IE 506: Machine Learning - Principles and Techniques **Jan-May 2023**

End-term Project Report: Langevin Autoencoders for Learning Deep Latent Variable Models

Team Name: Supernova *Team Members:* Tarun Bisht

Abstract

This project report aims to train deep latent variable model using Markov Chain Monte Carlo (MCMC) by simulating a stochastic differential equation called Langevin dynamics. This project study Langevin Autoencoder (LAE) [12] that uses Amortized Langevin dynamics (ALD) that replaces datapoint-wise MCMC iteration with updates on an encoder that maps datapoints into latent variable. Using MNIST dataset we compare generation capability and latent space of LAE and Variational Autoencoder (VAE). We also benchmark different parameters of LAE and their effect. We then proposed a new training algorithm based on deep latent variable models (DLVM's) that incorporates DLVM while training model to generate similar data samples. We bechmarked our proposed training method and show that this training procedure is robust towards adversarial attacks than normal training.

1 Introduction

In Unsupervised learning, the main task is to learn useful representations from unlabeled data. These tasks include finding the underlying structure or interesting patterns, clustering, statistical correlations, and generating data with the help of unlabeled data. This problem is harder than a supervised learning where we are given corresponding labels of training examples instances that supervises the learning algorithm. Here, instead of just predicting the target label, the model has to learn to describe the input itself without explicit supervision. If we have a model that explains data well then it can be used for semi-supervised learning using a small labeled dataset. As supervised learning methods cannot be scaled because we need labeled data and collecting large labelled dataset is expensive therefore using such models with small labelled dataset deals with data scaling problem.

One of the goals of unsupervised learning is to model the distribution of a given dataset $x \sim D$, where task is to model data distribution $p(x)$ using samples x from distribution such that $p(x) \sim p_D$. Machine learning models that can model $p(x)$ are called generative models. If we have access to such model then using it we can find the probability of an arbitary data point x or generate new points by sampling points from distribution, $x \sim p(x)$. These generative methods are used in: 1. Mapping one domain into another (ex. Text to Images, Text to Speech etc.), 2. Representation Learning, 3. Density Estimation 4. AI Art (ex. generating images, music etc.) etc.

Recently these unsupervised generative methods have shown success in many area of artificial intelligence. In NLP after the introduction of Transformers network [14] and introduction of models like GPT [10], BERT [3] and their variants has revolutionized the field. Similarly, in computer vision after introduction of GAN's [4], VAE [6] and their variants with recent introduction of diffusion models [5]. Recently we have also seen generative method ideas to map one domain into another domain. In speech we have models like wav2vec [2] for speech to text conversion and recent method Vall-E [15] for text to speech conversion. In text to image domain with introduction of methods like CLIP [9], GLIDE [7] we have seen advancement in creating images using just text prompt of image.

In this report we will only focus on deep latent variable models (DLVM) which is one of the methods in generative modelling. It computes the dataset's exact or approximate distribution functions using a hidden vari-

able called latent variable. These methods uses idea from Bayes theorem to generate data which require a posterior distribution which can be intractable for data of higher dimension. Variational inference (VI) and Markov chain Monte Carlo (MCMC) [11] are two practical tools to approximate intractable distributions. VI is dominantly used which approximate posterior distribution by using a tractable distribution and tuning its parameter so it becomes close to posterior distribution. This introduced Amortized VI which replace datapoint wise optimization with an encoder that map observation into latent variables this framework with DLVM is called variational autoencoder. Compared to VI, MCMC (e.g., Langevin dynamics) has high approximation ability and can approximate complex distributions by repeating sampling from a Markov chain with stationary distribution same as target distribution we want to approxiamte. MCMC in context with DLVM has not been explored because it takes long time to converge and running multiple MCMC interation is time consuming. This report study Langevin Autoencoder that uses amortized Langevin dynamics (ALD) which replaces datapoint wise MCMC iteration with udates of an encoder that map observation to a latent variable.

We provide a survey of existing literature in Section 2. Our proposal for the project is described in Section 3. We give details on experiments in Section 5. A description of future work is given in Section 8. We conclude with a short summary and pointers to forthcoming work in Section 7.

2 Literature Survey

2.1 Latent Variable Models

Given dataset $x \sim D$, our aim is to model distribution $p(x)$. The idea of the latent variable model is instead of modeling $p(x)$ directly, an unobserved/hidden variable called latent variable is used. Using this latent variable a conditional distribution $p(x|z)$ is defined for data also known as likelihood. We also make a assumption that given any data it can be represented using simple and lower dimensional representations ex. an image can be represented as colours, objects etc it it. Using latent variables we hope to find those lower dimension hidden representations of data. The process of finding such latent variables $x \mapsto z$ is called inference and process of mapping these latent variables back to data $z \mapsto x$ is called generation. These latent variables z can be continuous or discrete based on the models. Ex. VAE, LAE uses z as continuous while models like Vector Quantized VAE (VQ-VAE)[13] assume z as discrete. We also introduce a prior distribution $p(z)$ over the latent variables, and compute the joint distribution over observed and latent variables as $p(x, z)$. Using Bayes Rule we can connect inference and generation step as,

$$p(z|x) = \frac{p(x|z)p(z)}{p(x)} = \frac{p(x,z)}{\int p(x,z)dz}$$

2.2 Training Latent Variable Models

Training of model is done using Maximum Likelihood estimation.

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} \; -\sum_{i=1}^{m} \log p_\theta(x^i)$$

This is a standard optimization problem and it can be solved using gradient descent based algorithms. To apply gradient descent we need to calculate gradients of above objective function with respect to model pa-

rameters. Gradient computation gradient for a single data point x.

$$\begin{aligned}
\nabla_\theta \log p_\theta(x) &= \frac{\nabla_\theta p_\theta(x)}{p_\theta(x)} = \frac{\int \nabla_\theta p_\theta(x,z)dz}{p_\theta(x)} \\
\nabla_\theta \log p_\theta(x) &= \frac{\int p_\theta(x,z)\nabla_\theta \log p_\theta(x,z)dz}{p_\theta(x)} \\
\nabla_\theta \log p_\theta(x) &= \frac{\nabla_\theta p_\theta(x)}{p_\theta(x)} \\
\nabla_\theta \log p_\theta(x) &= \int p_\theta(z|x)\nabla_\theta \log p_\theta(x,z)dz
\end{aligned} \tag{1}$$

using, $\nabla_\theta \log p_\theta(x) = \frac{\nabla_\theta p_\theta(x)}{p_\theta(x)}$ from (1), we can see that, we need to compute posterior $p(z|x)$ in order to compute gradient of objective. This posterior calculation is intractable due to the lack of an analytic solution to the integral, therefore instead of calculating posterior distribution we try to approximate the it using VI or MCMC. This report focuses on MCMC based method using Langevin dynamics.

2.3 Langevin Dynamics

Langevin Monte Carlo is a class of Markov Chain Monte Carlo (MCMC) algorithm that generate samples from a probability distribution of interest by simulating the Langevin Equation, which is inspired from physics.

$$dz = -\nabla_z U(x,z;\theta)dt + \sqrt{2\beta^{-1}}dB \tag{2}$$

where, U is some potential/energy function in ML terminology it is a loss function and $\sqrt{2\beta^{-1}}dB$ is noise term where, B is brownian motion. The above equation (2) is a stochastic differential equation (SDE) and has $p^\beta(z|x;\theta) \propto exp(-\beta U(x,z;\theta))$ as its stationary distribution (SD). The idea of using Langevin dynamics is, if we somehow make target posterior distribution same as SD of (2) then we can sample points from posterior by just simulating the equation (2). By setting $\beta = 1$ we have, $p(z|x;\theta) \propto exp(-U(x,z;\theta))$ as,

$$exp(-U(x,z;\theta)) = p(z|x;\theta)p(x)$$

Therefore, by setting $U(x,z;\theta) = -\log p(z|x;\theta)p(x)$ we can obtain posterior $p(z|x;\theta)$ as stationary distribution.

$$U(x,z;\theta) = -\log p(x,z;\theta)$$

We are now interested in simulating the following SDE to sample from its steady state distribution $p(z|x;\theta)$

$$dz = -\nabla_z \log p(x,z;\theta)dt + \sqrt{2}dB$$

We can obtain samples from the posterior by simulating SDE using the Euler–Maruyama method.

$$z_{t+\eta} - z_t = -\eta\nabla_z \log p(x,z;\theta) + \sqrt{2}(B_{t+\eta} - B_t)$$

here $(B_{t+\eta} - B_t)$ becomes, $(B_{t+\eta} - B_t) \sim \mathcal{N}(0,\eta)$

$$\begin{aligned}
z_{t+\eta} &= z_t - \eta\nabla_z \log p(x,z;\theta) + \sqrt{2}(B_{t+\eta} - B_t) \\
z_{t+1} &\sim q(z_{t+1}|z_t) \\
q(z'|z) &= \mathcal{N}(z - \eta\nabla_z U(x,z;\theta), 2\eta I)
\end{aligned} \tag{3}$$

2.4 Estimating Gradients

After obtaining samples from posterior distribution the gradient $\nabla_\theta \log p_\theta(x, z)$ is approximated as

$$\int p_\theta(z|x)\nabla_\theta \log p_\theta(x, z)dz = \mathbb{E}_{p_\theta(z|x)}[\nabla_\theta \log p_\theta(x, z)] \approx \frac{1}{T}\sum_{t=1}^{T} \nabla_\theta \log p_\theta(x, z)$$

hence, for N samples gradients are calculated as

$$\frac{1}{N}\sum_{i=1}^{N} \mathbb{E}_{p_\theta(z^i|x^i)}[\nabla_\theta \log p_\theta(x^i, z^i)] \approx \frac{1}{N}\sum_{i=1}^{N}\frac{1}{T}\sum_{t=1}^{T} \nabla_\theta \log p_\theta(x^i, z^i) \tag{4}$$

Traditional MCMC starts with random initialization after updating model parameter θ, also it is required to run again from scratch for new data in test time, which hinders its practical use for learning DLVMs. To handle this inefficiency Hoffman has proposed to use a encoder $q(z|x)$ to initialize the datapoint-wise MCMC which can be defined as a Gaussian distribution similar to VAE:

$$q(z|x) = \mathcal{N}(z; \mu(x), diag(\sigma^2(x))) \tag{5}$$

where, μ and σ^2 are mappings from observation space to latent space. In training LD iterations of eq (3) are initialized using sample from distribution of eq (5), and model parameters are updated using stochastic gradient defined in eq (4). The encoder is trained similar to VAE where we maximize the evidence lower bound (ELBO):

$$L(\phi) = E_{q(z|x)\hat{p}(x)}\left[\log \frac{p(x, z)}{q(z|x)}\right]$$

This method of initilization LD with encoder speed up convergence but still this method require datapoint-wise MCMC iterations also encoder has to trained using different objective which makes it more complex.

3 Methods and Approaches

3.1 Amortized Langevin Dynamics

Instead of directly simulating langevin dynamics equation (2), an deterministic encoder $fz|x$ is defined that maps the observation into latent variable and its parameter ϕ are considered to follow eq. (2) as

$$\begin{aligned} d\phi &= -\nabla_\phi V(\phi)dt + \sqrt{2}dB \\ V(\phi) &= \sum_{i=1}^{n} U(x^{(i)}, f_{z|x}(x^{(i)}); \theta) \end{aligned} \tag{6}$$

Then applying Euler-Maruyama method to simulate (6) with discretization:

$$\begin{aligned} \phi_{t+1} &\sim q(\phi_{t+1}|\phi_t) \\ q(\phi'|\phi) &= \mathcal{N}(\phi'; \phi - \eta\nabla_\phi V(\phi), 2\eta I) \end{aligned} \tag{7}$$

additionally metropolis hastings (MH) rejection step can be used to remove discretization error as follow:

$$\alpha_t = \min\{1, \frac{\exp(-V(\phi'))q(\phi|\phi')}{\exp(-V(\phi))q(\phi'|\phi)}\}$$

Algorithm 1 describes the ALD procedure.

Algorithm 1 Amortized Langevin Dynamics

$\phi \leftarrow$ Initialize parameters
$Z(1), \ldots, Z(n) \leftarrow \emptyset$ ▷ Initialize sample sets for all n datapoints
repeat
  $\phi' \sim q(\phi'|\phi) := N(\phi'; \phi - \eta \sum_{i=1}^{n} \nabla_\phi U(x(i), z(i) = f_{z|x}(x^(i); \phi)), 2\eta I)$
  $\phi \leftarrow \phi'$ with probability $\min\{1, \frac{\exp(-V(\phi'))q(\phi|\phi')}{\exp(-V(\phi))q(\phi'|\phi)}\}$ ▷ MH rejection step.
  $Z(1), \ldots, Z(n) \leftarrow Z(1) \cup \{f_{z|x}(x(1); \phi)\}, \ldots, Z(n) \cup \{f_{z|x}(x(n); \phi)\}$ ▷ Add samples
until convergence of parameters
return $Z(1), \ldots, Z(n)$

3.2 Langevin Autoencoder

Based on ALD novel framework for learning latent variable model is proposed and is called Langevin Autoencoder(LAE). Algorithm 2 gives summary of LAE training procedure. We have a two functions, encoder ($f_{z|x}(x; \phi, \psi)$ and decoder ($f_{x|z}(z; \theta)$). These encoder and decoder functions can be defined using a neural network, such that encoder is defined as $f_{z|x}(x; \phi, \psi = \phi(g(x; \psi))$ and decoder $f_{x|z}(z; \theta) = g'(z; \theta)$. while training for each epoch data is passed to encoder and final layer of encoder ϕ is updated using ALD for some number of steps T and gradient is calculated using time average of V_t. These gradients will be used to update model parameters θ, ψ.

Algorithm 2 Langevin Autoencoders

$\theta, \Phi, \psi \leftarrow$ Initialize parameters
repeat
  $x^{(1)}, ..., x^{(n)} \sim \hat{p}(x)$ ▷ Sample a minibatch of n examples from the training data
  for $t = 0, ..., T-1$ **do** ▷ Run ALD iterations
    $V_t \leftarrow -\sum_{i=1}^{n} \log p(x^{(i)}, z(i) = \Phi g(x^{(i)}; \psi); \theta)$
    $\Phi' \sim q(\Phi'|\Phi) := \mathcal{N}(\Phi'; \Phi - \eta \nabla_\Phi V_t, 2\eta I)$
    $V_t' \leftarrow -\sum_{i=1}^{n} \log p(x^{(i)}, z^{(i)} = \Phi' g(x^{(i)}; \psi); \theta)$
    $\Phi \leftarrow \Phi'$ with probability $\min\{1, \frac{\exp(-V(\phi'))q(\phi|\phi')}{\exp(-V(\phi))q(\phi'|\phi)}\}$ ▷ MH acceptance step
  $V_T \leftarrow -\sum_{i=1}^{n} \log p(x^{(i)}, z^{(i)} = \Phi g(x^{(i)}; \psi); \theta)$
  $\theta \leftarrow \theta - \eta \nabla \theta \frac{1}{T} \sum_{t=1}^{T} V_t$ ▷ Update the decoder
  $\psi \leftarrow \psi - \eta \nabla_\psi \frac{1}{T} \sum_{t=1}^{T} V_t$ ▷ Update the encoder
until convergence of parameters
return θ, Φ, ψ

3.3 Training models with DLVM

A new training method to train models was proposed by team which uses DLVM's in training procedure. As encoder of DLVM $g_{x|z}$ has learned to map observation into latent space and provides distribution of given observation. Using this distribution some number of points N can be samples that can generate similiar class

images using decoder $h_{z|x}$ which are then added to training batch. Fig 1 shows the described training procedure and Algorithm 3 summarizes the described training procedure. This procedure can be used with small training sample dataset or class imbalanced dataset where new similar samples are generated by DLVM for training. The generated images although are similar but has some reconstruction noise that can also help in increasing adversarial robustness of model f_x. This reconstruction noise also add regularization effect while training model.

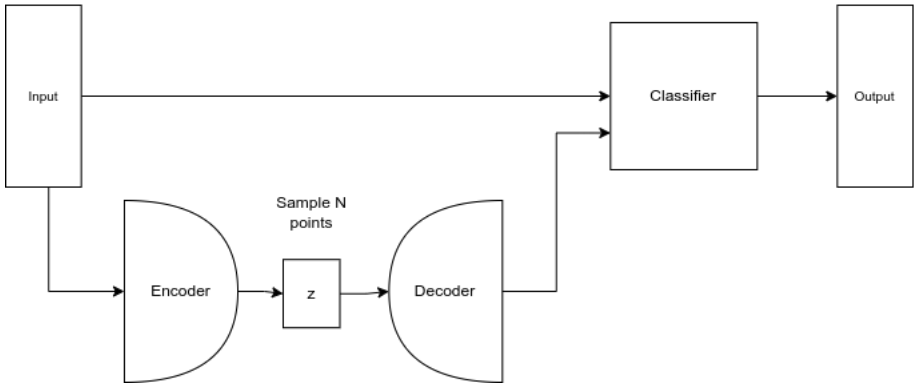


Figure 1: Proposed training method to train models incorporated with data from deep latent variable models.

Algorithm 3 Training Model with DLVMs

$g_\psi, h_\phi \leftarrow$ Load DLVM encoder and decoder
$f_\theta \leftarrow$ Instantiate model to train
$l \leftarrow$ Define loss function to train
$\mathcal{B} \leftarrow [\ ]$ ▷ Create a empty array to store samples
repeat
  $x^{(1)}, ..., x^{(n)} \sim \hat{p}(x)$ ▷ Sample a minibatch of n examples from the training data
  $\mathcal{D} \leftarrow g(x)$ ▷ Get distribution of training sample
  $\mathcal{B}[0] \leftarrow x^{(i)}$ ▷ Add training batch to batch array
  for $n = 1, ..., N$ **do**
    $z_n \sim \mathcal{D}$ ▷ Sample N points from $\mathcal{D}$
    $q_n = h(z_n)$ ▷ Generate sample using decoder
    $\mathcal{B}[n] \leftarrow q_n$ ▷ Add generated samples to training batch
  $\hat{y} = f(B)$ ▷ Make prediction
  $v_t = l(\hat{y}, y)$ ▷ Calculate loss
  $\theta \leftarrow \theta - \eta \nabla_\theta v_t$ ▷ Update model
until convergence of parameters
return θ

3.4 Work done before mid-term project review

Reading and understanding assigned paper along with overview of generative models and relevant referenced materials. Implementation of paper was done using PyTorch[8] and Torch distributions package [1]. Using the code implemented benchmarking was done between VAE and LAE on their capability to reconstruct given

input for MNIST dataset. To calculate reconstruction capability mean square error(MSE) was used as metric. Smaller value of MSE score denote better reconstruction of input.

3.5 Work done after mid-term project review

We trained LAE and VAE for longer epochs on MNIST dataset and start with generating some images by random sampling latent variable from Gaussian noise and passing it to decoder. Result of images generated are shown in section (6.1). Latent space learned by both VAE and LAE were analysed and results are shown in section (6.2). Effect of number of steps T and MH on input reconstruction, latent space and representation learning capability of LAE and effect of size of latent space dimension of LAE and VAE on input reconstruction and representation learning capability was analysed and results are shown in section (6.1). After these experiments a new training method was proposed which incorporate DLVMs while training a image classifier. In this method input batch for training classifier is passed into encoder of DLVMs and their latent space distribution is extracted using which new samples are drawn and added to training batch, this batch is then passed to classifier for training, section (3.3) describes this procedure in detail. This training procedure can be used to deal with adversarial examples during test time. The adversarial robustness of the proposed training method was analyzed by using a white box attack method called projected gradient descent(PGD) which generates adversarial samples and accuracy was used as metric to measure adversarial robustness. These results are shown in section (6.5). Finally, results on latent space interpolation are shown in section (6.6).

4 Data set Details

The MNIST (Modified National Institute of Standards and Technology) is a large database of handwritten digits $\{0, 1, ...9\}$. This dataset contains $60,000$ training samples and $10,000$ test samples. This dataset is subset of largers set made available by NIST. Each image in dataset have been size-normalized and centered in a fixed-size image. Each image is a grayscale image of size 28×28. This dataset is widely used for benchmarking in the field of machine learning.

5 Experiments

In this section we perform experiments to (1) evaluate the generation capability of LAE and compare it to VAE (2) compare the latent space of VAE and LAE (3) evaluate representation learning capability of LAE and compare with VAE (4) evaluate the effectiveness of our proposed training method, (5) evaluate whether our proposed training method is robust towards adversarial attacks, and (6) generation of samples from interpolation of latent space. Our experiments are primarily conducted on linear encoder and decoder using the MNIST datasets, and using the Pytorch, torch distributions framework. The linear encoder consist of 3 blocks with each block has 1 linear layer $\rightarrow$ layer norm $\rightarrow$ ReLU followed by a linear layer that maps output from last encoder block to latent dimension. Similarly linear decoder has also consist of 3 blocks with each block has 1 linear layer $\rightarrow$ layer norm $\rightarrow$ ReLU followed by a linear layer that maps output from last decoder block to input dimension. A detailed architecture of model is shown in Figure (2). This specific choice of model is choosen similar to author as its small and easy to do computation and experiments. Due to the choice of model and dataset we do not need high computational power. All experiments are done on machine with NVIDIA Geforce 940MX with 2GB VRAM and 12 GB RAM. For longer run Kaggle environment was used that provide NVIDIA Tesla P100 with 16gb VRAM and 13 GB of RAM. VAE and LAE are trained using Binary Cross-Entropy (BCE) with batch size of 512 for 1000 epochs with early stopping that track validation loss. Other parameters are varied in order to conduct different experiments, latent dimension is choosen

from $[2, 8, 16]$ for both LAE and VAE. For LAE number of steps T were choosen from $[2, 10, 50]$ and 2 models with latent dimension of 2 were trained with MH step enabled on one of the model. For experiment (4) and (5) simple classifier with 2 convlutional layer + maxpooling and 2 linear layers with dropout was used. It is trained for 10 epochs with negative log likelihood loss (NLL) calculated over 128 batches. All the above models are trained using Adam optimizer with initial learning rate of 10^{-4}

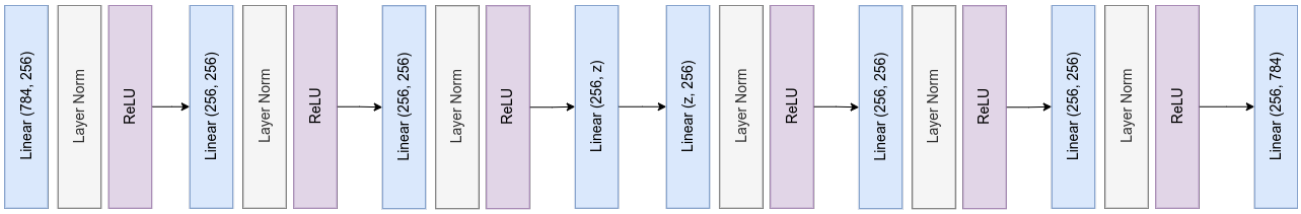


Figure 2: Architecture of Linear model for VAE and LAE, here z denotes latent dimension. In VAE the linear layer that maps encoded input to latent space is copied twice to get μ and σ^2

6 Results

6.1 Generation capability of LAE

In this section we show result of generation capability of LAE and its comparision with VAE. We used MSE as reconstruction loss so lower values indicate better results. We use LAE with number of steps $T = 2$ and no MH with step size of 0.001. It can be observed in table (1) that increasing latent dimension lowers the reconstruction loss in LAE and VAE as latent dimension increases models can encode more information from data this supports the observations we get. We can also observed LAE is consistently outperforms VAE. From table (2) we can observe, increasing number of steps in langevin step reduces the reconstruction loss at first but later it decreases, reason behind this behaviour is not clear and need more investigation. From table (3), when using MH step performance of LAE decreases by very little factor this is because we are rejecting some potential samples during MH step.

	VAE $\mid z = 2$	**LAE** $\mid z = 2$	**VAE** $\mid z = 8$	**LAE** $\mid z = 8$	**VAE** $\mid z = 16$	**LAE** $\mid z = 16$
MSE	0.00000732256	0.00000743449	0.00000317134	0.00000310958	0.00000257368	0.00000167048
BCE	141.74278	136.040208	103.07547	85.917400	98.79791	68.15598

Table 1: Comparision of Reconstruction loss (MSE) and Validation loss (BCE) of VAE and LAE

	LAE $\mid T = 2$	**LAE** $\mid T = 10$	**LAE** $\mid T = 50$
MSE	0.00000743449	0.00000716265	0.00000729323
BCE	136.040208	132.89959	134.36413

Table 2: Comparision of Reconstruction loss (MSE) and Validation loss (BCE) of LAE with change in number of steps T

	LAE | MH = False	**LAE | MH = True**
MSE	0.00000743449	0.00000748080
BCE	136.040208	136.238208

Table 3: Comparision of Reconstruction loss (MSE) and Validation loss (BCE) of LAE with and without MH

(a) LAE (b) VAE

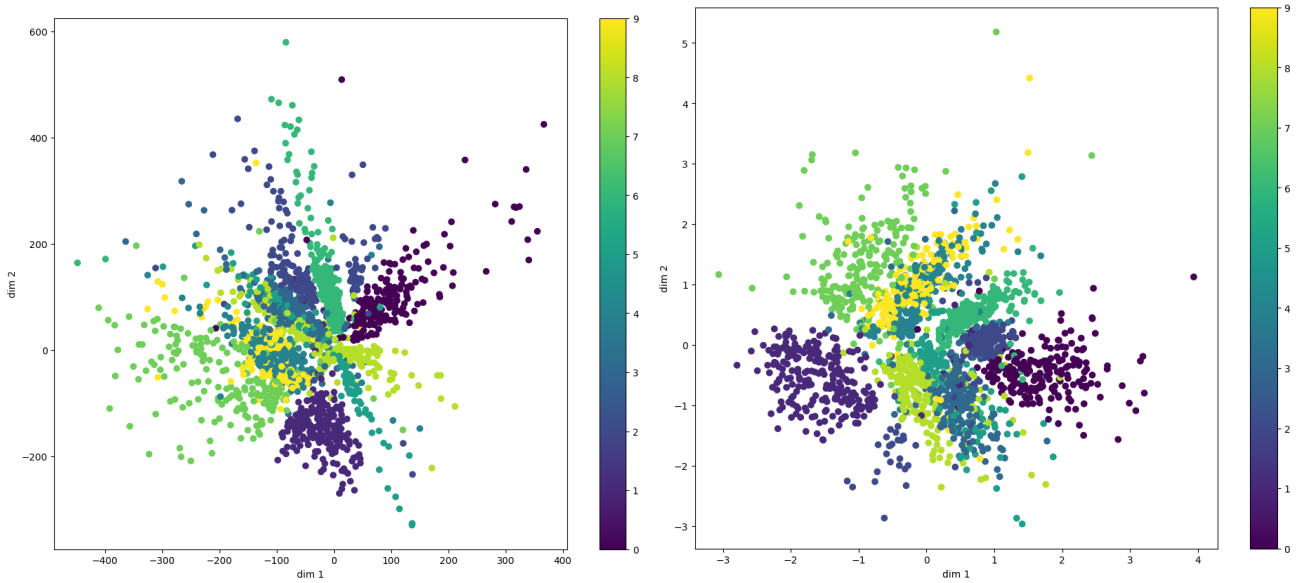
Figure 3: Image generation by random sampling from normal distribution

6.2 Comparision of latent space learned by LAE and VAE

Fig (4) shows 2 dimensional latent space plot of VAE and LAE. Both models clusters samples into multiple classes in encoded space. From the plot we can observe points related to digits 9 and 4 overlaps the most this might be due to similarity in digit 4 and 9 when drawn with hand. VAE latent space is more constrained than LAE. In VAE points are more clustered towards mean while in LAE they are much scattered. This is because VAE add additional penalty to loss function that makes its latent space closer to standard normal distribution $\mathcal{N}(0, 1)$. This shows that VAE has more control over the latent space than LAE although LAE can map distribution more precisely than VAE.

6.3 Compare representation learning capability of LAE and VAE

For this experiment we used latent space representations of samples from encoder of LAE and VAE and pass them to ReLU + Linear layer. Note that our goal here is not to provide the state-of-the-art results on classification benchmarks but to compare the representations learned by VAE and LAE in encoded space for classification task. As from fig (4) we can observe many digits in latent space overlaps and close to other classes, also latent space is clearly not linearly seperable because of this accuracies in this experiments are low. From



(a) LAE (b) VAE

Figure 4: Learned 2D Latent space of LAE and VAE

table (4) we can observe increasing the latent space dimension improves the representations in latent space. Table (5) shows that increasing langevin steps at first helps in representations learned but then it decreases. Reason behind this behaviour is not clear and need more investigation. Table (6) shows using MH improves representations in latent space.

	VAE $\mid z = 2$	**LAE** $\mid z = 2$	**VAE** $\mid z = 8$	**LAE** $\mid z = 8$	**VAE** $\mid z = 16$	**LAE** $\mid z = 16$
Train Loss	0.001352	0.00173053	0.00074552	0.00085344	0.00082766	0.00149354
Train Acc	0.0005407	0.0003460	0.00071939	0.00078394	0.00069669	0.00074028
Val Loss	0.001338	0.00166865	0.00074213	0.00061837	0.00080636	0.00086874
Val Acc	0.0005586	0.00037973	0.0007355	0.00080696	0.00072118	0.00081385

Table 4: Comparision of latent space representation of samples from LAE and VAE for classification

6.4 Evaluating effectiveness of our proposed training method

In this experiment we use training procedure described by Algorithm (3) and train a 2 layer convolutional network and benchmark effect of increasing number of points sampled N from latent space. Note that we train these models for only 10 epochs although these models were still converging but due to resource constraint these models cannot be trained till convergence. The goal here is to show that adding samples from latent distribution do not decrease the model performance by large factor and from table (7) we can see that's not the case. Adding samples from latent distribution is similar to data augmentation step which act as regularizer and add some noise to dataset which make model more robust towards small changes in inputs but this strategy needs more number of epochs to converge.

	LAE | $T = 2$	**LAE** | $T = 10$	**LAE** | $T = 50$
Train Loss	0.00173053	0.00183926	0.00186372
Train Accuracy	0.0003460	0.000370808	0.000375255
Val Loss	0.00166865	0.001644017	0.001683733
Val Accuracy	0.00037973	0.00043382	0.00045005

Table 5: Comparision of latent space representation of samples from LAE with change in number of steps T for classification

	LAE | MH = False	**LAE | MH = True**
Train Loss	0.00173053	0.00150817
Train Accuracy	0.0003460	0.00048625
Val Loss	0.00166865	0.00147085
Val Accuracy	0.00037973	0.00051563

Table 6: Comparision of latent space representation of samples from LAE with and without MH for classification

6.5 Adversarial Robustness of proposed training method

From table (8) and (9) we can observe that for higher value of ϵ proposed training method is more robust towards adversarial attacks. The validation loss is lower for proposed method with higher ϵ value, this shows that proposed training procedure was more robust as it gives low probability to samples which the model could not recognize hence decreases the validation loss.

6.6 Interpolation of Latent space of VAE and LAE

Fig (5) shows interpolation of latent space in VAE and LAE respectively. From these figures we can observe VAE has better interpolation results compare to LAE this is because of the latent space structure of LAE which is scattered. In VAE we have better control over the distribution we are learning and its space is constrained in $\mathcal{N}(0, 1)$ because of this interpolation is smooth in VAE compared to LAE.

7 Conclusion

This paper has proposed amortized langevin dynamics(ALD) which is an efficient MCMC method for training deep latent variable models(DLVMs). ALD uses a encoder that predict latent variable given observation and amortize the cost of datapoint-wise sampling. Using ALD Langevin Autoencoder(LAE) were proposed that updates encoder last layer using ALD procedure. From the experiments performed we demonstrated that LAE was able to learn latent space distribution for dataset and able to reconstruct input better than VAE in many cases. Although, from the experiments LAE was better than VAE, this factor is not too significant. The Latent space learned by VAE have much better structure and control than LAE. A novel method inco-

	None $\mid N = 0$	**LAE** $\mid N = 2$	**VAE** $\mid N = 2$	**LAE** $\mid N = 5$	**VAE** $\mid N = 5$
Val Loss	0.001153	0.001747	0.001770	0.002130	0.002134
Val Accuracy(%)	95.66	94.21	93.99	92.91	92.61

Table 7: Comparision of Validation loss and accuracy of proposed training procedure.

	None $\mid N = 0$	**LAE** $\mid N = 2$	**VAE** $\mid N = 2$	**LAE** $\mid N = 5$	**VAE** $\mid N = 5$
Accuracy(%)	95.66	94.21	93.99	92.91	92.61
$\epsilon = 0.01$	94.69	93.03	92.66	91.31	90.83
$\epsilon = 0.1$	70.85	66.42	63.40	62.62	60.89
$\epsilon = 0.2$	13.18	11.31	11.16	10.60	10.32
$\epsilon = 0.3$	0.16	0.33	0.35	0.41	0.41

Table 8: Accuracy comparision of PGD attack on models trained using proposed training procedure

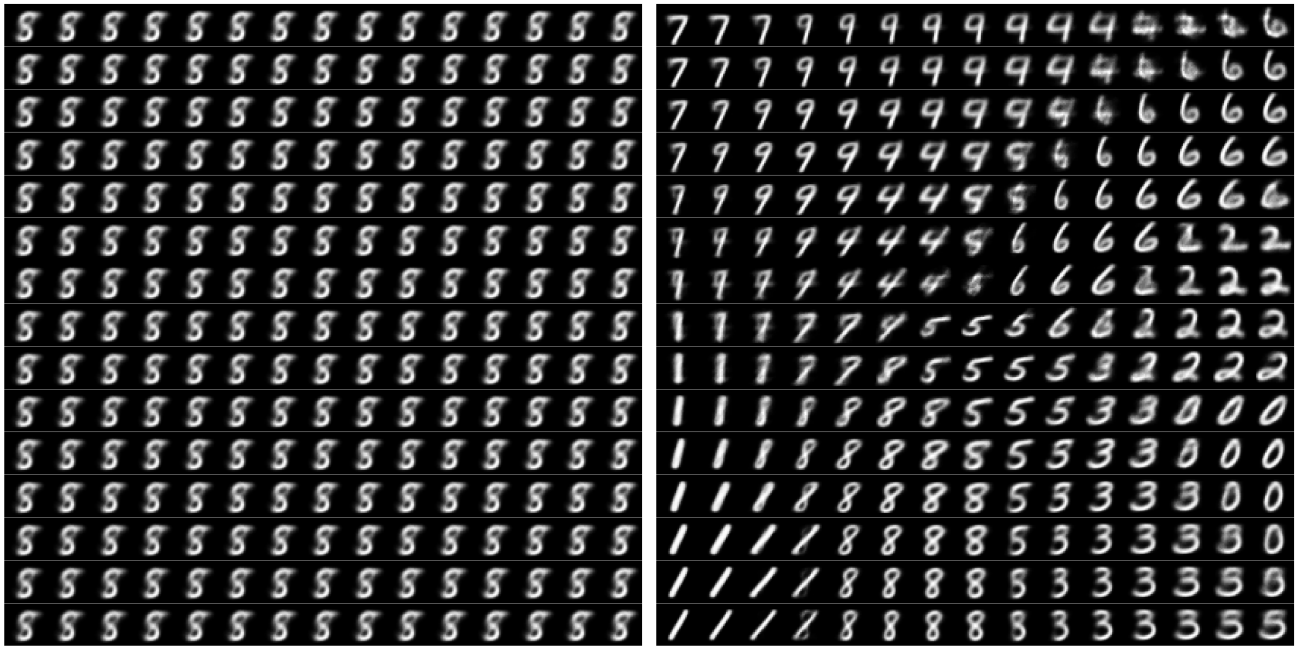
prating DLVM in training image classifier was proposed and it has shoen shown positive results on adversarial robustness of trained models.

8 Future Work

The adversarial robustness of proposed training method can be further analysed for larger and real world datasets. This training procedure also has potential to work with class imbalanced datasets and datasets with small number of training examples which can be analysed.

	None $\mid N = 0$	**LAE** $\mid N = 2$	**VAE** $\mid N = 2$	**LAE** $\mid N = 5$	**VAE** $\mid N = 5$
Val Loss	0.001153	0.001747	0.001770	0.002130	0.002134
$\epsilon = 0.01$	0.1702	0.2523	0.2630	0.3044	0.3114
$\epsilon = 0.1$	0.8187	0.9044	0.9968	0.9958	1.0703
$\epsilon = 0.2$	3.2136	2.9268	3.2381	3.0319	3.3443
$\epsilon = 0.3$	6.6067	5.5999	6.1211	5.6184	6.1619

Table 9: Validation loss comparision of PGD attack on models trained using proposed training procedure



(a) LAE (b) VAE

Figure 5: Interpolation of latent space between points $[-1, -1] \rightarrow [1, -1]$ horizontally and $[1, -1] \rightarrow [1, 1]$ vertically

References

[1] Pytorch distributions, 2023.

[2] Alexei Baevski, Henry Zhou, Abdelrahman Mohamed, and Michael Auli. wav2vec 2.0: A framework for self-supervised learning of speech representations, 2020.

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. 2019.

[4] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[5] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models, 2020.

[6] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2022.

[7] Alex Nichol, Prafulla Dhariwal, Aditya Ramesh, Pranav Shyam, Pamela Mishkin, Bob McGrew, Ilya Sutskever, and Mark Chen. Glide: Towards photorealistic image generation and editing with text-guided diffusion models, 2022.

[8] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019.

[9] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision, 2021.

[10] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

[11] Joshua S. Speagle. A conceptual introduction to markov chain monte carlo methods, 2020.

[12] Shohei Taniguchi, Yusuke Iwasawa, Wataru Kumagai, and Yutaka Matsuo. Langevin autoencoders for learning deep latent variable models, 2022.

[13] Aaron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural discrete representation learning, 2018.

[14] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[15] Chengyi Wang, Sanyuan Chen, Yu Wu, Ziqiang Zhang, Long Zhou, Shujie Liu, Zhuo Chen, Yanqing Liu, Huaming Wang, Jinyu Li, Lei He, Sheng Zhao, and Furu Wei. Neural codec language models are zero-shot text to speech synthesizers, 2023.