], Adam-based gradients known as adaptive Langevin gradient (adapt-LG) proposal distributions have been used in MCMC framework for convolutional graph neural networks- which produced better results when compared to LG proposal distribution. We note that the learning rate is a user chosen hyper-parameter that is fixed in the case of LG and LG proposal distribution. We use an enhanced form of gradients $\hat{E}(\theta, \mathbf{x})$ to obtain the adapt-LG proposal distribution as follows

$$\hat{\nu}_{k} = \beta_{1} \nu_{k-1} + (1 - \beta_{1}) \times \nabla E(\theta, \mathbf{x})$$

$$\mu_{k} = \beta_{2} \mu_{k-1} + (1 - \beta_{2}) \times \nabla E(\theta, \mathbf{x})$$

where $\beta_{1}$ and $\beta_{2}$ are first and second moment estimates, respectively. $\epsilon$ is a small scalar used to prevent division by 0. $\alpha$ is the user defined learning rate and typically, the default values are $\beta_{1} = 0.99$, $\beta_{2} = 0.999$, $\alpha = 10^{-3}$, and $\epsilon = 10^{-8}$ (implementation in PyTorch).

In previous work [77], Adam-based gradients known as adaptive Langevin gradient (adapt-LG) proposal distributions have been used in MCMC framework for convolutional graph neural networks- which produced better results when compared to LG proposal distribution. We note that the learning rate is a user chosen hyper-parameter that is fixed in the case of LG and LG proposal distribution. We use an enhanced form of gradients $\hat{E}(\theta, \mathbf{x})$ to obtain the adapt-LG proposal distribution as follows

$$\hat{\nu}_{k} = \beta_{1} \nu_{k-1} + (1 - \beta_{1}) \times \nabla E(\theta, \mathbf{x})$$

$$\mu_{k} = \beta_{2} \mu_{k-1} + (1 - \beta_{2}) \times \nabla E(\theta, \mathbf{x})$$

where $\beta_{1}$ and $\beta_{2}$ are first and second moment estimates, respectively. $\epsilon$ is a small scalar used to prevent division by 0. $\alpha$ is the user defined learning rate and typically, the default values are $\beta_{1} = 0.99$, $\beta_{2} = 0.999$, $\alpha = 10^{-3}$, and $\epsilon = 10^{-8}$ (implementation in PyTorch).

In previous work [77], Adam-based gradients known as adaptive Langevin gradient (adapt-LG) proposal distributions have been used in MCMC framework for convolutional graph neural networks- which produced better results when compared to LG proposal distribution. We note that the learning rate is a user chosen hyper-parameter that is fixed in the case of LG and LG proposal distribution. We use an enhanced form of gradients $\hat{E}(\theta, \mathbf{x})$ to obtain the adapt-LG proposal distribution as follows

$$\hat{\nu}_{k} = \beta_{1} \nu_{k-1} + (1 - \beta_{1}) \times \nabla E(\theta, \mathbf{x})$$

$$\mu_{k} = \beta_{2} \mu_{k-1} + (1 - \beta_{2}) \times \nabla E(\theta, \mathbf{x})$$

where $\beta_{1}$ and $\beta_{2}$ are first and second moment estimates, respectively. $\epsilon$ is a small scalar used to prevent division by 0. $\alpha$ is the user defined learning rate and typically, the default values are $\beta_{1} = 0.99$, $\beta_{2} = 0.999$, $\alpha = 10^{-3}$, and $\epsilon = 10^{-8}$ (implementation in PyTorch).
parallel tempering ensemble, and the inter-process communication for exchange of neighbouring replica states is shown in Figure 2 with further details in Algorithm 1.

Firstly, we define the autoencoder topology by specifying the number of inputs, hidden layers, and outputs for the encoder and (reversed) for the decoder. Secondly, we define the hyperparameters for the parallel tempering MCMC sampler, such as number of replicas (M), the geometric temperature ladder maximum temperature (T_{\text{max}}), replica swap interval (R_{\text{swap}}) that determines how often to propose an exchange between neighbouring replicas, and the maximum number of draws (samples) for each replica (R_{\text{max}}). In parallel processing, it is difficult to determine when to stop sampling and hence this is done by checking number of alive process. Therefore, we need to set the number of replicas in ensemble as alive = M. These are the key steps in initialisation as shown in Stage 0 of Algorithm 1.

The algorithm uses parallel tempering MCMC in its first phase, then switches to canonical MCMC in its second phase, which is defined by number of samples need to switch R_{\text{switch}}. During the switch, all the temperature values in the ladder is changed to 1 as done by [32, 84]. The algorithm implements adapt-LG* proposal distribution (adapt-LG in the first phase and LG in the second phase) using R_{\text{switch}}.

In Algorithm 1 we begin within replica sampling (Stage 1.1) by iterating through the respective replicas where the manager process executes and manages parallel processes. In Stage 1.2, each replica sample creates a proposal using either random-walk or Langevin-gradients (either LG or adapt-LG*) (Equation 3) and computes the likelihood (Stage 1.3) and uses Metropolis-Hastings condition in Stage 1.4 to checks if the proposal is good enough to be part of the posterior distribution. In Stage 1.5, the algorithm checks if it needs to use tempered MCMC or needs to transform into canonical MCMC where temperature values of the all the replicas in the ensemble are set to 1. In the case of tempered MCMC, the Langevin-gradients uses the adapt-LG proposal distribution and canonical MCMC uses LG. Note that the samples used by tempered MCMC and adapt-LG are discarded as a burn-in period. In Stage 2.1, the algorithm computes the likelihood of the replica exchange which is similar to the way its computed in Equation 6, and then it exchanges the neighbouring replica using Metropolis-Hastings condition (Stage 2.2).

In Stage 3, the algorithm checks if the maximum number of samples (R_{\text{max}}) has been reached, and if so, the number of replicas alive is decremented to eventually stop the entire sampling process. Finally, we reach the end in Stage 4 one the sampling process ends and we combine the respective replica posterior distribution of the autoencoder (weights and biases) for further analysis.

Rather than swapping the entire replica state (weights and biases in the autoencoder), it is possible to swap the respective neighbouring replica temperature values, which is cheaper computationally.

Figure 2: Bayesian autoencoder framework highlighting tempered MCMC utilising parallel computing and autoencoder neural network.
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 ($T_{\text{max}}$), replica swap interval ($R_{\text{swap}}$), and maximum number of samples for each replica ($R_{\text{max}}$).
* Set the number of replicas ($M$ in ensemble as $\text{alive}; \text{alive} = M$)
* Set the number-samples ($R_{\text{switch}}$) for first-phase (parallel tempering MCMC) and second-phase (canonical MCMC)
* Set $g_{\text{prob}}$ which determines how often to apply Langevin-grads.
* Set adapt-grad to True to ensure that LG is used by default.

while $\text{alive} \neq 0$ do
    Stage 1.0: Execute each replica via manager process
        for $i = 1$ to $M$ do
            first-phase: $T_i = \text{geometric}()$
            for $s = 1$ to $R_{\text{max}}$ do
                Stage 1.1: Within Replica Transition
                    for $k = 1$ to $R_{\text{swap}}$ do
                        1.2 if Unif(0, 1) $\leq g_{\text{prob}}$ then
                            if adapt-grad is True then
                                Create adapt-LG proposal using Equation (5)
                            else
                                Create LG proposal using Equation (3)
                            end
                        else
                            Create Random-walk proposal
                        end
                        1.3 Evaluate the respective likelihoods compute probability $\alpha$
                        1.4 if Unif(0, 1) $\leq \alpha$ then
                            Accept replica position, $\Theta_s \leftarrow \Theta^*$
                        else
                            Reject to retain previous state: $\Theta_s \leftarrow \Theta_{s-1}$
                        end
                        1.5 if $R_{\text{switch}} \leq s$ then
                            Update temperature, $T_i = 1$
                            Set adapt-grad to False
                        end
                    end
                Stage 2.0: Between Replica Transition:
                    2.1 Compute exchange replica acceptance probability $\beta$
                    2.2 if Unif(0, 1) $\leq \beta$ then
                        Signal() manager process Swap selected neighboring Replica, $\Theta_i \leftrightarrow \Theta_{i+1}$
                    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

Algorithm 1: Bayesian autoencoders that use tempered MCMC in the first phase and then canonical MCMC in the second phase based on $R_{\text{switch}}$. The algorithm implements adapt-LG* proposal distribution (LG first phase and adapt-LG second phase) using $R_{\text{switch}}$. The manager process is highlighted in black and replica processes running in parallel are highlighted in pink. Unif(0, 1) is shorthand for a single draw from a standard uniform distribution.
4. Experiments and Results

4.1. Dataset Description

We selected a variety of datasets commonly used in the literature to demonstrate similar methodology, making sure to consider a variety of sizes and other properties. They include Madelon [85], Coil-2000 [86] and Swiss Roll [87]. We summarise them in Table 1.

The synthetic Madelon dataset features data points grouped in 32 clusters, each on a vertex of a five-dimensional hypercube. The clusters are randomly labeled +1 or -1. In addition to these 5 coordinates, 15 of their linear combinations had been added for a total of 20 (redundant) informative features, as well as 480 random distractors. Coil-2000 has 86 variables containing customer information, including product usage data and socio-demographic data derived from postal codes, and whether the customer has a caravan insurance policy. Lastly, the synthetic Swiss Roll dataset arranges its data points in a shape a Swiss Roll in three dimensions and is a popular dataset for visualisation of data reduction methods [88, 89, 90].

The respective datasets were normalized before training in the range of [0,1]. This was done to ensure that all features had equal importance in the model.

4.2. Implementation

Table 2 provides the Bayesian autoencoder(adapt-LG*) topology that is used in the experiments described in this section. The autoencoder is trained, and the compressed or reduced dataset is used with another machine learning model for training to get classification accuracy, analogously to [91, 92, 93, 94, 95, 73].

We evaluate some of the critical parameters in MCMC sampling process to observe the effects on the sampling performance, computational time, and reconstruction accuracy. The Bayesian autoencoder is designed to reduce Madelon dataset with 500 features to 300. In the case of the Coil 2000 dataset, the Bayesian autoencoder reduces 85 features to 50 features. In the Swiss Roll dataset, the Bayesian autoencoder reduces 3 features to 2 features, essentially “unwrapping” the Swiss Roll, as shown in Figure 3.

In all experiments, we use MCMC sampler parameters as follows. In random-walk proposals and the random component of the Langevin proposal, the Gaussian noise is added with standard deviation v1 of 0.005, centred at 0 and we use v1 = 0 and v2 = 3 as shown in prior [3]. We use temperature maximum T_max = 2 and swap interval R_swap = 5 samples (iterations) for all experiments. We use M = 8 replicas with R_max = 6000 MCMC iterations per replica. After R_switch = 3000 iterations, parallel tempering MCMC is converted to canonical MCMC with neighbourhood exchange by setting the replica’s temperature to 1.

We implement the Bayesian autoencoder using pyTorch library [4] with Python multi-processing library [5] for parallel MCMC replica processes.

Using the reduced feature sets, we use standard classifiers and compare the results with the literature. We compare our results with several common classification algorithms: K-nearest neighbors (KNN) [96]; support vector machines (SVM) [97]; random subspaces ensemble (RSE) [93] method which uses ensemble of decision trees; and naive Bayes [95].

4.3. Preliminary investigation: parameter tuning

We first carry out a preliminary investigation to understand the effect of certain parameters on MCMC sampling and show visualisation of the results using the Swiss Roll dataset. Table 3 gives the MSE for different Bayesian autoencoder configurations in MCMC sampling with adaptive Langevin based proposal distribution for the Swiss Roll dataset. We show the trend of MSE as we change the MCMC parameters (step-size or variance in proposal distribution and learning rate for Langevin-gradients) to make it more difficult to accept weaker samples. Hence, we get a acceptance percentage and we try to find a balance between the acceptance percentage and MSE. We note that in some models, a acceptance of around 23.4 percent [28] is known to be optimum for MCMC sampling. The highlighted configuration in Table 3 is visualised in Figure 3. We also find that the MSE for the chosen configuration is close to the canonical autoencoder.

Figure 3 gives a visual representation of the Swiss Roll dataset for 2500 data-points. It shows the 3D and 2D views of the various stages of the visualization process. Panel (a) shows the original dataset; Panel (b) shows the reconstructed dataset after reducing and then reconstructing it through a canonical autoencoder; and Panel (c) uses the same architecture but the Bayesian autoencoder. The visualisation and analysis (Table 3) and Figure 3 demonstrate the effectiveness of the Bayesian autoencoder and its applicability to larger datasets.

4.4. Results

We evaluate specific aspects of or proposed Bayesian autoencoder framework using the Madelon dataset. The Bayesian autoencoder reduces number of the features from 500 to 300 features (encoder) and then reconstructs again from the 300 features into the 500 features (decoder). We report the MSE between the reconstructed dataset and the original dataset. We first present results in terms of computational efficiency, effect of Langevin-gradient rate and number or replicas (M) on MCMC sampling for the Madelon problem using 48,000 samples. We note that the Langevin-gradient is for one iteration or epoch which is defined by a single forward and backward pass via backpropagation.

Table 1: Overview of dataset used for dimensional reduction.

| Dataset      | No. Features | No. Train | No. Test |
|--------------|--------------|-----------|----------|
| Madelon      | 500          | 2000      | 1800     |
| Coil-2000    | 85           | 5822      | 4000     |
| Swiss Roll   | 3            | 3750      | 1250     |

4. PyTorch library: https://pytorch.org/
5. Python multi-processing library: https://docs.python.org/3/library/multiprocessing.html
Table 2: Bayesian autoencoder (adapt-LG*) topology showing the total number of parameters (weights and biases).

| Data       | Input Layer (Input–Output) | Hidden Layer 1 (Input–Output) | Hidden Layer 2 (Input–Output) | Hidden Layer 3 (Input–Output) | Hidden Layer 4 (Input–Output) | Output Layer (Input–Output) | Total Parameters |
|------------|---------------------------|-------------------------------|-------------------------------|-------------------------------|-------------------------------|-------------------------------|------------------|
| 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 | 3–10                      | 10–5                          | 5–2                           | 2–5                           | 5–10                          | 10–3                          | 224              |

Table 3: Performance (MSE) with different settings of MCMC parameters for the two instances of the Swiss Roll dataset using Topology shown in Table 2

| Method                                      | Topology    | MSE     | Acceptance Percentage | Step-size | Learning-rate |
|---------------------------------------------|-------------|---------|-----------------------|-----------|---------------|
| Canonical Autoencoder (Adam)                | Swiss Roll  | 0.012   | -                     | -         | -             |
| Bayesian Autoencoder (adapt-LG*)            | Swiss Roll  | 0.025   | 19.250                | 0.030     | 0.040         |
| **Bayesian Autoencoder (adapt-LG*)**        | Swiss Roll  | **0.010** | **0.125**            | **0.005** | **0.010**     |
| Bayesian Autoencoder (adapt-LG*)            | Swiss Roll  | 0.104   | 31.630                | 0.080     | 0.090         |

Table 4: MSE as a function of the Langevin-gradient rate for the Madelon dataset

| Langevin-gradient Rate | Train MSE Best (Mean,Std) | Test MSE Best (Mean,Std) | Swap % | Acceptance % | Time (mins.) |
|------------------------|---------------------------|--------------------------|--------|--------------|--------------|
| 0                      | 0.946 (1.908,0.952)       | 0.937 (1.899,0.951)      | 78.14  | 1.20         | 184          |
| 0.25                   | 0.019 (0.024,0.004)       | 0.019 (0.025,0.004)      | 77.23  | 15.78        | 242          |
| 0.5                    | 0.011 (0.027,0.004)       | 0.013 (0.027,0.004)      | 78.39  | 22.68        | 302          |
| 0.75                   | 0.012 (0.021,0.007)       | 0.013 (0.023,0.006)      | 71.57  | 34.64        | 359          |
| 1                      | 0.012 (0.019,0.005)       | 0.012 (0.021,0.002)      | 76.85  | 40.25        | 435          |

Table 5: Effect of the number of replicas on Madelon dataset

| Replicas | Train MSE Best (Mean,Std) | Test MSE Best (Mean,Std) | Swap % | Acceptance % | Time (mins.) |
|----------|---------------------------|--------------------------|--------|--------------|--------------|
| 2        | 0.012 (0.018,0.009)       | 0.013 (0.019,0.006)      | 27.29  | 32.0         | 564          |
| 4        | 0.013 (0.020,0.01)        | 0.015 (0.023,0.008)      | 42.67  | 32.5         | 527          |
| 6        | 0.013 (0.014,0.008)       | 0.014 (0.016,0.007)      | 56.93  | 32.0         | 483          |
| 8        | 0.012 (0.021,0.007)       | 0.013 (0.023,0.006)      | 71.57  | 34.64        | 359          |
Table 4 shows the effect of Langevin-gradient rate ($g_{prob}$) on the Bayesian autoencoder (adapt-LG*) performance, including training and testing mean of the MSE, its standard deviation (std), and the highest (worst) MSE. Since all of these are sampling from the same posterior, the MSE should be the same as well, but as the table shows, without the Langevin-gradient proposal, the sampler is unable to find the posterior modes. On the other hand, these proposals are much more computationally costly, so as they are made more frequently, the computational time also increases.

Next, we evaluate the effect of the number of replicas ($M$) using the Bayesian autoencoder (adapt-LG*). Table 5 summarises the results. As expected, we show a lower computational time with a similar MSE as the number of replicas increase, suggesting that parallel processing is useful in this case.

Figure 4 shows MCMC trace plots of the training and testing accuracy for one of the selected replicas of the Bayesian autoencoder (adapt-LG*), demonstrating convergence to the neighbourhood of a posterior mode. We present a quantitative assessment of convergence in Table 6 in the form of Gelman–Rubin [99] diagnostics. This diagnostic works by comparing the variance between replicates to the variance within them. A ratio sufficiently close to 1 indicates convergence to the MCMC’s stationary distribution – the posterior [99].

Figure 5 presents the log-likelihood of selected replicas (temperature level of 1.0) of the Bayesian autoencoder (adapt-LG*) for the respective problems. Figure 6 shows the trace plots and posterior distribution for the selected weights of the Bayesian autoencoder (adapt-LG*) with the convergence diagnostic given in Table 6.

Compression by autoencoders is typically benchmarked by applying machine learning methods, by calculating the MSE difference between the original and the reconstructed data, or by classifying the reduced dataset and comparing with classification on the original dataset [91, 92, 93, 94, 95, 74]. In our case, Figure 3 presents the MSE between the original and reconstructed dataset using multi-layer perceptron (MLP), where linear activation function in hidden layers are used with sigmoid activation function at the output layers.

Table 7 presents a comparison of the results to selected methods from the literature that use the compressed datasets for classification tasks. Caution should be exercised when making direct comparisons, because the autoencoder architectures may vary somewhat between papers. To the extent that these comparisons are valid, the Bayesian autoencoder approach with SVM outperforms the classification accuracy given in most methods from literature for the respective datasets. This shows the effectiveness of the Bayesian autoencoder as an alternative for dimensionality reduction, and motivates use of autoencoders as a possible data pre-processing step to increase classification accuracy.

Table 8 compares the canonical and the Bayesian autoencoder using SGD and Adam algorithms, including the best (lowest) MSE, the mean and the standard deviation of it under the posterior distribution. As evident in Table 7 we find that the standalone SGD optimizer does not perform well for large number of model parameters. We also find that the Bayesian autoencoder performs better in dimensionality reduction for all three datasets.

5. Discussion

Our primary goal was to incorporate Bayesian inference via MCMC into autoencoders. The proposed Bayesian autoen-
Table 6: Gelman–Rubin MCMC convergence diagnostics for the weight chains in parallel tempering

| Dataset       | ID-0 | ID-50 | ID-100 | ID-150 | ID-2000 | ID-3000 | ID-4000 | ID-6000 | ID-9000 | ID-10,000 |
|---------------|------|-------|--------|--------|---------|---------|---------|---------|---------|-----------|
| Madelon       | 1.14 | 1.14  | 1.15   | 1.21   | 1.16    | 1.13    | 1.18    | 1.25    | 1.13    | 1.25      |
| Coil 2000     | 1.14 | 1.16  | 1.18   | 1.23   | 1.15    | 1.17    | 1.11    | 1.24    | 1.12    | 1.13      |
| Swiss Roll    | 1.14 | 1.24  | 1.13   | 1.16   | -       | -       | -       | -       | -       | -         |

Table 7: Bayesian autoencoder accuracy rates comparison of established methods from literature

| Method                                    | Algorithm          | Madelon (Best Mean, Std) | Coil 2000 (Best Mean, Std) |
|-------------------------------------------|--------------------|--------------------------|----------------------------|
| Autoencoder based classifier [91]         | kNN                | 0.547 (-,-)              | 0.929 (-,-)                |
| Ensemble Classification using Dimensionality Reduction [93] | RSE                | 0.5565 (-, 0.0263)       | -                          |
| Autoencoder inspired unsupervised feature selection [94] | kNN                | 0.71 (-,-)               | -                          |
| Multi-objective Evolutionary Approach [95] | Naive Bayes        | -                        | 0.93 (-, 0.014)            |
| Multi-objective Evolutionary Approach [95] | SVM                | -                        | 0.948 (-, 0.002)           |
| Bayesian-Autoencoder (LG)                 | kNN                | 0.49 (0.45, 0.01)        | 0.89 (0.81, 0.05)          |
| Bayesian-Autoencoder (adapt-LG*)          | kNN                | 0.554 (0.523, 0.018)     | 0.958 (0.92, 0.015)        |
| Bayesian-Autoencoder (adapt-LG*)          | SVM                | 0.589 (0.554, 0.021)     | 0.96 (0.94, 0.009)         |

Table 8: Autoencoder MSE for the three datasets

| 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 (LG)                 | 0.020 (0.03,0.016)          | 0.050 (0.060,0.003)      | 0.033 (0.036,0.001)        |
| Bayesian Autoencoder (adapt-LG*)          | 0.010 (0.014,0.063)         | 0.013 (0.023,0.006)      | 0.016 (0.023,0.002)        |

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

Figure 4: MSE for replica with temperature=1

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

Figure 5: Log-Likelihood for replica with temperature=1

(a) Madelon
(b) Coil 2000
(c) Swiss Roll
coder provides an approach for uncertainty quantification in the reduced dataset. We can reconstruct an ensemble of reduced datasets from the posterior distribution rather than a single one given by single-point estimates. The results show that the Bayesian autoencoder framework can efficiently sample the posterior distribution of the autoencoder's weights and biases, which compared very well with canonical methods (such as SGD and Adam) in terms of accuracy. We also found that our approach is competitive with the other methods found in the literature.

It is unusual to use MCMC methods for deep learning, which often features models with tens of thousands of parameters; however, the field is slowly gaining momentum [70]. We have shown that with parallel computing and sophisticated gradient-based proposal distributions in MCMC, it is feasible for them to be used, nonetheless. We finally demonstrated how MCMC diagnostics apply to this problem.

One limitation is that convergence analyses for MCMC samplers – such as the Gelman–Rubin diagnostic – are generally designed for modest numbers of parameters. More work is needed to develop those more suited for deep learning applications.

Furthermore, in large deep learning models, although tens of thousands of parameters are used, not all nodes and weights are equally valuable. Optimisation and evolutionary computation methods have been used for pruning weights of large deep learning models, and there is potential for Bayesian inference methods to be used in model pruning. Hence, future research can focus on determining and differentiating essential weights from non-essential ones using specific metrics to determine which weights have a major effect on the model and determine which weights have the most effect on the performance of the model.
Figure 6: Posterior and trace plot for selected weights for the respective problems.
6. Conclusions

We present an alternative framework to the variational autoencoder that employs MCMC sampling for Bayesian autoencoder. We address the known limitations of MCMC methods for Bayesian deep learning model with the power of parallel computing and enhanced proposal distributions. 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.

Code and Data

We provide the code as open source software via a GitHub repository.

References

[1] P. Baldi, Autoencoders, unsupervised learning, and deep architectures, in: Proceedings of ICML workshop on unsupervised and transfer learning, 2012, pp. 37–49.

[2] C. Doersch, Tutorial on variational autoencoders, arXiv preprint arXiv:1606.05908 (2016).

[3] W. Luo, J. Li, J. Yang, W. Xu, J. Zhang, Convolutional sparse autoencoders for image classification, IEEE transactions on neural networks and learning systems 29 (7) (2018) 3289–3294.

[4] P. Zhou, J. Han, G. Cheng, B. Zhang, Learning compact and discriminative stacked autoencoder for hyperspectral image classification, IEEE Transactions on Geoscience and Remote Sensing 57 (7) (2019) 4823–4833.

[5] S. Gao, Y. Zhang, K. Jia, J. Lu, Y. Zhang, Single sample face recognition via learning deep supervised autoencoders, IEEE transactions on information forensics and security 10 (10) (2015) 2108–2118.

[6] M. Kan, S. Shan, H. Chang, X. Chen, Stacked progressive auto-encoders (spa) for face recognition across poses, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2014, pp. 1883–1890.

[7] G. Dong, G. Liao, H. Liu, G. Kuang, A review of the autoencoder and its variants: A comparative perspective from target recognition in synthetic-aperture radar images, IEEE Geoscience and Remote Sensing Magazine 6 (3) (2018) 44–68.

[8] S. Sahu, R. Gupta, G. Sivaraman, W. AbdAlmageed, C. Espy-Wilson, Adversarial auto-encoders for speech based emotion recognition, arXiv preprint arXiv:1806.02146 (2018).

[9] A. Ranjan, T. Bolkart, S. Sanjal, M. J. Black, Generating 3d faces using convolutional mesh autoencoders, in: Proceedings of the European Conference on Computer Vision (ECCV), 2018, pp. 704–720.

[10] J. Masci, U. Meier, D. Ciresan, J. Schmidhuber, Stacked convolutional auto-encoders for hierarchical feature extraction, in: International conference on artificial neural networks, Springer, 2011, pp. 52–59.

[11] J. Deng, Z. Zhang, E. Marchi, B. Schuller, Sparse autoencoder-based feature transfer learning for speech emotion recognition, in: 2013 Humaine Association Conference on Affective Computing and Intelligent Interaction, IEEE, 2013, pp. 511–516.

[12] G. Alain, Y. Bengio, What regularized auto-encoders learn from the data-generating distribution, The Journal of Machine Learning Research 15 (1) (2014) 3563–3593.

[13] D. P. Kingma, M. Welling, Auto-encoding variational bayes, arXiv preprint arXiv:1312.6114 (2013).

[14] C. K. Sønderby, T. Raiko, L. Maaløe, S. K. Sønderby, O. Winther, Ladder variational autoencoders, in: Advances in neural information processing systems, 2016, pp. 3738–3746.

[15] I. Tolstikhin, O. Bousquet, S. Gelly, B. Schoelkopf, Wasserstein auto-encoders, arXiv preprint arXiv:1711.01558 (2017).

[16] D. Liang, R. G. Krishnan, M. D. Hoffman, T. Jebara, Variational autoencoders for collaborative filtering, in: Proceedings of the 2018 World Wide Web Conference, 2018, pp. 689–698.

[17] J. Zhao, Y. Kim, K. Zhang, A. Rush, Y. LeCun, Adversarially regularized autoencoders, in: International Conference on Machine Learning, PMLR, 2018, pp. 5902–5911.

[18] L. Begggel, M. Pfeiffer, B. Bischof, Robust anomaly detection in images using adversarial autoencoders, in: Joint European Conference on Machine Learning and Knowledge Discovery in Databases, Springer, 2019, pp. 206–222.

[19] S. Pudhorskyi, R. Almohsen, G. Doretto, Generative probabilistic novelty detection with adversarial autoencoders, in: Advances in neural information processing systems, 2018, pp. 6822–6833.

[20] A. Creswell, A. A. Bharath, Denoising adversarial autoencoders, IEEE transactions on neural networks and learning systems 30 (4) (2019) 986–984.

[21] T. N. Kipf, M. Welling, Variational graph auto-encoders, arXiv preprint arXiv:1611.07308 (2016).

[22] Y. Ding, L.-P. Tian, X. Lei, B. Liao, F.-X. Wu, Variational graph auto-encoders for mirna-disease association prediction, Methods (2020).

[23] D. J. MacKay, Probable networks and plausible predictions—a review of practical bayesian methods for supervised neural networks, Network: Computation in Neural Systems 6 (3) (1995) 469–505.

[24] R. M. Neal, Bayesian learning for neural networks, Vol. 118, Springer Science & Business Media, 2012.

[25] D. J. MacKay, A practical bayesian framework for backpropagation networks, Neural computation 4 (3) (1992) 448–472.

[26] G. E. Hinton, D. Van Camp, Keeping the neural networks simple by minimizing the description length of the weights, in: Proceedings of the sixth annual conference on Computational learning theory, ACM, 1993, pp. 5–13.

[27] A. Graves, Practical variational inference for neural networks, in: Advances in neural information processing systems, 2011, pp. 2348–2356.

[28] Y. Burda, R. Grosse, R. Salakhutdinov, Importance weighted autoencoders, arXiv preprint arXiv:1509.00519 (2015).

[29] G. O. Roberts, R. L. Tweedie, Convergence exponential of langevin distribution and their discrete approximations, Bernoulli (1996) 341–363.

[30] G. O. Roberts, J. S. Rosenthal, Optimal scaling of discrete approximations to langevin diffusions, Journal of the Royal Statistical Society: Series B (Statistical Methodology) 60 (1) (1998) 255–268.

[31] M. Welling, Y. W. Teh, Bayesian learning via stochastic gradient langevin dynamics, in: Proceedings of the 28th International Conference on Machine Learning (ICML-11), 2011, pp. 681–688.

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

[33] R. Chandra, A. Kapoor, Bayesian neural multi-source transfer learning, Neurocomputing 378 (2020) 54 – 64.

[34] B. L. Betechuoh, T. Marwala, T. Tettey, Autoencoder networks for hiv classification, Current Science (2006) 1467–1473.

[35] D. Avino, D. Cazzolino, G. Poggi, L. Verdoliva, Autoencoder with recurrent neural networks for video forgery detection, Electronic Imaging 2017 (7) (2017) 92–99.

[36] F. Zhao, J. Feng, J. Zhao, W. Yang, S. Yan, Robust lstm-autoencoders for face de-occlusion in the wild, IEEE Transactions on Image Processing 27 (2) (2018) 778–790.

[37] A. Taheri, K. Gimpel, T. Berger-Wolf, Learning graph representations with adversarial autoencoders, in: Joint European Conference on Machine Learning and Knowledge Discovery in Databases, Springer, 2019, pp. 681–688.

[38] H. Pardede, E. Suryawati, R. Sustika, V. Zilvan, Unsupervised convolutional auto-encoders for hierarchical feature extraction, in: International conference on artificial neural networks, Springer, 2011, pp. 52–59.

[39] S. Sahu, R. Gupta, G. Sivaraman, W. AbdAlmageed, C. Espy-Wilson, Adversarial auto-encoders for speech based emotion recognition, arXiv preprint arXiv:1806.02146 (2018).

[40] A. Ranjan, T. Bolkart, S. Sanjal, M. J. Black, Generating 3d faces using convolutional mesh autoencoders, in: Proceedings of the European Conference on Computer Vision (ECCV), 2018, pp. 704–720.

[41] J. Masci, U. Meier, D. Ciresan, J. Schmidhuber, Stacked convolutional auto-encoders for hierarchical feature extraction, in: International conference on artificial neural networks, Springer, 2011, pp. 52–59.

[42] J. Deng, Z. Zhang, E. Marchi, B. Schuller, Sparse autoencoder-based feature transfer learning for speech emotion recognition, in: 2013 Humaine Association Conference on Affective Computing and Intelligent Interaction, IEEE, 2013, pp. 511–516.

[43] G. Alain, Y. Bengio, What regularized auto-encoders learn from the data-generating distribution, The Journal of Machine Learning Research 15 (1) (2014) 3563–3593.

[44] D. P. Kingma, M. Welling, Auto-encoding variational bayes, arXiv preprint arXiv:1312.6114 (2013).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3163270, IEEE Access.
An unsupervised fuzzy rule-based approach (04 2020).

[89] L. Van Der Maaten, E. Postma, J. Van den Herik, Dimensionality reduction: a comparative review, J Mach Learn Res 10 (2009) 66–71.

[90] D. Peluffo, J. Salazar Castro, J. Alvarado Pérez, Interactive interface for efficient data visualization via a geometric approach, 2015. doi:10.31344/0753194.2015.73303397

[91] F. Pulgar, F. Charte, A. Rivera Rivas, M. J. Del Jesus, Aeknn: An autoencoder knn-based classifier with built-in dimensionality reduction, International Journal of Computational Intelligence Systems 12 (02 2018). doi:10.2991/ijcis.2018.125905686

[92] P. Rauber, A. Falcão, A. Telea, Projections as visual aids for classification system design, Information Visualization 17 (2017) 1473871617713333. doi:10.1177/1473871617713337

[93] A. Schclar, L. Rokach, A. Amit, Ensembles of classifiers based on dimensionality reduction (2013). arXiv:1305.4345

[94] K. Han, Y. Wang, C. Zhang, C. Li, Autoencoder inspired unsupervised feature selection, 2018, pp. 2941–2945. doi:10.1109/ICASSP.2018.8462261

[95] D. Singh, D. S. Sisodia, P. Singh, Multi-objective evolutionary approach for the performance improvement of learners using ensembling feature selection and discretization technique on medical data (2020). arXiv:2004.07478

[96] Z. Zhang, Introduction to machine learning: K-nearest neighbors, Annals of Translational Medicine 4 (2016) 218–218. doi:10.21037/atm.2016.03.37

[97] M. A. Hearst, S. T. Dumais, E. Osuna, J. Platt, B. Scholkopf, Support vector machines, IEEE Intelligent Systems and their applications 13 (4) (1998) 18–28.

[98] M. Bédard, Optimal acceptance rates for metropolis algorithms: Moving beyond 0.234, Stochastic Processes and their Applications 118 (12) (2008) 2198–2222. doi:https://doi.org/10.1016/j.spa.2007.12.005

URL https://www.sciencedirect.com/science/article/pii/S0304414907002177

[99] D. Vats, C. Knudson, Revisiting the Gelman-Rubin diagnostic (2020). arXiv:1812.09394