Improved Autoguides for Probabilistic Programs

Jiaqing Xie

4th Year Project Report
Electronics and Computer Science
School of Informatics
University of Edinburgh

2023

Abstract

Variational inference (VI) can usually be used to simulate a complex posterior distribution. Using automatic differentiation in VI facilitates inference as optimization since it does not require specifying an importance distribution, which is implemented in Pyro as autoguides. Pyro's two simplest autoguides are based on the full-rank and mean-field assumptions. However, the mean-field assumption does not account for dependent variables while full-rank assumptions lack the enrichness choices of covariance matrices, which may compromise inference performance. In this project, we developed six different covariance matrices based autoguides. We proposed another two autoguides based on the inverse model dependencies to guide the poesterior more accurately. We have evaluated our new autoguides on different datasets, the degree of improvement when using new autoguides, and the relevant scenarios for the autoguides. In general, the autoguide based on a covariance matrix is appropriate for low dimensional tasks, whereas the autoguide based on an inversed dependency model is appropriate for high dimensional tasks with enriched dependency structures. It is important to note that autoguides are not suitable for many deep generative models such as variational encoders.

Research Ethics Approval

This project was planned in accordance with the Informatics Research Ethics policy. It did not involve any aspects that required approval from the Informatics Research Ethics committee.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Jiaqing Xie*)

Acknowledgements

It has been a real challenge to do this project. When I worked on this project, I often did not make progress, and it was very difficult to realize the code. Because of insufficient coding, mathematical, and logic knowledge, I sometimes felt pressure and failed to reproduce some algorithms and source code myself. I nearly collapsed and fell into depression for two months. Now that I have followed people's advice, I feel much better and would like to thank them all individually here.

I am very grateful to my supervisor Prof. Siddharth Narayanaswamy, and my second marker Prof. Mary Cryan, for helping me achieve small goals instead of being unable to accomplish anything. In the end, there are some significant improvements based on their careful advice, although the results are not as good as expected. Thanks to Dr. Eli Bingham, the developer and leader of Pyro, for helping me solve a lot of conceptual problems. He also explained some of his ideas as I worked on Pyro. He played an extremely crucial role in assisting me understand Pyro's entire framework. My thanks also extend to Dr. Tuan Anh Le for helping me understand the paper. I had a lot of misguided ideas about the inference compilation part of the paper. Additionally, I would like to express my gratitude to Dr. Nicholas Polydorides, who taught me advanced algebraic techniques.

As well, I'd like to express my gratitude to my parents, Mr. Shengliang Xie and Mrs. Zhaodi Shan, as well as my two best friends, Jingwen Pan and Jiaqi Huang. They helped me when I nearly collapsed. Thanks to all of you, this dissertation would not have been possible without your assistance and understanding.

Table of Contents

1 Introduction **1**
- 1.1 Motivation . . . 1
- 1.2 Objectives . . . 1
- 1.3 Main contributions . . . 2
- 1.4 Report outlines . . . 2

2 Background **3**
- 2.1 What is inference . . . 3
- 2.2 Bayesian inference . . . 4
  - 2.2.1 Concepts . . . 4
  - 2.2.2 Extension on Bayes . . . 5
- 2.3 Variational inference . . . 6
  - 2.3.1 Concepts . . . 6
  - 2.3.2 Optimization problem . . . 7
  - 2.3.3 Variational expectation maximization . . . 8
  - 2.3.4 Forms of variational inference . . . 8
- 2.4 Probabilistic Programming . . . 10
  - 2.4.1 Definition and Languages . . . 10
  - 2.4.2 Main PPL: Pyro . . . 10
- 2.5 Bayesian Networks (BN) . . . 12
  - 2.5.1 Definition . . . 12
  - 2.5.2 Important Properties of BN . . . 12
  - 2.5.3 BN with Variational Inference . . . 13

3 ADVI and potential improvements **14**
- 3.1 Problem Restatement . . . 14
- 3.2 ADVI . . . 15
- 3.3 Learning Inverse Dependencies . . . 18
  - 3.3.1 Observed to latent (Upstream) . . . 19
  - 3.3.2 Latent to observed (Downstream) . . . 20
- 3.4 Covariance Matrices' Variants . . . 22
  - 3.4.1 PolyDiagNorm . . . 22
  - 3.4.2 SymmetricNorm . . . 23
  - 3.4.3 LowRankNorm . . . 23
  - 3.4.4 BlockDiagNorm . . . 24

3.4.5 ToeplitzNorm . . . 24
3.4.6 CirculantNorm . . . 25
3.5 Inference Compilation . . . 25
3.5.1 Difference from Variational Inference . . . 26
3.5.2 Architecture . . . 26

4 Evaluation **28**
4.1 Environment settings . . . 28
4.2 Datasets and tasks . . . 28
4.2.1 GMM data . . . 28
4.2.2 National Income Dataset . . . 29
4.2.3 MNIST . . . 29
4.2.4 OSIC Pulmonary Fibrosis . . . 29
4.2.5 Daily S&P 500 dataset . . . 29
4.2.6 SARS-COV-2 . . . 30
4.3 Tested autoguides . . . 30
4.4 Aims of experiments (tasks) . . . 30
4.5 Results . . . 31

5 Discussions and Conclusions **39**
5.1 Results overview . . . 39
5.2 Discussions and future works . . . 39
5.3 Conclusions . . . 40

Bibliography **41**

A Others **46**
A.1 Code implementations . . . 46

Chapter 1

Introduction

1.1 Motivation

Analyzing latent variables or distribution parameters from observed data is the central question of inference. The problem is also referred to as posterior estimation. Studies have been conducted on solving the inference problems either using sampling methods [1, 2, 3], or variational inference [4, 5, 6, 7, 8, 9]. The majority of variational inferences make use of mean-field or full-rank assumptions [10]. According to the mean-field assumption, latent variables are independent of each other, while the full-rank assumption extends this by using Cholesky factorization [11] and extending the covariance matrix to be unconstrained by positive definite requirements. Both of these methods have been implemented in Pyro, a mainstream programming language, as a form of autoguide. Essentially, they use the automated differentiation during variational inference (ADVI) method [12]. When autoguides are used, each model's original parameters are transformed into a Gaussian latent space.

The two previous autoguides have some drawbacks. Taking the mean-field approach might lead to a loss of latent variable dependencies. Assuming the full-rank situation, we will always have to perform Cholesky factorization with inserting it into the parameter scale tril of the Gaussian distribution. It removes the positive definite requirement, but neglects the importance of different kinds of special covariance matrices. Therefore, we aim at extending the covariance matrix group of autoguides as well as getting the inverse dependency by either using faithful inversion [13] or stochastic inverse [4] of the model dependencies to create new autoguides. Though AutoStructured is implemented, the node order is given in an unconvincing reversed order that might lose dependency information at higher dimensions. In order to construct a convincing dependency-based guide, there is a need to present it in a more structured manner.

1.2 Objectives

This project is primarily intended to build new autoguides and perform inference tests on different datasets and models. The objectives of this project are,

- To design autoguides using various covariance matrices
- To design autoguides using inverse dependency models
- To test those new autoguides on different datasets, observe the loss, and report the latent variable patterns in several models and datasets
- To generalize the autoguides' usage range by comparing them to other baseline autoguides or just simple guides.

1.3 Main contributions

Contributions to this project are as follows:

- I contributed eight new autoguides, including six based on covariance matrices, and two others based on different inverse dependency models, including topological ordering of the graph and minimum distance ordering given the observed node.
- I compared the performance of different autoguides on different datasets, and combined them with images to explain why some parameters did not fit, the limitations of generating new data using posterior probabilities, and some potential tradeoffs.
- I reviewed a lot of literature on variational inference as well as advanced algebra books, with a more thorough and detailed description of the background and method part, finally translating the theory to code.

1.4 Report outlines

The following is the order in which I organize this dissertation:

- Chapter 1 outlines project's goals, main objectives, and main personal contributions to the project.
- Chapter 2 introduces the fundamentals of the project, including Bayesian inference, variational inference, probabilistic programming, and Bayesian networks.
- Chapter 3 describes the sources of autoguide, as well as improvements based on inversed model dependencies, covariance matrices, and RNN autoguide.
- Chapter 4 describes how some of our improved methods developed in chapter 3 are applied to the datasets used of inference and how some rules and conclusions about the use of autoguides are formulated.
- Chapter 5 summarizes the results of implementations, discusses the deficiencies, suggests some improvements to be made in the future work, and summarizes the whole project.

Chapter 2

Background

Chapter 2 has provided the basic understanding of the project's fundamentals and we will discuss: **1)** an explanation of inference problems and some real-world examples **2)** what Bayesian inference is and its pros and cons **3)** what methods have been used to improve Bayesian inference; **4)** a guide to variational inference **5)** probabilistic programming as a tool to translate theory into practice **6)** the relationship between Bayesian networks and inference **7)** Bayesian network representation with probability distributions.

2.1 What is inference

Understanding what a hypothesis is is the first step to understanding inference. We are affected by the weather conditions on a daily basis. If nothing happens, rain is predicted to occur 50% of the time. Thus, a hypothesis can be defined mathematically as the probability that a probabilistic event occurs without given conditions. A rainy day and a non-rainy day are two distinct events, which means 50% probability individually. The probability of rain could be higher than 60% or 70%, however, if the sky is cloudy because weather conditions are given, so we will make different probability predictions based on the weather. In this case, inference can be defined mathematically as the modification of the original hypothesis probability based on the conditions (data), that is, the probability resulting from the conditions. However, conditional probabilities do not always constitute inferences. The general rule is that premises are inferred from results.

Here is another classic example of inference that can help understand it, which is cancer diagnosis. The probability of cancer occurring is extremely small, with a less than 1% chance recorded. Considering this scenario, we would like to know if the chances of getting cancer, even if we test positive, are still that small. Patients that suffer from cancer are assumed to have a positive test rate of 99%, while those who are not affected have a negative test rate of 99%. Using the joint probability distribution formula, the probability of both having this disease and testing positive is 0.99%. The marginal probability calculation formula (to simplify, it will not be expanded in detail) claims a positive rate of 10.9%, so the conditional probability that patients are diagnosed as

positive but still suffer from the cancer is only 8.33%. It is very rare to be diagnosed with cancer by being positive. In other words, the risk of cancer itself is small, and it is also unnecessary to worry even if being tested positive.

2.2 Bayesian inference

2.2.1 Concepts

As described in 2.1, both examples fall within Bayesian inference. Generally, Bayesian statistical inference problems aim to learn unobserved variables from observed variables. This includes exploring confidence intervals and estimating latent variable distributions from observed data in the field of statistics. As more evidence and information become available, the probability of a particular hypothesis can be updated continuously. Here we abstract data and hypotheses into symbolic representations and write out a general form of Bayesian inference.

We posit observed data $\mathcal{X}$ and unobserved patterns $\mathcal{Z}$. As the center of a statistical inference problem, one has to determine how to infer the underlying pattern of z $\in \mathcal{Z}$ from unlabeled x $\in \mathcal{X}$. Bayesian inference describes such inference dependency as a **posterior** probability distribution. Suppose that θ is a shared parameter for sets $\mathcal{X}$ and $\mathcal{Z}$, along with a probability function $p(\mathcal{Z}|\mathcal{X};\theta)$ that maps the variable x from original parameter space to latent (variable) space. The classic Bayes' theorem describes an inference problem as follows:

$$p(\mathcal{Z}|\mathcal{X};\theta) = \frac{p(\mathcal{X}|\mathcal{Z};\theta)p(\mathcal{Z};\theta)}{p(\mathcal{X};\theta)} \tag{2.1}$$

Our hypotheses in 2.1 $p(\mathcal{Z};\theta)$ are referred to as a prior probability distribution, whereas $p(\mathcal{X}|\mathcal{Z};\theta)$ is our conditional probability distribution. There is prior knowledge of domain experts to estimate $p(\mathcal{Z};\theta)$ and $p(\mathcal{X};\mathcal{Z},\theta)$. The decision on these two models is therefore subjective. Models may vary from scenario to scenario, but they are already known before any inference is made. We can present this marginal probability $p(\mathcal{X};\theta)$ in an integral form if z is continuous in space $\mathcal{Z}$:

$$p(\mathcal{X};\theta) = \int_z p(\mathcal{X}|z \in \mathcal{Z};\theta)p(z \in \mathcal{Z};\theta)dz \tag{2.2}$$

Bayesian inference has the following advantages:

- Using the posterior as a prior for new parameter updates, only a few new observations are required.
- It is the foundation for all inference methods.
- Different prior inference designs can get the same likelihood function, where it can be interpreted easily.

But it also has some obvious shortcomings:

- Prior can't be generalized and must be designed in independent cases.

- Posterior is heavily dependent on prior design.
- Calculating such integral (equation 2.2) is intractable in high-dimensional vector spaces for $\mathcal{Z}$ and sometimes we can't achieve an analytical solution. Even if we can calculate it, it has a high computation cost.

2.2.2 Extension on Bayes

The integration problem in equation 2.2 can be approached in two ways. The first approach is to use sampling simulations. We discuss sampling in this section and a variational approach to approximating the posterior distribution in 2.3. Stochastic simulation poses the problem of generating samples for a probability distribution p(x). The generation of samples may be difficult when p(x) has a complex distribution or when p(x) is a high-dimensional distribution, it is required to use more complex random simulation methods. Three different methods for sampling have been discussed: (**MCMC**) sampling, **MH** sampling, and **Gibbs** sampling. Despite the fact that algorithms and concepts of sampling are not the primary focus of the project, I will explain them here for completeness' sake.

MCMC sampling [2] Two of the concepts introduced by MCMC sampling are Monte Carlo simulations and Markov Chain models. According to certain probabilistic rules, a Markov chain experiences state transitions. One of the most important properties is that the probability of state transition is determined solely by the previous state. According to the convergence theorem, the probability distribution of a Markov Chain will converge itself to a stationary distribution. Based on the assumption that after n steps of convergence, the transition sequences after n steps will all be samples of a stationary distribution. When n reaches an infinite value, it will no longer be related to nodes or time. To satisfy the above detailed stationarity conditions, we need to construct the following transition matrix P that it satisfies:

$$\pi(i)P(i,j) = \pi(j)P(j,i) \tag{2.3}$$

where $\pi(i)$ is the stationary distribution of P(i, j). For the above formula to hold generally, an extra α(i, j) is introduced, which is also referred to as an acceptance rate since the equation 2.3 does not always hold:

$$\pi(i)\underbrace{P(i,j)\alpha(i,j)}_{\textbf{new P'(i,j)}} = \pi(j)\underbrace{P(j,i)\alpha(j,i)}_{\textbf{new P'(j,i)}} \tag{2.4}$$

Using the sample x_t, we could sample a x_* from the original transition matrix at time t. The acceptance rate would then be $\alpha(x_t, x_*)$. The uniform distribution U $\sim$ (0,1) is supposed and a random value u is sampled from this distribution. The transition from x_t to x_* will be accepted if this value is less than acceptance rate. Otherwise, at time t-1, it will still have the same sample. The integral can then be transformed into a summation over a series of finite samples sampled from stationary distributions.

MH sampling [3] Likewise, MH sampling is based on the state transition of the Markov chain, but Metropolis [3] designed a new sampling method to optimize MCMC sampling.

The acceptance rate α(i, j) may be too low, resulting in low sampling efficiency. In other words, sampling once is likely to reject a state transition, which may not lead to convergence. Hence, we adjust the acceptance rate as follows:

$$\alpha(x_t, x_*) = min\{\frac{\pi(j)P(j,i)}{\pi(i)P(i,j)}, 1\} \tag{2.5}$$

The advantage of this modification is that it increases the value of the acceptance rate, which in turn increases the probability of a state being transited. There are, however, two problems associated with MH sampling. In first place, calculating the acceptance rate at high dimensions is inefficient for datasets with too many features. In addition, researchers are still not satisfied with the acceptance rate. Ideally, the probability of a state transition occurring at the next moment should be 100%.

Gibbs sampling [1] Gibbs sampling then solves both of these issues. For Gibbs sampling, the stationary condition is met by using point transitions. Consider a 2-dimensional dataset with variables x_1 and x_2. Therefore, the sampling value at time t-1 would be x_1^{t-1} and x_2^{t-1}. To determine the new value of x_2^t, the sampling of x_2^t can be calculated from the value of x_1 at time t-1 based on its conditional probability distribution. The sampling of x_1^t is based on x_2^t and its conditional probability on x_2^t. These two values $\{x_1^t, x_2^t\}$ represent new sampling values of the distribution at time t. Higher dimension situations imply the following: one point is sampled based on the sample values of all the other points at the previous moment and the other points are sampled based on the sample values of some points at this moment, or the moment according to the dependencies between the states. During the sampling process, all other points are fixed, which is similar to the coordinate gradient (axis) descent method [14]. Alternatively, this type of sampling technique can be called cyclic sampling.

2.3 Variational inference

2.3.1 Concepts

In addition to sampling, variational inference (VI) is another important method to solve the Bayesian posterior integration problem [15]. To begin, we introduce the notion of conjugate distributions, where the posterior is in the same distribution group as the prior, or in other words, they exhibit the same form. This is best illustrated by a Gaussian distribution. In the case of a prior Gaussian distribution, the posterior distribution must also be Gaussian. Therefore a Gaussian distribution can be constructed to approximate this posterior Gaussian distribution, which is also known as the **Gaussian approximation**. The Dirichlet distribution is used to infer the parameters of a multinomial distribution, and the categorical distribution is used to infer discrete variables. VI considers how to approximate intractable distributions. The problem is generalized that given any posterior distribution $p(Z|X)$, where Z is set of the latent variables and X is a group of data, if an approximated distribution $q(Z)$ can be found so that it is close to $p(Z|X)$.

2.3.2 Optimization problem

Variational methods are based on optimization problems and allow for approximate inference. This is an optimization problem because we must constantly approximate the true distribution with approximate distributions, and the key to optimization is to reduce the difference between distributions. It is aimed at determining an approximate posterior distribution $p(\mathcal{Z}|\mathcal{X};\theta)$ with $q(\mathcal{Z};\phi) \in \mathcal{Q}$ [15] by implementing KL-divergence loss function:

$$q(\mathcal{Z};\phi) = \underset{q(\mathcal{Z};\phi)\in\mathcal{Q}}{argmin}\,(KL(q(\mathcal{Z};\phi)||p(\mathcal{Z}|\mathcal{X};\theta))) \tag{2.6}$$

, where $\mathcal{Q}$ is a space containing a set of possible approximation distributions. KL is an abbreviation for **Kullback-Leibler** divergence, also referred to as relative entropy in the context of information systems, and is often used to describe the difference between two distributions. We can measure the difference between approximated probability distribution $q(\mathcal{Z};\phi)$ and real probability distribution $p(\mathcal{Z}|\mathcal{X};\theta)$ in terms of an integral form [15] where we assume that the parameter ϕ is the parameter to $\mathcal{Z}$ merely:

$$KL(q(\mathcal{Z};\phi)||p(\mathcal{Z}|\mathcal{X};\theta)) = \int_z q(\mathcal{Z};\phi) log \frac{p(\mathcal{Z}|\mathcal{X};\theta)}{q(\mathcal{Z};\phi)} dz \tag{2.7}$$

Optimally, $q(\mathcal{Z};\phi) = p(\mathcal{Z}|\mathcal{X};\theta)$, where the value of KL-divergence becomes zero under this circumstance since the log term will become zero. But actually KL-divergence will never reach zero and will also experience diverging during optimization step, The value of KL$(\cdot)$ is always larger than zero which can be proved by Jensen's Inequality theorem [15]. We can rewrite the equation 2.6 in the form of evidence lower bound (**ELBO**) [15]:

$$KL(q(\mathcal{Z};\phi)||p(\mathcal{Z}|\mathcal{X};\theta)) = -ELBO(\mathcal{X};\theta;\phi) + log(p(\mathcal{X};\theta)) \tag{2.8}$$

A couple of reasons have led to the emergence of ELBO. The first reason is due to the fact that the calculation of KL divergence still includes the calculation of the posterior distribution, making the calculation difficult. It is found that using ELBO, only the approximate distribution and the previously known joint probability distribution in the calculation formula make it easier to deal with the approximate distribution, such as differential or integral of variables. Additionally, the distributed maximum likelihood estimation (MLE) and KL loss do not seem to be linked directly, whereas, the ELBO is related with MLE. The ELBO represents the upper limit of the joint probability distribution (according to Jensen's inequality) and our optimization process maintains the ELBO close to this upper limit. In other words, KL loss is usually calculated by subtracting these two indicators, which means that KL loss cannot be calculated directly.

Minimization of KL-divergence in equation 2.7 is equivalent to the maximization of ELBO since calculating $p(\mathcal{X};\theta)$ is independent of $q(\mathcal{Z};\phi)$ as mentioned. Current methods are solving how to optimize ELBO. ELBO can also be rewritten as:

$$\begin{aligned} ELBO(x;\theta,\phi) &= E_{q(\mathcal{Z};\phi)} log \frac{p(\mathcal{X},\mathcal{Z};\theta)}{q(\mathcal{Z};\phi)} && (2.9)\\ &= E_{q(\mathcal{Z};\phi)} log(p(\mathcal{X}|\mathcal{Z};\theta)) - KL(q(\mathcal{Z};\phi)||p(\mathcal{Z}|\mathcal{X};\theta)) && (2.10) \end{aligned}$$

The equation is easier to understand now since the first term can be regarded as a likelihood function after reconstructing the original space from the latent space. The

equation functions as a loss function during actual training in some models such as variational auto-encoders [5], and graph variational auto-encoders [6]. In part 2.3.4, various variational inference methods are illustrated.

2.3.3 Variational expectation maximization

Maximum likelihood solutions can be found using an expectation maximization algorithm for probabilistic models with latent variables. The general EM algorithm can be brought into VI to demonstrate the relationship between ELBO and maximum likelihood. It can also demonstrate how to find the best variational distribution q and optimal distribution parameters.

A typical EM procedure consists of two steps, in which the first step is to keep the parameters fixed. ELBO maximizes the log likelihood of the joint probability distribution by maintaining the distribution parameters and θ unchanged. In this case, the posterior probability equals the variational distribution q. During the second step of maximization, the variational distribution q is fixed, and θ is optimized. When the variational distribution is not too complicated, the EM algorithm is very efficient in training.

2.3.4 Forms of variational inference

Mean-field Assumption Measuring the mutual interaction between $z_i \in \mathcal{Z}$ and $z_{j\neq i} \in \mathcal{Z}$ is difficult where Bayesian networks cannot represent the conditional probabilities. A classical assumption about variational inference is that the variational posterior distribution is an entirely decomposable distribution:

$$q(\mathcal{Z};\phi) = \prod_{i=1}^{m} q_i(z_i) \tag{2.11}$$

where z_i and $z_{j\neq i}$ are independent and identically distributed. It ignores inference networks' enrich structures, losing the variational distribution's flexibility. One possible substitution is to add auxiliary variables as the conditions of ϕ in latent space. Based on this idea, many works sought to minimize the gap between true posteriors and approximate posteriors, such as structured stochastic variational inference [16], inference with auxiliary variables [17], variational Gaussian process [18], Copula variational inference [19], and hierarchical variational models [20].

Gradient Descent Because the space of parameters and the space of distributions are different, stochastic gradient descent directly does not produce good results [4]. In order to improve SGD, the natural gradient stated in information geometry is used, and the so-called stochastic variational inference (SVI) is proposed [4]. The second-order derivative (Hessian) information is the essence of SVI. The SVI mainly discusses models with mean-field and conjugacy assumptions. The advantage of these models is that the Hessian is easy to calculate and has an explicit form, but for more complex models, calculating the Hessian will greatly increase the computational complexity of the algorithm. Pyro takes SVI as the main optimized methods [21].

Reparameterization Tricks We perform the gradient descent to the equation 2.6, which will lead to:

$$\nabla_\phi ELBO(x;\theta,\phi) = E_{q(Z;\phi)}[\nabla_\phi log(q(Z;\phi)) log \frac{p(X,Z;\theta)}{q(Z;\phi)}]$$

(2.12)

The stochastic gradient descent based on this equation is called Black Box Variational Inference (BBVI) [7]. The problem of a high variance under the framework of BBVI is common. Using reparameterization tricks in latent space given by Max Welling's variational auto-encoder (VAE) is proved to be efficient [5]. We assume that there exists a function $Z = f_\phi(\varepsilon)$ where $\varepsilon \sim \mathcal{N}(0,1)$, then the equation 2.9 will become

$$\nabla_\phi ELBO(x;\theta,\phi) = E_\varepsilon(\nabla_\phi log(p(X, f_\phi(\varepsilon))) - log(q_\phi(f_\phi(\varepsilon)))) \tag{2.13}$$

Recent works [5, 8] have proven that VAE can reduce variance and make the Monte Carlo estimation probable. But VAE also has apparent limitations: it can not deal with discrete variables, and the number of distribution groups to do reparameterization is limited. Two papers [22, 23] took the advantage of Gumbel-Softmax distribution to relax discrete variables then it is able to do reparameterization to discrete variables.

Another problem with reparameterization is that the number of distributions that can do reparameterization is very limited. Automatic Differentiation Variational Inference (ADVI) [12] was proposed to map the variational distribution in the original individual parameter space to an unconstrained common space. Blei et al. tried to use acceptance-rejection sampling to enlarge distributions used for reparameterization [24]. ADVI is the method that we will mainly discuss in this project.

Flexible Transformations Variable transformation can be done in a more flexible manner. That is the point of normalizing flows (NFs) [8]. NFs are generalizations of the reparameterization trick: In the past, we transformed with a function, but now we can transform with a multi-layer function, as long as the Jacobian of the composite function is easy to solve. A key challenge at this point is to find enough representation power, and Jacobian is a family of functions that is more easily available. Research in this area has been extensive. A simple linear-time transformation was demonstrated [8], and it was also mentioned that infinite flows could be used, such as Langevin Flow and Hamiltonian Flow. Max Welling et al. proposed inverse autoregressive flows (IAFs) [25].

***Better Optimization Aims** Given a main model, we need to design optimization objectives (such as KL divergence/ELBO), variational models, and corresponding optimization algorithms. Thus far, we have only discussed improvements in the latter two parts. Another optimization goal can be chosen in addition to ELBO. Compared to ELBO, importance-weighted auto-encoders (IWAE) [26] use a lower bound that is better. Additionally, operator variational inference (OPVI) [27] re-examines the design of this optimization objective and proposes a more general framework with KL loss. As

a general matter, this type of problem is less work than the first two, since we can always design better optimization algorithms to compensate for the defects in the optimization goal.

2.4 Probabilistic Programming

2.4.1 Definition and Languages

Programming paradigm of probabilistic nature is probabilistic programming (**PP**). Under this framework, the Bayesian probabilistic models are specified and inferred automatically [28]. The field of probabilistic programming combines machine learning, statistics, and programming languages, and develops evaluators for machine learning-based inference models [28]. In general, a probabilistic programming system provides researchers with direct answers when there is uncertainty about the parameters. Probabilistic reasoning hand-written programs can be used to assist in decision making under uncertainty. With probabilistic programming, such programs are more easily to implement, providing a convenient framework or an interface to define the probability model as well as automatically learn probability models. By using probabilistic programming, researchers can avoid having to calculate the posterior based on their own models, or even write the parameter update process by hand. They can write their own programs quickly if they are familiar with a probabilistic programming language.

Programming languages for probabilistic applications are commonly implemented using C, C++, Java, and Python, which are often extended from mainstream basic programming languages. For example, **Alchemy** [29] is extended by C++, **Probabilistic-C** [30] is extended by C, **BLOG** [31] is extended by Java and **PyMC3** [32] is extended by Python. Different frameworks are used under different situations and domains, selecting the most appropriate language in accordance with the model descriptions and the environment settings, depending on personal preferences and easier implementations. Below we provide an overview of some mainstream probabilistic programming languages within the machine learning and statistics domain, [28], from which researchers can choose:

- machine learning: **Church** [33], **Anglican** [34], **BayesDB** [35], **CPProb** [36]
- statistics: **Birch** [37], **STAN** [38], **Infer.NET** [39]
- deep generative models: **Probtorch** [40], **Pyprob** [41], **Pyro** [21]

2.4.2 Main PPL: Pyro

Pyro [21], as described in part 2.4.2, is an open-source probabilistic programming framework available in Python and supported by PyTorch, an open-source machine learning library for researchers. Compared to other **programming languages**, it has the advantage of inferring for generative models and solving variational inference problems. Its torch-based API appeals to many users, as well. Although a new framework called NumPyro [42] has been released recently which is based on Jax for automatic differentiation (*automatically compute gradients*) and is faster than pyro, we introduce pyro

instead to better understand how it is used for inference problems and not the advanced calculations for gradient optimization.

A first step in Python is to construct your own dataset, or use a built-in dataset in Pyro. The next step is to construct your own model. It is possible to set some parameters of the prior distribution that are confirmed because one needs to know the prior distribution of the parameters first. Latent variables create new samples under the plate. In Pyro, the plate ensures conditional independence between new data. Designing the guide is the third step. The variational distribution in pyro is called **guide**, and it is only determined by the latent variables, so we only need to create the joint probability distribution among all latent variables. The fourth step is to identify our optimizer, usually Adam, but it can also be SGD. The final step is to bring the model and guide into SVI (stochastic variational inference). The ELBO parameters' update is automatically calculated by SVI in Pyro. It will calculate KL loss when using the step function. We can then train the model to figure out the latent variables' parameters. Consequently, by comparing the prior to get a better posterior probability that matches the input data as a knowledge, we can generate more data according to the model to see if the posterior is better than the prior. This is the general framework for training in Pyro, which demonstrates that a reasonable posterior, one dataset, and only one model are needed.

Some scholars might still feel that designing a suitable guide is difficult, and hope that the program can guides the model automatically on its own. Fortunately, Pyro has already provided these guides, where the base is called AutoGuide. The aim of this project is to generate new autoguides or improve the original autoguide to make the variational distribution closer to the ideal posterior distribution. In Pyro, we mainly concern ourselves with the classes *AutoGuide*, *AutoContinuous* and some of their inheritance, including Mean-field Gaussian posterior *AutoDiagonalNormal* and Full-Rank Gaussian posterior *AutoMultivariateNormal*. With regard to an example, here we are demonstrating:

```
from pyro.infer.autoguide import AutoDiagonalNormal
from pyro.infer.svi import SVI
guide1 = AutoDiagonalNormal(model) #  Mean-field
svi = SVI(model, guide1, ...) # Stochastic Variational Inference
for i in len(num_steps):
    loss = svi.step(x, y) # record loss
```

Listing 2.1: Example of using Pyro AutoGuide

Pyro has provided the function **get_param_store()** to extract the learned parameters, allowing us to plot the data given the posterior distribution and model parameters. The performance of mean-field and full-rank Gaussian distributions can be improved by providing new classes and using them as shown in the code block. The details of building new models in Pyro will be explained in part 3.4.

2.5 Bayesian Networks (BN)

2.5.1 Definition

A graphical model (**PGM**) represents the structure of conditional dependencies between random variables with a graph. The model is commonly used in Bayesian inference and probability theory. There are three types of probabilistic graphical models: *Undirected Graphical Model* (DG) , *Directed Acyclic Graphical Model* (DAG) and *Directed Cyclic Graphical Model* (DCG) [43].A Bayesian Network (**BN**) is equivalent to a DAG, where edges are directed and its joint probability can be expressed as a multiplication of conditional probabilities. We care about BN only, since we intend to explore inverse dependencies of models, which is an indication that the model needs to be directed and acyclic. Two commonly known examples about Bayesian Networks are: Deep Belief Networks [44] and Gaussian Mixture Model [45].

2.5.2 Important Properties of BN

We have discussed in this section how the BN is related to a probability distribution, what its essential properties are, and how it is used in our project.

Factorization The general expression of a joint probability for a BN $\mathcal{G}(\mathcal{V},\mathcal{E})$ is:

$$p(x) = \prod_{v\in\mathcal{V}} p(x_v|x_{pa(v)}) \tag{2.14}$$

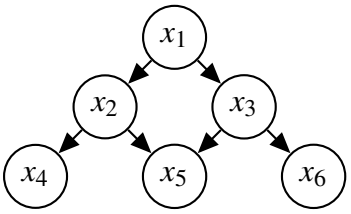

Figure 2.1: An example of a Bayesian Network: $G(\mathcal{V},\mathcal{E})$, where $\mathcal{V} = \{x_1,x_2,...,x_6\}$ and $\mathcal{E} = \{x_1 \to x_2, x_1 \to x_3, x_2 \to x_4, x_2 \to x_5, x_3 \to x_5, x_3 \to x_6\}$

We give an example to illustrate the construction of a joint probability of Bayesian Network, also called the factorization process. Given a graph $\mathcal{G}(\mathcal{V},\mathcal{E})$, where $\mathcal{V} = \{x_1,x_2,...,x_6\}$ and $\mathcal{E} = \{x_1 \to x_2, x_1 \to x_3, x_2 \to x_4, x_2 \to x_5, x_3 \to x_5, x_3 \to x_6\}$, as shown in Figure 2.1. The observation of x_2 depends on the observation of x_1, indicating that x_1 is the parent of x_2 so the joint probability of $p(x_1,x_2) = p(x_1)p(x_2|x_1)$. The joint probability of the total graph is:

$$p(x_1,x_2,...,x_6) = p(x_6|x_3)p(x_5|x_2,x_3)p(x_4|x_2)p(x_3|x_1)p(x_2|x_1)p(x1) \tag{2.15}$$

The inverse conditional probability is a posterior distribution. Now x_4, x_5, and x_6 are latent nodes and x_1 is the observed node. Therefore, it is an inference problem. Due to the fact that inference cannot be induced directly, it is necessary to assume an inverse dependency, reconstructing observed latent nodes from unobserved ones. Methods of analyzing a model's inversed dependency structure will be discussed in part 3.

Local Markov property Based on its parent variables, the local Markov property states that each variable is independent of its non-descendent variables:

$$X_v \perp\!\!\!\perp X_{V\setminus de(v)}|X_{\mathrm{pa}(v)} \quad \textit{for all v in V} \tag{2.16}$$

We can use this condition to remove redundant variables in calculating joint probability.

d-separation and I-map Based on a third set Z, d-separation determines whether a particular set of variables X is independent from a different set Y on a Bayesian network. Local Markov property is a particular case of d-separation. If any node in the trail between A and B is not observed and the trail is *converging*, then node A and node B are d-separated. The easiest way is to perform the moralization of graphs and remove observed variables to see if an edge is added between node A and node B. More examples are presented in Koller's PGM book [43].

Graphs that satisfy all the conditions of an independent distribution are called I-maps, where the conditional independence contained in the graph is a subset of the conditional independence satisfied by the distribution. A distribution can be derived from a list of graphs that includes all dependencies, which allows us to assume less distribution at a time.

2.5.3 BN with Variational Inference

A plate model describes the dependencies between variables in a Bayesian network. The plate model eliminates the need to account for the interdependencies of each variable in the group. In addition to presenting variational inference in a Bayesian network, the plate model can differentiate between the conditional prior probability distribution and the posterior distribution.

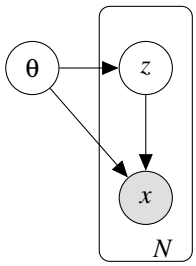

Figure 2.2: Plate model of $p_\theta(z)p_\theta(x|z)$

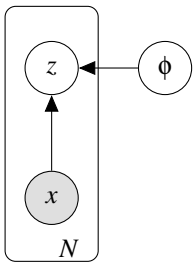

Figure 2.3: Plate model of $q_\phi(z|x)$

θ is the shared parameter for computing prior and conditional probability based on $\mathcal{Z}$ and $\mathcal{X}$ and is none of the observed and latent variables, so it's out of the plate. Computing posterior is the opposite process which is directed from $\mathcal{X}$ to $\mathcal{Z}$. In the approximation space, ϕ is the only parameter to $\mathcal{Z}$. In general, we build a plate model to see dependencies of variational inference clearly.

Chapter 3

ADVI and potential improvements

This chapter attempts to explain the methodology used in the experiments in terms of theories that could improve basic variational methods and autoguides. Section 3.1 describes the inference problem to be solved. Section 3.2 presents ADVI, one of the most effective solutions to the central problem based on variational inference. Section 3.3 presents improved methods of estimating the posterior by accumulating the inverse dependency of the latent variables. In Section 3.4, we present novel methods of improving autoguides by using different types of covariance matrices, where Gaussian distributions dominate the approximation space. The third improvement method called inference compilation, is discussed in Section 3.5.

3.1 Problem Restatement

In the previous paragraph, the concept and nature of the inference problem were discussed. Here is a recap of the problem and below we will provide a solution. In essence, one central issue of this essay is to determine the posterior distribution of latent variables given any custom probability model containing latent variables, the prior distribution of these latent variables and the data points. By finding a more accurate posterior distribution, some unknown data points can be predicted from the pattern of the latent variables.

In section 2, the inference problem has been formulated. One example here is that the problem can be pertained to solve Gaussian posterior distributions. As far as the one-dimensional Gaussian distribution is concerned, mean and variance are latent variables (scalars). For the multivariate Gaussian distribution, mean vectors and covariance matrices are latent variables. In general, the posterior distribution of the Gaussian distribution is in the same distribution group as the prior distribution, which is known as a conjugate prior. A posterior with such priors has an analytical solution when they are converted. In other words, under the denominator of Bayes theorem formula 2.1, the integral of the observed variable is also generally integrable.

As with the posterior and prior, the gamma distribution and the beta distribution are conjugate as well. For discrete variables, the classic conjugate priors are the multinomial

prior, poisson prior, and bernoulli distribution. It is generally necessary to map a one-dimensional integral to a high-dimensional integral for MCMC with Gibbs sampling if the denominator integral cannot be calculated for any other discrete or continuous variable distribution. This greatly increases the time and space complexity of the calculation. Ideally, we would like to keep away from the time-consuming process of integrating or sampling. In this way, we introduce a variational distribution, in which variational clusters reduce the distance from the original posterior distribution. Data points are not considered in the estimation as they do not influence the parameters of the variational distribution.

The method does, however, have some obvious restrictions. In some cases, it is impossible to determine what the true posterior distribution is, making it impossible to decide on a variational factor that is reasonable. A second problem is that its variables have constraints on their distributions and ranges. It follows that the variables of the variational distribution will be strictly within the same distribution cluster as the parameters of the posterior and prior distribution. During the maximization of expectations process (EM), the variational distribution q is optimized, as outlined in section 2. During this process, KL divergence will also be calculated so that the gradient of ELBO will be calculated automatically. It will be necessary to make sure that the parameter value is still within that limited range after the gradient update. Therefore, it is crucial to always be aware of the parameter value's size in order to avoid the update parameter running over the set limit. This process requires experts and humans to predetermine the parameters. Depending on the model and the data, there will be different parameters and range settings, which can be very troublesome, since setting parameters can't be generalized.

In order to ensure that any model can use the same parameter space, a new method is introduced in which all latent variables, regardless of distribution, are reparameterized into a high-dimensional space. ADVI discusses this method extensively. Detailed discussion of ADVI, including further approximations to latent distributions, is provided in section 3.2.

3.2 ADVI

The ADVI process consists of four stages: 1) remap each input's parameter distribution into a new shared space, remove restrictions on its values 2) approximate the shared space to a Gaussian distribution 3) transform into a standard normal for automatic differentiation and Monte Carlo integration 4) specify parameters (latent variable) update based on Monte Carlo sampling and gradient descent.

ADVI requires no conjugate prior, but only a differentiable variable distribution, since the gradient must be continuous. In addition, the input model should also be differentiable, and the joint probability distribution type should support the priority. The authors have mentioned a number of models that were used for testing autoguides, including linear regression, GMMs, and LDAs [12] , which can marginalize out variables to ensure that they meet the continuous and differentiable conditions.

1) Given Latent variables θ, and its prior distribution: $p(\theta)$. The aim is to transform

the support of θ in order to be live in $\mathbb{R}^K$ space Q by function T. The transformed joint distribution can be expressed as:

$$p(x,q) = p(x,T^{-1}(q)) \mid detJ_{T^{-1}}(q) \mid \tag{3.1}$$

where $p(x,T^{-1}(q))$ represents the original distribution factor and $J_{T^{-1}}$ is the Jacobian matrix of the inverse of T [46]. An example is the Gaussian distribution where its covariance matrix is diagonal. To ensure that the diagonal of its covariance matrix is positive, the values on the diagonal must be greater than 0, which is identical to the determinant of the covariance matrix $\mid \text{diag}(\sigma^2)\mid > 0$. Afterward, you can alter θ into the real number domain either by using the softplus function log(1+e(σ^2)), or by directly using the log function.

2) Any given transformed parameter can be approximated to this space by a common distribution group, which is defined as a Gaussian distribution in ADVI. The following two cases should be discussed in this context. The first case is to assume that the latent variables in the Q space are independent of each other, which is the so-called mean-field Gaussian distribution with factorization separated:

$$q(\eta,\Theta) = Normal(\eta \mid \mu, diag(\sigma^2)) \tag{3.2}$$

$$= \prod_{i=1}^{K} Normal(\eta_k \mid \mu_k, \sigma_k^2) \tag{3.3}$$

where $\Theta = \{\mu_1, ..., \mu_k\} \cup \{\sigma_1^2, ..., \sigma_k^2\}$. In the Pyro implementation, each of the variables is concatenated with each other and will lead to a $\mathbb{R}^{2K}$ vector. Obviously the variances have been constrained to the positive real coordinates, therefore we should remove this constraint by mapping these positive variance to the whole real number space, where the function might be a softplus function or just a simple logarithm transformation. One problem of this method is that we assume the variables are independent of each other, but actually latent variables might include some dependencies between each other where we need to accmulate these dependency distribution to reach a joint distribution. So a multivariate normal distribution is needed. It is also called full-rank Gaussian, since each row or column entry of a covariance matrix is not a combination or other row or column vectors in the matrix. Likewise, the form of the full-rank approximation is in this form:

$$q(\eta;\Theta) = Normal(\eta \mid \mu, \sum) \tag{3.4}$$

We must make sure that $\sum$ should be also positive semi-definite as definition of the covariance matrix. Therefore, we could use a special LU factorization: Cholesky factorization which assumes that $\sum = LL^T$ and that covariance matrix is positive defintie. Only one factorization for a positive definite and symmetric matrix exists. It can help explain the fact that dependence between each pair of two variables might help improve the posterior distribution, while it could include more parameters to train than a mean-field assumed distribution and still requires a cholesky constraint for the covariance matrix. The number of parameters to train in a mean-field Gaussian is equal to 2*K, where K is the latent dimension (number of variables), but the number of parameters to train in a full-rank Gaussian is equal to K+K*(K+1)/2.

3) We need to revisit the loss function for estimating the best q function in the transformed Q space, which is shown in background part that the metrics to measure two distribution's differences is the KL-divergence Loss and is equivalent to measuring the ELBO. However, in the transformed space, we need to rewrite it to be related with q function and the common parameters Θ in the Q space, which is:

$$L(\Theta) = \mathbb{E}_{q(\eta;\Theta)}[log\frac{p(x;\theta)}{q(\eta;\Theta)}] \tag{3.5}$$

$$= \mathbb{E}_{q(\eta;\Theta)}[logp\left(x,T^{-1}(\eta)\right) + log|detJ_{T^{-1}}(\eta)|] + \mathbb{H}[q(\eta;\Theta)] \tag{3.6}$$

It is intractable to calculate the expected differential for the ELBO when updating it and doing back propagation. Thus, we can calculate the differentiation of the term inside the expectation first, and then find the expectation by applying the final transformation, which is called the elliptical standardization [47]. S_θ can be regarded as a transformation that encapsulates the variational parameters in the latent space. This will convert the non-standard Gaussian distribution into a standard Gaussian distribution. The transformation for the mean-field assumption is $\zeta = S_\theta(\eta) = diag\,(exp(w))^{-1}(\eta - \mu)$, but for the full-rank assumption, it is $\zeta = S_\theta(\eta) = L^{-1}(\eta - \mu)$. As a result, it will generally lead to an approximation of the standard normal: $q(\zeta) = Normal\,(\zeta|0,I)$. As entropy is independent of both the model and the transformation, it does not require transformation. It is also stated that a simple analytic form is given for the entropy of the Guassian and its gradient, which can be implemented once and reused for all other models [12].

Algorithm 1 Parameter updating process for ADVI [12]

Require: Dataset x, model p(x;θ), iteration i = 1, threshold δ
  μ^1 = **0**, w^1 = **0** (mean-field) and L^1 = **0** (full-rank)
  while $\nabla ELBO$ less than δ **do**
    Draw M samples $\zeta_m \sim$ Normal(**0**, **I**)
    Approximate the gradient to the mean $\nabla_\mu L$ using MC integration
    Approximate the gradient to the covariance $\nabla_w L$ or $\nabla_L L$ using MC integration
    Calculate the current step size ρ^i
    Update $\mu^{i+1} \Leftarrow \mu^i$ + diag(ρ^i) $\nabla_\mu L$
    Update $w^{i+1} \Leftarrow w^i$ + diag(ρ^i) $\nabla_w L$
    Update $L^{i+1} \Leftarrow L^i$ + diag(ρ^i) $\nabla_L L$
    Increment iteration counter i = i + 1
  end while
  Return $\mu^* \Leftarrow \mu^i$
  Return $w^* \Leftarrow w^i$
  Return $L^* \Leftarrow L^i$

4) As a final step in ADVI, the update of the variational distribution parameters plays a significant role. Since we approximate the variational distribution as a Gaussian score, there is a parameter of μ regardless of whether it is the mean-field assumption or the full rank assumption. To conclude, the rest parameters associated with the mean-field assumption are w, while the parameters associated with the full rank assumption are L. If we separately differentiate these three variables, we will get $\nabla_\mu L$, $\nabla_w L$ and $\nabla_L L$.

As a result of approximate standard normalization of the terms within the expectation, they are now differentiable, so the expectation is the only thing left to do. Monte Carlo Integration can therefore be applied to draw samples based on the standard normal. In some conditions, the algorithm converges to the ELBO local maximum. The algorithm is implemented in one class called *AutoGuide* in Pyro. Two assumptions *AutoNormal* and *AutoMultivariateNormal* are derived from *AutoGuide*. They are the most basic type of autoguides and we compare them with our novel autoguides.

3.3 Learning Inverse Dependencies

Although ADVI successfully maps the parameters of any model into the same latent space, the algorithm assumes independence between variables. We hope that by incorporating the conditional distribution between the latent variables into the calculation of the posterior probability, we can fully take advantage of the dependencies between variables to infer parameters. In part 2, we discussed how the original probability from a latent variable node to an observed node can be represented as a probabilistic graphical model. In contrast, for the calculation of the posterior probability, it must be done by creating a reverse probabilistic graphical model from the observed node (or leaf node) to the latent node. The reverse probability model must be constructed by first converting the directed graph into an undirected graph [48], which we can then construct the posterior from the moralized graph. It maximizes the expression of the original information about the original paths in the graph. It minimizes the loss of the conditional independence that is contained in the directed graph.

Moralization and Markov Blanket As moralization is not the main point of the project, it serves as a prerequisite for inverting the models, so I will simply explain how it works here. By using a moral graph in graph theory, a directed acyclic graph can be translated into its equivalent undirected form. For the junction tree algorithm, it is one the most important step, which is used to propagate beliefs on graphical models. An undirected equivalent of a directed acyclic graph is formed by enforcing edges between all pairs of nonadjacent nodes that have a common child, and then removing all direct edges from the graph. In equivalent terms, an undirected graph is a moral graph of an acyclic directed graph in which the nodes of the original G are now connected to their Markov blankets. In a trusted Bayesian network, a Markov blanket of node A refers to a set of nodes related to A ,which contains the nodes of A's parents, its children, and its child nodes' parents (excluding A). In a Markov random field, a Markov blanket is simply represented as a node adjacent to node A. A Markov blanket usually represents the first case in machine learning. At this point, we can compute the moral graph for any directed acyclic graphical model.

Example As a simple example, figure 3.1 on the left shows a DAC in which C is a latent node. It has nodes A and B as parents. E is a child node. D is the parent node of child node E apart from node C. Based on the definition of moralization, the parent node of C shares a child node. However, the parent node does not have a link. Therefore, we need to add an edge between A and B to indicate the relationship between A and B. Similarly, another edge needs to be added between C and D, so two operations are

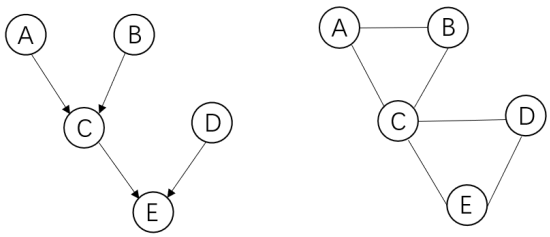

Figure 3.1: A directed acyclic graph and its moral graph [4]

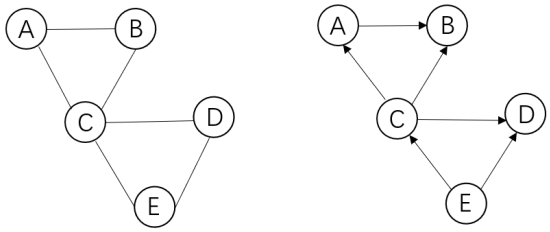

Figure 3.2: A moral graph and its reverse graph of 3.1 left [4]

performed in this graph. When it comes to C's Markov blanket, it is the parent node of C, which are A and B, the child node E of C, and the parent node D of the child node of C. In other words, node C's markov blanket is all nodes except C.

The inverse dependency of the model can be obtained in two ways. Starting from the observed node and upstreaming to the latent node is the first method. In this method, the node distance between an observed node and current node is measured and topological sorting or shortest distance sorting is performed. Starting from the latent node, the second method proceeds down in a downstreaming process. To calculate joint probability distributions, the metrics are used to determine the minimum number of variables needed to eliminate the variables. Alternatively, we are considering the smallest number of additional edges required to form the smallest clique for a node which will be the parent (or root) of the reverse model.

3.3.1 Observed to latent (Upstream)

Analyzing from observed data to latent variables is mentioned in two articles [49, 50]. In a directed graph $\mathcal{G}(\mathcal{V},\mathcal{E})$, $\mathcal{V}$ and $\mathcal{E}$ represent the vertices and edges, respectively. And given the observed node v_{obs}, then any v_k in $\mathcal{V}$ except v_{obs} has a distance from v_{obs}, which is expressed as: dist(v_{obs}, v_k). We can sort all points in V using this distance metric. In this list, the most frontal node is the one closest to the observed nodes. In order to maintain the conditional probability, we connect all points except the parent nodes in the original forward diagram to the current point. Based on the above explanation of d-separation, the point and all the other unconnected points in the graph are independent of one another. We will proceed to the next node in the list after that until we have explored all latent variables.

With respect to Figure 3.1, we will begin with E, where E is the observed node. According to the moral graph, the original E's parents C and D become its children in

the reverse graph. Distance from E to C and D is the same, i.e. dist(C,E) = dist(D,E). The distance between D and A is now equal to infinity since A is a leaf node and there is no path from D to A. D is ranked before C because D is more distant from the leaf node whereas the distance between A and C is 1. We add a directed edge $\mathcal{V}''_{C\to D}$, and similarly add $\mathcal{V}''_{A\to B}$. The implemented algorithm arranges the nodes either in topological order or according to their distance from the observed node. First, the inverse dependency of the original priority is added, followed by a directed edge from the markov blanket (excluding its parents in original fowarding graph) to the node according to the order of the order, with the directed edge pointing to the node.

In the papers relating to the two methods, two distinct sampling methods were discussed. In order to perform online forward and reverse sampling in stochastic inverse, the MCMC and MH sampling algorithms are used [50]. The second method involves simplifying the inverse graph to produce distinct latent variable(s) by using its connection relationship as a basis [49]. The model is regarded as a forward neural network. To learn the revised conditional density, sequential Monte Carlo is employed [49]. New sampling methods were not considered in this study. In contrast, we examined whether adding such inverse dependency would improve inference effects.

3.3.2 Latent to observed (Downstream)

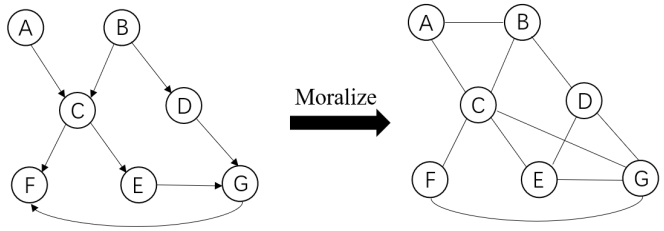

Figure 3.3: A DAG with 7 nodes. First moralize the graph, which means additional edges should be added between A and B, D and E, and C and G [13].

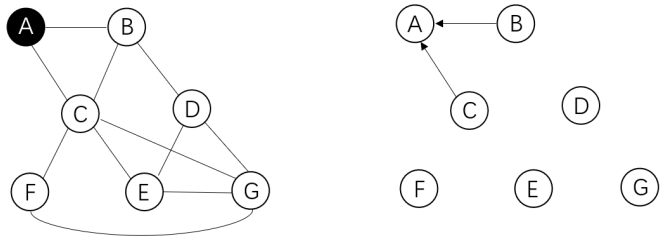

Figure 3.4: Start with latent node, A and B. For B, we need to add another edge CD and AD to calculate the joint probability p(A, B, C, D), so A is prior than B since calculating p(A, B, C) does not require adding any edge in the graph [13].

Another method explores the inverse dependency structure from latent nodes to observed nodes [13]. In essence, a directed graph is constructed from the latent nodes to the observed nodes. Examples are given in figures 3.3 through 3.8. We begin by constructing the list of nodes that have been visited already (null), followed by the list of frontier variables that are ready for elimination. In 3.3, we reach a moral graph of the left DAG.

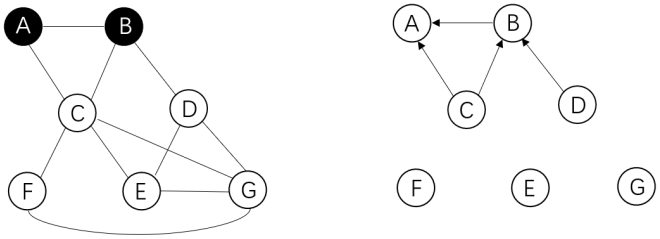

Figure 3.5: After A is eliminated and marked as visited, since B is not visited so we need to visit B. We need to calculate the joint probability p(B, C, D). Therefore, an edge CD is added. C and D are added to the latent node list and B is removed from the list [13].

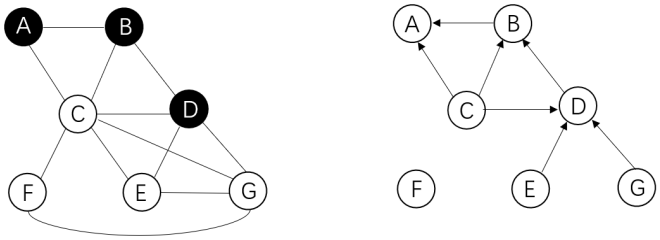

Figure 3.6: Similarly, eliminating D does not require adding any edges (edge CD has already been added in the last round), so D is prior than C [13].

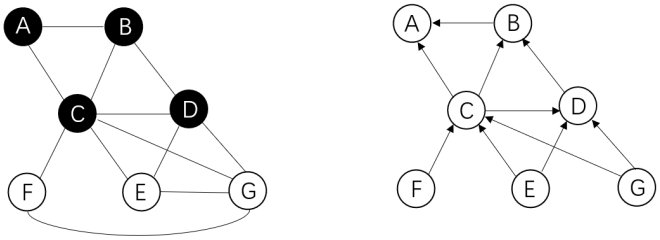

Figure 3.7: Eliminating C requires adding an additional edge EF. After that, add E to the latent node list. E is the last node to eliminate since F and G are observed nodes (roots) [13].

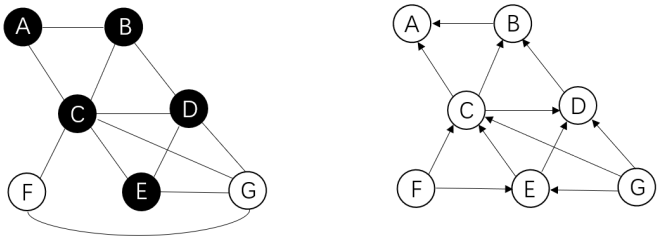

Figure 3.8: Finish traversing and the directed graph on the right is the inverse dependency graph of the original model (posterior) [13].

Latent nodes A and B are referred to as frontier variables. If we wish to eliminate joint probability p(A, B, C), we must form a minimal clique with A's neighbors B and C. Checking B reveals that eliminating p(A, B, C, D) does require an additional edge between C and D. This means there are no ACD or BAD cliques, so A is more prior than B. Afterward, A is selected with its neighbors pointed to it, and A is then marked as visited. Since B is still in the frontier variable list, visit B. To eliminate p(A, B, C, D), two variables C and D are needed, so C and D are added to the frontier variables list. We repeat the above process until we reach the last observed node. Finally, directed edges are added from observed nodes to the last observed node. We can also express this algorithm this way: we sort nodes according to the number of variables to be eliminated, then add edges between Markov blankets (between nodes). This algorithm has been implemented in Pyro which is called *AutoStructured*.

3.4 Covariance Matrices' Variants

We discovered improvements to autoguide by passing covariance matrices, in addition to inverting model dependencies. We wonder, for example, if we can use methods other than softplus constraints to make the variance and mean always greater than 0, for the class of *AutoDiagonalNormal* (corresponding to the mean-field assumption in ADVI). *AutoMultivariateNormal* is another example. To ensure positive definiteness of the covariance matrix, we only use the lower triangular matrix for this class (corresponding to the full-rank assumption in ADVI) and pass it as the scale tril to the Gaussian distribution. To reduce the number of parameters or improve inference, we considered designing different covariance matrices. Six new autoguides were designed using different covariance matrices, with the only condition being that the covariance matrix must always be positive definite.

3.4.1 PolyDiagNorm

We observe from the class *AutoNormal* or *AutoDiagonalNormal* that the posterior distribution obeys the rule that the scale term should be transformed by a softplus function to ensure that each variance term in the scale vector is positive. The main idea for the new autoguide *PolyDiagNorm* is to perform linear transformation to each term in the scale. Given a scale $\mathbf{x} = [x_1, x_2, ..., x_n]$, looking from the intermediate results of any sampled training in our tested datasets, any trained x_i would not be larger than 10 for example. More strictly, since the original scale was assigned to a small value so it's not likely to show big gradient jump in the training. Therefore, 0.1 * x_i will be smaller than 1. Recall the theorem that:

$$\frac{1}{1-x} = 1 + x + x^2 + ... + x^n \tag{3.7}$$

for any x within -1 to 1 where n is close to infinity. Given any n, the value on the right side of the equation will be larger than zero. Therefore we can define a function F(x) = 1 + x + x^2 for example, the new scale that is passed to the distribution function will become: **x'** = [F(x_1), F(x_2), ..., F(x_n)], where each term in **x'** is larger than zero and thus removing the softplus constraints. With a high-dimensional dataset, we can set the

coefficient of x_i to a smaller value, which may be task-specific. It is conceivable that there is a better way to remove constraints, such as assigning a sine or cosine function to the original value, which is considered to be included in our future work.

3.4.2 SymmetricNorm

In the *AutoMultivariateNormal* class, cholesky factorization has been performed, and the results have been passed to scale tril so that they satisfy the full-rank assumption in ADVI. We find that the matrix scale tril is a lower triangular matrix. The disadvantage is that it still requires a lot of parameters during the training process. In addition, we want to investigate how to generate new types of positive definite covariance matrices without cholesky factorization. In order to generate the symmetric matrix, we perform a matrix-level multiplication with the scale vector and its transpose. Suppose that the latent dimension is N. Then the total parameters for training in *AutoMultivariateNormal* is equal to N*(N-1)/2. In our method *SymmetricNorm*, the number of parameters is still 2*N.

According to the symmetric matrix theorem, given any matrix or vector x, xx^T is symmetric. It is easy to prove that $(xx^T)^T = (x^T)^T x^T = xx^T$ so that the matrix is symmetric. Then we scale the matrix by a small number c, such as 0.01. From linear algebra, we know that adding a few small elements to the diagonal of a symmetric matrix can make it positive definite. Therefore our covariance matrix **Cov** will become:

$$Cov = c * \mathbf{x}\mathbf{x}^T + K * \mathbb{I}_n \tag{3.8}$$

where n is the latent dimension (dim) and $x \in \mathbb{R}^{1*dim}$. Typically c is equal to 0.01 and K is equal to 1.

We could also perform matrix decomposition to this positive definite matrix to extract its eigenvalues which can be further transformed into a diagonal matrix, where Cov $= U\sum U^T$, where $U^T = U^{-1}$ and $\sum$ is a diagonal matrix containing Cov's eigenvalues. We can pass $\sum$ to covariance matrix since eigenvalues for a positive definite matrix are larger than zero, but it is our choice whether or not to do so.

3.4.3 LowRankNorm

Our new autoguide *LowRankNorm* is a special case of *SymmetricNorm*. Suppose that we would like to create a covariance matrix by a low-rank matrix **Cov**, where **Cov** $\in \mathbb{R}^{m*n}$ and m $\neq$ n. The implementation of the class *AutoLowRankMultivariateNormal* entails self-multiplicating Cov with the transpose of Cov's diagonal and adding small numbers to it along with Cholesky factorization. It may have disadvantages, such as adding squared scales to the diagonals instead of a unified scaled identity matrix, and it does not account for the scaling fact of $Cov.Cov^T$. The idea is to multiply *Cov* by another trained matrix and add a scaled number c:

$$Cov' = c * (U \cdot Cov)(U \cdot Cov)^T + K * \mathbb{I}_n \tag{3.9}$$

where U $\in \mathbb{R}^{n*m}$. c is typically equal to 0.01 and K is equal to 1. Moreover, Cov' may be substituted into equation 3.8 to make it the case of *SymmetricNorm*.

3.4.4 BlockDiagNorm

The block diagonal matrix, for instance, is one of the useful but special types of matrix in linear algebra. We create the autoguide *BlockDiagNorm* based on the block diagonal matrix. This type of matrix is square diagonal matrices that have square elements on the diagonal while containing zeroes on the off diagonal. The block diagonal matrix C is typically in this form:

$$C = \begin{pmatrix} A & 0 & \cdots & 0 \\ 0 & B & \cdots & 0 \\ \vdots & \vdots & \ddots & 0 \\ 0 & \cdots & \cdots & K \end{pmatrix}$$

where A, B..., K could be any kind of matrix. There is only one requirement: C must have the same number of rows and columns. We only used two matrices A and B to construct such C in the real implementation. Therefore, both A and B must be positive definite in order for this block diagonal matrix to be positive definite. We use the same operations as described in 3.4.2 and 3.4.3. It also cuts down the number of parameters for training, since A $\in \mathbb{R}^{\lceil N/2 \rceil * \lceil N/2 \rceil}$ and B $\in \mathbb{R}^{\lfloor N/2 \rfloor * \lfloor N/2 \rfloor}$ for example, then the total parameters is less than equal to N/2 * N/2 * 2 = $N^2/2$, which is then less than N^2. Since more non-zero terms are included in the covariance matrix, that means the mutual information between two latent variables has been included, so it might perform better than a pure diagonal matrix.

3.4.5 ToeplitzNorm

Toeplitz matrices are another type of covariance matrix that can be utilized. We create the autoguide *ToeplitzNorm*. The toeplitz matrix is a matrix in which the diagonals from left to right are constant. Toeplitz matrix C with n+1 rows and n+1 columns can be written as:

$$C = \begin{pmatrix} a_0 & a_1 & \cdots & a_{n-1} & a_n \\ a_{-1} & a_0 & a_1 & \ddots & a_{n-1} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ a_{-n+1} & \ddots & \ddots & a_0 & a_1 \\ a_{-n} & a_{-n+1} & \cdots & a_{-1} & a_0 \end{pmatrix}$$

We have

$$C_{i+1,j+1} = C_{i,j} = a_{j-i} \tag{3.10}$$

where 0≤i≤n-1 and 0≤j≤n-1.

We use the above equation when implementing the covariance matrix. There are two choices. If we only use the scale vector, then we enforce a_{-n} to be equal to a_n. Alternatively, if we consider that the row vector differs from the column vector, then we can build a new vector that assumes another N parameter space to construct this matrix and force it to be positive definite. It takes up a maximum of 3*N parameters, where N refers to the number of latent variables, but is computationally expensive for high-dimensional latent variables.

3.4.6 CirculantNorm

A circulant matrix C is the special case of Toeplitz matrix, where C can be written as:

$$C = \begin{pmatrix} a_0 & a_1 & \cdots & a_{n-1} & a_n \\ a_n & a_0 & a_1 & \ddots & a_{n-1} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ a_2 & \ddots & \ddots & a_0 & a_1 \\ a_1 & a_2 & \cdots & a_n & a_0 \end{pmatrix}$$

We create a new autoguide *CirculantNorm* based on the circulant matrix. By using the scale vector, we can generate the matrix easily. Unfortunately, making sure that the matrix is positive definite is difficult, so we use an extreme example. Specifically, we only take into account the value of a_0 and a_1 and disregard other terms. The matrix now looks like this:

$$C = \begin{pmatrix} a_0 & a_1 & \cdots & 0 & 0 \\ 0 & a_0 & a_1 & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & a_0 & a_1 \\ a_1 & 0 & \cdots & 0 & a_0 \end{pmatrix} or, C = \begin{pmatrix} a_0 & 0 & \cdots & 0 & a_1 \\ a_1 & a_0 & 0 & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & a_0 & 0 \\ 0 & 0 & \cdots & a_1 & a_0 \end{pmatrix}$$

In order to make the matrix positive definite, we need to make sure that a_0 is larger than zero and a_1's absolute value since the determinant value of the matrix is equal to a_0^n + a_1^n in the left case while the answer is a_0^n - a_1^n in the right case if C $\in \mathbb{R}^{n*n}$. Regardless of n, this value should always be greater than zero.

In the next round of training, if the scale vector does not satisfy the requirement, an increase in a_0 or decrease in a_1 will be necessary to meet the requirement (suppose a_0 and a_1 are all positive).

3.5 Inference Compilation

Making the repeated inferences fast is critical . It is referred to as adaptive Monte Carlo method, or amortized inference [51]. Inferences about past related models are reused by humans to speed up current inferences [52]. Besides, according to the Brooks Paige et al. [49], the inverse dependency network can be equivalent to a neural network. A neural network can be constructed for each inverse graph. As the directed graphical model is extended to universal probabilistic programs, inversing dependencies becomes increasingly difficult and impossible. Thus, the author proposes a program-specific method that ignores variable dependencies and uses a non-domain RNN model and domain-specific observation embedding to guide latent variables. LSTMs and RNNs can represent relationships between latent variables through their transfer relationship, which makes deep learning superior. Similarly, some other works focused amortized inference on one model, such as learning inference before seeing data [53, 54], which

sample training data in an neural network and learning sequential Monte Carlo (SMC) proposals for fixed graphical models [49].

Inference compilation A compilation of inference is a technique that transforms an inference problem written in a universal programming language (STAN or Pyro) into a trained deep learning model written in a neural network specification language [41], such as Pytorch or Tensorflow. By feeding observational data into this neural network at test time, a probabilistic model is approximated inferenced using the original model.

3.5.1 Difference from Variational Inference

Think about how we define variational inference: the goal is to find a posterior in the approximation family Q that could minimize the divergence from our true posterior. As part of amortized inference, the given family is a set of conditional distributions of z given x instead of marginal distributions. It is thus a matter of identifying a member of the family whose divergence from the true posterior minimizes the expected divergence. KL divergence for amortized inference is $D_{KL}(p(x|y)||q(x|y;\phi))$ [41], which is different from the KL-divergence for variational inference in equation 2.6.

Methods of learning inference differ according to the difference. Variational autoencoders targets $KL(q||p)$ while some methods such as reweighted wake-sleep [55, 56] and their SMC counterparts [57, 58] target $KL(p||q)$.

3.5.2 Architecture

Recurrent Neural Networks To achieve amortized inference in inference compilation, a recurrent neural network model is used to process the input without regard to the domain [41]. Figure 3.9 illustrates the structure of a basic RNN.

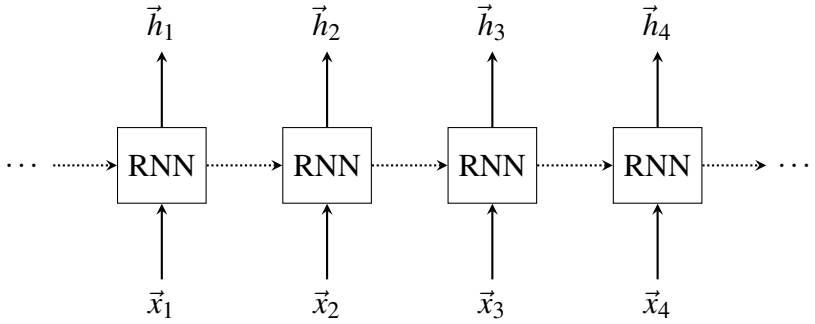

Figure 3.9: Recurrent Neural Network

Particularly, the long short-term memory (LSTM) [59] architecture is utilized which helps mitigate the problem that the gradients of RNN(s) tends to vanish and explode when time increases. [59] Inputs of the LSTM are the concatenation of the observed embedding, sampled embedding and one-hot encodings of current program address, instance number and proposal type of the proposal distribution [41]. Figure 2.5 shows the LSTM block where we need to pass the hidden states h_t to proposal layers.

Neural network architecture In particular, the long short term memory (LSTM) [59] architecture is utilized for mitigating the gradient vanishing and explosion problems of a RNN [59]. We start the evaluation by computing the observe embedding f_{obs}(x), which is domain specified. By executing a probabilistic program successively deterministically, an execution trace can be generated that looks like this: $(z_t, a_t, i_t)_{t=1}^{T}$. z_t is the sampled value, a_t is the address of the sample and i_t is the instance number of the sample, and T is a trace-dependent length [41]. An LSTM network architecture is automatically generated by combining an LSTM core with embedded layers, proposal layers, and a probabilistic program for training the network with an infinite stream of training data generated from the model.

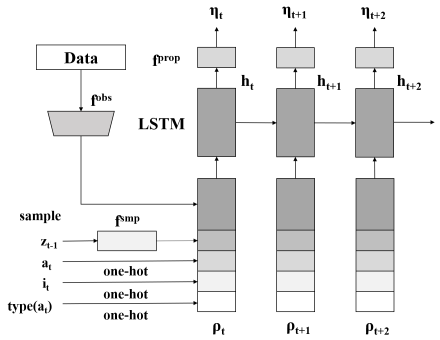

Figure 3.10: LSTM structure in inference compilation

The details are shown in figure 3.10. The sampling embedding layer is applied to the sampled value z_{t-1} at the last time step. The one-hot encoding layer is also applied to address, instance, and type of address (proposal type). All of the elements constitute the LSTM input. h_t is the output of LSTM. We apply proposal layers to the output h_t which will lead to proposal parameters η_t. Note that the LSTM core can possibly be a stack of multiple LSTMs. After the compilation stage is finished, weights ϕ are trained and neural networks are specialized for the given model. Then, we use importance sampling in conjunction with the model and network parameters to obtain the posterior, which is fast and cheap when compared to compilation stage.

Chapter 4

Evaluation

4.1 Environment settings

The entire project requires Python 3.8.12, as well as Pyro-ppl version 1.8.1. As Pyro is a dependency of Pytorch, we must install Pytorch, where version 1.11.0 is used. VSCode, command line, and bash are used for development. In order to create training loss graphs, view latent embeddings, and generate data, we will also need to download matplotlib and seaborn, whose versions are 3.5.1 and 0.11.2 respectively.

4.2 Datasets and tasks

As part of our study, we tested the benchmarking autoguides and our generated autoguides on six datasets, including national income dataset, MNIST dataset, OSIC pulmonary fibrosis dataset, SARS-COV-2 dataset and Daily S&P 500 dataset. As for the remaining one, it is created using synthetic data in order to examine the latent variables of the Gaussian Mixture Models (**GMM**). For each dataset, tasks are mentioned.

4.2.1 GMM data

A number of Gaussian distribution patterns can be found in given data using Gaussian Mixture Models (GMM). The model parameters that we want to explore are mixture weights Φ, means μ and a covariance matrix Σ. The following example illustrates the problem and the expected result. Suppose we have a list of data. The data consists of five elements, each of which is equal to 0, 1, 10, 11, 12. We give a Dirichlet distribution to the weights with normal distribution to the means and covariance matrix, as well as make sure that the parameter $\phi_{i=1,...,K}$ are reasonable since mixture models are sensitive and succeptible to local modes. Under the Gaussian distribution, a trained mean will be 0.5 and 11, and a trained variance might be 0.05 and 0.2. We have designed three tests based on the different lengths of data with different numbers of potential Gaussian distribution patterns. K = 2, 3 and 5 with dataset lengths of 20, 60, and 100 respectively.

4.2.2 National Income Dataset

As part of supervised learning, linear regression is one of the most basic tasks. The Bayesian linear regression will be applied to the national income dataset adapted from this paper [60]. The purpose of exploring this dataset is to determine if the level of ruggedness of a country is related to its GDP level. Specifically, we look at non-African countries and African countries. Previously, a study stated that poor roads and terrains generally result in low income levels, but this is not always true because poor geographical conditions can elevate African countries' GDP levels [61]. As a result, the results are positive for African countries. By looking at the original data, but also by inferring latent patterns and predicting the data, we can find out if this rule holds true for African countries. Three features have been extracted from the dataset.

4.2.3 MNIST

MNIST is a large-scale dataset comprised of 60,000 training and 10,000 test examples of handwritten digits [62]. MNIST is a well-known benchmark dataset that is used mostly for computer vision oriented tasks such as image classification [63] and image reconstruction [64, 65]. To determine whether autoguides can infer latent variables in some classical deep generative models, we are interested in the reconstruction tasks in this project. Specifically, a variational autoencoder with the structure of an encoder-decoder network is being developed. Autoguides are used for guiding the model (the decoder part) where the corresponding plate model is shown in figure 2.3.

4.2.4 OSIC Pulmonary Fibrosis

From Kaggle competition, we selected OSIC Pulmonary Fibrosis Dataset [66]. Pulmonary fibrosis has no known cause or cure. We are trying to predict the severity of lung function decline in this dataset. Spirometers measure forced vital capacity (FVC). When it comes to medical applications, a model's level of confidence can be useful. The 176 patients who were measured for FVC made an average of nine visits. Visits occurred in the interval [-12, 133] during specific weeks. The decline in lung capacity is very evident and varies widely from patient to patient. FVC measurement for each patient for each week in the interval of [-12, 133] was predicted, along with their confidence to match. In this application, we will use Bayesian hierarchical linear regression with partial pooling using parameters α and β. Even though α and β differ for each patient, the coefficients all share a similarity. The individual coefficients can be modelled by assuming they all come from the same group distribution.

4.2.5 Daily S&P 500 dataset

Daily S&P 500 is the dataset that tracks the performance of 500 large companies in the United States that deal with stock exchanges. As input data, we chose the close price of the market. The model that we used is a stochastic volatility model [67], which is utilized as a mainstream market prediction model [68]. Predicting the data that matches the market's direction is our goal. Under normal conditions, autoguide will encounter a memory overflow on training latent variables due to the large size of the dataset,

which is why we have selected data within the last 100 days. The number of days is the parameter that we can adjust in the experiments.

4.2.6 SARS-COV-2

Many studies have focused on the epidemic situation of SARS-COV-2 (Covid-19) in the past two years, which is why we included it to complement the timeliness of the dataset. Based on public data collected from different PANGO lineage viral genome samples around the world over time [69], this dataset includes the relative growth rates of different SARS-CoV-2 strains . There are about 2 million sequences in total. The plate structure of the model would return a block-diagonal like posterior covariance matrix with an estimated 500,000 latent variables. Due to the high dimensions, the program would also suffer from gradient explosion. We therefore set an autoguide list, add the autoguide first, and then add a local guide to the list that blocks some latent variables. This will aid in solving gradient explosion in a high dimensional latent space. Our goal is to see if structured autoguides will improve in the large dimensional space.

4.3 Tested autoguides

We have selected *AutoDiagonalNormal*, *AutoMultivariateNormal*, *AutoLowRankMultivariateNormal* and *AutoStructured-faithful* as our baseline autoguides. *PolyDiagNorm*, *SymmetricNorm*, *LowRankNorm*, *BlockDiagNorm*, *ToeplitzNorm*, and *CirculantNorm* mentioned in 3.4 will be tested as new autoguides, as well as two inversed dependencies based methods mentioned in 3.3 including *Structured-topo* and *Structured-min-dis*.

4.4 Aims of experiments (tasks)

1. To find out whether the new autoguides can further reduce the loss, we test all those autoguides on different datasets. This is the main purpose of all experiments. In light of the fact that the results of any random seed will be similar,**we are not testing random seeds**. Fixed seed number is **2022**.
2. Different datasets and models serve different purposes. The purpose of testing GMM data was to see approximate Gaussian patterns in the data. The income dataset was tested to determine if the bootstrap parameters confirm whether the ruggness has a reverse effect on African countries' incomes. The MNIST dataset was used to test if autoguide could guide deep generative models. The SARS-CoV-2 dataset was used to test the inverse dependency model's strength in the case of high dimensionality. In testing the OSIC dataset and S&P 500 dataset, one goal was to expand application scenarios for autoguides, and another goal was to validate their capabilities by guiding models in latent space with relatively high dimensions. **Based on standard benchmark tests, we used Pyro's existing code to generate images of parameters, losses, and different inference patterns**.
3. We will outline the possible scenarios of the new autoguides based on the data we have selected, including some advantages and disadvantages in the case of both

high-dimensional and low-dimensional latent variables, as well as the impact of structured autoguides when dependencies are included.

4.5 Results

GMM dataset *SymmetricNorm* and *LowRankNorm* performed the best when K equals 2, which indicates that they reached the lowest loss during the training. Nevertheless, they might suffer from a high variance problem, as shown in the table 4.1. When K is equal to 3, *PolyDiagNorm* and *ToeplitzNorm* were the best according to the loss. Structured autoguide performed the best among all autoguides when K is equal to 5. The reason why *Structured-topo* and *Structured-min-ds* have shown the same results with *Structured-faithful* is that in the corresponding graphical model, the parents of "obs" are "weight", "loc" and "scale" with no potential path between each two parents. Regardless of the type of inversed model we use, the last parameter in the plate (in order) will always be the first node to be considered for adding markov blankets.

Table 4.1: Testsing autoguides on GMM dataset

Model	K = 2	WEIGHT (K = 2)	LOC (K = 2)	SCALE (K = 2)	K = 3	K =5
DiagonalNorm	72.65	0.67, 0.33	17.42, 10.27	1.57	194.77	334.02
MultivariateNorm	72.44	0.67, 0.33	17.30, 9.99	1.76	195.49	334.90
LowRankMultivariateNorm	72.09	0.53, 0.47	17.83, 9.38	1.22	195.65	331.27
PolyDiagNorm	**70.68**	0.67, 0.33	19.22, 7.54	**1.44**	**192.48**	340.17
SymmetricNorm	**69.20**	0.71, 0.29	17.82, 10.04	8.46	195.37	334.43
LowRankNorm	**69.21**	0.71, 0.29	17.90, 10.07	8.39	194.25	336.00
BlockDiagNorm	72.47	0.60, 0.40	17.32, 7.25	3.13	194.82	**332.77**
ToeplitzNorm	71.02	0.63, 0.37	17.35, 9.60	**1.63**	**193.24**	333.78
CirculantNorm	**70.69**	0.54, 0.46	18.99, 10.52	8.30	194.10	339.18
Structured-faithful	72.09	0.68, 0.32	18.62, 9.00	**1.62**	194.06	**330.62**
Structured-topo	72.09	0.68, 0.32	18.62, 9.00	1.62	194.06	330.62
Structured-min-dis	72.09	0.68, 0.32	18.62, 9.00	1.62	194.06	330.62

Furthermore, we were interested in inferences that could be drawn from the generative model. We focused on a situation where K was equal to 3. Accordingly, different autoguides will produce different gaussian patterns. We find that *CirculantNorm* leads to a larger variance for each distribution, but the three other autoguides seem much more concentrated and reasonable. The reason might be that the covariance matrix for *CirculantNorm* is sparse as mentioned before while the covariance matrix for other three autoguides is not sparse.

Also, we could take a look at the entire training records of the four autoguides listed above. This time we choose K = 5. From figure 4.6 to figure 4.9, we can find that the *Strucutred* and *AutoNormal* have shown a better level of stability during the training. *ToeplitzNorm* is somewhat unstable after 100 epochs, but for *CirculantNorm*, it is obviously unstable. To infer parameters in Gaussian mixture models, we could prefer to use *ToeplitzNorm* instead of *CirculantNorm*. In order to resolve the stability problem for **CirculantNorm**, we could increase the learning rate, for example 0.2 instead of 0.1, or train with more epochs, for example 1000 epochs or more.

Income dataset We cared about the coefficients of this Bayesian linear regression problem, which are bias a_1, the coefficient of the variable "is_cont_africa" a_2, the

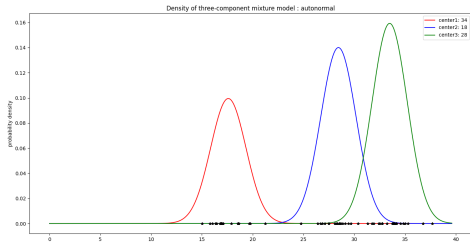

Figure 4.1: pattern(AutoNormal)

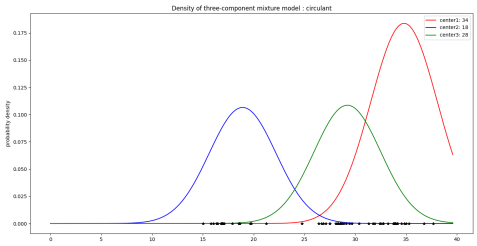

Figure 4.2: pattern(CirculantNorm)

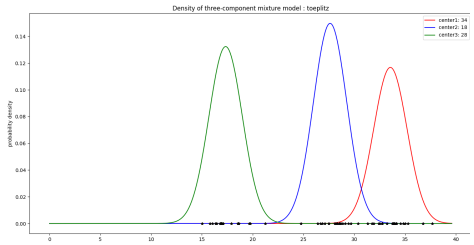

Figure 4.3: pattern(ToeplitzNorm)

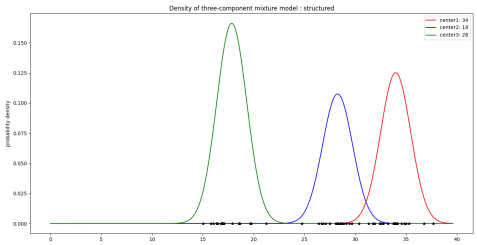

Figure 4.4: pattern(Structured)

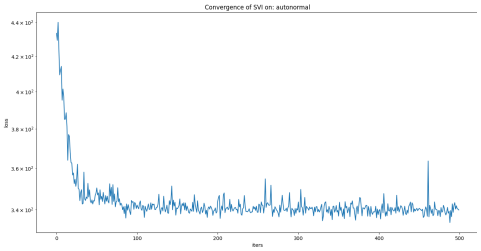

Figure 4.5: loss(AutoNormal)

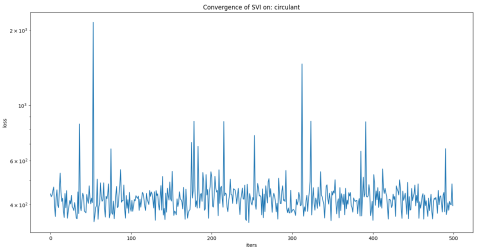

Figure 4.6: loss(CirculantNorm)

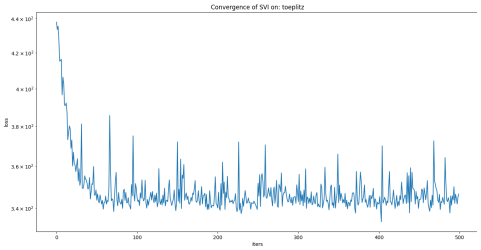

Figure 4.7: loss(ToeplitzNorm)

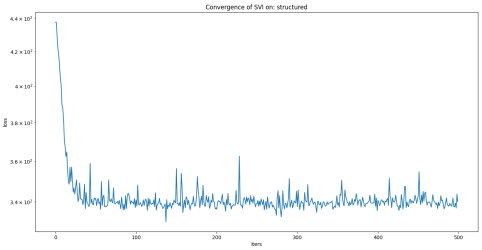

Figure 4.8: loss(Structured)

Figure 4.9: Patterns and losses of four selected autoguides in the case of GMM with K = 3 and length of 60.

coefficient of the variable ”ruggedness” a_3, and the coefficient of their product a_4. We infer the scale parameter as well. Since it's low dimensional, it's easy to estimate latent variables. Default learning rate is 0.15. The purpose is to determine whether terrain ruggedness correlates positively with income in African countries while the opposite is true for non-African countries.

The training performance is displayed for each autoguide. In comparison with all other new autoguides, *ToeplitzNorm* performed the best with a loss of 240.38 . It is the same

Table 4.2: Testing autoguides on income dataset

Model	LOSS	LOC(a_1)	LOC(a_2)	LOC(a_3)	LOC(a_4)	LOC(SC)
DiagonalNorm	243.19	9.24	-1.87	-0.06	0.41	-2.21
MultivariateNorm	243.01	9.17	-1.85	-0.18	0.34	-2.20
LowRankMultivariateNorm	239.93	9.08	-1.98	-0.24	0.27	-2.30
PolyDiagNorm	256.25	9.47	-1.67	0.37	-0.05	-0.32
SymmetricNorm	251.32	8.57	-1.56	0.03	-0.53	-0.76
LowRankNorm	299.80	9.15	-1.02	-0.08	0.04	-0.32
BlockDiagNorm	243.30	9.15	-1.81	-0.17	0.34	-2.30
ToeplitzNorm	**240.38**	**9.20**	**-1.78**	**-0.14**	**0.46**	**-2.15**
CirculantNorm	395.07	2.52	-0.68	-0.10	-1.03	5.26
Structured-faithful	243.31	9.23	-1.96	-0.16	0.45	-2.23
Structured-topo	243.31	9.23	-1.96	-0.16	0.45	-2.23
Structured-min-dis	243.31	9.23	-1.96	-0.16	0.45	-2.23

inference level as *LowRankMultivariateNorm*, which is the baseline autoguide. Next, we can measure the density of slope in their regression lines, and also the record of training losses. The density shows no difference for *LowRankNorm*, so there is half the uncertainty for this distribution. It is not surprising because in the original dataset, the data seems to be similarly spread among two distributions . But for *ToeplitzNorm*, the data distribution of non-African countries is more concentrated. So we dig into its posterior predictive uncertainty.

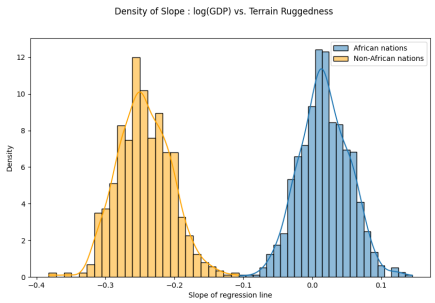

Figure 4.10: Slope Density(LowRank) from Income Data

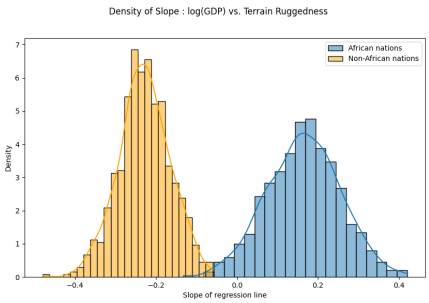

Figure 4.11: Slope Density(Toeplitz) from Income Data

Additionally, their posterior predict ability is different. **ToeplitzNorm** has shown a more apparent positive slope when fitting non-African data than **LowRankNorm**. Therefore, although **LowRankNorm** has given the best training loss, its posterior predictability it not as good as **ToeplitzNorm**, which can infer that our new autoguides are better. In

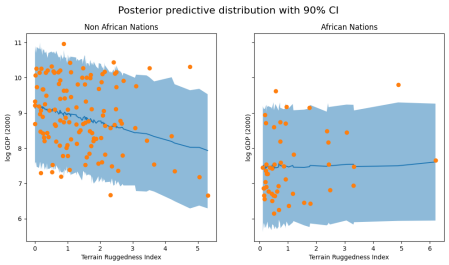

Figure 4.12: Posterior(LowRank) from Income Data

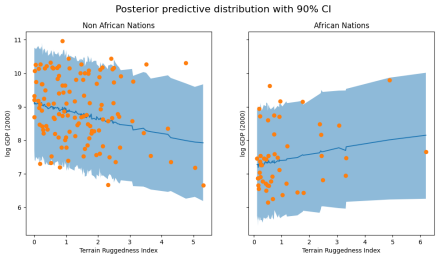

Figure 4.13: Posterior(Toeplitz) from income Data

terms of the stability of autoguides' training loss, we find that overall the loss is stable

except for *PolyDiagNorm* and *SymmetricNorm*. During training, the learning rate is relatively high, which makes convergence less likely. It may be necessary to adjust the learning rate to a smaller value such as 0.05 or 0.02 instead of 0.15 to better achieve a more stable training loss. The loss may also take a longer time to converge.

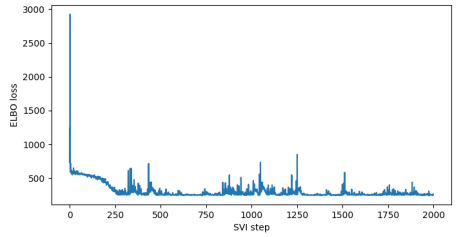

Figure 4.14: Loss(BlockDiagNorm) from Income Data

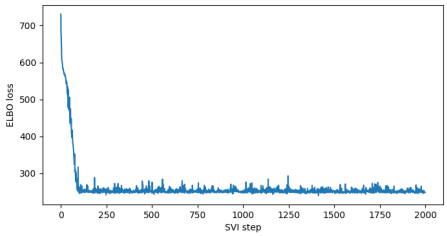

Figure 4.15: Loss(ToeplitzNorm) from Income Data

MNIST On the MNIST dataset, we test the autoguides to guide the model. Specifically, the model is our decoder module in the variational autoencoder structure. First, we disregard the inference network (encoder module) and use autoguide to guide the model parameters. Then we compare the results with those that use a traditional neural network as the guide, which is a traditional non-autoguide type inference network.

SymmetricNorm, *LowRankNorm* and *CirculantNorm* performed better on training loss than the baseline autoguides. In our comparison of their embeddings, we find that, despite the autoguides having a lower training loss, no significant difference can be observed between the two classes in the t-SNE visualization. Despite better performance, given the latent variables, the new autoguides cannot recover to the original data. We illustrate this by looking at the reconstructed data generated by the program. When we guide the model using the same parameter settings without autoguide, the data is not well reconstructed, but we can deduce the number from each image. When it comes to the autoguide, we can see that none of the images are clear because it is badly reconstructed. Therefore, even with the plate model shown in figure 2.3, we conclude that it is better to use a simple guide (neural network encoder) instead of an autoguide.

Table 4.3: Testing autoguides on MNIST dataset

Model	TEST ELBO	LATENT DIM	HIDDEN DIM
DiagonalNorm	206.35	50	100
MultivariateNorm	208.38	10	10
LowRankMultivariateNorm	206.52	20	10
PolyDiagNorm	206.37	50	100
SymmetricNorm	**205.89**	20	10
LowRankNorm	**205.90**	20	10
BlockDiagNorm	212.85	20	10
ToeplitzNorm	/	20	10
CirculantNorm	**205.87**	10	10
Structured-topo	206.30	10	10

OSIC dataset Our new autoguides are not as well as the baseline autoguides when considering the lost on OSIC datasets. In addition, the autoguide *BlockDiagNorm* suffers from gradient exploding, which is a common problem in high-dimensional dataset training, even if we use the clipping Adam optimizer for training. *ToeplitzNorm*

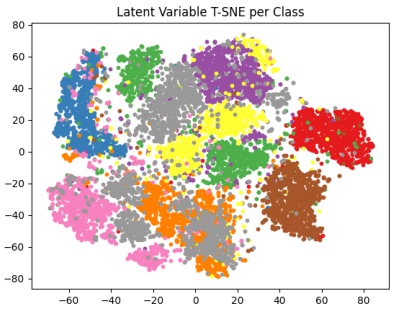

Figure 4.16: embedding(PolyDiag)

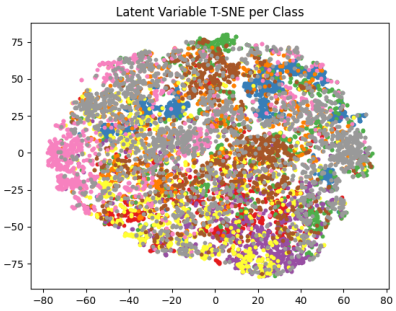

Figure 4.17: embedding(Circulant)

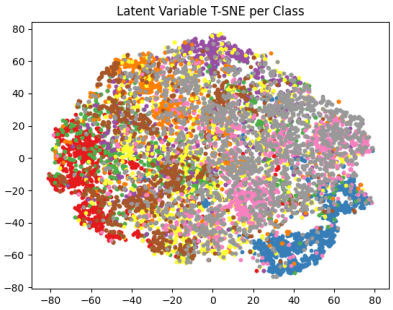

Figure 4.18: embedding(Symmetric)

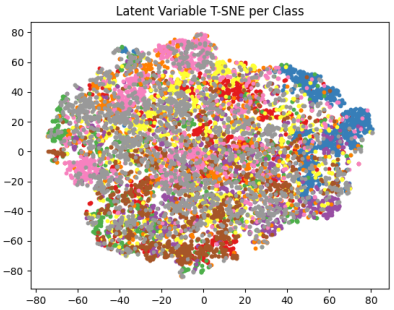

Figure 4.19: embedding(BlockDiag)

Figure 4.20: Embeddings from four autoguides on MNIST

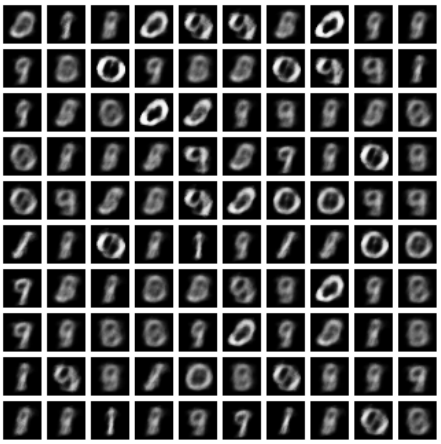

Figure 4.21: No autoguide

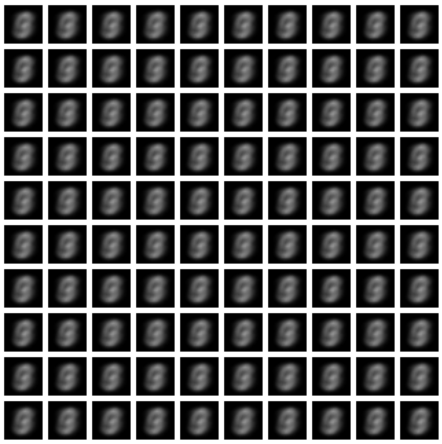

Figure 4.22: With autoguide

is slow in training, and *CirculantNorm* is unstable in its loss. *BlockDiagNorm* requires a small scaled coefficient of 0.0001 multiplied by the identity matrix in order to work. To reduce a high computation time cost, *ToeplitzNorm* should not include matrix generation by *for* operation in python but instead by performing the Toeplitz function within the Scipy package. On the other hand, although training *CirculantNorm* was not stable, it returned the lowest initial loss, which might be regarded as a good initialization method.

Observing the learned FVC for *AutoNormal* and *SymmetricNorm*, we can see that the *AutoNormal* learned Bayesian linear regressions well. Orange and red lines are almost in line with each other. It predicts a higher uncertainty where the data points are not

Table 4.4: Testing autoguides on OSIC dataset

Model	TEST LOSS	LR
DiagonalNorm	12340.73	0.11
MultivariateNorm	13093.07	0.01
LowRankMultivariateNorm	13293.03	0.01
PolyDiagNorm	14158.62	0.05
SymmetricNorm	13913.47	0.05
LowRankNorm	14720.32	0.06
BlockDiagNorm	/	0.001
ToeplitzNorm	slow	0.02
CirculantNorm	14756.31	0.001
Structured-topo	15404.57	0.004
Structured-min-dis	15433.26	0.004

showing a predicted slope as shown in the first patient. However, it predicts a higher confidence for patient 2 and patient 3. When it comes to *SymmetricNorm*, it adequately learned Bayesian Linear Regressions, but it does not ensure a good confidence interval. As a result, there is a tradeoff between a high fitness and a small variance (confidence interval). Several autoguides follow the same pattern, so if we want to explore more concentrated patterns for new autoguides, new parameters' initialization should be fully explored. From this, it can be concluded that it is difficult to determine whether our new method performs better than any other autoguide in all aspects. Although it might improve the loss, level up the ELBO, it might damage concentration on data patterns.

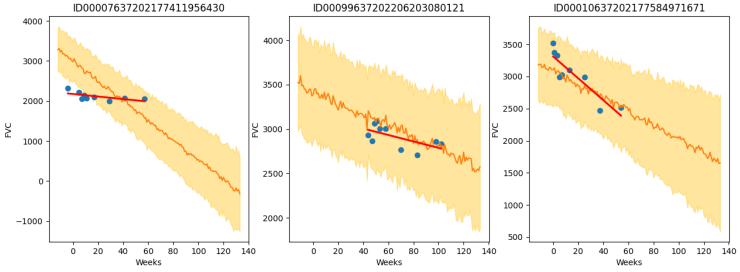

Figure 4.23: FVC AutoNormal from OSIC data

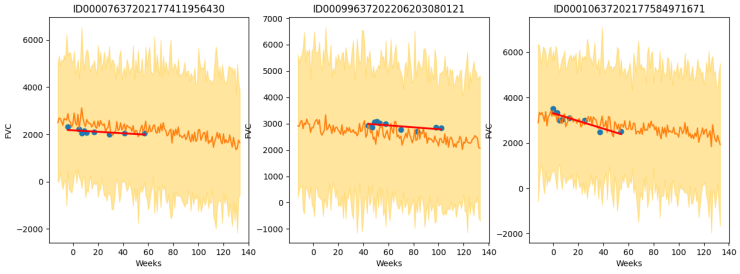

Figure 4.24: FVC SymmetricNorm from OSIC data

SARS-COV-2 We found that in such a high dimensional dataset with structured latent variables, two methods based on inverse dependency models performed better than covariance based autoguides with a loss of 12.04. They showed a good mean average

error (MAE) in the inference step. Again, *BlockDiagNorm* suffers from gradient explosion. Performing *ToeplitzNorm* is too slow because there are 538452 latent variables and 4806046 learnable parameters, which is computationally difficult as mentioned before.

Table 4.5: Testing autoguides on SARS-COV-2

Model	LOSS	MAE(OVERALL)	MAE(MASSACHUSETTS)	LR
DiagonalNorm	10.34	0.19	0.10	0.01
MultivariateNorm	691.6	1.33	0.58	0.005
LowRankMultivariateNorm	20.71	0.25	0.26	0.009
PolyDiagNorm	73.03	0.85	0.41	0.005
SymmetricNorm	81.85	0.82	0.43	0.005
LowRankNorm	634.50	1.13	0.47	0.005
BlockDiagNorm	/	/	/	/
ToeplitzNorm	slow	/	/	/
CirculantNorm	77.61	0.73	0.39	0.005
Structured-topo	12.04	0.16	0.20	0.1
Structured-min-dis	12.04	0.16	0.20	0.1

Looking into the training session of *structured-topo* and *CirculantNorm*, for example, *structured-topo* shows a sharper gradient descent and lower loss after 1000 epochs. Other parameters also decreased significantly. We can see from the figure that the loss can be further reduced even though it is at the same level as the *AutoDiagonalNorm* in baseline autoguides. By setting better learning rates or optimizer's parameters, loss can be further reduced. In general, covariance-based autoguides with the same parameters have relatively small gradients and slow convergence. As a result, we should take into consideration the structured autoguide when analyzing the dataset with high dimensional latent variables with structured model.

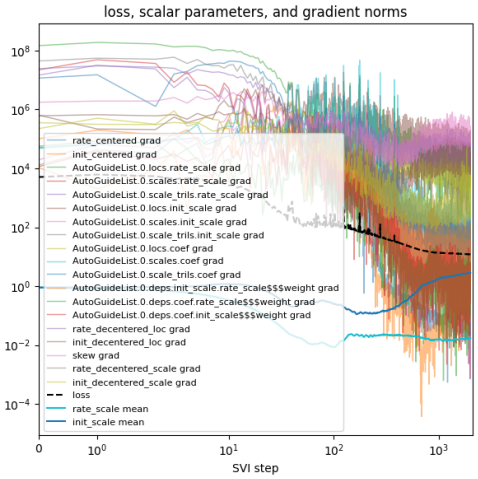

Figure 4.25: Info for structured-topo from SARS-COV-2 data

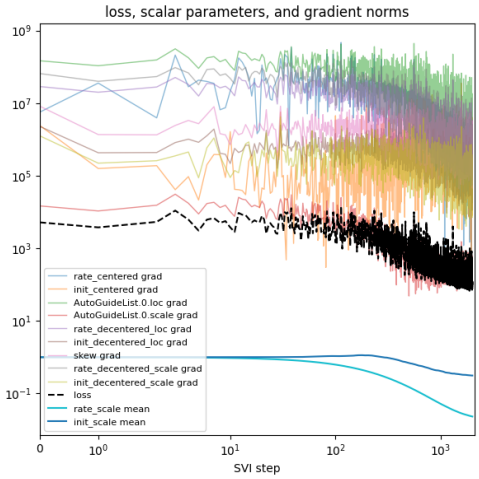

Figure 4.26: Info for CirculantNorm from SARS-COV-2 data

Daily S&P 500 dataset We found that the new autoguides, except *PolyDiagNorm*, do not show any improvement in loss. Furthermore, we found that the learned latent variables have similar ranges, except for h_0. Since h_0 is the initial point, we infer that if the initial position of Brownian motion can be learned in the Levy model, then the loss will be small. The loss fluctuated greatly because the new autoguides may not learn

this initial point. Therefore, we can have an idea, for example, let the autoguide in a baseline learn this initial position to determine the empirical distribution, and then use the learned parameters to re-initialize the model and reduce the variance.

Table 4.6: Testing autoguides on daily S&P 500 dataset

Model	LOSS	h_0	r_{loc}	r_{skew}	$r_{stability}$	SIGMA
DiagonalNorm	-2.81	-0.054 ± 0.101	0.011 ± 0.004	-0.026 ± 0.013	1.722 ± 0.017	2.021 ± 0.058
MultivariateNorm	-0.20	0.037 ± 0.064	0.008 ± 0.008	-0.178 ± 0.021	1.152 ± 0.0302	2.182 ± 0.068
LowRankMultivariate	-0.36	0.095 ± 0.065	0.001 ± 0.007	-0.298 ± 0.022	0.930 ± 0.034	2.189 ± 0.035
PolyDiagNorm	-1.10	0.005 ± 0.194	0.131 ± 0.077	-0.026 ± 0.064	1.674 ± 0.036	0.835 ± 0.010
SymmetricNorm	8.34	0.482 ± 0.00	-0.005 ± 0.000	-0.082 ± 0.000	1.049 ± 0.000	1.668 ± 0.000
LowRankNorm	19.43	0.484 ± 0.043	0.040 ± 0.045	-0.071 ± 0.027	1.047 ± 0.023	1.667 ± 0.069
BlockDiagNorm	7.54	0.560 ± 0.036	-0.003 ± 0.035	-0.043 ± 0.020	1.082 ± 0.020	1.540 ± 0.051
ToeplitzNorm	slow	/	/	/	/	/
CirculantNorm	8.90	0.560 ± 0.000	-0.003 ± 0.000	-0.043 ± 0.000	1.082 ± 0.000	1.543 ± 0.000
Structured-topo	7.96	/	/	/	/	/
Structured-min-dis	7.96	/	/	/	/	/

We compared the priority order of nodes when running two structured autoguides. The results obtained in the actual running process are similar, showing that different inverses can achieve similar results. *AutoStructured* object in Pyro has no attribute quantiles so we do not generate the latent variable distributions for *AutoStructured* . Below are the results of two reversing algorithms applied to the daily S&P 500 dataset: *Structured-topo*: {'r': 0, '$r\_t\_exponential$': 1, '$r\_t\_uniform$': 2, '$r\_z\_exponential$': 3, '$r\_z\_uniform$': 4, '$r\_stability$': 5, 'r_skew': 6, 'r_loc': 7, 'v_dct': 8, '$sigma$': 9, 'h_0': 10}

Structured-min-dis: {'r': 0, '$sigma$': 1, '$r\_stability$': 2, 'v_dct': 3, 'r_loc': 4, '$r\_z\_exponential$': 5, '$r\_t\_uniform$': 6, '$r\_z\_uniform$': 7, '$r\_t\_exponential$': 8, 'r_skew': 9, 'h_0': 10}

Both of the inverse dependency models will lead to a similar loss and predictability. The figures 4.27 and 4.28 show that volatility is approximately equal to the areas of log

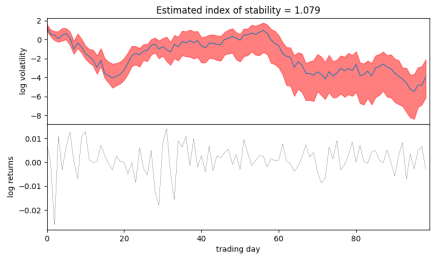

Figure 4.27: Log return CirculantNorm from S&P 500

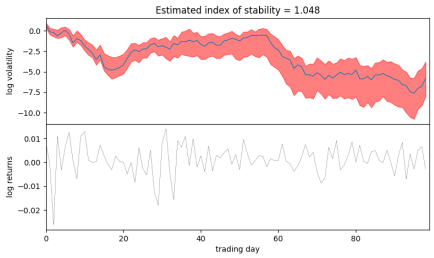

Figure 4.28: Log return SymmetricNorm from S&P 500

returns in the last 100 days for *CirculantNorm* and *SymmetricNorm*. This uncertainty has been underestimated because *AutoDiagonalNormal* was used as an approximate guide. Hamiltonian Monte Carlo (HMC) [70] or No-U-Turn Sampler (NUTS) [71] provide more precise uncertainty estimates. We recommend using MCMC sampling over autoguides in this case as the result is too rough even if the guide is created with the best of intentions. The gradient exploding problem also occurs when running *ToeplitzNorm* on this dataset with high dimensional latent variables.

Chapter 5

Discussions and Conclusions

5.1 Results overview

In comparison to the baseline's four autoguides, our eight new autoguides have been tested on six datasets. It is shown that *SymmetricNorm* and *LowRankNorm* (new) are optimized on GMM dataset, and that these two also have relatively large variance problems. On income dataset, it is found that *ToeplitzNorm* is the best, and it is able to learn better density and slope than the basic autoguides. The new autoguide is able to reduce loss on the data of MNIST, but not as much as the loss obtained by designing with a traditional inference network. The new autoguide may not perform as well on OSIC data in terms of loss, but it has a relatively good degree of data fitting, which is achieved through the posterior. The performance of the structured autoguide on the SARS-CoV-2 dataset is on par with the baseline autoguides.

In light of the above analysis, one can conclude that autoguides based on covariance matrices can be useful in models that have relatively low design dimensions and not so strong dependencies among variables. However, some autoguides based on an inverted dependency model can be used when the dimensions are relatively large and the relationships between variables are complicated. The initial autoguides, in general, can generally be improved. We also discovered that autoguide cannot be used in deep generative models. As an example, variational auto-encoders still need to use an nn-based encoder as an inference network rather than an autoguide model.

5.2 Discussions and future works

There are a lot of problems during training, such as initialization. When testing SARS-CoV-2 and OSIC datasets, the initial value for the loc and scale is relatively large, which leads to the problem of gradient explosion in some algorithms when dealing with high dimensions, so it is necessary to greatly reduce the initial value of loc and scale. *ToeplitzNorm* autoguide runs very slowly when dealing with large dimensions. It is also necessary to judge that the determinant of all submatrices is larger than zero to keep positive definite. Thus, for high-dimensional matrices, it may be necessary to

use Scipy's algorithm to expand the matrix and convert it to a positive definite matrix. Scipy does not yet support computing torch tensor with gradients, so the scale cannot be trained. We can solve this problem by adding scale to loc or by multiplying scale by the product of covariance matrices. Besides, we must compromise between mean and variance. Daily S&P 500 and OSIC dataset indicate that although the new autoguide performs well in inferring mean parameters, the variances are large, which means that the confidence interval is uncertain. It is necessary to devise a method for gradually reducing the scale, leaving aside the manual adjustment method mentioned above.

Baseline selection is the subject of another discussion. At present, datasets are selected based on autoguide availability, but for some datasets with hidden variables of higher dimensions, other methods may be needed to compare autoguides, such as MCMC [2], NUTS [71], and other well-performed sampling methods, rather than only baseline autoguides. Our future plans include incorporating state-of-the-art inference methods and sampling-based methods. Furthermore, we might include the IC-based RNN style autoguide in the future.

Also, we found that although different seeds get similar results, the more accurate method of getting report loss or accuracy still needs to be averaged by testing different seeds. This is to demonstrate that the method is robust.

5.3 Conclusions

Six new autoguides based on different types of covariance matrices were implemented in this project, which are *PolyDiagNorm*, *SymmetricNorm*, *LowRankNorm*, *BlockDiagNorm*, *ToeplitzNorm*, and *CircularNorm*. With latent-to-observed style and observed-to-latent style, we use the implemented *AutoStructured-faithful* and our new autoguides *Structured-topo* and *Structured-min-dis*, which are based on topological ordering of the original graph and minimum distance orderings from the observed node. Together with another three baseline autoguides *AutoDiagonalNormal*, *AutoMultivariateNormal* and *AutoLowRankMultivariateNormal*, we perform the guide tests on some benchmarking datasets in classical inference problems and come into the conclusions for each individual dataset. In addition, we find some general rules that covariance-based autoguides improve some aspects when latent variables are low dimensional, but inverse dependency-based autoguides improve some aspects when latent variables are high dimensional and complicated in structure. Autoguides are not suitable for inferring latent variable parameters in deep generative models, such as variational autoencoders and deep markov models. The inference network should instead be an neural network based encoder (guide).

Bibliography

[1] George Casella and Edward I George. Explaining the gibbs sampler. *The American Statistician*, 46(3):167–174, 1992.

[2] Christian Robert and George Casella. A short history of markov chain monte carlo: Subjective recollections from incomplete data. *Statistical Science*, 26(1):102–115, 2011.

[3] Christian P Robert and George Casella. The metropolis—hastings algorithm. In *Monte Carlo statistical methods*, pages 231–283. Springer, 1999.

[4] Matthew D Hoffman, David M Blei, Chong Wang, and John Paisley. Stochastic variational inference. *Journal of Machine Learning Research*, 2013.

[5] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[6] Thomas N Kipf and Max Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016.

[7] Rajesh Ranganath, Sean Gerrish, and David Blei. Black box variational inference. In *Artificial intelligence and statistics*, pages 814–822. PMLR, 2014.

[8] Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *International conference on machine learning*, pages 1530–1538. PMLR, 2015.

[9] Mike Wu, Kristy Choi, Noah Goodman, and Stefano Ermon. Meta-amortized variational inference and learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 6404–6412, 2020.

[10] David M Blei, Alp Kucukelbir, and Jon D McAuliffe. Variational inference: A review for statisticians. *Journal of the American statistical Association*, 112(518):859–877, 2017.

[11] Nicholas J Higham. Cholesky factorization. *Wiley Interdisciplinary Reviews: Computational Statistics*, 1(2):251–254, 2009.

[12] Alp Kucukelbir, Dustin Tran, Rajesh Ranganath, Andrew Gelman, and David M Blei. Automatic differentiation variational inference. *The Journal of Machine Learning Research*, 18(1):430–474, 2017.

[13] Stefan Webb, Adam Golinski, Rob Zinkov, Tom Rainforth, Yee Whye Teh, Frank Wood, et al. Faithful inversion of generative models for effective amortized inference. *Advances in Neural Information Processing Systems*, 31, 2018.

[14] Paul Tseng and Sangwoon Yun. A coordinate gradient descent method for non-smooth separable minimization. *Mathematical Programming*, 117(1):387–423, 2009.

[15] David M. Blei, Alp Kucukelbir, and Jon D. McAuliffe. Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 112(518):859–877, Apr 2017.

[16] Matthew D Hoffman and David M Blei. Structured stochastic variational inference. In *Artificial Intelligence and Statistics*, pages 361–369, 2015.

[17] Tihomir Asparouhov and Bengt Muthén. Auxiliary variables in mixture modeling: Three-step approaches using m plus. *Structural equation modeling: A multidisciplinary Journal*, 21(3):329–341, 2014.

[18] Dustin Tran, Rajesh Ranganath, and David M Blei. The variational gaussian process. *arXiv preprint arXiv:1511.06499*, 2015.

[19] Dustin Tran, David Blei, and Edo M Airoldi. Copula variational inference. In *Advances in Neural Information Processing Systems*, pages 3564–3572, 2015.

[20] Rajesh Ranganath, Dustin Tran, and David Blei. Hierarchical variational models. In *International Conference on Machine Learning*, pages 324–333. PMLR, 2016.

[21] Eli Bingham, Jonathan P Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul Szerlip, Paul Horsfall, and Noah D Goodman. Pyro: Deep universal probabilistic programming. *The Journal of Machine Learning Research*, 20(1):973–978, 2019.

[22] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

[23] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

[24] Christian Naesseth, Francisco Ruiz, Scott Linderman, and David Blei. Reparameterization gradients through acceptance-rejection sampling algorithms. In *Artificial Intelligence and Statistics*, pages 489–498. PMLR, 2017.

[25] Durk P Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improved variational inference with inverse autoregressive flow. *Advances in neural information processing systems*, 29, 2016.

[26] Yuri Burda, Roger Grosse, and Ruslan Salakhutdinov. Importance weighted autoencoders. *arXiv preprint arXiv:1509.00519*, 2015.

[27] Rajesh Ranganath, Dustin Tran, Jaan Altosaar, and David Blei. Operator variational inference. *Advances in Neural Information Processing Systems*, 29, 2016.

[28] Jan-Willem van de Meent, Brooks Paige, Hongseok Yang, and Frank Wood. An introduction to probabilistic programming, 2021.

[29] Marenglen Biba. Integrating logic and probability: Algorithmic improvements in markov logic networks. *PHD, university of Bari, italy*, 2009.

[30] Brooks Paige and Frank Wood. A compilation target for probabilistic programming languages. In *International Conference on Machine Learning*, pages 1935–1943. PMLR, 2014.

[31] Brian Milch, Bhaskara Marthi, and Stuart Russell. Blog: Relational modeling with unknown objects. In *ICML 2004 workshop on statistical relational learning and its connections to other fields*, pages 67–73, 2004.

[32] John Salvatier, Thomas V Wiecki, and Christopher Fonnesbeck. Probabilistic programming in python using pymc3. *PeerJ Computer Science*, 2:e55, 2016.

[33] Noah Goodman, Vikash Mansinghka, Daniel M Roy, Keith Bonawitz, and Joshua B Tenenbaum. Church: a language for generative models. *arXiv preprint arXiv:1206.3255*, 2012.

[34] Frank Wood, Jan Willem Meent, and Vikash Mansinghka. A new approach to probabilistic programming inference. In *Artificial Intelligence and Statistics*, pages 1024–1032. PMLR, 2014.

[35] Vikash Mansinghka, Richard Tibbetts, Jay Baxter, Pat Shafto, and Baxter Eaves. Bayesdb: A probabilistic programming system for querying the probable implications of data. *arXiv preprint arXiv:1512.05006*, 2015.

[36] Mario Lezcano Casado. *Compiled inference with probabilistic programming for large-scale scientific simulations*. PhD thesis, University of Oxford, 2017.

[37] Lawrence M Murray and Thomas B Schön. Automated learning with a probabilistic programming language: Birch. *Annual Reviews in Control*, 46:29–43, 2018.

[38] Bob Carpenter, Andrew Gelman, Matthew D Hoffman, Daniel Lee, Ben Goodrich, Michael Betancourt, Marcus Brubaker, Jiqiang Guo, Peter Li, and Allen Riddell. Stan: A probabilistic programming language. *Journal of statistical software*, 76(1):1–32, 2017.

[39] T Minka, JM Winn, JP Guiver, Y Zaykov, D Fabian, and J Bronskill. Infer .net 2.7.(2018). *URL http://research. microsoft. com/infernet. Microsoft Research Cambridge*, 1, 2018.

[40] Narayanaswamy Siddharth, Brooks Paige, Jan-Willem Van de Meent, Alban Desmaison, Noah D Goodman, Pushmeet Kohli, Frank Wood, and Philip HS Torr. Learning disentangled representations with semi-supervised deep generative models. *arXiv preprint arXiv:1706.00400*, 2017.

[41] Tuan Anh Le, Atilim Gunes Baydin, and Frank Wood. Inference compilation and universal probabilistic programming. In *Artificial Intelligence and Statistics*, pages 1338–1348. PMLR, 2017.

[42] Du Phan, Neeraj Pradhan, and Martin Jankowiak. Composable effects for flexible and accelerated probabilistic programming in numpyro. *arXiv preprint arXiv:1912.11554*, 2019.

[43] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[44] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[45] Douglas A Reynolds. Gaussian mixture models. *Encyclopedia of biometrics*, 741:659–663, 2009.

[46] David J Olive. *Statistical theory and inference*. Springer, 2014.

[47] Richard Arnold Johnson, Dean W Wichern, et al. *Applied multivariate statistical analysis*, volume 6. Pearson London, UK:, 2014.

[48] Paul Rozin. The process of moralization. *Psychological science*, 10(3):218–221, 1999.

[49] Brooks Paige and Frank Wood. Inference networks for sequential monte carlo in graphical models. In *International Conference on Machine Learning*, pages 3040–3049. PMLR, 2016.

[50] Andreas Stuhlmüller, Jacob Taylor, and Noah Goodman. Learning stochastic inverses. *Advances in neural information processing systems*, 26, 2013.

[51] Tuan Anh Le. *Amortized inference and model learning for probabilistic programming*. PhD thesis, University of Oxford, 2019.

[52] Samuel Gershman and Noah Goodman. Amortized inference in probabilistic reasoning. In *Proceedings of the annual meeting of the cognitive science society*, volume 36, 2014.

[53] Peter Dayan, Geoffrey E Hinton, Radford M Neal, and Richard S Zemel. The helmholtz machine. *Neural computation*, 7(5):889–904, 1995.

[54] Geoffrey E Hinton, Peter Dayan, Brendan J Frey, and Radford M Neal. The" wake-sleep" algorithm for unsupervised neural networks. *Science*, 268(5214):1158–1161, 1995.

[55] Jörg Bornschein and Yoshua Bengio. Reweighted wake-sleep. *arXiv preprint arXiv:1406.2751*, 2014.

[56] Tuan Anh Le, Adam R Kosiorek, N Siddharth, Yee Whye Teh, and Frank Wood. Revisiting reweighted wake-sleep for models with stochastic control flow. In *Uncertainty in Artificial Intelligence*, pages 1039–1049. PMLR, 2020.

[57] Shixiang Gu, Zoubin Ghahramani, and Richard E Turner. Neural adaptive sequential monte carlo. *arXiv preprint arXiv:1506.03338*, 2015.

[58] Yura N Perov, Tuan Anh Le, and Frank Wood. Data-driven sequential monte carlo in probabilistic programming. *arXiv preprint arXiv:1512.04387*, 2015.

[59] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[60] Richard McElreath. rethinking: Statistical rethinking book package. *R package version*, 1, 2016.

[61] Nathan Nunn and Diego Puga. Ruggedness: The blessing of bad geography in africa. *Review of Economics and Statistics*, 94(1):20–36, 2012.

[62] Li Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE signal processing magazine*, 29(6):141–142, 2012.

[63] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and Andre Van Schaik. Emnist: Extending mnist to handwritten letters. In *2017 international joint conference on neural networks (IJCNN)*, pages 2921–2926. IEEE, 2017.

[64] Zhimeng Pan, Brian Rodriguez, and Rajesh Menon. Machine-learning enables image reconstruction and classification in a "see-through" camera. *OSA Continuum*, 3(3):401–409, 2020.

[65] Smitha Milli, Ludwig Schmidt, Anca D Dragan, and Moritz Hardt. Model reconstruction from model explanations. In *Proceedings of the Conference on Fairness, Accountability, and Transparency*, pages 1–9, 2019.

[66] Sampurna Mandal, Valentina E Balas, Rabindra Nath Shaw, and Ankush Ghosh. Prediction analysis of idiopathic pulmonary fibrosis progression from osic dataset. In *2020 IEEE International conference on computing, power and communication technologies (GUCON)*, pages 861–865. IEEE, 2020.

[67] Eric Jacquier, Nicholas G Polson, and Peter E Rossi. Bayesian analysis of stochastic volatility models. *Journal of Business & Economic Statistics*, 20(1):69–87, 2002.

[68] Stephen J Taylor. Modeling stochastic volatility: A review and comparative study. *Mathematical finance*, 4(2):183–204, 1994.

[69] Andrew Gelman, Aki Vehtari, Daniel Simpson, Charles C Margossian, Bob Carpenter, Yuling Yao, Lauren Kennedy, Jonah Gabry, Paul-Christian Bürkner, and Martin Modrák. Bayesian workflow. *arXiv preprint arXiv:2011.01808*, 2020.

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

[71] Matthew D Hoffman, Andrew Gelman, et al. The no-u-turn sampler: adaptively setting path lengths in hamiltonian monte carlo. *J. Mach. Learn. Res.*, 15(1):1593–1623, 2014.

Appendix A

Others

A.1 Code implementations

This github repository contains more results and the full implementation of the code: `https://github.com/JIAQING-XIE/advi_nips`. The dataset and plotting implementation were adapted from `http://github.com/pyro-ppl/pyro`.