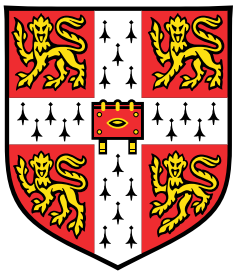
Manifold Hamiltonian Dynamics for Variational Auto-Encoders



Yuanzhao Zhang

Department of Engineering
University of Cambridge

This dissertation is submitted for the degree of
Master of Philosophy

Darwin College August 2018

I would like to dedicate this thesis to my loving parents ...

Declaration

I, Yuanzhao Zhang of Darwin College, being a candidate for the MPhil in Machine Learning, Speech and Language Technology, hereby declare that this report and the work described in it are my own work, unaided except as may be specified below, and that the report does not contain material that has already been used to any substantial extent for a comparable purpose.

Total word count: 9271

Yuanzhao Zhang
August 2018

Acknowledgements

I would like to acknowledge my supervisor Dr. Yichuan Zhang, who have been extremely supportive throughout the project. He is very knowledgeable in this domain and has provided me with clear and useful ideas. I am really grateful for his patience with me and I wouldn't have finished this project without him.

I would also like to thank Prof. Bill Byrne for his help and advice during my early stage at Cambridge. It has been a great pleasure studying at one of the best engineering departments in the world.

Last but not least, I would like to thank my beloved parents and my girlfriend Ting who have always been there for me. My gratitude also goes to my friends and classmates at Cambridge. You are the ones who make my experience at this lovely town unforgettable.

Abstract

Variational Auto-Encoder (VAE) has been a very successful generative model that is able to compress high-dimension data into low-dimension latent space, as well as to generalize to domain outside the training data and generate unseen new data. What is different about VAE from the vanilla auto-encoder, is that VAE can model a continuous latent space using a Gaussian distribution, whereas auto-encoder can only model a discrete latent space. This enables us to sample from the latent space and generate variations different from the original input.

However, VAE also has its limitations. One of the most significant one is that we assume the latent distribution is a Gaussian, which is a over-simplified assumption because in reality the true data distribution can be much more complicated. Since it was first introduced, researchers have been trying to improve the vanilla VAE and we have seen some promising results in the literature. In this dissertation, we try to address this issue by taking a specific approach, which is to combine variational inference with Markov Chain Monte Carlo methods. We examine the performance of Hamiltonian variational inference model by conducting extensive experiments, and come up with a novel idea of Riemannian manifold variational inference.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

1.1 Motivation

Approximating an intractable posterior distribution is one of the core problems in Bayesian inference. With Bayes's rule:

$$p(z|x) = \frac{p(z)p(x|z)}{p(x)} = \frac{p(z)p(x|z)}{\int p(z)p(x|z)dz} \tag{1.1}$$

we have a simple recipe for the posterior. However, the integration part in the denominator is often intractable for practical problems. There are two major approaches to solve this problem: variational inference and Markov chain Monte Carlo (MCMC) methods. Both methods have their pros and cons.

Variational Auto-Encoder (VAE) [8] is a powerful generative model which utilizes variational inference to learn the distribution of the given data. VAE has gained a lot of research interests given its nice interpretability and overall good performance to model the data distribution and generate new data. However, VAEs also suffer from several drawbacks, one of them being the over-simplified posterior approximation. The vanilla VAE uses a multivariate Gaussian with a diagonal covariance to approximate the true posterior, which has a significant impact on the quality of inferences made using variational methods.

On the other hand, a popular alternative to variational inference is Markov chain Monte Carlo (MCMC) method. Samples of the desired distribution can be obtained by running a Markov chain for a number of steps. The more steps there are, the closer we can approach the equilibrium distribution.

1.2 Contribution

The central idea of the project is to combine variational inference with MCMC methods to obtain the best of both worlds. More specifically, we aim to first fully examine the performance of Hamiltonian variational inference (HVI), which was presented in paper [16], then we try to come up with a novel idea of Riemannian Manifold Hamiltonian variational inference (RMHVI) based on the principle of HVI. It incorporates Riemannian Manifold Hamiltonian Monte Carlo (RMHMC) [3], which is a Hamiltonian Monte Carlo sampling method defined on the Riemannian manifold and it is designed to resolve the shortcomings of existing Monte Carlo algorithms when sampling from target densities that may be high dimensional and exhibit strong correlations.

We augment the inference networks (both fully-connected and convolutional networks) in vanilla Variational Auto-Encoders (VAE) with HVI and test the model on different datasets to prove the effectiveness of combining variational inference with MCMC methods for image generation tasks. We also implement RMHVI with VAEs by using fully-connected neural network configuration and compare its performance with HVI.

1.3 Dissertation organization

The remaining chapters are organized as follows: chapter 2 provides the necessary background information on Variational Auto-Encoders, Hamiltonian Monte Carlo and Riemannian Manifold Hamiltonian Monte Carlo. Chapter 3 introduces several related work in the current literature. Chapter 4 presents the detailed algorithms and principles behind HVI and RMHVI. Chapter 5 gives the experiment details including datasets, neural network configurations, experiment results and analysis. Finally chapter 6 draws a conclusion of our work and provide possible methodologies for future work.

Chapter 2

Background

2.1 Variational Auto-Encoder

2.1.1 Problem scenario

Given a dataset $\boldsymbol{X} = \{\boldsymbol{x}^{(i)}\}_{i=1}^{N}$ consisting of N i.i.d samples of variable $\boldsymbol{x}$, if we assume the data are generated in two steps: first a latent variable $\boldsymbol{z}$ is generated from a prior $p_\theta(\boldsymbol{z})$; then a variable $\boldsymbol{x}$ is generated from $p_\theta(\boldsymbol{x}|\boldsymbol{z})$. We are then interested in the parameters θ and latent variable $\boldsymbol{z}$ which are usually hidden from us. Unfortunately, the posterior is usually intractable, algorithms like expectation–maximization (EM) cannot be directly used here.

2.1.2 Structure

A graphical representation of VAE is shown in Figure 2.1. Solid lines denote $p_\theta(\boldsymbol{z})$ and $p_\theta(\boldsymbol{x}|\boldsymbol{z})$, which are the generative section of VAE. Dashed lines denote $q_\phi(\boldsymbol{z}|\boldsymbol{x})$, which is the variational approximation to the true posterior. During training, the parameters θ and ϕ are jointly learned.

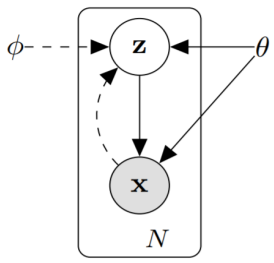


Fig. 2.1 Graphical representation of VAE model from [8].

2.1.3 The variational lower bound

The marginal likelihood for each individual datapoint $\boldsymbol{x}^{(i)}$ can be written as:

$$\log p_\theta(\boldsymbol{x}^{(i)}) = D_{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x}^{(i)})||p_\theta(\boldsymbol{z}|\boldsymbol{x}^{(i)})) + \mathcal{L}(\theta,\phi;\boldsymbol{x}^{(i)}) \tag{2.1}$$

where the first term is the KL divergence of our approximation from the true posterior, and the second term is called the variational lower bound. KL divergence measures how one distribution diverges from another one, and it is always non-negative. Thus we have:

$$\log p_\theta(\boldsymbol{x}^{(i)}) \geq \mathcal{L}(\theta,\phi;\boldsymbol{x}^{(i)}) = \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\big[-\log q_\phi(\boldsymbol{z}|\boldsymbol{x}) + \log p_\theta(\boldsymbol{x},\boldsymbol{z})\big] \tag{2.2}$$

if we rearrange the terms of the above expression, we get another expression of the lower bound:

$$\mathcal{L}(\theta,\phi;\boldsymbol{x}^{(i)}) = -D_{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x}^{(i)})||p_\theta(\boldsymbol{z})) + \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\big[\log p_\theta(\boldsymbol{x}^{(i)}|\boldsymbol{z})\big] \tag{2.3}$$

To optimize the lower bound we need to compute the gradient w.r.t. to parameter ϕ. However, if we use the Monte Carlo gradient estimator, the gradient w.r.t. ϕ can be a bit tricky because the variance can be very high because of sampling from $q_\phi(\boldsymbol{z}|\boldsymbol{x})$, and it needs to be controlled as in [10] and [15].

2.1.4 The SGVB estimator

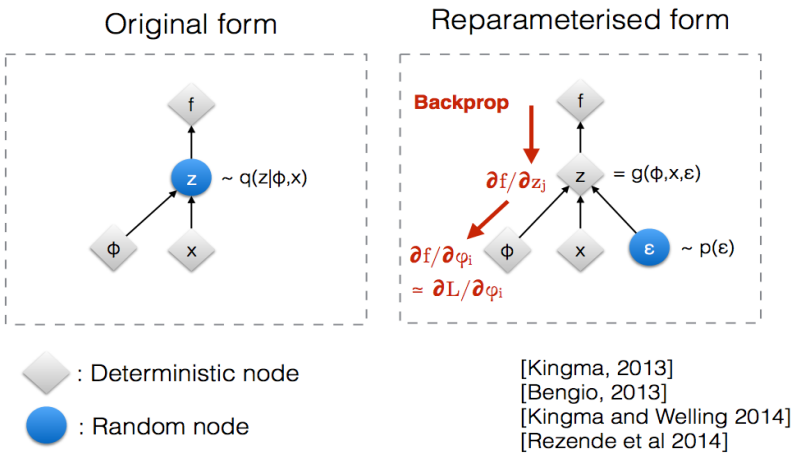


Fig. 2.2 Graphical explanation of why reparameterization trick introduces differentiability. Figure taken from [5].

To solve the above problem, we apply a technique called the reparameterization trick. This trick can make the Monte Carlo estimate of the expectation w.r.t. $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ differentiable w.r.t.

ϕ. Figure 2.2 illustrates the reason. In the original form, we can not apply backpropagation through the graph because $\boldsymbol{z}$ is derived from a random sampling process. However, if we introduce an additional parameter ε which is sampled from a fixed distribution, and then apply a deterministic transformation $\boldsymbol{z} = g_\phi(\boldsymbol{\varepsilon}, \boldsymbol{x})$, the whole process becomes differentiable because now we can backpropagate through deterministic nodes.

Using this technique, we can get the Stochastic Gradient Variational Bayes (SGVB) estimator of the lower bound:

$$\hat{\mathcal{L}}(\theta, \phi; \boldsymbol{x}) = -D_{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x}^{(i)})||p_\theta(\boldsymbol{z})) + \frac{1}{L}\sum_{l=1}^{L}(\log p_\theta(\boldsymbol{x}^{(i)}|\boldsymbol{z}^{(i,l)})) \tag{2.4}$$

where $\boldsymbol{z}^{(i,l)} = g_\phi(\boldsymbol{\varepsilon}^{(i,l)}, \boldsymbol{x}^{(i)})$ and $\boldsymbol{\varepsilon}^{(l)} \sim p(\boldsymbol{\varepsilon})$. If we look at equation 2.4, we can see that the first term acts as a regularizer which measures how closely the latent variables match a unit Gaussian, and the second term gives a reconstruction error.

2.2 Hamiltonian Monte Carlo

2.2.1 Hamiltonian dynamics

In physics, Hamiltonian dynamics explicitly defines the time evolution of the system by a set of Hamilton's equations:

$$\frac{d\theta}{dt} = \frac{\partial H}{\partial \boldsymbol{p}} \tag{2.5}$$

$$\frac{d\boldsymbol{p}}{dt} = -\frac{\partial H}{\partial \theta} \tag{2.6}$$

where θ is the *position* vector, $\boldsymbol{p}$ is the *momentum* vector and H is the *Hamiltonian*. For a closed system which does not allow energy transfer in or out of the system, it is the sum of the kinetic and potential energy. A simple example of such a system can be a frictionless ball that slides over a surface of varying height. The potential energy $U(\theta)$ is associated with the position/height of the ball, and the kinetic energy $K(\boldsymbol{p})$ is associated with the momentum of the ball.

2.2.2 Proprieties of Hamiltonian dynamics

- The total energy is preserved: $H(\theta(t), \boldsymbol{p}(t)) = H(\theta(0), \boldsymbol{p}(0))$

- The volume in $(\theta, \boldsymbol{p})$ space is preserved: $d\theta(t)d\boldsymbol{p}(t) = d\theta(0)d\boldsymbol{p}(0)$. Because the determinant of the Jacobian of the mapping $T_s : (\theta(t), \boldsymbol{p}(t)) \rightarrow (\theta(t+dt), \boldsymbol{p}(t+dt))$ is equal to one.
- The Hamiltonian dynamics is reversible, meaning that the mapping T_s from the state at time t to the state at time $t+dt$ is one-to-one because the state transitions are explicitly guided by Hamilton's equations, and hence reversible.

2.2.3 Link to the target distribution

To relate an energy function to a distribution, we can use a concept adopted from statistical mechanics known as the *canonical distribution*. For an energy function $E(\theta)$ over θ, the corresponding canonical distribution is defined as: $p(\theta) = \frac{1}{Z} e^{\frac{-E(\theta)}{T}}$, where Z is the normalizing constant, and T is the temperature of the system. In practice, we do not need to worry about Z because MCMC methods can sample from unnormalized probability distributions, and T is set to be 1.

The Hamiltonian can be written as:

$$H(\theta, \boldsymbol{p}) = U(\theta) + K(\boldsymbol{p}) \tag{2.7}$$

Using the concept of canonical distribution, we can define the joint probability density of θ and $\boldsymbol{p}$ to be:

$$p(\theta, \boldsymbol{p}) \propto e^{-H(\theta, \boldsymbol{p})} = e^{-U(\theta)} e^{-K(\boldsymbol{p})} \propto p(\theta) p(\boldsymbol{p}) \tag{2.8}$$

We can see that the joint distribution factorizes, which means θ and $\boldsymbol{p}$ are independent. We denote the variable of interest by θ and the auxiliary variables by p which allows the Hamiltonian dynamics to operate. Since θ and $\boldsymbol{p}$ have independent distributions, we are free to choose any distribution of momentum $\boldsymbol{p}$. A common practice is to use a multivariate Gaussian with zero mean and diagonal covariance, which corresponds to a quadratic kinetic energy: $\frac{\boldsymbol{p}^T M^{-1} \boldsymbol{p}}{2}$.

2.2.4 The Hamiltonian Monte Carlo algorithm

The Hamiltonian Monte Carlo (HMC) algorithm can be used to sample from a continuous distribution (up to an unknown normalizing constant).

The algorithm typically consists of three steps. The first step only changes the momentum $\boldsymbol{p}$ by drawing an initial sample from its Gaussian distribution independent of the θ vector.

The second step updates the position and momentum vector using the *leapfrog* method, which is a discretized approximation to the Hamilton's equation. The leapfrog method works as follows:

$$\boldsymbol{p}(t+\frac{\varepsilon}{2}) = \boldsymbol{p}(t) - \frac{\varepsilon}{2}\frac{\partial U(\theta(t))}{\partial \theta} \tag{2.9}$$

$$\theta(t+\varepsilon) = \theta(t) + \varepsilon \frac{\boldsymbol{p}(t+\frac{\varepsilon}{2})}{\boldsymbol{M}} \tag{2.10}$$

$$\boldsymbol{p}(t+\varepsilon) = \boldsymbol{p}(t+\frac{\varepsilon}{2}) - \frac{\varepsilon}{2}\frac{\partial U(\theta(t+\varepsilon))}{\partial \theta} \tag{2.11}$$

It is easy to see that each complete leapfrog step is reversible by simply negating $\boldsymbol{p}$ due to symmetry. Likewise, as the Jacobian of the transformations has unit determinant then the volume is preserved. However, the total Hamiltonian is only approximately conserved. This is an artifact known as *energy drift* due to the approximations used to discretize time.

In the third step, a Metropolis acceptance probability is computed and the proposed state $(\boldsymbol{q}^*, \boldsymbol{p}^*)$ from the leapfrog steps is either accepted or rejected. This ensures the states stay in high-density regions of the target probability, while at the same time be able to occasionally travel to low-density regions. The probability of acceptance is computed as:

$$\min\left[1, \exp(-U(\theta^*) + U(\theta_0) - K(\boldsymbol{p}^*) + K(\boldsymbol{p}_0))\right] \tag{2.12}$$

If the proposed state is rejected, the next state is the same as the current state. If the integration error in the total Hamiltonian is small, we would expect that this acceptance probability will remain at a very high level.

2.3 Riemannian Manifold Hamiltonian Monte Carlo

2.3.1 Riemannian manifold

A Riemannian manifold is a real, smooth manifold M equipped with an inner product g_p on the tangent space T_pM at each point p that varies smoothly from point to point. [18]. The Euclidean space itself carries a natural structure of the Riemannian manifold, but the general concept of Riemannian manifold is broader and can contain much more complicated geometric structures than Euclidean manifold.

Another important concept is *metric tensor*, which is a function that takes a pair of tangent vectors v and w as inputs and produces a scalar $g(v,w)$. This function can be seen as a generalization of dot product in Euclidean space. It defines the length and angle between tangent vectors so that we can compute the length of curves in the manifold through

integration. By definition, a manifold equipped with a positive definite metric tensor is known as a Riemannian manifold. In a Riemannian manifold, the curve/line connecting two points that has the smallest length is called a *geodesic*. Its length is the distance that a point needs to traverse to go from one point to the other [17]. In Euclidean space, the geodesic between two points is simply the straight line connecting the two points, whereas in Riemannian space it's not the case. If we define the distance between two points p and q as $d(p,q)$ (equivalent to the length of geodesic), then we can see the metric tensor as the derivative of the distance function.

2.3.2 Exploiting Riemannian manifold concepts in MCMC

There are various ways of measuring the "distance" between two probability density functions. One of them is the symmetric KL divergence: $D_S(p||q) = D_{KL}(p||q) + D_{KL}(q||p)$. For a probability density function $p(y;\theta)$, if we permute the parameter θ by a small displacement $\delta\theta$ and derive a new probability function $p(y;\theta+\delta\theta)$, we have $p(y;\theta+\delta\theta) = p(y;\theta) + \delta\theta^T \nabla_\theta p(y;\theta) + O(2)$. Since $\log(1+\varepsilon) \approx \varepsilon$, naturally we can get:

$$\begin{aligned} D_S(p(y;\theta+\delta\theta)||p(y;\theta)) &= \delta\theta^T \mathbb{E}_{y|\theta}\{\nabla_\theta \log p(y;\theta) \nabla_\theta \log p(y;\theta)^T\}\delta\theta \\ &= \delta\theta^T G(\theta)\delta\theta \end{aligned} \tag{2.13}$$

where $G(\theta)$ is called the Fisher information matrix. Paper [13] noted that $G(\theta)$ is by definition positive-definite, therefore it is a metric tensor of a Riemannian manifold. We can conclude that the space of probability functions is endowed with a Riemannian geometry.

Intuitively, we can see $\delta\theta$ as the term that determines the direction of the geodesic between two points (in this case, two distributions) in the manifold, and the matrix $G(\theta)$ determines the distance that we should move along each dimension. If we define $G(\theta)$ to be an identity matrix, it then reduces to the metric tensor in Euclidean manifold and we lose the nice property of Riemannian manifold where the position specific metric tensor is able to reflect the local geometric structure.

2.3.3 The Riemannian Manifold Hamiltonian Monte Carlo algorithm

In this subsection, we will briefly introduce the concepts behind the Riemannian Manifold Hamiltonian Monte Carlo (RMHMC) algorithm.

Following on from section 2.3.2, we can now define the Hamiltonian on a Riemannian manifold. We can see the standard HMC as described in section 2.2.4 as an algorithm defined in Euclidean space, where we need to manually tune the mass matrix M defining a globally

constant metric. Whereas in RMHMC the mass matrix is replaced by $G(\theta)$ which exploits the local geometric structure automatically. More specifically, the norm of each $\dot{\theta}$ becomes:

$$||\dot{\theta}||^2_{G(\theta)} = \dot{\theta}^T G(\theta)\dot{\theta} = \boldsymbol{p}^T G^{-1}(\theta)\boldsymbol{p} \tag{2.14}$$

instead of:

$$||\dot{\theta}||^2_M = \dot{\theta}^T M\dot{\theta} = \boldsymbol{p}^T M^{-1}\boldsymbol{p} \tag{2.15}$$

thus the kinetic energy has a new form defined by the inverse of $G(\theta)$. Therefore the Hamiltonian on Riemann manifold can be written as:

$$H(\theta,\boldsymbol{p}) = -\mathcal{L}(\theta) + \frac{1}{2}\log((2\pi)^D|G(\theta)|) + \frac{1}{2}\boldsymbol{p}^T G(\theta)^{-1}\boldsymbol{p} \tag{2.16}$$

so that the marginal density of θ is the desired target density:

$$p(\theta) \propto \int \exp(H(\theta,\boldsymbol{p}))d\boldsymbol{p} = \frac{\exp(\mathcal{L}(\theta))}{\sqrt{2\pi^D|G(\theta)|}}\int \exp(-\frac{1}{2}\boldsymbol{p}^T G(\theta)^{-1}\boldsymbol{p})d\boldsymbol{p} = \exp(\mathcal{L}(\theta)) \tag{2.17}$$

The dynamics is defined by a new set of Hamilton's equations:

$$\frac{d\theta}{dt} = \frac{\partial H}{\partial \boldsymbol{p}} = G(\theta)^{-1}\boldsymbol{p} \tag{2.18}$$

$$\frac{d\boldsymbol{p}}{dt} = -\frac{\partial H}{\partial \theta} = \frac{\partial \mathcal{L}(\theta)}{\partial \theta} - \frac{1}{2}Tr[G(\theta)^{-1}\frac{\partial G(\theta)}{\partial \theta}] + \frac{1}{2}\boldsymbol{p}^T G(\theta)^{-1}\frac{\partial G(\theta)}{\partial \theta}G(\theta)^{-1}\boldsymbol{p} \tag{2.19}$$

With this new set of Hamilton's equations, a generalized leapfrog algorithm is proposed and the details are shown in algorithm 5 in section 4.1.4. A nice visualization of a comparison between HMC and RMHMC is shown in Figure 2.3 from paper [3]. The left image shows the path of a Markov chain using HMC with a unit mass matrix, and the second image shows the path from the same starting point using RMHMC. Note how the use of a position specific $G(\theta)$ makes the Markov chain converges much faster.

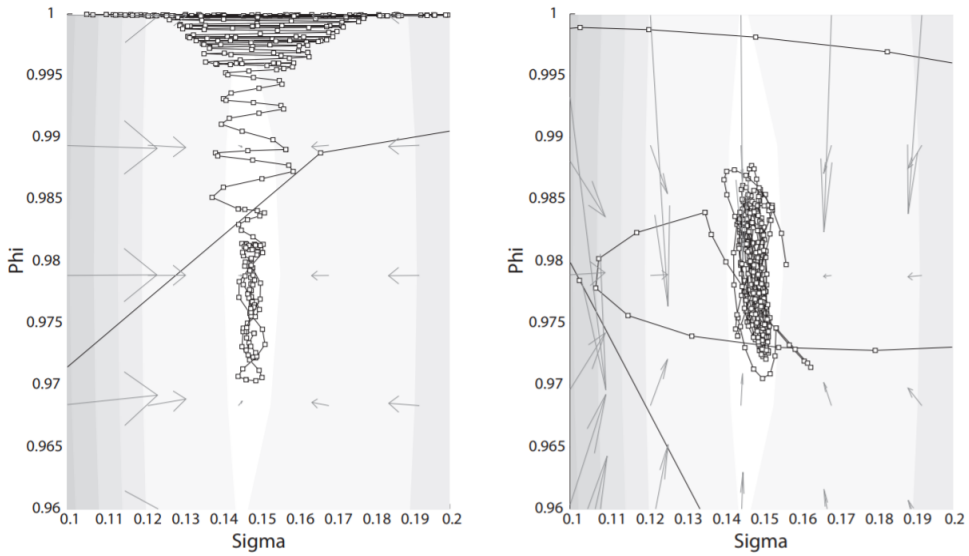


Fig. 2.3 Comparison between HMC and RMHMC paths from a stochastic volatility model investigated in paper [3].

Chapter 3

Related Work

3.1 Importance Weighted Auto-Encoder

The Importance Weighted Auto-Encoder (IWAE) was first proposed in [1]. The central idea is to use a tighter evidence lower bound (ELBO) by applying importance sampling in the evaluation of the ELBO. It has the same architecture as the standard VAE, the major distinction lies in the way the ELBO is computed. For the standard VAE, the ELBO is the Monte Carlo estimate for a single sample $\boldsymbol{z} \sim q_\phi(\boldsymbol{z} \mid \boldsymbol{x})$ where $\boldsymbol{z}$ is the latent variable, whereas for IWAE, it considers not just one sample, but a fixed number of k samples:

$$\mathcal{L}_k(\boldsymbol{x}) = \mathbb{E}_{\boldsymbol{z}_1,\ldots\boldsymbol{z}_k \sim q_\phi(\boldsymbol{z}|\boldsymbol{x})}\big[\log \frac{1}{k}\sum_{i=1}^{k} \frac{p(\boldsymbol{x},\boldsymbol{z}_i)}{q(\boldsymbol{z}_i|\boldsymbol{x})}\big] \tag{3.1}$$

The larger k is, the closer the ELBO gets to the true marginal likelihood of the data. When $k = 1$, it is the same as the ELBO of the standard VAE. Ideally, if we increase k to infinity, equation 3.1 gives the true marginal likelihood.

However, as shown in paper [12], as k increases, the gradient estimator becomes problematic. More specifically, the gradient of the parameters to be optimized becomes zero. The paper concludes that using a tighter ELBO can be detrimental to the process of learning.

3.2 Normalizing Flows

In paper [14], Normalizing Flows were introduced to enrich the posterior approximation family. It is able to specify flexible, arbitrarily complex and scalable posterior approximations. The basic idea is to transform a simple initial density into a more complex one by applying a sequence of invertible transformations to reach a certain level of complexity.

To start with, let's consider a simple example where "change of variables" is involved. Let $\boldsymbol{z}$ be a random variable that follows a distribution $\boldsymbol{z} \sim q(\boldsymbol{z})$, and f is an invertible transformation that maps $\boldsymbol{z}$ to $\boldsymbol{y}$. The resulting variable $\boldsymbol{y} = f(\boldsymbol{z})$ has the following distribution:

$$q(\boldsymbol{y}) = q(\boldsymbol{z}) |\det \frac{\partial f^{-1}}{\partial \boldsymbol{z}}| = q(\boldsymbol{z}) |\det \frac{\partial f}{\partial \boldsymbol{z}}|^{-1} \tag{3.2}$$

If we apply a series of transformations $f_k, k \in 1, ...K$, we get a normalizing flow:

$$\boldsymbol{z}_K = f_K \circ f_{K-1} \circ ... f_1(\boldsymbol{z}_0) \tag{3.3}$$

$$q_K(\boldsymbol{z}_K) = q_0(\boldsymbol{z}_0) \prod_{k=1}^{K} |\det \frac{\partial f_k}{\partial \boldsymbol{z}_{k-1}}|^{-1} \tag{3.4}$$

where $\boldsymbol{z}_0$ is sampled from an initial distribution $q_0(\boldsymbol{z}_0)$. From equation 3.4 we can see that the determinant of the Jacobian need to be computed and this could easily be the bottleneck of the algorithm. Thus we require to have normalizing flows that allow for inexpensive computation of the determinant, or where the Jacobian is not needed at all. To achieve this, the paper proposed two families of transformations.

3.2.1 Planar flow

Planar flows have the form:

$$f(\boldsymbol{z}) = \boldsymbol{z} + \boldsymbol{u} h(\boldsymbol{w}^T \boldsymbol{z} + b) \tag{3.5}$$

where $\lambda = \{\boldsymbol{w} \in \mathbb{R}^D, \boldsymbol{u} \in \mathbb{R}^D, b \in \mathbb{R}\}$ and $h(\cdot)$ is a smooth element-wise non-linearity, which is typically the activation function in neural networks. If we define $\psi(\boldsymbol{z}) = h'(\boldsymbol{w}^T \boldsymbol{z} + b)\boldsymbol{w}$, the determinant is then easily computed as:

$$|\det \frac{\partial f}{\partial \boldsymbol{z}}| = |1 + \boldsymbol{u}^T \psi(\boldsymbol{z})| \tag{3.6}$$

We can think of this flow as slicing the $\boldsymbol{z}$ space with the hyperplane defined using $\boldsymbol{w}^T \boldsymbol{z} + b = 0$.

3.2.2 Radial flow

An alternative to the planar flow is the radial flow. The transform is defined as:

$$f(\boldsymbol{z}) = \boldsymbol{z} + \beta h(\alpha, r)(\boldsymbol{z} - \boldsymbol{z}_0) \tag{3.7}$$

where $r = |\boldsymbol{z} - \boldsymbol{z}_0|$ and $h(\alpha, r) = \frac{1}{\alpha + r}$, the free parameters are $\lambda = \{\boldsymbol{z}_0 \in \mathbb{R}^D, \alpha \in \mathbb{R}^+, \beta \in \mathbb{R}\}$. The determinant is computed as:

$$|\det \frac{\partial f}{\partial \boldsymbol{z}}| = \big[1 + \beta h(\alpha, r)\big]^{d-1} \big[1 + \beta h(\alpha, r) + 1 + \beta h^{'}(\alpha, r) r\big] \tag{3.8}$$

This family of transformations also allow for efficient linear-time computation of the determinant. The radial flows introduce reference points in the z-space and apply radial contractions and expansions around the reference points.

3.2.3 Alternative flows

The paper [14] also proposed Langevin flow and Hamiltonian flow. The Langevin flow is defined by the Langevin stochastic differential equation. In this case, the transformation of densities can be obtained by the Fokker-Planck equation, and the stationary solution for $q_t(\boldsymbol{z})$ is given by the Boltzmann distribution: $q_\infty(\boldsymbol{z}) \propto e^{-\mathcal{L}(\boldsymbol{z})}$.

For the Hamiltonian flow, it can be seen as a kind of normalizing flow on an augmented space $(\boldsymbol{z}, \boldsymbol{p})$ where $\boldsymbol{p}$ is the momentum of the Hamiltonian dynamics. It is an instance of an *infinitesimal volume-preserving flow*.

Both flows are examples of utilizing transition operators in the MCMC literature. As we know that as the number of transitions goes to infinity, the distribution $q(\boldsymbol{z})$ will converge to the true posterior $p(\boldsymbol{z}|\boldsymbol{x})$. In particular, the Hamiltonian flow has a very similar way of turning a simple distribution into a complex one by utilizing the Hamiltonian dynamics when compared with the focus of this MPhil project. A disadvantage of using such normalizing flows is that they require additional time for computing the likelihood and the corresponding gradients in the leapfrog steps.

3.3 Inverse Autoregressive Flow

To improve the expressivity of the normalizing flows introduced in the above section, Inverse Autoregressive Flow (IAF) was introduced in [7]. To start with, let's first consider autoregressive transformations, which belong to a family of transformations where the i^{th} dimension of the resulting variable $\boldsymbol{y}$ only depends on the 1^{st} to the i^{th} dimension of the input variable $\boldsymbol{x}$ like the following:

$$y_1 = \mu_1 + \sigma_1 x_1 \tag{3.9}$$

$$y_i = \mu(\boldsymbol{y}_{1:i-1}) + \sigma(\boldsymbol{y}_{1:i-1}) x_i \tag{3.10}$$

To get the complete output variable $\boldsymbol{y}$ we need to perform $O(D)$ sequential computations which cannot be parallelized thus making the overall computation expensive. However, we can see that its inverse transformation can be parallelized because the inverse has the form:

$$x_i = \frac{y_i - \mu(\boldsymbol{y}_{1:i-1})}{\sigma(\boldsymbol{y}_{1:i-1})} \tag{3.11}$$

we can vectorize the above equation:

$$\boldsymbol{x} = \frac{\boldsymbol{y} - \mu(\boldsymbol{y})}{\sigma(\boldsymbol{y})} \tag{3.12}$$

The major difference between the forward and inverse autoregressive transformations is that the output variable of the inverse autoregressive transformation does not have the dependencies between dimensions as in the forward transformation, making it possible to parallelize the computation. In addition, the Jacobian of equation 3.12 is lower triangular with a simple diagonal. As we know, the determinant of a lower triangular matrix equals to the product of the elements on the diagonal, which makes the computation extremely efficient.

Chapter 4

Methodology

4.1 Combining variational inference with MCMC

4.1.1 Basic idea

As mentioned in chapter 1, variational inference and MCMC methods are the two major methods for estimating the intractable posterior distribution $q(\boldsymbol{z}|\boldsymbol{x})$ in machine learning problems. The major benefit we get from variational inference is that we can explicitly optimize an ELBO using any available optimization algorithms. The lower bound usually has the form:

$$\log p(\boldsymbol{x}) \geq \log p(\boldsymbol{x}) - D_{KL}(q_\theta(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z}|\boldsymbol{x})) = \mathbb{E}_{q_\theta(\boldsymbol{z}|\boldsymbol{x})}\left[\log p(\boldsymbol{x},\boldsymbol{z}) - \log q_\theta(\boldsymbol{z}|\boldsymbol{x})\right] = \mathcal{L} \quad (4.1)$$

Maximizing the lower bound w.r.t. θ will minimize the KL-divergence.

As for MCMC, it usually starts by taking an initial sample $\boldsymbol{z}_0$ from an initial distribution $q(\boldsymbol{z}_0)$ or $q(\boldsymbol{z}_0|\boldsymbol{x})$, and the apply a stochastic transition operator to the sample $\boldsymbol{z}_0$:

$$\boldsymbol{z}_t \sim q(\boldsymbol{z}_t|\boldsymbol{z}_{t-1},\boldsymbol{x}) \quad (4.2)$$

If we apply this transition operator for a sufficient number of times, we can approach the true posterior, but it takes a longer time to do so.

The central idea to combine variational inference and MCMC is that we can regard the intimidate samples: $\boldsymbol{y} = \boldsymbol{z}_0, \boldsymbol{z}_1, ...\boldsymbol{z}_{t-1}$ as *auxiliary variables* for computing the ELBO.

Namely, we can integrate the auxiliary variables into the expression of the ELBO:

$$\begin{aligned}\mathcal{L}_{aux} &= \mathbb{E}_{q(\boldsymbol{y},\boldsymbol{z}_T|\boldsymbol{x})}\big[\log\big[p(\boldsymbol{x},\boldsymbol{z}_T)r(\boldsymbol{y}|\boldsymbol{x},\boldsymbol{z}_T)\big] - \log q(\boldsymbol{y},\boldsymbol{z}_T|\boldsymbol{x})\big] \\ &= \mathcal{L} - \mathbb{E}_{q(\boldsymbol{z}_T|\boldsymbol{x})}\{D_{KL}\big[q(\boldsymbol{y}|\boldsymbol{z}_T,\boldsymbol{x})||r(\boldsymbol{y}|\boldsymbol{z}_T,\boldsymbol{x})\big]\} \\ &\leq \mathcal{L} \leq \log\big[p(\boldsymbol{x})\big]\end{aligned} \tag{4.3}$$

Here $r(\boldsymbol{y}|\boldsymbol{x},\boldsymbol{z}_T)$ is another approximation we made to approximate the true reverse model $q(\boldsymbol{y}|\boldsymbol{x},\boldsymbol{z}_T) = \int q(\boldsymbol{y},z_T|\boldsymbol{x})dy$, and it's called the inverse model. For this project we assume it has a Markov structure: $r(\boldsymbol{z}_0,...,\boldsymbol{z}_{T-1}|\boldsymbol{x},\boldsymbol{z}_T) = \prod_{t=1}^{T} r_t(\boldsymbol{z}_{t-1}|\boldsymbol{x},\boldsymbol{z}_t)$. Thus we can further rewrite the ELBO as:

$$\begin{aligned}\log p(\boldsymbol{x}) &\geq \mathbb{E}_q\big[\log p(\boldsymbol{x},\boldsymbol{z}_T) - \log q(\boldsymbol{z}_0,...,\boldsymbol{z}_T|\boldsymbol{x}) + \log r(\boldsymbol{z}_0,...,\boldsymbol{z}_{t-1}|\boldsymbol{x},\boldsymbol{z}_T)\big] \\ &= \mathbb{E}_q\big[\log\big[\frac{p(\boldsymbol{x},\boldsymbol{z}_T)}{q(\boldsymbol{z}_0|\boldsymbol{x})}\big] + \sum_{t=1}^{T}\log\big[\frac{r_t(\boldsymbol{z}_{t-1}|\boldsymbol{x},\boldsymbol{z}_t)}{q_t(\boldsymbol{z}_t|\boldsymbol{x},\boldsymbol{z}_{t-1})}\big]\big]\end{aligned} \tag{4.4}$$

If we want to optimize this lower bound, we can specify q_t and r_t in some parametric form and then optimize 4.4 w.r.t. the parameters. A typical example is to use neural networks to represent these two models.

4.1.2 ELBO of general MCMC variational inference

In many cases, the lower bound in equation 4.4 cannot be computed analytically. However we can obtain its Monte Carlo estimate without bias using the following algorithm.

Algorithm 1 MCMC lower bound estimate

Require: Model with joint distribution $p(\boldsymbol{x},\boldsymbol{z})$
Require: Number of iterations T
Require: Transition operator $q_t(\boldsymbol{z}_t|\boldsymbol{x},\boldsymbol{z}_{t-1})$
Require: Inverse model $r_t(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t,\boldsymbol{x})$
Require: Step size ε and mass matrix M
Draw an initial random $\boldsymbol{z}_0 \sim p(\boldsymbol{z}_0|\boldsymbol{x})$
Calculate the initial lower bound as $\mathcal{L} = \log p(\boldsymbol{x},\boldsymbol{z}_0) - \log q(\boldsymbol{z}_0|\boldsymbol{x})$
for $t = 1:T$ **do**
    Perform random transition $\boldsymbol{z}_t \sim q_t(\boldsymbol{z}_t|\boldsymbol{x},\boldsymbol{z}_{t-1})$
    Calculate the ratio $\alpha_t = \frac{p(\boldsymbol{x},\boldsymbol{z}_t)r_t(\boldsymbol{z}_{t-1}|\boldsymbol{x},\boldsymbol{z}_t)}{p(\boldsymbol{x},\boldsymbol{z}_{t-1})q_t(\boldsymbol{z}_t|\boldsymbol{x},\boldsymbol{z}_{t-1})}$
    Update $\mathcal{L} = \mathcal{L} + \log\alpha_t$
return $\mathcal{L}$

To optimize the above lower bound, we need to obtain its unbiased gradients w.r.t. model parameters. Here we encounter the same problem where we need to apply backpropagation

through random sampling processes $q_t(\boldsymbol{z}_t|\boldsymbol{x},\boldsymbol{z}_{t-1})$, which is not feasible unless we utilize the reparameterization trick mentioned in the SGVB estimator of the VAE in section 2.1.4: first we draw a set of primitive random variables $\boldsymbol{u}_t$ from distribution $p(\boldsymbol{u}_t)$, and then transform $\boldsymbol{u}_t$ into $\boldsymbol{z}_t$ with transformation $\boldsymbol{z}_t = g_\theta(\boldsymbol{u}_t,\boldsymbol{x})$ which ensures that $\boldsymbol{z}_t$ follows the distribution $q_t(\boldsymbol{z}_t|\boldsymbol{x},\boldsymbol{z}_{t-1})$.

Once we obtain the gradients, we can optimize the lower bound using any stochastic gradient-based optimization algorithm as shown in algorithm 2.

Algorithm 2 Markov Chain Variational Inference (MCVI)

Require: Forward model $q_\theta(\boldsymbol{z})$ and backward model $r_\theta(\boldsymbol{z}_0,...,\boldsymbol{z}_{t-1}|\boldsymbol{z}_T)$
Require: Parameter θ
Require: Stochastic estimate $\mathcal{L}(\theta)$ of the lower bound from algorithm 1
while not converged **do**
    Obtain unbiased estimate $\hat{g}$ with $\mathbb{E}_q[\hat{g}] = \nabla_\theta \mathcal{L}_{aux}(\theta)$ by differentiating $\mathcal{L}(\theta)$
    Update θ using gradient $\hat{g}$ and a stochastic optimization algorithm
return final optimized variational parameters θ

4.1.3 ELBO of Hamiltonian variational inference

Hamiltonian Monte Carlo (HMC) is one of the most efficient MCMC method and thus we will choose HMC as the MCMC algorithm to incorporate it into the auxiliary ELBO in equation 4.4. A momentum variable $\boldsymbol{v}$ is introduced to make the HMC sampler operate. The algorithm for calculating the auxiliary ELBO is shown in algorithm 3.

Algorithm 3 Hamiltonian variational inference (HVI)

Require: Unnormalized log posterior $\log p(\boldsymbol{x},\boldsymbol{z})$
Require: Number of iterations T
Require: Momentum initial distribution $q_t(\boldsymbol{v}_t^{'}|\boldsymbol{x})$ and inverse model $r_t(\boldsymbol{v}_t|\boldsymbol{z}_t,\boldsymbol{x})$
Require: Step size ε and mass matrix M
Draw an initial random $\boldsymbol{z}_0 \sim p(\boldsymbol{z}_0|\boldsymbol{x})$
Calculate the initial lower bound as $\mathcal{L} = \log p(\boldsymbol{x},\boldsymbol{z}_0) - \log q(\boldsymbol{z}_0|\boldsymbol{x})$
for $t = 1 : T$ **do**
    Draw initial momentum $\boldsymbol{v}_t^{'} \sim q_t(\boldsymbol{v}_t^{'}|\boldsymbol{x})$
    Pass $(\boldsymbol{z}_{t-1},\boldsymbol{v}_t^{'})$ to HMC sampler and obtain $(\boldsymbol{z}_t,\boldsymbol{v}_t)$
    Calculate the ratio $\alpha_t = \frac{p(\boldsymbol{x},\boldsymbol{z}_t)r_t(\boldsymbol{v}_t|\boldsymbol{x},\boldsymbol{z}_t)}{p(\boldsymbol{x},\boldsymbol{z}_{t-1})q_t(\boldsymbol{v}_t^{'}|\boldsymbol{x})}$
    Update $\mathcal{L} = \mathcal{L} + \log \alpha_t$
return $\mathcal{L}$

We can see that the way we calculate the auxiliary ELBO is by sampling from the transisiton q_t and then evaluate the inverse model r_t at those samples. Adding the ratio α_t through multiple iterations $t = 1 : T$ will result in $\mathcal{L} = \mathcal{L} + \log\left[\frac{p(\boldsymbol{x},\boldsymbol{z}_1)r_1(\boldsymbol{v}_1|\boldsymbol{x},\boldsymbol{z}_1)}{p(\boldsymbol{x},\boldsymbol{z}_0)q_1(\boldsymbol{v}_1'|\boldsymbol{x},\boldsymbol{z}_{t-0})}\frac{p(\boldsymbol{x},\boldsymbol{z}_2)r_2(\boldsymbol{v}_2|\boldsymbol{x},\boldsymbol{z}_2)}{p(\boldsymbol{x},\boldsymbol{z}_1)q_2(\boldsymbol{v}_2'|\boldsymbol{x},\boldsymbol{z}_1)}...\right]$ where the intimidate $p(\boldsymbol{x},\boldsymbol{z}_t)$ terms will be cancelled out.

The ELBO is then optimized w.r.t. θ where θ represents the parameters of the forward model $q_\theta(\boldsymbol{z})$ and inverse model $r_\theta(\boldsymbol{z}_0,...\boldsymbol{z}_{t-1}|\boldsymbol{z}_T)$. Note that in this case, since the transition from $(\boldsymbol{z}_{t-1},\boldsymbol{v}_t')$ to $(\boldsymbol{z}_t,\boldsymbol{v}_t)$ is deterministic, we have $q(\boldsymbol{v}_t,\boldsymbol{z}_t|\boldsymbol{z}_{t-1},\boldsymbol{x}) = q(\boldsymbol{v}_t'|\boldsymbol{z}_{t-1},\boldsymbol{x})$ and $r(\boldsymbol{v}_t',\boldsymbol{z}_{t-1}|\boldsymbol{z}_t,\boldsymbol{x}) = r(\boldsymbol{v}_t|\boldsymbol{z}_t,\boldsymbol{x})$. The gradient of the auxiliary ELBO is obtained with the help of the reparameterization trick.

HVI also requires the unnormalized log posterior $\log p(\boldsymbol{x},\boldsymbol{z})$, which in many cases can be obtained by adding the log prior $p(\boldsymbol{z})$ with log likelihood $p(\boldsymbol{x}|\boldsymbol{z})$. In addition, we found that in practice one HMC iteration is enough for a large performance improvement without introducing too much computation overhead.

The distributions are parameterized with neural networks so that we can explicitly optimize the weights and biases with backpropagation. For the subsequent experiments, we tested two types of neural networks: fully-connected neural networks and convolutional neural networks.

A graphical representation of the fully-connected configuration is shown in Figure 4.1. As we can see, the initial distribution $q(\boldsymbol{z}_0|\boldsymbol{x})$, the forward model $q_t(\boldsymbol{v}_t'|\boldsymbol{x})$, the inverse model $r_t(\boldsymbol{v}_t|\boldsymbol{x},\boldsymbol{z}_t)$ and the decoder $p(\boldsymbol{x}|\boldsymbol{z}_t)$ are all parameterized with fully-connected neural networks.

Similarly, a graphical representation of the convolutional configuration is shown in Figure 4.2 where the yellow layers represent convolutional/deconvolutional layers.

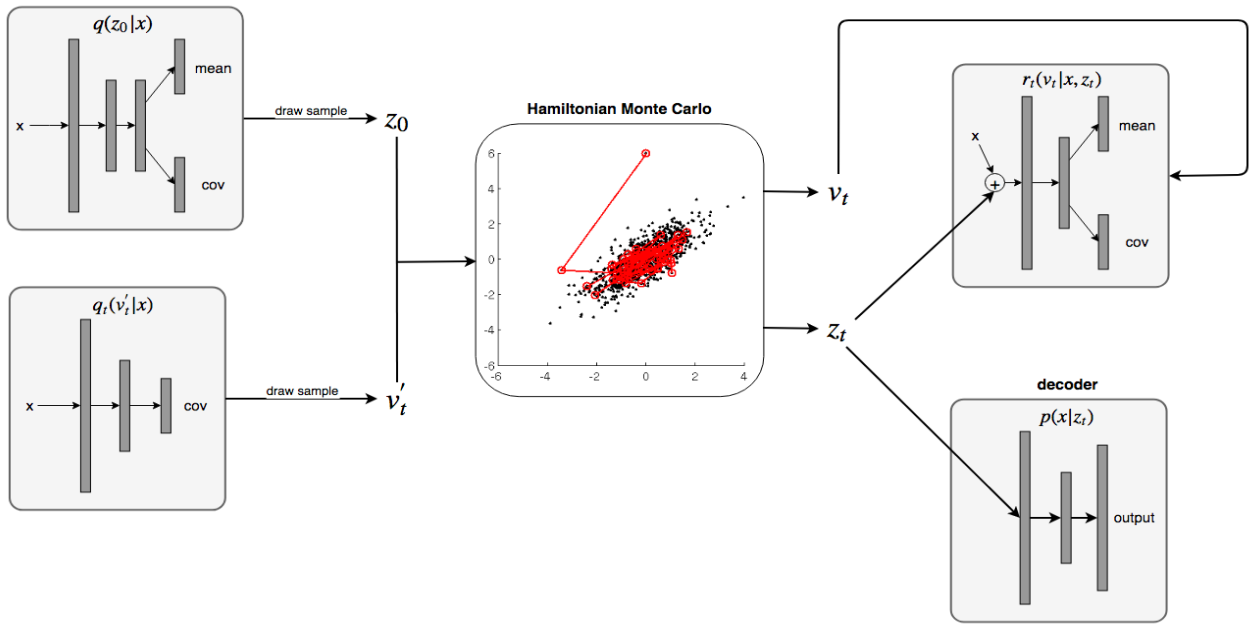


Fig. 4.1 Graphical representation of HVI model with fully-connected networks.

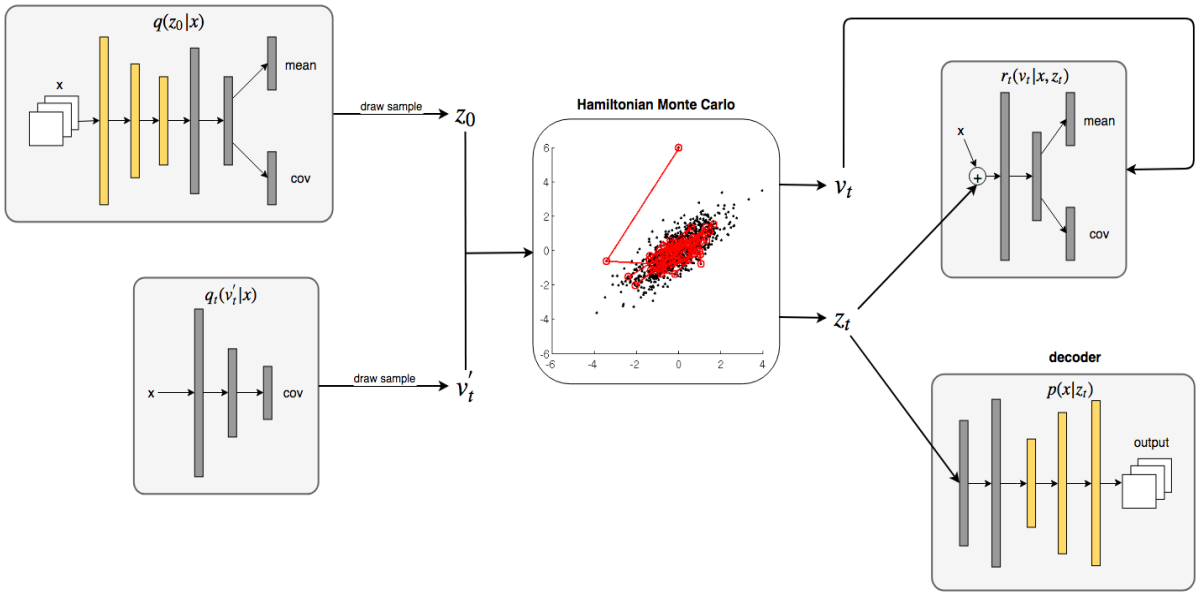


Fig. 4.2 Graphical representation of HVI model with convolutional networks, yellow layers represent convolutional/deconvolutional layers

4.1.4 ELBO of Riemann manifold Hamiltonian variational inference

The ELBO of Riemann manifold Hamiltonian variational inference is calculated in a similar way as the ELBO of Hamiltonian variational inference except for some minor modifications. The algorithm is shown in algorithm 4.

Algorithm 4 Riemann manifold Hamiltonian variational inference (RMHVI)

Require: Unnormalized log posterior $\log p(\boldsymbol{x}, \boldsymbol{z})$
Require: Number of iterations T
Require: Inverse model $r_t(\boldsymbol{v}_t|\boldsymbol{z}_t, \boldsymbol{x})$
Require: Step size ε
Draw an initial random $\boldsymbol{z}_0 \sim p(\boldsymbol{z}_0|\boldsymbol{x})$
Calculate the initial Fisher information matrix $G(\boldsymbol{z_0})$
Sample initial momentum $\boldsymbol{v}_1^{'}$ from $\mathcal{N}(0, G(\boldsymbol{z_0}))$
Calculate the initial lower bound as $\mathcal{L} = \log p(\boldsymbol{x}, \boldsymbol{z}_0) - \log q(\boldsymbol{z}_0|\boldsymbol{x})$
for $t = 1 : T$ **do**
    Pass $(\boldsymbol{z}_{t-1}, \boldsymbol{v}_t^{'})$ to manifold HMC sampler and obtain $(\boldsymbol{z}_t, \boldsymbol{v}_t)$
    Calculate the ratio $\alpha_t = \frac{p(\boldsymbol{x}, \boldsymbol{z}_t) r_t(\boldsymbol{v}_t|\boldsymbol{x}, \boldsymbol{z}_t)}{p(\boldsymbol{x}, \boldsymbol{z}_{t-1}) q_t(\boldsymbol{v}_t^{'}|\boldsymbol{x}, \boldsymbol{z}_{t-1})}$
    Update $\mathcal{L} = \mathcal{L} + \log \alpha_t$
return $\mathcal{L}$

A graphical representation of the system architecture is shown in Figure 4.3. As we can see from the figure, the most significant difference from HVI is that we do not need to optimize the parameters for the momentum initial distribution model $q_t(\boldsymbol{v}_t^{'}|\boldsymbol{x})$. The forward model is replaced by a module that calculates $G(\boldsymbol{z}_0)$. The momentum is drawn from a multivariate Gaussian with zero mean and a covariance matrix defined by the Fisher information matrix $G(\boldsymbol{z})$ instead. Here $G(\boldsymbol{z})$ plays the same role as the diagonal covariance matrix does in HVI. The difference is that $G(\boldsymbol{z})$ automatically adapts to the natural geometric structure of the density model $p(\theta)$ in Riemannian manifold, and it should yield more effective transitions in the overall algorithm.

In statistics, the Fisher information matrix measures the amount of information that an observable random variable $\boldsymbol{x}$ carries about an unknown parameter θ of a distribution that models $\boldsymbol{x}$. More specifically, it is the variance of the *score*, where score is the gradient of the log-likelihood with respect to θ. It can be shown that the first moment (mean) is zero. In this case, the variance can be written as:

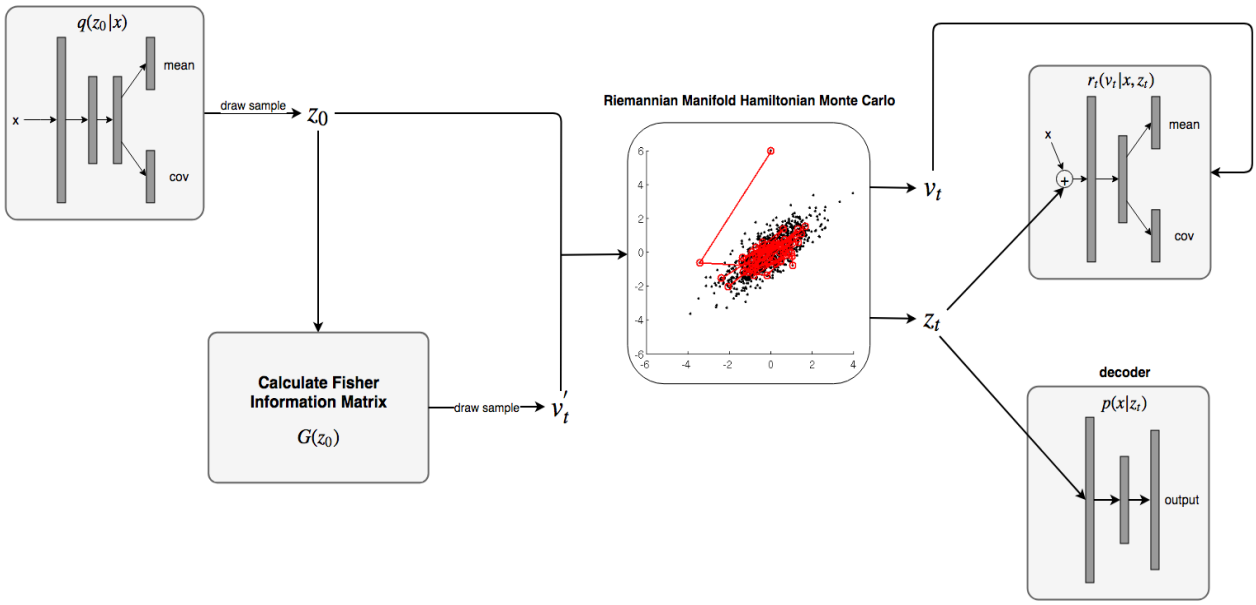


Fig. 4.3 Graphical representation of RMHVI model with fully-connected networks.

$$\begin{aligned} G(\theta) &= \mathbb{E}[(\frac{\partial}{\partial\theta}\log f(\boldsymbol{X};\theta))^2|\theta] \\ &= \int (\frac{\partial}{\partial\theta} f(\boldsymbol{x};\theta))^2 f(\boldsymbol{x};\theta) d\boldsymbol{x} \end{aligned} \tag{4.5}$$

If the integration in equation 4.5 is intractable, we can use the Monte Carlo estimate of it:

$$G(\theta) \approx \frac{1}{N}\sum[(\frac{\partial}{\partial\theta}\log f(\boldsymbol{x};\theta))^2|\theta] \tag{4.6}$$

where $\boldsymbol{x}$ is sampled from the conditional distribution $p(\boldsymbol{x}|\theta)$.

In our experiment setting, θ is the latent variable $\boldsymbol{z}$ and $\boldsymbol{x}$ is the output of the neural decoder. Following the procedure described above, we can obtain the Fisher information matrix $G(\boldsymbol{z})$ as follows:

1. Pass $\boldsymbol{z}$ through the decoder and obtain a set of logits that represent the probabilities of particular pixels taking the value 1 (black), which can be viewed as a multivariate Bernoulli distribution. For MNIST dataset, the size of this distribution has dimension $28 \times 28 = 784$.

2. Take n samples $\boldsymbol{y}$ from the multivariate Bernoulli distribution.

3. Calculate the gradients of $\boldsymbol{y}$ w.r.t. $\boldsymbol{z}$: $\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{z}}$.

4. Calculate the mean of the square of the gradients.

Once we get $G(\boldsymbol{z})$, we can use it to calculate the Hamiltonian. Unlike in HMC, the manifold version of Hamiltonian has a different expression:

$$H(\boldsymbol{q},\boldsymbol{p}) = -\mathcal{L}(\boldsymbol{q}) + \frac{1}{2}\log((2\pi)^D|G(\boldsymbol{q})|) + \frac{1}{2}\boldsymbol{p}^T G(\boldsymbol{q})^{-1}\boldsymbol{p} \tag{4.7}$$

In this case, the joint density is no longer factorizable and therefore the log-likelihood does not correspond to a separable Hamiltonian. The dynamics are defined by the Hamilton's equation:

$$\frac{d\boldsymbol{q}}{dt} = \frac{\partial H}{\partial \boldsymbol{p}} = G(\boldsymbol{q})^{-1}\boldsymbol{p} \tag{4.8}$$

$$\frac{d\boldsymbol{p}}{dt} = -\frac{\partial H}{\partial \boldsymbol{q}} = \frac{\partial \mathcal{L}(\boldsymbol{q})}{\partial \boldsymbol{q}} - \frac{1}{2}Tr[G(\boldsymbol{q})^{-1}\frac{\partial G(\boldsymbol{q})}{\partial \boldsymbol{q}}] + \frac{1}{2}\boldsymbol{p}^T G(\boldsymbol{q})^{-1}\frac{\partial G(\boldsymbol{q})}{\partial \boldsymbol{q}}G(\boldsymbol{q})^{-1}\boldsymbol{p} \tag{4.9}$$

With the modified Hamilton's equations, we can derive the generalized leapfrog algorithm [9] as shown below in algorithm 5. Note that here we omit the Metropolis-Hastings step that is typically used with HMC.

Algorithm 5 RMHMC with Generalised Leapfrog

Initialize current $\boldsymbol{q}$
for IterationNum = 1 to NumSamples **do**
    Sample new momentum $\boldsymbol{p}^1$
    Calculate current $H(\boldsymbol{q},\boldsymbol{p}^1)$
    $\boldsymbol{q}^1$=current $\boldsymbol{q}$
    for n = 1 to N (leapfrog steps) **do**
        // Update the momentum with fixed point iterations
        $\hat{\boldsymbol{p}}^0 = \boldsymbol{p}^n$
        for i = 1 to NumOfFixedPointSteps **do**
            $\hat{\boldsymbol{p}}^i = \boldsymbol{p}^n - \frac{\varepsilon}{2}\nabla_{\boldsymbol{q}}H(\boldsymbol{q}^n,\hat{\boldsymbol{p}}^{i-1})$
        $\boldsymbol{p}^{n+\frac{1}{2}} = \hat{\boldsymbol{p}}^i$
        // Update the parameters with fixed point iterations
        $\hat{\boldsymbol{q}}^0 = \boldsymbol{q}^n$
        for i = 1 to NumOfFixedPointSteps **do**
            $\hat{\boldsymbol{q}}^i = \boldsymbol{q}^n + \frac{\varepsilon}{2}\nabla_{\boldsymbol{p}}H(\boldsymbol{q}^n,\boldsymbol{p}^{n+\frac{1}{2}}) + \frac{\varepsilon}{2}\nabla_{\boldsymbol{p}}H(\hat{\boldsymbol{q}}^{i-1},\boldsymbol{p}^{n+\frac{1}{2}})$
        $\boldsymbol{q}^{n+1} = \hat{\boldsymbol{q}}^i$
        // Update the momentum exactly
        $\boldsymbol{p}^{n+1} = \boldsymbol{p}^{n+1} - \frac{\varepsilon}{2}\nabla_{\boldsymbol{q}}H(\boldsymbol{q}^{n+1},\boldsymbol{p}^{n+\frac{1}{2}})$

	Speed	ELBO
VAE	fast	worst
HVI:	moderate	moderate
RMHVI	slow	best

Table 4.1 Comparison between fully-connected HVI and vanilla VAE for MNIST.

4.1.5 Theoretic comparison between models

In theory, a general comparison between three models is shown in Table 4.1. VAE is the fastest model to be trained because we do not need any leapfrog steps. HVI is *mk* times more expensive than VAE where m is the number of MCMC iterations and k is the number of leapfrog steps. RMHVI is the slowest because it not only has k leapfrog steps, within each leapfrog step we also need to computer the Fisher information matrix $G(\mathbf{z})$ by computing the gradient of n samples. Moreover, we need to compute this $G(\mathbf{z})$ multiple times because there are multiple fixed point iterations within in each leapfrog step as shown in algorithm 5.

RMHVI should be able to give the highest ELBO because RMHMC converges faster than HMC. If given the same number of leapfrog steps, The samples we get from RMHMC is more likely to fall into the high-density regions of the target posterior. HVI should be able to outperform vanilla VAE because of the additional complexity of the approximation to the true posterior distribution.

The actual model performance is tested in the next chapter through extensive experiments.

Chapter 5

Experiments

5.1 Hamiltonian variational inference (HVI)

5.1.1 Fully-connected neural network configuration

To test the performance of the HVI algorithm, we replace the inference network (encoder) in VAE with HVI as describe in algorithm 3. We first use fully-connected neural networks to parameterize the internal models.

- The initial distribution $q(\boldsymbol{z}_0|\boldsymbol{x})$ is parameterized with a fully-connected neural network with two hidden layers, softplus activations and 300 hidden units in each hidden layer. The output are mean and diagonal covariance of a Gaussian. The default dimension of output variable ($\boldsymbol{z}$) is 20.

- The forward model $q_t(\boldsymbol{v}_t^{'}|\boldsymbol{x})$ is parameterized using a shallow fully-connected network with one hidden layer that has 300 hidden units, softplus activations and a Gaussian output variable that only contain the diagonal covariance because the mean is zero.

- Similarly, the inverse model $r_t(\boldsymbol{v}_t|\boldsymbol{x}, \boldsymbol{z}_t)$ is parameterized using a shallow fully-connected network with one hidden layer that has 300 hidden units, softplus activations and outputs mean and diagonal covariance of a Gaussian.

- The decoder is parameterized using a shallow fully-connected network with one hidden layer that has 400 hidden units and relu activations, and an output layer with 784 (varies depending on the dataset) output size and sigmoid activations.

(a) Before binarization (b) After binarization

Fig. 5.1 Comparison of MNIST image before and after binarization.

5.1.1.1 MNIST experiments

The MNIST dataset is a large dataset of handwritten digits which is commonly used for training image processing systems. It contains 60,000 training images and 10,000 testing images. Each image has a size of 28×28 and pixel values are in greyscale. For the subsequent experiments we discard the labels that come with images because we are not interested in classification task.

5.1.1.1.1 Data preparation

First the data in MNIST is binarized by sampling from a multivariate Bernoulli distribution where the parameter p for each pixel is determined by the greyscale value of the original image. This binarized version of MNIST is then used as the target when computing the reconstruction loss. A comparison of images before and after binarization is shown in Figure 5.1. Those images can be re-binarized every epoch to prevent overfiting issue.

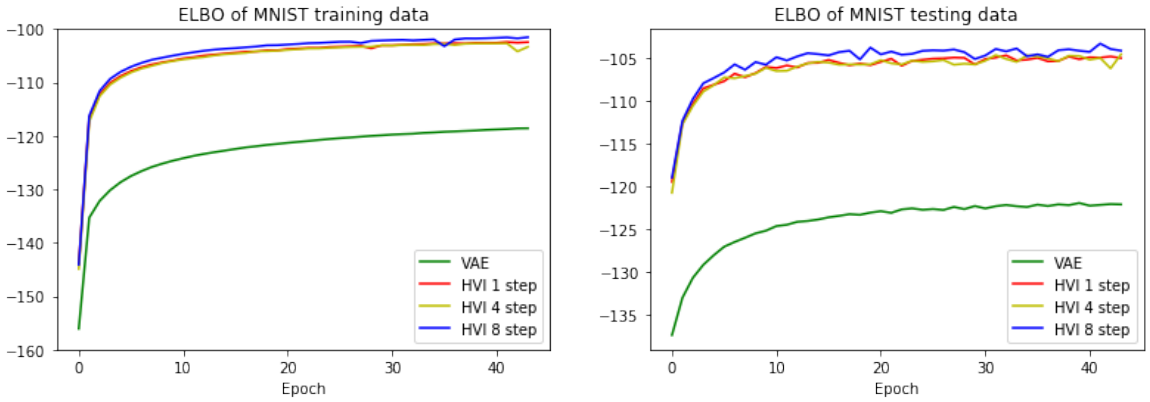
5.1.1.1.2 Implementation details

The models are trained using mini-batches of size 64. Algorithm 3 gives the ELBO for each individual image $\boldsymbol{x}$. To avoid having to loop through every image, the code is vectorized so that we can compute the ELBO for all the training images in a batch in parallel.

The gradients within each HMC leapfrog step are computed using the *torch.autograd* libraries. The gradients of each tensor variable in PyTorch are accumulated during forward pass and need to be manually zero out. This is something that is different from TensorFlow that we need to pay attention to. An Adam [6] optimizer with adaptive learning rate is used. The initial learning rate is set to be 0.003 and it is multiplied by 0.95 for every 10 epochs.

Model type	ELBO of different data type		
	Training	Testing	Log-likelihood
VAE	-118.51	-121.26	-
HVI+fully-connected:			
1 leapfrog step	-102.45	-104.96	-101.24
4 leapfrog steps	-103.34	-104.52	-103.97
8 leapfrog steps	-101.54	-104.10	-103.60

Table 5.1 Comparison between fully-connected HVI and vanilla VAE for MNIST.



(a) Training data

(b) Testing data

Fig. 5.2 Comparison of ELBO between different models.

Apart from the neural networks mentioned in section 5.1.1, the optimizer also tries to optimizer the step size ε, so that the step size can be different for each individual dimension of the position vector $\boldsymbol{z}$. We can see in the experiments that by optimizing the step size ε we can achieve a better performance when compared with using a fixed step size. The step size for each dimension is clamped between [0.001, 0.5] for numeric stability and also to control the numeric error of leapfrog integrator. If the step size is too big, the Hamiltonian will fluctuate by a large margin and the leapfrog approximation becomes highly inaccurate, whereas if the step size is too small, we can not fully utilize the dynamics of HMC sampler.

5.1.1.1.3 Results

The results of HVI model is compared with a vanilla VAE. To make the results comparable, the VAE's encoder and decoder architecture are the same as the $q(\boldsymbol{z}_0|\boldsymbol{x})$ and $p(\boldsymbol{x}|\boldsymbol{z}_t)$ in the HVI.

Model type	ELBO of different data type	
	Training	Testing
HVI+fully-connected:		
5 dimension	-126.99	-128.13
10 dimension	-105.78	-106.98
20 dimension	-101.54	-104.10

Table 5.2 Comparison between different latent variable size with 8 leapfrog steps.

The model is trained for 50 epochs and the resulting ELBO values in log scale are shown in Table 5.1 and Figure 5.2. The "log likelihood" column is estimated with importance sampling by Monte Carlo estimate of $\mathbb{E}_{p(\boldsymbol{z}|\boldsymbol{x})}p(\boldsymbol{x})$. Following algorithm 3, we can estimate this marginal as:

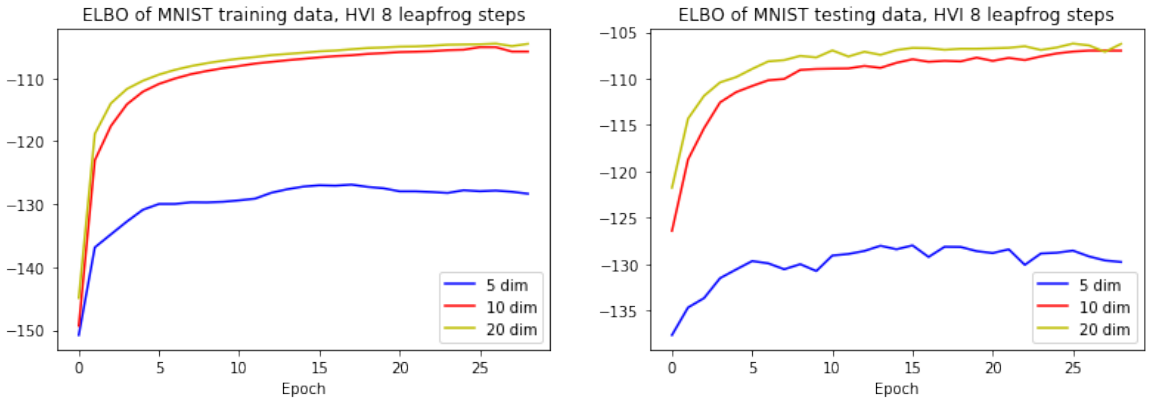
$$\begin{aligned}\mathbb{E}_{p(\boldsymbol{z}|\boldsymbol{x})}p(\boldsymbol{x}) &= \int p(\boldsymbol{x})p(\boldsymbol{z}|\boldsymbol{x})d\boldsymbol{z} = \int p(\boldsymbol{x},\boldsymbol{z})d\boldsymbol{z} = \int p(\boldsymbol{x},\boldsymbol{z}_1)r(\boldsymbol{v}_1|\boldsymbol{z}_1,\boldsymbol{x})d\boldsymbol{z}_1 d\boldsymbol{v}_1 \\ &= \int \frac{p(\boldsymbol{x},\boldsymbol{z}_1)r(\boldsymbol{v}_1|\boldsymbol{z}_1,\boldsymbol{x})}{q(\boldsymbol{z}_0|\boldsymbol{x})q(\boldsymbol{v}_1'|\boldsymbol{x})}q(\boldsymbol{z}_0|\boldsymbol{x})q(\boldsymbol{v}_1'|\boldsymbol{x})d\boldsymbol{z}_0 d\boldsymbol{v}_1'\end{aligned} \tag{5.1}$$

where $w = \frac{p(\boldsymbol{x},\boldsymbol{z}_1)r(\boldsymbol{v}_1|\boldsymbol{z}_1,\boldsymbol{x})}{q(\boldsymbol{z}_0|\boldsymbol{x})q(\boldsymbol{v}_1'|\boldsymbol{x})}$ is the importance weight of samples. To estimate the marginal, we calculate the average of the weight w with 1000 $\boldsymbol{z}_0$ and $\boldsymbol{v}_1$ sampled from $q(\boldsymbol{z}_0|\boldsymbol{x})$ and $q(\boldsymbol{v}_1'|\boldsymbol{x})$.

As shown in the Table 5.1, HVI gives much better result compared with VAE. As the number of leapfrog steps increases, the time for training an HVI model also increases, but the performance gets better. The major computation bottleneck is the gradient calculations within each leapfrog step.

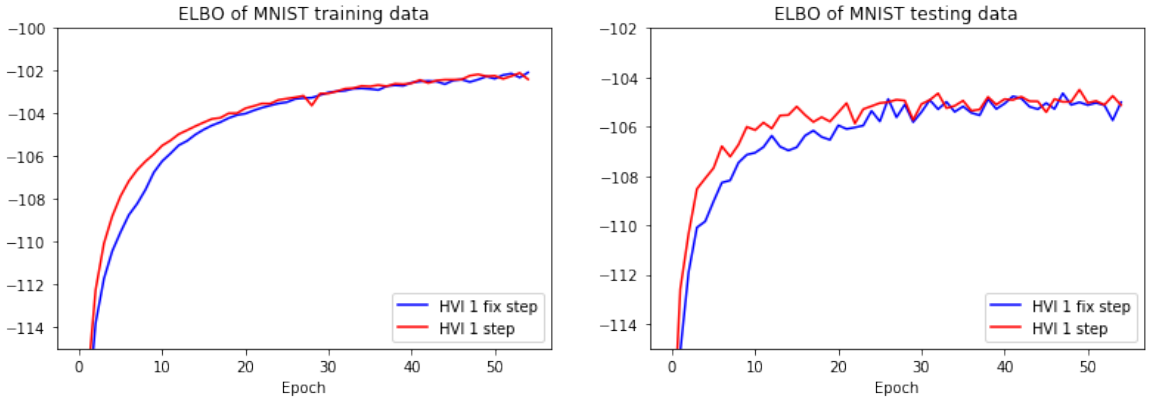
We also explored the impact of hidden variable size. We tested three different latent variable dimensions: 5, 10, 20 for HVI model with 8 leapfrog steps. The results are shown in Table 5.2 and Figure 5.3. A large latent space is able to retain more information compared with a smaller latent space, thus we observe that a latent variable $\boldsymbol{z}$ with 20 dimensions gives the best result among the three. The difference between 20 and 10 dimensions is very minimum, indicating that sometimes we can sacrifice performance for memory usage when the latter is very critical.

Another thing we explored is the effect of optimizing step size ε. We use HVI model with 1 leapfrog step to explore the effect of optimizing step size ε versus if it is fixed at 0.01 for all the dimensions. The results are shown in Figure 5.4. We can see if ε is optimized, we are able to converge to the highest ELBO faster. However, the final ELBO values between two models are almost identical. This is because in the leapfrog algorithm, we need to multiply ε



(a) Training data (b) Testing data

Fig. 5.3 Comparison of ELBO between different latent variable size with 8 leapfrog steps.



(a) Training data (b) Testing data

Fig. 5.4 Comparison of ELBO between models with fixed/optimized step size with 1 leapfrog step.

with $\frac{1}{M}$ as shown in the equation below. If we optimize the $\frac{\varepsilon}{M}$ term jointly, it has the similar effect of only optimizing ε. Although in principle they are not exactly the same because M also determines the distribution where we need to sample the initial momentum from.

$$\theta(t+\varepsilon) = \theta(t) + \varepsilon \frac{\boldsymbol{p}(t+\frac{\varepsilon}{2})}{\boldsymbol{M}} \tag{5.2}$$

To further compare the model performance, we sample from a multivariate unit Gaussian which has the same dimension as $\boldsymbol{z}_t$ and pass it through the trained decoder network. The resulting images are shown in Figure 5.5. As we can see, all four images are not very clear but the first image is obviously more blurred than the remaining three images. The difference between the second, third and fourth images are not significant, which is an expected behavior since their difference in ELBO value is not big.

(a) VAE

(b) HVI with 1 leapfrog steps

(c) HVI with 4 leapfrog steps

(d) HVI with 8 leapfrog steps

Fig. 5.5 Comparison of generated images between different models.

5.1.2 Convolutional neural network configuration

Convolutional neural networks (CNN) are more powerful than fully-connected neural networks in particular for computer vision and image processing domain. In this section we will explore the performance of HVI model with convolutional/deconvolutional layers.

- The initial distribution $q(\boldsymbol{z}_0|\boldsymbol{x})$ is parameterized with three convolutional layers followed by two fully-connected layers. The convolutional layers have 5×5 filters, [16,32,32] feature maps, stride of 2 and softplus activations. There is also a padding of 2 for each convolutional layer. The fully-connected layers have 300 hidden units in the hidden layer and softplus activations. The output are mean and diagonal covariance of a Gaussian. The default dimension of output variable ($\boldsymbol{z}$) is 20.
- The forward model $q_t(\boldsymbol{v}_t^{'}|\boldsymbol{x})$ is the same as in section 5.1.1. It is parameterized using a shallow fully-connected network with one hidden layer that has 300 hidden units, softplus activations and a Gaussian output variable that only contain the diagonal covariance because the mean is zero.
- The inverse model $r_t(\boldsymbol{v}_t|\boldsymbol{x}, \boldsymbol{z}_t)$ also remains the same as in section 5.1.1. It is parameterized using a shallow fully-connected network with one hidden layer that has 300 hidden units, softplus activations and outputs mean and diagonal covariance of a Gaussian.
- The decoder architecture mirrors the inference model $q(\boldsymbol{z}_0|\boldsymbol{x})$. It has two fully-connected layers followed by three deconvolutional layers. To make the output size of each deconvolutional layer the same as convolutional layers of the inference model, an output padding of 1 is also added for the second and third deconvolutional layers. This architecture is similar to [2].

5.1.2.1 MNIST experiments

5.1.2.1.1 Data preparation

Again the data in MNIST is binarized by sampling from a multivariate Bernoulli distribution.

5.1.2.1.2 Implementation details

The models are trained using mini-batches of size 64. PyTorch does not have automatic padding functions like “VALID” or “SAME” in TensorFlow, which means the padding size need to be manually decided. The output size of convolutional layers are [14x14,7x7,4x4], and the output size of deconvolutional layers are [7x7,14x14,28x28].

Similar to the settings in the fully-connected configuration, an Adam optimizer with adaptive learning rate is used to optimize both the neural networks and the step size ε. The initial learning rate is set to be 0.0002 and it is multiplied by 0.95 for every 10 epochs. The step size for each dimension is clamped between [0.001, 0.5].

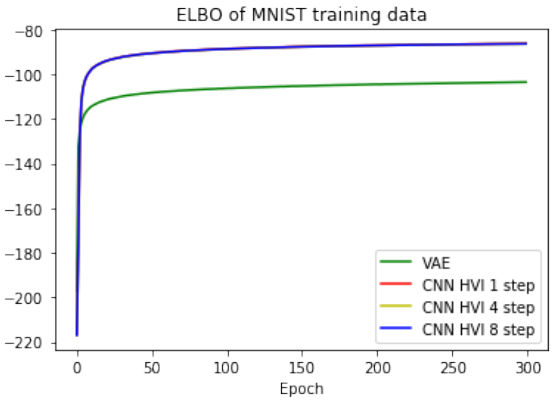
5.1.2.1.3 Results

The results of HVI model is compared with a convolutional VAE whose encoder and decoder has the same architecture as $q(\boldsymbol{z}_0|\boldsymbol{x})$ and $p(\boldsymbol{x}|\boldsymbol{z}_t)$. Such special VAE has been explored by papers like [11] where the convolutional VAE was used to train on ImageNet.

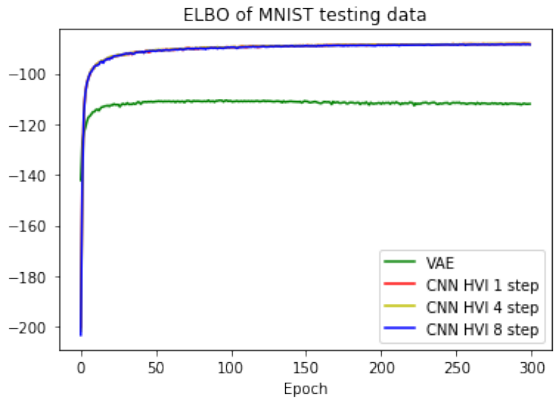
The results are obtained by training for 300 epochs and are shown in Table 5.3 and Figure 5.6. We can see HVI model outperforms VAE by a large margin, but the difference of between three HVI models is very minimum. Using 4 leapfrog steps gives the best result, which is not expected. This could be because the step size ε is a bit too big so that the leapfrog approximation which is used to discretize Hamilton's equations dose not perform well when there are too many steps. The samples gradually wander off the correct trajectory as the number of leapfrog steps increases. A simpler explanation is that their difference is so small that we cannot say HVI with 4 leapfrog steps is necessarily the best due to the stochastic nature of the training process.

We also explored the impact of latent space dimension. As shown in Table 5.4 and Figure 5.7, we observe the same trend as we observed in the fully-connected neural network configuration. A latent space with 20 dimensions gives the best results.

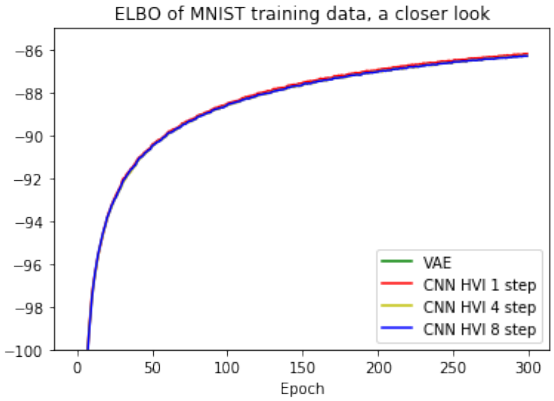
After training, we draw random samples from a unit Gaussian and pass them through the trained decoder to generate a set of images. A comparison between different models is shown in Figure 5.8. All four models are able to generate images that are much clearer than images in Figure 5.5, indicating that convolutional neural networks do a better job in capturing high-level features of images compared with fully-connected neural networks. In addition, if we compare these four images in Figure 5.8, we can see the first image generated by convolutional VAE is more blurred than the remaining three images, especially when we look at the boundaries between the digits and background, where the boundaries of convolutional HVI models tend to be sharper.



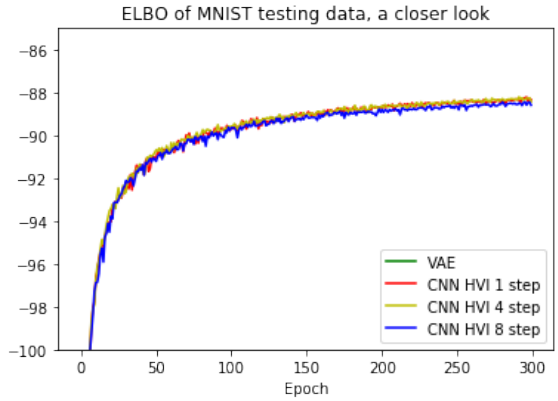
(a) Training data



(b) Testing data

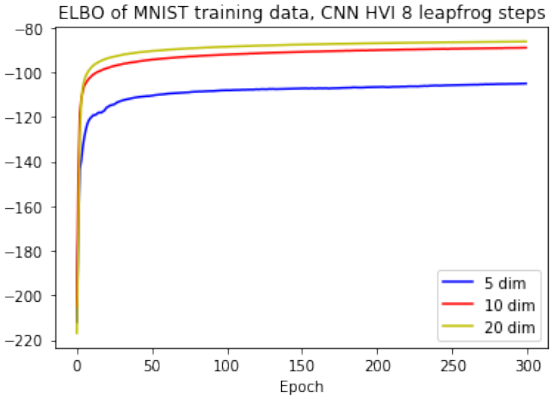


(c) Training data, a closer look

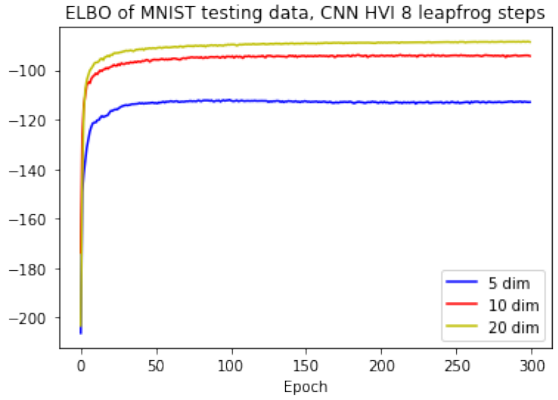


(d) Testing data, a closer look

Fig. 5.6 Comparison of ELBO between different models with convolutional layers.



(a) Training data



(b) Testing data

Fig. 5.7 Comparison of ELBO between different latent variable size with 8 leapfrog steps.

(a) convolutional VAE

(b) convolutional HVI with 1 leapfrog step

(c) convolutional HVI with 4 leapfrog steps

(d) convolutional HVI with 8 leapfrog steps

Fig. 5.8 Comparison of generated images between different models.

Model type	ELBO of different data type		
	Training	Testing	Log-likelihood
VAE	-105.17	-110.78	-
HVI+fully-connected:			
1 leapfrog steps	-86.49	-88.34	-88.36
4 leapfrog steps	-86.29	-88.21	-88.19
8 leapfrog steps	-86.29	-88.23	-88.30

Table 5.3 Comparison between convolutional HVI and convolutional VAE.

Model type	ELBO of different data type	
	Training	Testing
HVI+convolutional :		
5 dimension	-105.08	-112.87
10 dimension	-89.04	-94.04
20 dimension	-86.29	-88.23

Table 5.4 Comparison between different latent variable size with 8 leapfrog steps.

5.1.2.2 CIFAR-10 experiments

The CIFAR-10 dataset is also a collection of images that are commonly used in machine learning community. It consists of 60000 32×32 colour images in 10 classes, with 6000 images for each class. Some sample images can be seen in Figure 5.9. The major difference between CIFAR-10 and MNIST is that each image in CIFAR-10 has 3 channels representing color coding in RGB, whereas images in MNIST only have one channel. The CIFAR-10 dataset is a good dataset for testing whether the HVI model we initially developed for MNIST generalizes well to other datasets.

5.1.2.2.1 Data preparation

The data is downloaded directly from the torchvision package of PyTorch. The output of torchvision datasets are PILImage images of range [0, 1] which means they are already normalized. For the subsequent experiments, we will use these normalized images as the targets during supervised training.

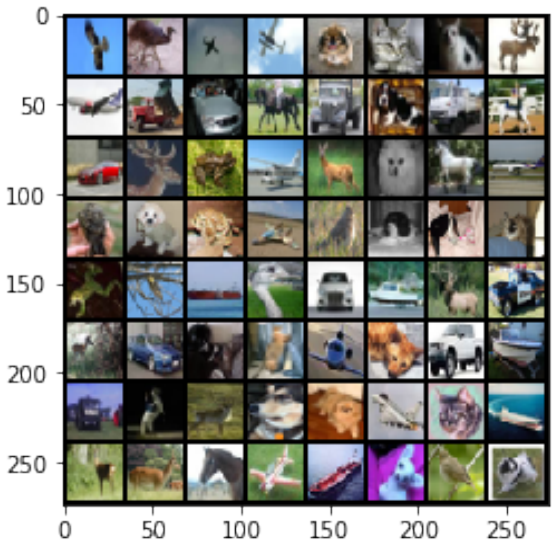


Fig. 5.9 Sample images from CIFAR-10 dataset.

5.1.2.2.2 Implementation details

To generate CIFAR-10 images we need to modify the convolutional and deconvolutional architecture that we used for MNIST dataset. The architecture is described below.

- The initial distribution $q(\boldsymbol{z}_0|\boldsymbol{x})$ is parameterized with four (instead of three) convolutional layers followed by two fully-connected layers. The convolutional layers have 3×3 filters, [16,32,32,16] feature maps, stride of [1,2,1,2] and relu activations. There is a padding of 1 for each convolutional layer. The fully-connected layers have 512 hidden units in the hidden layer and relu activations. The output are mean and diagonal covariance of a Gaussian. We also added batch normalization after every convolutional and fully-connected layer. The default dimension of output variable ($\boldsymbol{z}$) is increased from 20 to 40. We increase the number of convolutional layers and reduce the size of filter so that we can capture more details in the image since CIFAR-10 images are more sophisticated than MNIST images. The dimension of latent space is also increased because we have more information to encode.

- The forward model $q_t(\boldsymbol{v}_t^{'}|\boldsymbol{x})$ and the inverse model $r_t(\boldsymbol{v}_t|\boldsymbol{x},\boldsymbol{z}_t)$ have the same architecture as before except that we increased the number of hidden units from 300 to 512.

- The decoder architecture mirrors the inference model $q(\boldsymbol{z}_0|\boldsymbol{x})$. It has two fully-connected layers followed by four deconvolutional layers. Each layer is followed by a batch normalization operation. An output padding of 1 is also added for the first

and third deconvolutional layers. The output of the last layer is passed through sigmoid activations.

There are two choices for the likelihood function $p(\boldsymbol{x}|\boldsymbol{z})$: the first one is a multivariate Bernoulli whose probabilities are computed from $\boldsymbol{z}$ with a neural network:

$$\log p(\boldsymbol{x}|\boldsymbol{z}) = \sum_{i=1}^{D} x_i \log y_i + (1-x) \cdot \log(1-y_i) \tag{5.3}$$

where $\boldsymbol{y}$ is the output from the decoder which has the same size as the target. For CIFAR-10 images, this size is $32 \times 32 \times 3 = 3072$ (in comparison, for MNIST image it is $28 \times 28 = 784$). This is the choice we used for the previous MNIST experiments.

Alternatively, we can use a multivariate Gaussian with a diagonal covariance to express the likelihood function:

$$\log p(\boldsymbol{x}|\boldsymbol{z}) = \log \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{\sigma^2 I}) \tag{5.4}$$

where $\boldsymbol{\mu}$ and $\boldsymbol{\sigma^2}$ are outputs from the decoder. Intuitively we are measuring the "distance" of the target from the output, which is represented by a multivariate Gaussian.

The first choice is best for targets with binary data, like the binarized MNIST we used for previous experiments. The second choice is better suited for continuous data like images. However, for the subsequent experiments we will use the first choice as we discovered that it already works pretty well during training in practice. To do that, we have sigmoid activations at the end of the decoder ouput layer to squeeze the output into the range [0, 1].

To test the performance of our HVI model, we also implemented a VAE with the same convolutional/deconvolutional architecture and their results are compared in the next section.

An Adam optimizer with adaptive learning rate is used to optimize both the neural networks and the step size ε. The initial learning rate is set to be 0.0001 and it is multiplied by 0.95 for every 10 epochs. The initial step size ε is set to be $e^{-4.6} = 0.01$ for every dimension. The step size for each dimension is clamped between [0.001, 0.5].

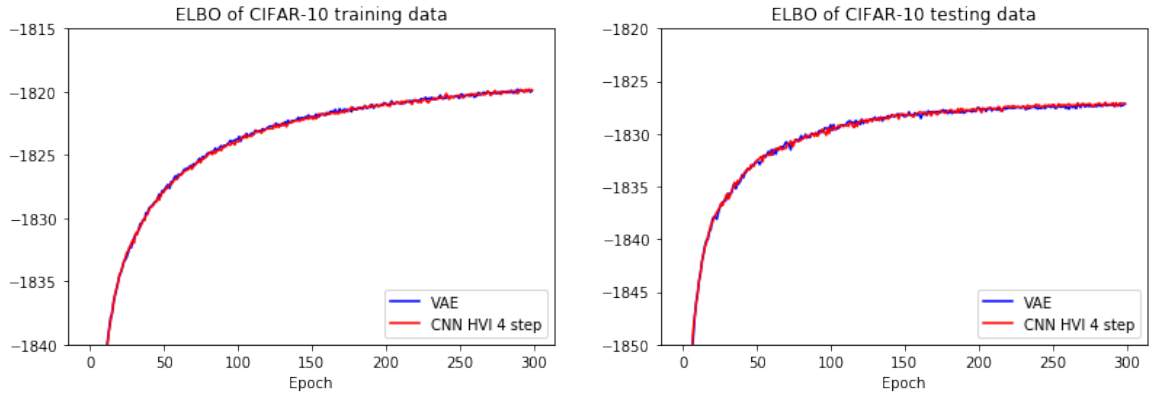
5.1.2.2.3 Results

The results are obtained by training for 300 epochs. A comparison between convolutional VAE and convolutional HVI is shown in Table 5.5 and Figure 5.10. Due to time constraint, we only tested the convolutional HVI model with 4 leapfrog steps. From Figure 5.10 we can't see any huge difference between the ELBO, though convolutional HVI does give a slightly better result as can be seen from Table 5.5.

After training, the trained decoders are used to generate 64 images and the results are shown in Figure 5.11. As we can see, there is no significant difference between the images,

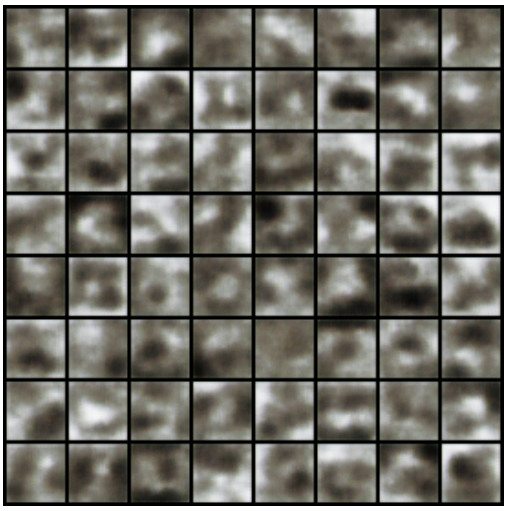
Model type	ELBO of different data type		
	Training	Testing	Log-likelihood
VAE	-1819.92	-1827.13	-
HVI+fully-connected: 4 leapfrog steps	-1819.85	-1827.10	-1826.92

Table 5.5 Comparison between convolutional HVI and convolutional VAE.

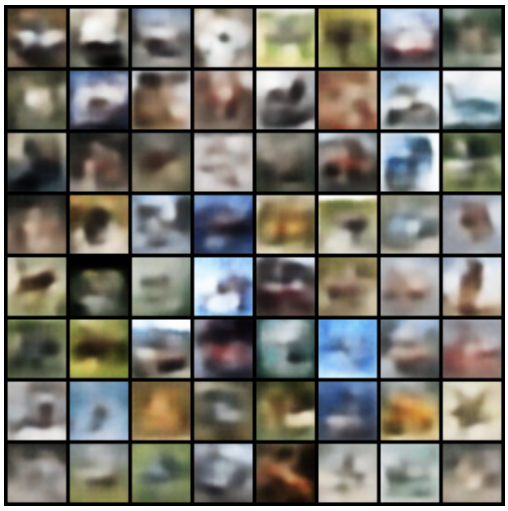


(a) Training data (b) Testing data

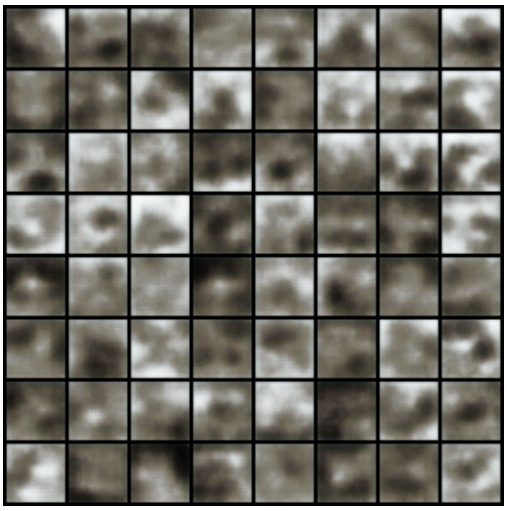
Fig. 5.10 Comparison of CIFAR-10 ELBO between convolutional VAE and convolutional HVI.



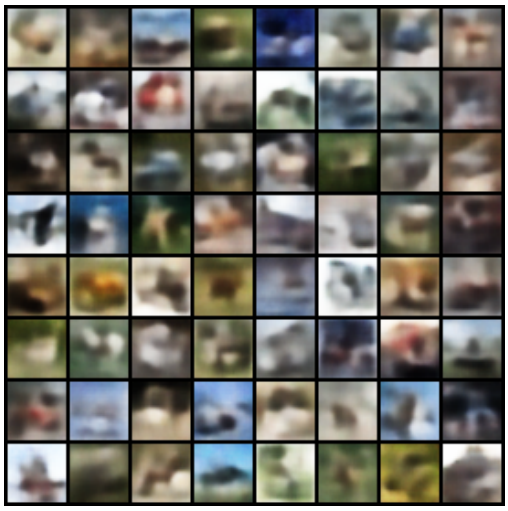
(a) convolutional VAE, 1st epoch



(b) convolutional VAE, 300th epoch



(c) convolutional HVI, 1st epoch



(d) convolutional HVI, 300th epoch

Fig. 5.11 Comparison of generated images between different models at different epoch.

though 5.11d does seem to have more details retained compared with Figure 5.11b. In summary, HVI does not seem to have a clear advantage on CIFAR-10 dataset, we are able to get a small improvement but need to sacrifice computation time, which can be not worthy under certain circumstances.

5.2 Riemannian Manifold Hamiltonian variational inference (RMHVI)

In this section we will attempt to implement Riemannian Manifold Hamiltonian variational inference (RMHVI) and test it on the MNIST dataset.

5.2.1 Fully-connected neural network configuration

The most significant difference is that in RMHVI, we no longer need a forward model $q_t(\boldsymbol{v}_t^{'}|\boldsymbol{x})$. It is replaced by a function that calculates the Fisher information matrix $G(\boldsymbol{z}_0)$. This matrix has the same functionality as the diagonal covariance matrix from the forward model does. We draw initial samples of the momentum variable $\boldsymbol{v}_t^{'}$ from a multivariate Gaussian distribution $\mathcal{N}(0, G(\boldsymbol{z_0}))$.

The overall architecture is shown in Figure 4.3.

- The initial distribution $q(\boldsymbol{z}_0|\boldsymbol{x})$ is parameterized with a fully-connected neural network with two hidden layers, softplus activations and 300 hidden units in each hidden layer. The output are mean and diagonal covariance of a Gaussian. The default dimension of output variable ($\boldsymbol{z}$) is 20.
- The inverse model $r_t(\boldsymbol{v}_t|\boldsymbol{x}, \boldsymbol{z}_t)$ is parameterized using a shallow fully-connected network with one hidden layer that has 300 hidden units, softplus activations and outputs mean and diagonal covariance of a Gaussian.
- The decoder is parameterized using a shallow fully-connected network with one hidden layer that has 400 hidden units and relu activations, and an output layer with 784 output size and sigmoid activations.

5.2.1.1 MNIST experiments

5.2.1.1.1 Data preparation

The data in MNIST is binarized by sampling from a multivariate Bernoulli distribution.

5.2.1.1.2 Implementation details

The number of fixed point iterations is typically set to be around 5 and 6 to achieve convergence. For the second fixed point iteration (starting from state 14 of algorithm 5), we reduce this number to 1 as it is very costly. When computing the Fisher information matrix $G(\boldsymbol{z})$, by default we draw 5 samples $\boldsymbol{y}$ from the multivariate Bernoulli distribution to calculate the

Model type	ELBO of different data type		
	Training	Testing	Log-likelihood
VAE	-118.51	-121.26	-
HVI+fully-connected: 1 leapfrog step	-102.45	-104.96	-101.24
RMHVI+fully-connected: 1 leapfrog step	-108.73	-110.94	-110.35

Table 5.6 Comparison between VAE, HVI and RMHVI with 1 leapfrog step.

variance of the gradients, as we found that this is the major computation bottleneck during training so we keep this number small for now. We have also decided to keep the matrix $G(\boldsymbol{z})$ as a diagonal matrix for now.

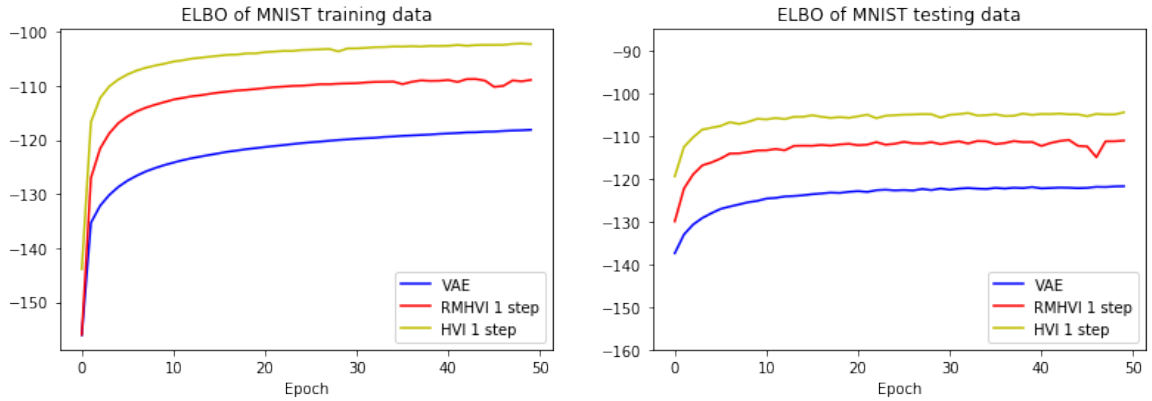
All the gradients are computed using the autograd package in PyTorch. Though we have an explicit expression of the gradient of the Hamiltonian w.r.t. $\boldsymbol{q}$ as shown in equation 4.9, we decide not to compute the trace of the matrix but to differentiate the Hamiltonian w.r.t. $\boldsymbol{q}$ directly as this way is more efficient.

5.2.1.1.3 Results

The results are obtained by training for 50 epochs and are shown in Table 5.6 and Figure 5.12. We can see that when using only 1 leapfrog step, RMHVI model is still able to outperform vanilla VAE, but it's worse than HVI model.

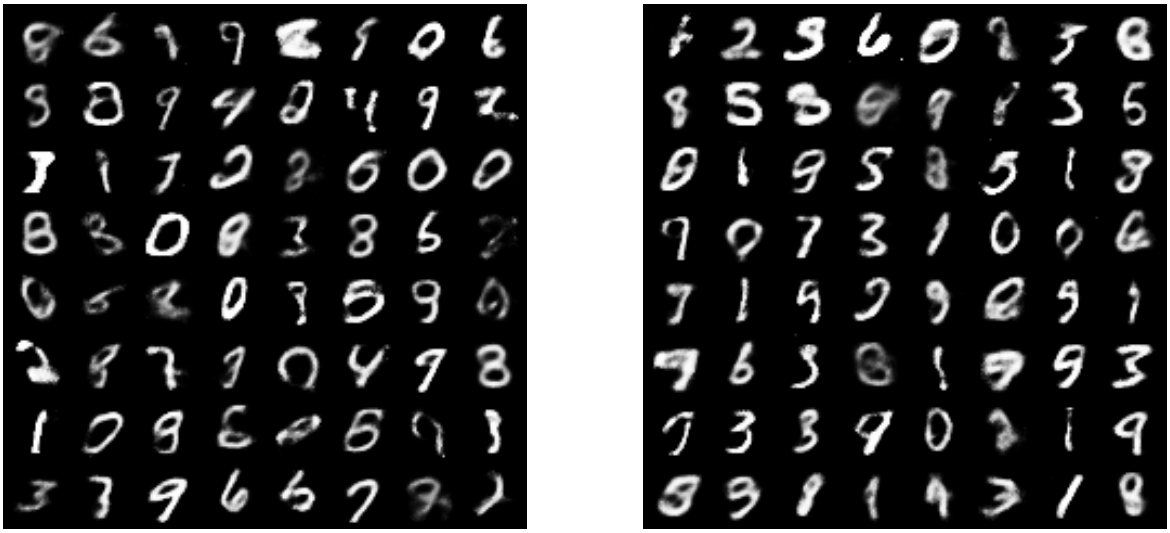
There are several possible explanations. **1)** First, we are using a diagonal Fisher information matrix $G(\boldsymbol{z})$ which assumes independence between dimensions, however, in reality the correlation between dimensions does exist, so our simplified assumption makes the matrix $G(\boldsymbol{z})$ inefficient in adapting to the natural geometric structure of the density model $p(\boldsymbol{z})$ in Riemannian manifold. **2)** We are taking 5 samples $\boldsymbol{y}$ from the multivariate Bernoulli distribution when calculating $G(\boldsymbol{z})$, which is a bit small for obtaining a Monte Carlo estimate of the variance. However the number of samples n heavily influences the computation overhead, thus if we set n too big, the computation overhead will undermine RMHMC's benefits.

Two sets of generated images after 50 epochs are shown in Figure 5.13. Both images can reconstruct the original MNIST images reasonably well, with the second image generated by RMHVI looking slightly better than the first one generated by HVI. This contradicts the fact that HVI gives a better ELBO value. Thus we have reason to believe that the reconstruction error of RMHVI is not much worse than that of HVI, it might be even better.



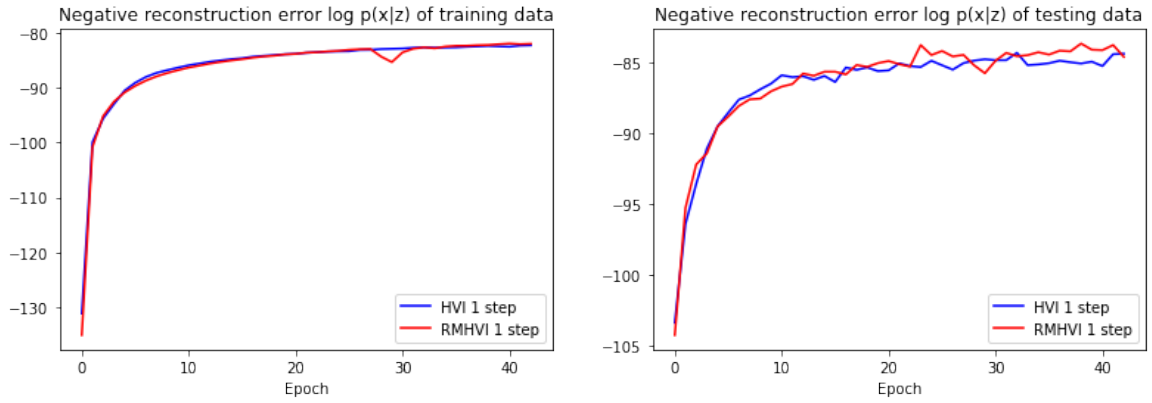
(a) Training data (b) Testing data

Fig. 5.12 Comparison of ELBO between different models, 1 leapfrog step.



(a) HVI with 1 leapfrog step (b) RMHVI with 1 leapfrog step

Fig. 5.13 Comparison of generated images between HVI and RMHVI.



(a) Training data (b) Testing data

Fig. 5.14 Comparison of negative reconstruction error between HVI and RMHVI.

To further compare the performance of HVI and RMHVI models and confirm our hypothesis, we store the average negative reconstruction error $\log p(\boldsymbol{x}|\boldsymbol{z})$ separately and they are compared in Figure 5.14. As we can see there is no significant difference in the reconstruction error, meaning that HVI is able to achieve a smaller KL-divergence: $-D_{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x}^{(i)})||p_\theta(\boldsymbol{z}))$, which leads to a higher ELBO value as we see in Figure 5.12.

$$\mathcal{L}(\theta,\phi;\boldsymbol{x}^{(i)}) = -D_{KL}(q_\phi(\boldsymbol{z}|\boldsymbol{x}^{(i)})||p_\theta(\boldsymbol{z})) + \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_\theta(\boldsymbol{x}^{(i)}|\boldsymbol{z})\right] \tag{5.5}$$

5.2.2 Convolutional neural network configuration

5.2.2.1 MNIST experiments

5.2.2.1.1 Implementation problems

In this section, we attempt to implement the RMHVI model with convolutional neural network configuration. This is a failed attempt and we will explore the possible reasons. The model architecture is shown in Figure 5.15.

The first thing we notice during implementation is that, if we use the default initialization mechanism of convolutional neural networks in PyTorch, we encounter numerical errors during training. The elements of Fisher information matrix $G(\boldsymbol{z})$ become very small, the mean of diagonal elements is around 6.67×10^{-6}.

1) Therefore we switch to Xavier initialization, which is a method that automatically determines the scale of initialization based on the number of input and output neurons. It was first introduced in paper [4]. We use the *xavier_normal_* function which initializes the weights by sampling from a zero mean Gaussian with variance $Var(\boldsymbol{w})$. The variance is

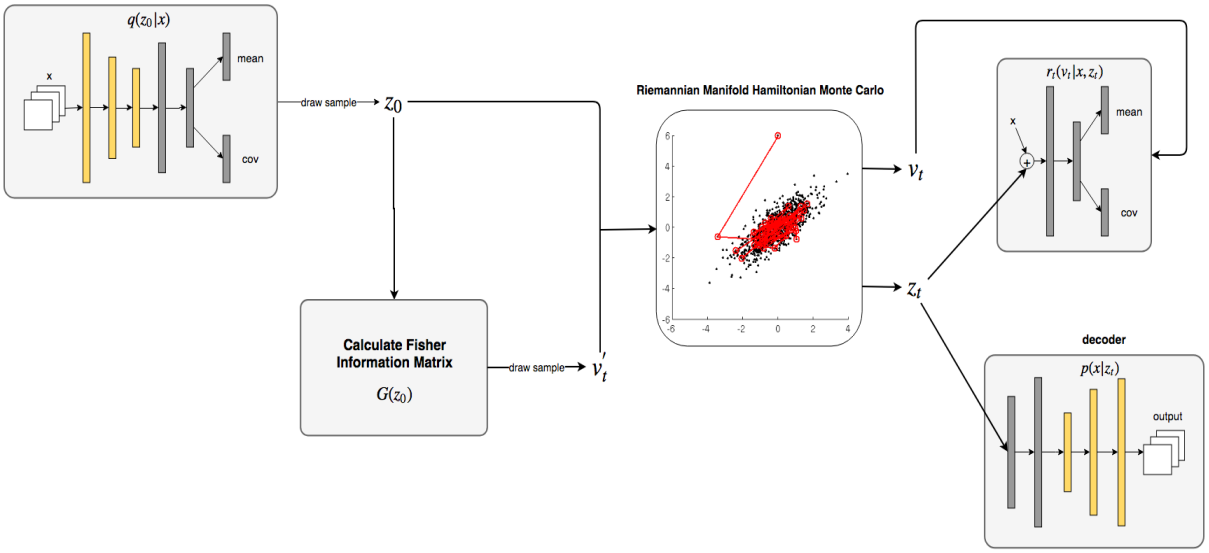


Fig. 5.15 Graphical representation of RMHVI model with convolutional networks.

determined by the following expression:

$$Var(\boldsymbol{w}) = \frac{2}{n_{in} + n_{out}} \tag{5.6}$$

where n_{in} is the number of input neurons and n_{out} is the number of input neurons of the next layer. Xavier initialization ensures the variance of ouput from a layer is the same as the variance of input to this layer. After doing this, the mean of diagonal elements in $G(\boldsymbol{z})$ increases to around 3.78×10^{-4}, which is still too small.

2) We then try to clamp the value of diagonal elements in $G(\boldsymbol{z})$ into a reasonable range. But the result is that all elements have the same value, which is the minimum value of the range. This is undesired because it forces the Riemannian manifold to have the same geometric structure everywhere, which is incorrect.

3) We also try to divide the diagonal elements by the minimum value among all the elements, expecting all the elements to fall in a reasonable range while in the same time remain the relative ratio among each other. But this could result in some massive elements.

4) Finally, we try to add a constant to all the diagonal elements. From the fully-connected configuration we can obtain the mean of the diagonal elements, which is around 1.6. Therefore we add this quantity to all the diagonal elements. This would have a similar result as in solution **2)**, because now all the diagonal elements are around 1.6.

In summary, it is very difficult to obtain a reasonable $G(\boldsymbol{z})$ to train the model. The reason is straightforward: we have a much more complicated decoder, indicating that the Riemannian manifold for $p(\boldsymbol{x}|\boldsymbol{z})$ has more geometric details that can be hard to grasp.

In comparison, when we use fully-connected neural networks as decoder, the underlying Riemannian structure is much easier to be learned because we have fewer nonlinearities.

(All the detailed PyTorch implementation of the models mentioned in this chapter can be found at: https://github.com/ericZYZ/HVI-RMHVI-for-VAE)

Chapter 6

Summary and Future Work

6.0.1 Summary

The dissertation addresses one important problem in machine learning, which is to approximate the intractable posterior when applying Bayesian inference. In particular, we use VAE as the framework for testing the performance of different algorithms because the vanilla VAE suffers from the over-simplified assumption of the latent space distribution. Naturally we can use it as a baseline when comparing model performance.

We started by introducing this problem and the motivation of this dissertation in Chapter 1. Then in Chapter 2 we introduce the necessary background knowledge for our methodology. There have been some existing methods in the current literature that aim to solve the same issue in various angles, and they are briefly explained in Chapter 3. Chapter 4 and 5 describe the model architecture, implementation details and the corresponding results.

To conclude, we have tested two frameworks: HVI and RMHVI. They follow the same principle introduced in paper [16], which is to combine variational inference with MCMC methods to enrich the posterior family for VAE. Two different neural network configurations are implemented for HVI and they both outperform vanilla VAE for two different datasets. We also discover the increase in leapfrog steps and latent variable dimensions are able to improve HVI performance, but at the same time more computation time and memory usage are needed. The fully-connected neural network configuration is successfully implemented for the RMHVI model and it outperforms vanilla VAE as well. However it does not perform as good as the HVI model, which can be due to various reasons detailed in section 5.2.1.1.3. We also attempt to implement convolutional RMHVI model but failed to come up with a workable solution mainly because the geometric structure of the convolutional decoder is very complicated in Riemannian manifold.

6.0.2 Future work

One obvious future work is to to explore more stable implementation of the convolutional RMHVI model. We have outlined how we attempted to overcome the problem and the possible reasons it failed in section 5.2.2. With more extensive experiments we may be able to come up with a workable implementation.

So far we have been using Fisher information matrix as our Riemannian metric. However, there can be other types of Riemannian metric as well. One possible extension of the dissertation is to replace $G(\boldsymbol{z})$ with different types of Riemannian metric and see whether the performance of RMHVI improves.

Ideally, RMHVI should be able to outperform HVI, however in our experiment results we see the opposite. One way to eliminate the potential engineering issues in the implementation of the RMHVI model, we can first test it on some simpler problems like the Bayesian logistic regression problem shown in paper [3]. Once we have proven the effectiveness of RMHVI in such problems then we can test it on more complicated datasets like MNIST.

Another optimization can be made is the training speed. So far we have vectorized the calculation of the ELBO in algorithm 3 and algorithm 4 for all the training images. However there are certain places in the code where we did not fully vectorize. For example, during the calculation of $G(\boldsymbol{z})$ we need to use a for loop for n samples. Some of the limitations come from PyTorch itself. If a different library is used we may be able to mitigate such issues and make the training process faster.

References

[1] Burda, Y., Grosse, R. B., and Salakhutdinov, R. (2015). Importance weighted autoencoders. *CoRR*, abs/1509.00519.

[2] Dosovitskiy, A., Springenberg, J. T., and Brox, T. (2015). Learning to generate chairs with convolutional neural networks. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1538–1546.

[3] Girolami, M. and Calderhead, B. (2011). Riemann manifold langevin and hamiltonian monte carlo methods. *Journal of the Royal Statistical Society Series B*, 73(2):123–214.

[4] Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS'10). Society for Artificial Intelligence and Statistics*.

[5] Kingma, D. (2015). Slides in black box inference and learning workshop, nips'15, montreal, canada.

[6] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

[7] Kingma, D. P., Salimans, T., Jozefowicz, R., Chen, X., Sutskever, I., and Welling, M. (2016). Improved variational inference with inverse autoregressive flow. In Lee, D. D., Sugiyama, M., Luxburg, U. V., Guyon, I., and Garnett, R., editors, *Advances in Neural Information Processing Systems 29*, pages 4743–4751. Curran Associates, Inc.

[8] Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. *CoRR*, abs/1312.6114.

[9] Leimkuhler, B. J. and Reich, S. (2004). *Simulating Hamiltonian dynamics*. Cambridge monographs on applied and computational mathematics. Cambridge Univ., Cambridge.

[10] Paisley, J. W., Blei, D. M., and Jordan, M. I. (2012). Variational bayesian inference with stochastic search. In *ICML*. icml.cc / Omnipress.

[11] Pu, Y., Gan, Z., Henao, R., Yuan, X., Li, C., Stevens, A., and Carin, L. (2016). Variational autoencoder for deep learning of images, labels and captions. In Lee, D. D., Sugiyama, M., Luxburg, U. V., Guyon, I., and Garnett, R., editors, *Advances in Neural Information Processing Systems 29*, pages 2352–2360. Curran Associates, Inc.

[12] Rainforth, T., Kosiorek, A. R., Le, T. A., Maddison, C. J., Igl, M., Wood, F., and Teh, Y. W. (2018). Tighter variational bounds are not necessarily better. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pages 4274–4282.

[13] Rao, C. R. (1992). Information and the accuracy attainable in the estimation of statistical parameters. In *Springer Series in Statistics*, pages 235–247. Springer New York.

[14] Rezende, D. J. and Mohamed, S. (2015). Variational inference with normalizing flows. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 1530–1538. JMLR.org.

[15] Rezende, D. J., Mohamed, S., and Wierstra, D. (2014). Stochastic backpropagation and approximate inference in deep generative models. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, pages 1278–1286.

[16] Salimans, T., Kingma, D. P., and Welling, M. (2015). Markov chain monte carlo and variational inference: Bridging the gap. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 1218–1226. JMLR.org.

[17] Wikipedia contributors (2018a). Metric tensor — Wikipedia, the free encyclopedia. [Online; accessed 11-August-2018].

[18] Wikipedia contributors (2018b). Riemannian manifold — Wikipedia, the free encyclopedia. [Online; accessed 11-August-2018].