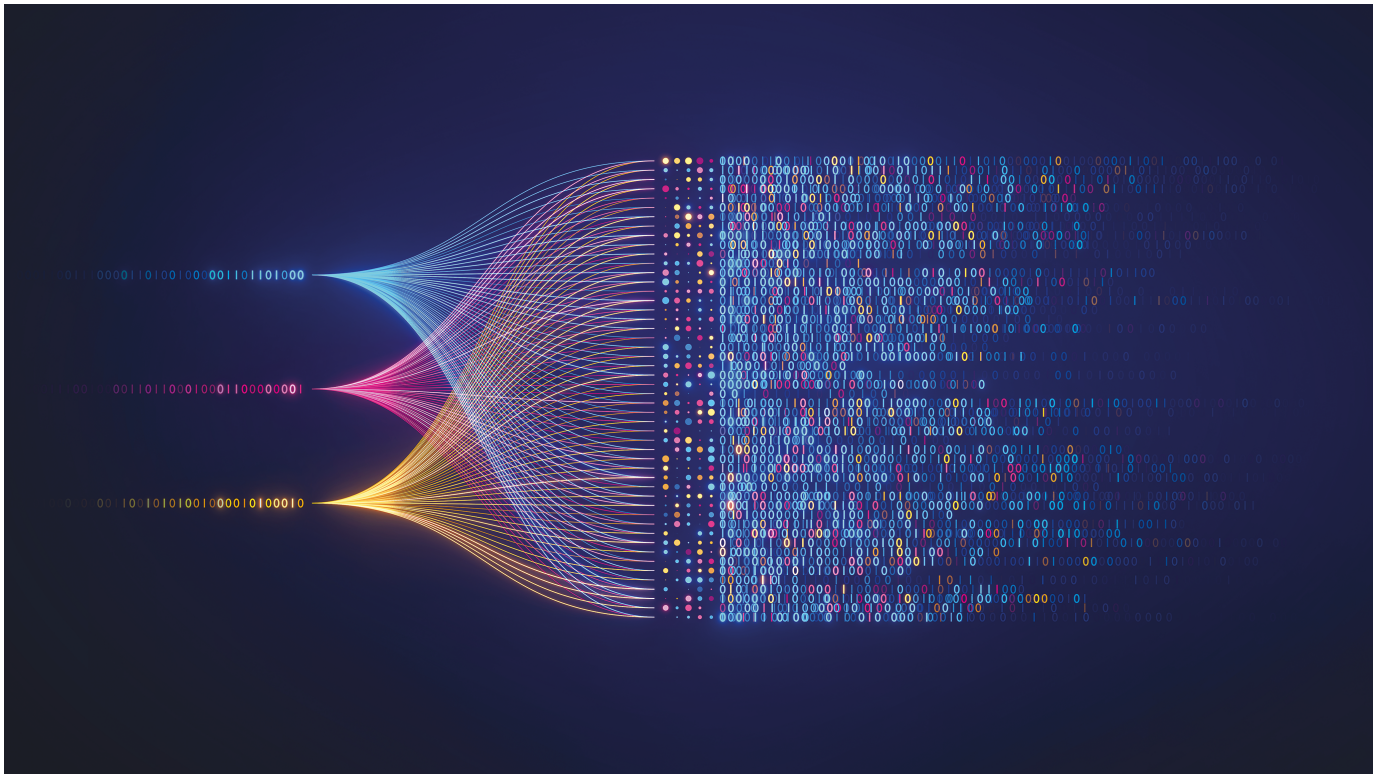
Normalizing Flows Aided Variational Inference: A Useful Alternative to MCMC?



Sumegha Premchandar, Bhattacharya Shrijita, and Maiti Tapabrata

1. Introduction

A major area of contemporary statistics research is learning to model probability distributions of varying complexity. The problem of learning to characterize probability distributions broadly takes two forms: estimating a probability density given samples from it and approximating densities that are known only up to a normalizing constant. The latter avenue of research has applications in Bayesian inference, where we wish to generate samples from the posterior distribution of model parameters given observed data.

This review aims to discuss the use of normalizing flows for variational inference (VI), a method wherein we can approximate and sample from complex probability densities [RM15]. This type of probabilistic modeling lies in the second avenue of research, where we do not have a normalizing constant for probability densities of interest. VI is a tool that emerged in machine learning to

Sumegha Premchandar is a graduate student at Michigan State University. Her email address is premchan@msu.edu.

Bhattacharya Shrijita is an assistant professor in the Department of Statistics and Probability at Michigan State University. Her email address is bhatta61@msu.edu.

Maiti Tapabrata is a professor in the Department of Statistics and Probability at Michigan State University. His email address is maiti@msu.edu.

Communicated by Notices *Associate Editor Richard Levine.*

DOI: https://doi.org/10.1090/noti2742

approximate probability densities. It is often applied in Bayesian statistics as a more scalable alternative to Markov Chain Monte Carlo (MCMC) methods for large datasets. Although scalable, earlier works such as mean-field or structured VI are limited when approximating more complex, multimodal probability distributions. Normalizing flows are mappings from a simple base distribution to a more complex probability distribution. They are primarily used for modeling continuous distributions and can be used to specify very flexible probability models, thus improving the accuracy of VI algorithms.

There already exist comprehensive reviews for normalizing flow methods in general. An overview of different normalizing flow families is provided in [KPB2005], while [PNR+21] goes into depth on each family of flow models and extends this discussion to newer areas, such as flows for discrete variables. These reviews are an overarching look at flows for probabilistic modelling and are focussed on applications in the machine learning literature. Discussion of applications of a more classical statistical nature is limited. An excellent exposition and survey of VI from a statistical lens is given in [BKM17]. However, they only cover variational families of a parametric nature, such as mean-field and structured VI. We extend the discussion to variational families specified by normalizing flow models. Further, our review is written for readers entirely new to the area.

In latent variable modeling, we aim to learn the conditional distribution of latent variables $\boldsymbol{z} = (z_1, z_2, \ldots z_d)$ given observed data $\boldsymbol{x} = (x_1, x_2, \ldots x_n)$, that is, $\pi(\boldsymbol{z}|\boldsymbol{x})$. We explain how solving this problem is useful in Bayesian statistics. In parametric statistics, stochasticity in the observed data is often described using a specific probability distribution $p(\boldsymbol{x}|\boldsymbol{z})$, where $\boldsymbol{z}$ needs to be estimated from the data $\boldsymbol{x}$. In Bayesian inference, we assume a prior distribution $\pi(\boldsymbol{z})$ on $\boldsymbol{z}$ representing our beliefs about the model parameter *prior* to observing the data. Based on the data, we update our beliefs via the posterior distribution $\pi(\boldsymbol{z}|\boldsymbol{x})$. The posterior can be calculated by Bayes theorem:[1]

$$\pi(\boldsymbol{z}|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|\boldsymbol{z})\pi(\boldsymbol{z})}{\int_{\boldsymbol{z}} p(\boldsymbol{x}|\boldsymbol{z})\pi(\boldsymbol{z})d\boldsymbol{z}}.$$

For cases where the marginal likelihood $m(\boldsymbol{x}) = \int_{\boldsymbol{z}} p(\boldsymbol{x}|\boldsymbol{z})\pi(\boldsymbol{z})d\boldsymbol{z}$ is intractable we resort to approximate inference. MCMC methods have long been the go-to for sampling from posterior distributions when $m(\boldsymbol{x})$ cannot be computed. MCMC algorithms generate samples from a Markov Chain whose stationary distribution converges to the target distribution of interest. One prominent example is the Metropolis-Hastings method [CG95], of which the Gibbs sampling algorithm [CG92] is a special case. However, these methods may not always scale well to high-dimensional models and can be slow to converge for multimodal distributions. VI has shown promise as a scalable alternative to MCMC. In VI, the target distribution is approximated by a family of distributions Q among which we choose the optimal distribution q_{ϕ^*} to be "closest" to the target. To determine "closeness," KL-divergence is often used. Intuitively, KL-divergence is something akin to a distance between 2 probability distributions. Thus, probabilistic modeling with VI becomes an optimization problem:

$$q_{\phi^*} \in \operatorname{argmin}_{q_\phi \in Q} KL(q_\phi || \pi(.|\boldsymbol{x})).$$

Mean-field VI (MF-VI) is a popular approach in which the variational family Q is defined based on the assumption that latent variables are independent. The mean-field assumption is useful for faster computations during optimization but is restricted in the complexity of densities we can approximate. Structured VI takes this one step further by allowing dependencies across latent variables. However, even with Structured VI we cannot guarantee that we can approximate any density arbitrarily well. This is where normalizing flows come in.

When should we use normalizing flows VI? In [BKM17], the authors observe that "VI is suited to large data sets and scenarios where we want to quickly explore many models; MCMC is suited to smaller data sets and scenarios where we happily pay a higher computational cost for more precise samples." While this is generally true of MF-VI, normalizing flows VI lies somewhere between MCMC and other variational approximation approaches in terms of computational efficiency and accuracy. To shed some light on how normalizing flows VI compares to other sampling methods such as MCMC and MF-VI, we implement variational inference with neural autoregressive flows [HLCK18] for several examples. These examples cover classical Bayesian statistical applications in exponential family models, Gaussian linear regression and logistic regression. We cover scenarios of varying dimensions and complexity of the target distribution. This gives us a high-level idea of scalability vs. accuracy for these methods but is by no means a rigorous treatment of the topic.

We begin the following section by introducing normalizing flows and elaborate on how to use them for VI. We then proceed to examples in Section 3. Finally, we discuss some important takeaways & challenges remaining in the area in Section 4.

2. Normalizing Flows

The main idea behind normalizing flows is to transform some simple base distribution on a continuous support into a "target" distribution that is usually more complex, via a series of bijective, differentiable transformations (diffeomorphisms).

[1] *In much of the Bayesian inference literature, θ will be used for the unknown parameter instead of $\boldsymbol{z}$. We use $\boldsymbol{z}$ to be consistent with general VI literature* [BKM17], [RM15].

Let $Z \in \mathbb{R}^d$ be a random variable whose density we wish to model. We begin with a random variable U sampled from some base distribution $p_U(\boldsymbol{u})$ also defined on support $\mathbb{R}^d$ and apply a diffeomorphism $T : \mathbb{R}^d \to \mathbb{R}^d$ such that $Z = T(U)$. The density of Z is then given by the change of variable formula:

$$p_Z(\boldsymbol{z}) = p_U(\boldsymbol{u})|J_T(\boldsymbol{u})|^{-1}.$$

$|J_T(\boldsymbol{u})|$ denotes determinant of Jacobian of T w.r.t $\boldsymbol{u}$. Thus, the function T transforms the density $p_U(\boldsymbol{u})$ into $p_Z(\boldsymbol{z})$. This process, wherein samples from one probability density "flow" through a mapping to obtain another density is called a normalizing flow.

A natural question to ask is whether normalizing flows can be used to transform a simple base distribution (e.g., uniform or standard normal distribution) into ***any*** target distribution. [PNR+21] contains a constructive argument to show that normalizing flows can indeed recover any target density under rather general conditions. In practice, this is heavily dependent on the transformations T that we employ.

Discrete and continuous-time flows. Normalizing flows are mainly of two types—discrete time (finite flows) and continuous time (infinitesimal flows) [PNR+21]. Discrete-time normalizing flows are constructed by choosing a finite sequence of transformations $T_1, T_2, ... T_K$ and applying them successively to some base distribution $p_U(\boldsymbol{u})$ such that $\boldsymbol{z}_K = T_K \circ T_{K-1} ... \circ T_1(\boldsymbol{u})$. Since we choose all transformations to be diffeomorphisms, the change of variables formula applies and we have:

$$p_{Z_K}(\boldsymbol{z}_K) = p_U(\boldsymbol{u}) \times |J_{T_K}(\boldsymbol{z}_{K-1})|^{-1} \times |J_{T_{K-1}}(\boldsymbol{z}_{K-2})|^{-1} \cdots \times |J_{T_1}(\boldsymbol{u})|^{-1}.$$

The number of transformations K, is often called the flow depth. Increasing flow depth can help us model progressively more complex densities at the expense of increased computational cost due to the calculation of the determinant for Jacobian matrices $J_{T_k}(.)$.

We can think of discrete time flows as modelling the evolution of a probability density at K-many time points. In contrast, continuous-time normalizing flows model this evolution continuously from some time $t = 0$ to T as an ordinary differential equation $\frac{dz_t}{dt} = f(t, \boldsymbol{z}_t)$. A well-known example of a continuous time flow is the Hamiltonian flow, which is used for MCMC sampling [Nea11].

2.1. Normalizing flows for variational inference.

We now expand on how normalizing flows are used to aid VI. As before, let $\boldsymbol{z} = z_{1:d}$ be the latent variables, $\boldsymbol{x} = x_{1:n}$ be the observed data and $\pi(\boldsymbol{z}|\boldsymbol{x})$ be the conditional distribution we wish to sample from:

$$\pi(\boldsymbol{z}|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|\boldsymbol{z})\pi(\boldsymbol{z})}{m(\boldsymbol{x})}.$$

VI approximates the target distribution by choosing a family of distributions $Q = \{q_\phi | \phi \in \Phi\}$ and selecting the optimal distribution in this family q_{ϕ^*} "closest" to the target density in terms of KL-divergence:

$$q_{\phi^*} \in \operatorname{argmin}_{q_\phi \in Q} KL(q_\phi || \pi(.|\boldsymbol{x})). \tag{1}$$

Other metrics such as more generalized α-divergence measures [LT16] can be used in place of KL-divergence. However, KL-divergence is popular due to its versatility and relative ease of implementation. The optimization in (1) is difficult to work with due to the presence of the intractable marginal likelihood $m(\boldsymbol{x})$. In practice, we maximize the **e**vidence **l**ower **bo**und (**ELBO**) with respect to the variational parameters ϕ due to its equivalence to (1). The **ELBO** is the negative KL-divergence between the variational distribution q and the joint distribution $p(\boldsymbol{x}, \boldsymbol{z})$ of latent variables and observed data:

$$\begin{aligned} &\max_{q_\phi \in Q} \mathbf{ELBO}(q_\phi, \pi(.|\boldsymbol{x})) \\ &= \max_{q_\phi \in Q} \Big\{ \mathbb{E}_{q_\phi(\boldsymbol{z})}\big[\ln p(\mathbf{x}, \boldsymbol{z})\big] - \mathbb{E}_{q_\phi(\boldsymbol{z})}\big[\ln q_\phi(\boldsymbol{z})\big] \Big\}. \end{aligned} \tag{2}$$

Using normalizing flows to aid variational inference was first popularized in [RM15]. The idea is to start with some base distribution $q_0(\boldsymbol{z}_0)$ and then apply diffeomorphisms $T_1, T_2 ... T_K$ successively so that $\boldsymbol{z}_K = T_K \circ T_{K-1} ... T_1(\boldsymbol{z}_0)$. The transformations $(T_k)_{k=1}^K$, parameterized by ϕ, induce a flexible variational family $Q = \{q_\phi(\boldsymbol{z}) | \phi \in \Phi\}$. We have the following useful relations:

$$\ln q_\phi(\boldsymbol{z}_K) = \ln q_0(\boldsymbol{z}_0) - \sum_{k=1}^K \ln \left| \frac{\partial T_k}{\partial \boldsymbol{z}_{k-1}} \right| \tag{3}$$

$$\mathbb{E}_{q_\phi(\boldsymbol{z})} h(\boldsymbol{z}) = \mathbb{E}_{q_0(\boldsymbol{z}_0)} h(T_K \circ T_{K-1} ... T_1(\boldsymbol{z}_0)). \tag{4}$$

(3) follows from the change of variable formula and (4) is a well-known property of expectation. We simplify the maximization of the ELBO in (2):

$$\begin{aligned} &\max_{q_\phi \in Q} \mathbb{E}_{q_\phi(\boldsymbol{z})}\big[\ln p(\boldsymbol{x}|\boldsymbol{z})\pi(\boldsymbol{z}) - \ln q_\phi(\boldsymbol{z})\big] \\ &= \max_{q_\phi \in Q} \Big\{ \mathbb{E}_{q_0(\boldsymbol{z}_0)}\big[\ln p(\boldsymbol{x}|\boldsymbol{z}_K)\pi(\boldsymbol{z}_K)\big] \\ &+ \mathbb{E}_{q_0(\boldsymbol{z}_0)}\Big[\sum_{k=1}^K \ln \left| \frac{\partial T_k}{\partial \boldsymbol{z}_{k-1}} \right| \Big] - \mathbb{E}_{q_0(\boldsymbol{z}_0)}\big[\ln q_0(\boldsymbol{z}_0)\big] \Big\} \end{aligned} \tag{5}$$

$$\begin{aligned} &= \max_{q_\phi \in Q} \Big\{ \mathbb{E}_{q_0(\boldsymbol{z}_0)}\big[\ln p(\boldsymbol{x}|\boldsymbol{z}_K)\pi(\boldsymbol{z}_K)\big] \\ &+ \mathbb{E}_{q_0(\boldsymbol{z}_0)}\Big[\sum_{k=1}^K \ln \left| \frac{\partial T_k}{\partial \boldsymbol{z}_{k-1}} \right| \Big] \Big\}. \end{aligned} \tag{6}$$

Equations (3) and (4) jointly imply (5). We are essentially re-parametrizing the expectation in terms of the base distribution q_0. In (6), we are able to drop $\mathbb{E}_{q_0(\boldsymbol{z}_0)}\big[\ln q_0(\boldsymbol{z}_0)\big]$ because it is free of the parameter ϕ. In practice, optimizing over $q_\phi \in Q$ effectively becomes optimizing over the parameters ϕ of transformations $(T_k)_{k=1}^K$. ϕ are referred

to as *flow parameters*. In general, for d-dimensional latent variables $\boldsymbol{z}$, calculating the determinant of Jacobian $J_{T_k}(\boldsymbol{z}_{k-1})$ takes $O(d^3)$ time [PNR+21]. Therefore, in addition to $T_1, T_2, \ldots, T_K$ being diffeomorphisms, they are often selected such that computational complexity of calculating $J_{T_k}(\boldsymbol{z}_{k-1})$ is $O(d)$.

There are a myriad of ways in which we can choose the normalizing flow transformations. Intuitively, if we choose T_k to be deep neural networks we should be able to approximate almost any well-behaved function. But how do we ensure computational feasibility? Neural autoregressive flows (NAF) [HLCK18] manage to achieve this balance. NAF satisfy the "Universal approximation property." This means that they can approximate ***any*** probability distribution within an arbitrarily small error margin provided the flow depth K is large enough. Further, the autoregressive structure of these flows ensures the Jacobian determinants can be computed in $O(d)$ time. Note that this is just one among many families of normalizing flows. Given these nice properties we choose to use NAF for our examples in Section 3.

2.1.1. *Neural autoregressive flows.* Autoregressive flows are among the most popular normalizing flows discussed in the literature. We discuss some of the principals behind autoregressive normalizing flows. We concentrate on describing NAF since we use these for the examples in which we contrast normalizing flows, MCMC and MF-VI.

Continuing with the same notation, we denote the input from the base distribution by $\boldsymbol{u} = u_{1:d}$ and transformed latent variable by $\boldsymbol{z} = z_{1:d}$. Autoregressive flows are constructed such that each transformed variable $z_i, 1 \leq i \leq d$ is dependent only on the first i inputs $u_{1:i}$. More specifically, the transformer $T = (\tau_1, \tau_2, \ldots \tau_d)$ is made up of d many diffeomorphisms such that:

$$z_i = \tau_i(u_i, c_i(u_{1:i-1})) \quad 2 \leq i \leq d.$$

τ_i is parameterized by the vector $c_i(u_{1:i-1})$. The function $c_i : \mathbb{R}^{i-1} \to \mathbb{R}^m$ is referred to as conditioner and it enforces the autoregressive property for the normalizing flow (see Figure 1). As the name suggests, NAF uses a neural network for τ_i. The 2 types of transformations used are:

1. **Deep Sigmoidal Flow (DSF)** - This neural network uses a single hidden layer.
2. **Dense Deep Sigmoidal Flow (DDSF)** - This uses a deep neural network.

For readers new to the topic, think of a neural network as a somewhat complex function that takes some inputs and applies a series of operations and transformations to them. They generally involve multiplication of inputs with weight matrices, translation, and the application of certain "activation" functions. The DSF network is formally

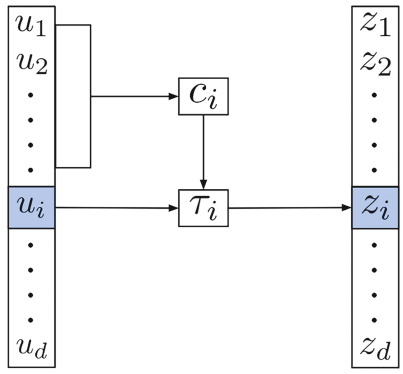


Figure 1. Autoregressive flows.

defined as:

$$z_i = \sigma^{-1}(\boldsymbol{w}_i^\top \sigma(\boldsymbol{a}_i.u_i + \boldsymbol{b}_i)) \quad \boldsymbol{a}_i, \boldsymbol{w}_i, \boldsymbol{b}_i \in \mathbb{R}^k \quad 1 \leq i \leq d.$$

Here k is the number of nodes in the hidden layer and $\sigma(x) = 1/(1 + e^{-x})$ is an activation function. $\boldsymbol{a}_i$ and $\boldsymbol{w}_i$ are constrained as $a_{i,j} > 0 \ \forall i, j$, $0 < w_{i,j} < 1, \sum_j w_{i,j} = 1$. This ensures invertibility of τ_i [HLCK18]. The DDSF transformation has the capacity to be more expressive than DSF due to the universal approximation properties of deep neural networks, albeit at an increased computational cost.

Until now we have discussed the choice of the transformers τ_i for NAF. To construct the conditioner there are no constraints such as invertibility on the functions c_i, $1 \leq i \leq d$. A natural choice is to use a neural network for the conditioner as well. However, using a distinct neural network for each c_i is computationally infeasible as d increases. This is because we have to store and optimize over d networks each with different parameters. [PNR+21] discusses a range of conditioners that leverage parameter sharing across c_i. Following [HLCK18], we adopt the popular masked conditioner approach. Masked conditioners take $u_{1:d}$ as inputs to a neural network and calculate all the parameters $c_1, \left(c_i(u_{1:i-1})\right)_{i=2}^d$ for the transformers in a single forward pass. For a network with a single hidden layer, the autoregressive dependency structure is enforced by multiplying the weight matrices $\mathcal{W}_1$ & $\mathcal{W}_2$ by masking matrices $\mathcal{M}_1, \mathcal{M}_2$ of the same dimension. $\mathcal{M}_1, \mathcal{M}_2$ consist of binary $1 - 0$ entries such that a 0 entry in $\mathcal{M}_i$ implies the corresponding weighted connection is dropped from the network. Therefore entries in $\mathcal{M}_1, \mathcal{M}_2$ are chosen such that there is no connection between the i^{th} input u_i and $1, 2 \ldots, (mi)$ outputs of the network. Here m is a multiplier which tells us how many parameters are required for each τ_i. The weight matrices $\mathcal{W}_1 \in \mathbb{R}^{d \times k}$ and $\mathcal{W}_2 \in \mathbb{R}^{k \times md}$ correspond to the hidden and output layer respectively for the conditioner network. See Figure 1 in [GGML15].

2.1.2. *Implementation.* Recall that for normalizing flows aided variational inference (FAVI) we maximize the ELBO:

$$\mathcal{L}(q_\phi) = \mathbb{E}_{q_0(z_0)}[\ln p(\boldsymbol{x}, \boldsymbol{z}_K)] + \mathbb{E}_{q_0(z_0)}[\sum_{k=1}^{K} \ln |\frac{\partial T_k}{\partial \boldsymbol{z}_{k-1}}|].$$

$(T_k)_{k=1}^K$ and $\boldsymbol{z}_K$ depend on ϕ in the equation above. In general, $\mathcal{L}(q_\phi)$ will not have a closed form expression. Additionally, standard coordinate-wise gradient ascent algorithms are computationally inefficient for large datasets. As a result, the Stochastic Gradient Ascent (SGA) algorithm is often used for optimizing the ELBO.

SGA is an iterative method that uses the following update for the flow parameters ϕ at step t:

$$\phi_{t+1} = \phi_t + \alpha_t l(\phi_t).$$

Here $l(\phi)$ is a realization for an unbiased estimator of $\nabla_\phi \mathcal{L}(q_\phi)$, the gradient for the ELBO. We can calculate it by sampling $\boldsymbol{z}_0^{(1)}, \boldsymbol{z}_0^{(2)}, \ldots \boldsymbol{z}_0^{(S)}$ from $q_0(.)$ and passing them through the transformations $T_1, T_2, \ldots, T_K$ to get $\boldsymbol{z}_K^{(1)}, \boldsymbol{z}_K^{(2)}, \ldots, \boldsymbol{z}_K^{(S)}$:

$$l(\phi) = \frac{1}{S} \sum_{s=1}^{S} \left[\nabla_\phi \ln p(\boldsymbol{x}, \boldsymbol{z}_K^{(s)}) + \sum_{k=1}^{K} \nabla_\phi \ln |\frac{\partial T_k}{\partial \boldsymbol{z}_{k-1}^{(s)}}| \right].$$

SGA almost surely converges to a local minimum for non-convex functions and global minimum for pseudo-convex functions when learning rates satisfy $\sum_{t=1}^{\infty} \alpha_t = \infty$ & $\sum_{t=1}^{\infty} \alpha_t^2 < \infty$. ([RM51], [Bot98]).

In practice, choosing the learning rate α_t is nontrivial. When α_t is too large we may overshoot the maxima and when α_t is too small then SGA will learn too slowly. For our experiments, we use the Adam algorithm [KB14] which uses an adaptive learning rate that incorporates information about the scale of different components in the parameter vector ϕ. We use a standard normal distribution for $q_0(z_0)$.

Note that, the outputs of the normalizing flow transformations $\boldsymbol{z}_k$ are unconstrained, i.e., they belong to $\mathbb{R}^d$, since they are outputs of a neural network. Sometimes the latent variable space is restricted, for example, our model may have a variance parameter $\sigma^2 > 0$. More formally, when $z_i \in S \subset \mathbb{R}$ we apply a final transformation T_{K+1} to constrain $z_{K,i}$. For instance, if $z_i > 0$ then we set $\tilde{z}_{K,i} = \ln(1 + e^{z_{K,i}})$.

3. Illustrative Examples

Here, we implement the FAVI algorithm on some examples. We also provide comparisons to MCMC and MF-VI where applicable. Note that MCMC comprises a wide class of algorithms ranging from the more basic Random Walk Metropolis Hastings (RW-MH) and Gibbs sampling methods to approaches that make use of gradient information such as the Hamiltonian Monte-Carlo (HMC) [Nea11].

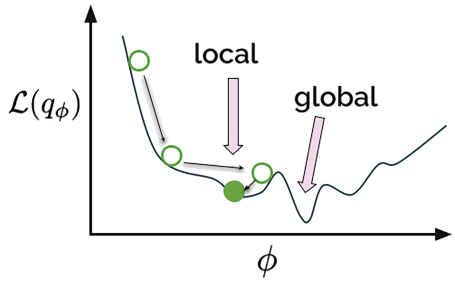
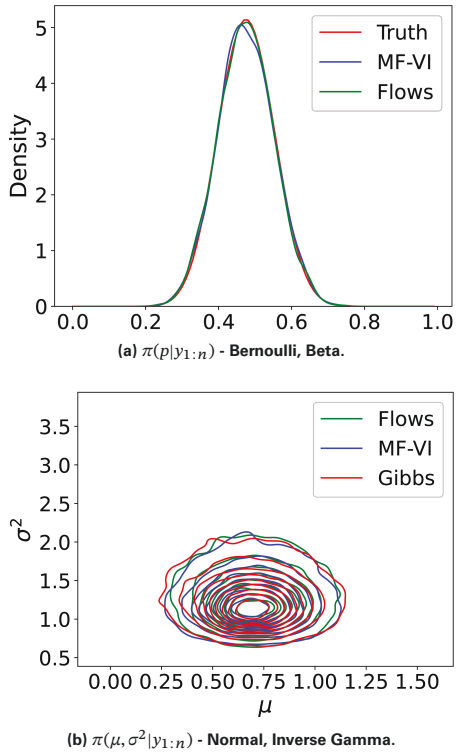


Figure 2. Stochastic gradient ascent for $\phi \in \mathbb{R}$.



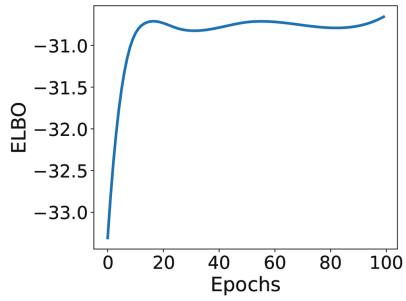
(a) $\pi(p|y_{1:n})$ **- Bernoulli, Beta.**

(b) $\pi(\mu, \sigma^2|y_{1:n})$ **- Normal, Inverse Gamma.**

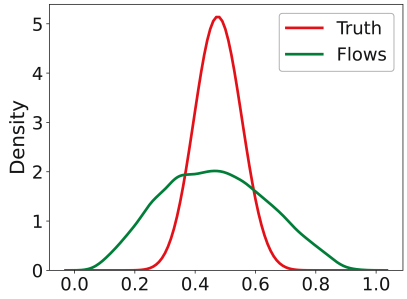
Figure 3. Exponential family density plots.

We use either the RW-MH or Gibbs sampling methods as a baseline since these are widely used in classical applications of Bayesian inference. See Section 4 for a detailed discussion of contemporary MCMC literature. Section 3.1 discusses FAVI for exponential family models, followed by 3.2 in which we sample from un-normalized energy density functions. We then move onto Bayesian linear and logistic regression in 3.3 and 3.4. Through these examples we hope to elucidate how FAVI works in different contexts.

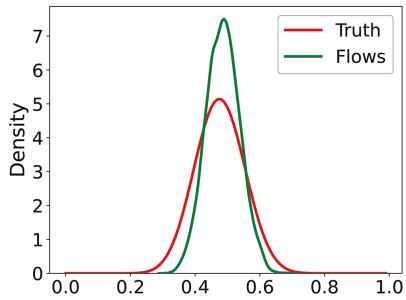
3.1. **The exponential family.** In many applications of Bayesian inference the complete conditionals $p(z_i|z_{-i}, \boldsymbol{x})$ $1 \le i \le d$ of latent variables belong to the exponential family $\mathcal{P} = \{\frac{h(z_i)}{A(\eta)} \exp \eta^t t(z_i)\}$. This class of models is known as conditionally conjugate exponential family models and its broad applicability makes it of interest to statistical



(a) ELBO vs. Training Epochs.



(b) $\pi(p|y_{1:n})$: epochs = 2.



(c) $\pi(p|y_{1:n})$: epochs = 20.

Figure 4. Convergence of FAVI - Bernoulli, Beta.

practitioners. [BKM17] discusses the derivation for coordinate ascent variational inference (CAVI), an MF-VI method, for this class of models. It is natural to extend this discussion to the FAVI algorithm for exponential family models.

FAVI performing well on low-dimensional examples is a necessary but not sufficient condition for them to be reliable for high-dimensional VI problems. This motivates our choice of examples:

1. $(y_i)_{i=1}^n | p \overset{\text{i.i.d}}{\sim} \text{Bernoulli}(p) \quad \pi(p) : \; p \sim \text{Beta}(a, b)$
2. $(y_i)_{i=1}^n | \mu, \sigma^2 \overset{\text{i.i.d}}{\sim} N(\mu, \sigma^2),\ \pi(\mu, \sigma^2) : \; \mu \sim N(0, \tau^2) \perp\!\!\!\perp \sigma^2 \sim \text{Inv-Gamma}(v_1, v_2)$

In the first case the posterior has a closed form with which we can compare the density obtained by flows.

$$p|y_{1:n} \sim \text{Beta}(a + n\bar{y}, b + (n - n\bar{y}))$$

For the second example, we compare with results obtained by Gibbs sampling. We also include results from MF-VI.

We see from Figure 3 that FAVI, MF-VI, and Gibbs produce similar results. Figure 4 is a demonstration of how the density obtained from FAVI converges to the true distribution over the training epochs. From the plot of ELBO against epochs we see that at around the 20^{th} epoch there is a plateau. This indicates the density from FAVI has changed shape and is approaching the true distribution.

3.2. Sampling from multimodal densities.

The primary advantage of normalizing flows is their ability to recover highly multimodal target distributions with complex dependencies. In [HLCK18], authors use NAF to sample from multimodal energy density functions, for which the normalizing constant is unknown. They do not however, provide a comparison to MCMC methods. Given that MCMC methods are theoretically guaranteed to converge to the target distribution of interest, we believe it would be useful to include this comparison. We contrast both accuracy and computational time for NAF and the RW-MH Algorithm, for sampling from the energy density functions $U1 - U9$ (see Figure 5).

We compare the two methods based on run-time and kernel density estimates (k.d.e). Run-time is measured from the first iteration for the algorithm till convergence. For comparing the densities generated by both methods we calculate kernel density estimates (k.d.e) from the samples. We use a Gaussian kernel, on a grid of size 200×200. We then calculate square root of sum of squares error ($\sqrt{\textbf{SSE}}$) for k.d.e over the grid as $\sqrt{\sum_{i=1}^{N=40{,}000} (\hat{f}(\boldsymbol{x}_i) - f_{\text{True}}(\boldsymbol{x}_i))^2}$. Here, $\hat{f}(.)$ is the kernel density estimator obtained from either the FAVI/RW-MH algorithm and $f_{\text{True}}(.)$ is the true density.[2] This is equivalent to the Frobenius norm of errors between true and estimated density on our grid.

Determining convergence. For assessing convergence of the random-walk Metropolis Hastings we visually inspect the autocorrelation and trace plots. The plots for many energy functions display non-negligible autocorrelation upto lag 40, therefore we thin the samples by 40 and run the chain for 400,000 samples. We choose this run-time in order to obtain a sufficient sample size of 10k to get richer kernel density plots. We run FAVI for 15k epochs based on stabilization of the loss function and also generate 10k samples after training. For both, the RW-MH and FAVI algorithms there is a degree of subjectivity to determining convergence since we use visual inspection. Empirical convergence criteria such as trace plots, $\hat{R}$ [GR92] and zero autocorrelation in the samples does not guarantee convergence of the Markov chain. Although satisfying these criteria is not sufficient for convergence, it is necessary for us to gain confidence that the Markov chain is approaching the stationary distribution.

[2] *Since we do not have the closed form for the true density we normalize the energy functions using numerical integration from SciPy's integrate module.*

Results. Table 1 reports the average $\sqrt{\text{SSE}}$ of k.d.e for both FAVI and RW-MH algorithms across the best three of five trials (based on loss) ± standard deviation. We chose the best three because the loss does not converge in all cases for the FAVI algorithm, due to sensitivity to choice of initialization and the presence of local minima.[3] We observe that the RW-MH algorithm outperforms FAVI in terms of k.d.e metrics by a small–medium margin with one exception, $U6$. We also see from the standard deviations that the RW-MH algorithm is more stable than FAVI, which is highly sensitive to the initialization of flow parameters ϕ. For instance, $U5$ has a standard deviation of $\sqrt{\text{SSE}}$ 0.95 for FAVI and only 0.01 for RW-MH. In terms of computational time, approximately half of the energy functions have run-time of a similar order for the FAVI and RW-MH algorithms. For the remaining functions we observe:

1. For $U3$ and $U5$ the RW-MH algorithm takes approximately 60% of the run-time that FAVI does.
2. For $U4$ and $U9$ the trend is reversed and FAVI takes only 30% of the RW-MH algorithm run-time.

Upon closer examination of the function forms we see that $U3$ and $U5$ are relatively simple functions to evaluate over a particular sample whereas $U4$ and $U9$ are complex functions, being a mixture of multiple densities. $U4$ is a mixture Gaussian density with four components and $U9$ is a mixture of $U3$ and part of $U8$. Thus, unlike FAVI, the RW-MH algorithm does not scale as complexity of the target density increases.

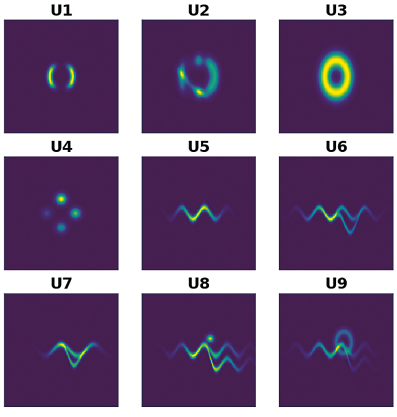
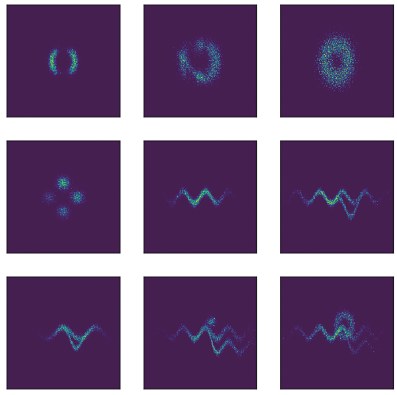
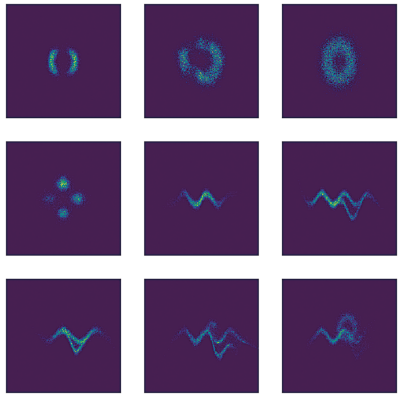


Figure 5. True density.



(a) Histogram - RW-MH.



(b) Histogram - FAVI.

Figure 6. Energy density functions $U1 - U9$.

	Avg. $\sqrt{\text{SSE}}$		Avg. Run-time	
Ef	FAVI	RW-MH	FAVI	RW-MH
$U1$	0.98 ± 0.05	0.94 ± 0.01	114 ± 1	119 ± 4
$U2$	0.46 ± 0.03	0.42 ± 0.01	118 ± 7	173 ± 31
$U3$	0.20 ± 0.01	0.18 ± 0.01	109 ± 3	68 ± 8
$U4$	0.62 ± 0.04	0.56 ± 0.02	**122 ± 1**	**465 ± 84**
$U5$	1.72 ± 0.95	1.17 ± 0.01	111 ± 2	69 ± 4
$U6$	**1.21 ± 0.03**	**1.25 ± 0.00**	145 ± 15	212 ± 40
$U7$	1.07 ± 0.01	1.07 ± 0.02	122 ± 7	166 ± 8
$U8$	1.11 ± 0.04	1.10 ± 0.00	127 ± 12	251 ± 13
$U9$	1.25 ± 0.04	1.15 ± 0.01	**131 ± 2**	**303 ± 45**

Table 1. Avg. $\sqrt{\text{SSE}}$ ± s.d. of k.d.e. for $U1 - U9$ (smaller values are better). | Avg. algorithm run-time in seconds ±s.d. for $U1 - U9$ (smaller values are better).

3.3. **Linear regression.** In this section we implement the FAVI algorithm on a Bayesian linear regression example to sample from the posterior of regression parameters given the data, $\pi(\beta, \sigma^2|D)$. We use the framework below:

$$y_i \sim \boldsymbol{x}_i^\top \beta + \varepsilon_i, \quad \beta \in \mathbb{R}^p, \ \varepsilon_i \overset{\text{i.i.d}}{\sim} N(0, \sigma^2)\ 1 \le i \le n$$
$$\pi(\beta, \sigma^2) : \ \beta \sim N(0, \tau^2 I_p) \perp\!\!\!\perp \sigma^2 \sim \text{Inv-Gamma}(a, b).$$

[3] *Although not standard practice, we report results across different initializations to contrast the stability across FAVI and MCMC. Further, run-time varies across trials and averaging gives us a better idea of the true run-time.*

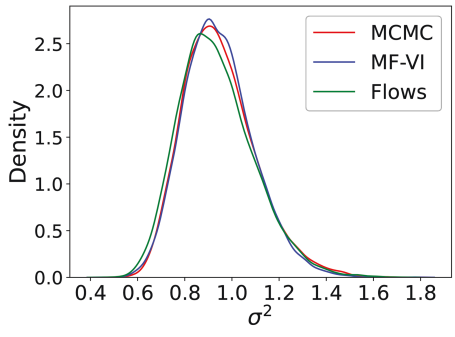
We compare FAVI to both MF-VI and the Gibbs sampling algorithm. We use Gibbs sampling because it relies on the complete conditionals for latent variables $\pi(\beta|y_{1:n}, \sigma^2)$ and $\pi(\sigma^2|y_{1:n}, \beta)$ which are easily available in this case. Through this example, we can gain some insight on the scalability and accuracy contrast between the three methods in a classical statistical setup. To assess effect of both sample size and dimensionality on the performance of these methods we use a grid of (n, p) combinations. We allow n (sample size) to take values 50, 100, and 200, while p (β dimension) takes values 2, 20, 50, and 100.

For our experiments, we simulate the true data generating β_0 from the $Uniform(\frac{1}{2}, 2)$ distribution. We assume $\sigma_0 = \tau = 1$ where σ_0 is the true value of model parameter σ. The p-dimensional predictor variables $\boldsymbol{x}_1, \boldsymbol{x}_2 \ldots \boldsymbol{x}_n \in \mathbb{R}^p$ are simulated from a multivariate normal distribution $N(0, I_p)$. For the cases where $p \geq 20$, we set only 20% of the variables to be non-zero in order to ensure the latent variable space is sparse, that is, $K << p$ where K is the number of non-zero components in β_0.

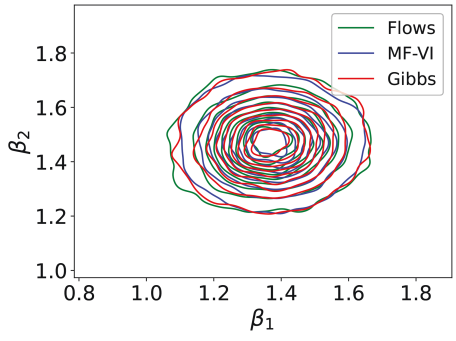
Similar to Section 3.2, convergence of the Gibbs sampling algorithm is determined by a combination of trace and autocorrelation plots. We thin the samples by a factor of 10 to ensure 0 autocorrelation. We initialize β with its O.L.S estimate for faster convergence. Convergence of FAVI and MF-VI is ascertained via stabilization of the loss function.

Results. In order to visualize the difference between densities approximated by the three approaches (FAVI, MF-VI and Gibbs) we use kernel density plots. For the case where $p = 2$, we can easily visualize the posterior distributions of β and σ^2. For higher-dimensional examples we use the kernel density plots for SSE of β; $g(\beta) = \|\beta - \beta_0\|_2^2$ where β is sampled from the posterior $\pi(\beta|y_{1:n})$. We present density plots for $n = 100$ and varying p in Figure 8. We report the model predictive root mean squared error ($\sqrt{\textbf{MSE}}$) on test data $\sqrt{\sum_{i=1}^{n_{\text{test}}}(y_i - \hat{y}_i)^2/n_{\text{test}}}$. Here $\hat{y}_i = \boldsymbol{x}_i^\top \hat{\beta}$ is the predicted value for the i^{th} sample based on mean of the posterior samples $\hat{\beta} = 1/N \sum_{n=1}^{N} \beta_n$. Here N is the number of β samples generated and is set to be 10k. To get a sense of variance of the posterior distribution for β we also report $\overline{s_\beta}$. This is obtained by first computing sample standard deviation from posterior samples for each $\pi(\beta_i|y_{1:n})$ as $s_{\beta_i} = (1/(N-1)) \sum_{n=1}^{N} (\beta_i^n - \overline{\beta_i})^2$ for $1 \leq i \leq p$. We then aggregate these by averaging across dimensions as $\overline{s_\beta} = (1/p) \sum_{i=1}^{p} s_{\beta_i}$. By reporting $\sqrt{\text{MSE}}$ and $\overline{s_\beta}$ we are able to capture model predictive performance and uncertainty quantification for the three methods.

We observe from density plots that for cases where $p = 2$ the difference across 3 algorithms is insubstantial (Figure 7). As dimension p increases we see that the FAVI results



(a) $\pi(\sigma^2|y_{1:n})$



(b) $\pi(\beta_1, \beta_2|y_{1:n})$.

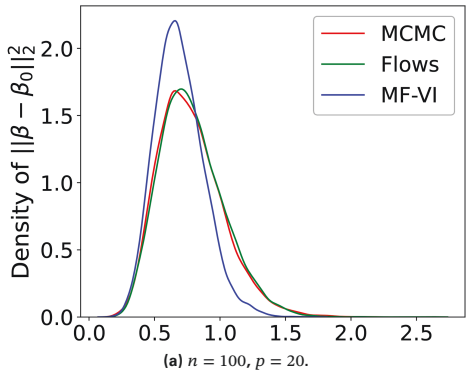
Figure 7. Linear regression: $n = 100$, $p = 2$.

are reasonably close to Gibbs sampling (the gold standard) but MF-VI produces a peaked distribution, indicating it under estimates uncertainty in the samples (Figure 8). Performance metrics in Table 2 show that all 3 methods have similar predictive performance. For uncertainty quantification, $\overline{s_\beta}$ shows that MF-VI has lower posterior variance in general than the other two methods, while FAVI is comparable to Gibbs. There are 2 notable exceptions when $n = 100, p = 100$ and $n = 50, p = 100$. In these cases MF-VI has larger $\overline{s_\beta}$ by 0.01 points than FAVI but this difference is insubstantial.

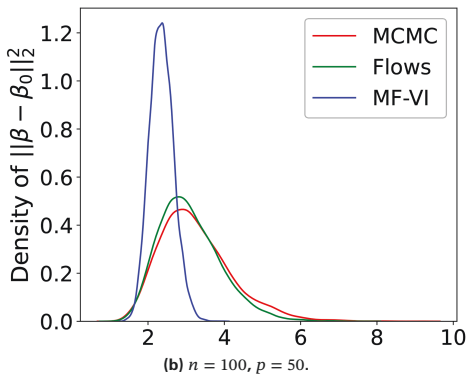
For the 3 cases where $p \geq n$, the Gibbs sampling algorithm breaks down due to the instability of the following matrix inversion: $(I_p/\tau^2 + X^\top X/\sigma^2)^{-1}$. Since FAVI, like MF-VI, does not require the matrix inversion step and can specify an arbitrarily flexible family of densities, it could be a promising alternative to Gibbs sampling in such a set-up.[4]

Table 3 reports average algorithm run-times ± s.d. across 5 trials. In general, MF-VI runs the fastest, followed by Gibbs sampling and then FAVI. However, for the highlighted cases of $p = 50$ and $n \geq 100$ this trend switches and Gibbs sampling becomes slower than FAVI. Given that Gibbs sampling breaks down in high dimension, it is difficult to discern a pattern and contrast scalability of the two methods.

[4] *There are modifications of Gibbs sampling that can circumvent this and a more thorough exploration is required.*



(a) $n = 100$, $p = 20$.



(b) $n = 100$, $p = 50$.

Figure 8. Linear regression: Density plots of $||\beta - \beta_0||_2^2$ where β is sampled from $\pi(\beta|y_{1:n})$.

	Pred $\sqrt{\text{MSE}}$			Avg. β s.d. $\overline{s_\beta}$		
(n, p)	**Gibbs**	**FAVI**	**MF-VI**	**Gibbs**	**FAVI**	**MF-VI**
(50, 2)	1.06	1.06	1.06	0.17	0.17	0.16
(50, 20)	0.88	0.92	0.88	0.24	0.23	0.17
(50, 50)	*	3.06	3.41	*	0.47	0.46
(50, 100)	*	2.93	2.64	*	0.77	0.78
(100, 2)	1.04	1.04	1.04	0.11	0.11	0.11
(100, 20)	1.05	1.05	1.05	0.14	0.14	0.12
(100, 50)	1.93	1.90	1.93	0.17	0.16	0.11
(100, 100)	*	2.64	2.92	*	0.46	0.47
(200, 2)	1.07	1.07	1.07	0.08	0.09	0.08
(200, 20)	1.10	1.09	1.10	0.09	0.09	0.08
(200, 50)	1.26	1.26	1.26	0.10	0.10	0.08
(200, 100)	1.86	1.87	1.86	0.14	0.13	0.09

Table 2. Linear regression: Model predicted $\sqrt{\text{MSE}}$ (smaller values are better). | Avg. s.d. of β samples. We use * when the result can't be computed.

Avg. Run-time (in seconds)			
(n, p)	**Gibbs**	**FAVI**	**MF-VI**
(50, 2)	55 ± 0	252 ± 8	22 ± 1
(50, 20)	79 ± 5	374 ± 21	23 ± 1
(50, 50)	*	606 ± 41	122 ± 10
(50, 100)	*	1055 ± 84	922 ± 69
(100, 2)	59 ± 4	267 ± 10	24 ± 2
(100, 20)	85 ± 5	383 ± 24	24 ± 1
(100, 50)	**953 ± 54**	**549 ± 23**	36 ± 2
(100, 100)	*	1087 ± 56	778 ± 80
(200, 2)	57 ± 3	254 ± 13	23 ± 1
(200, 20)	80 ± 0	568 ± 10	23 ± 0
(200, 50)	**997 ± 110**	**567 ± 31**	24 ± 2
(200, 100)	833 ± 37	1011 ± 28	35 ± 1

Table 3. Linear regression: Avg. algorithm run-time ± s.d. over five trials (smaller values are better). We use * when the result can't be computed.

3.4. **Logistic regression.** We consider the following model:

$$y_i \sim \text{Bernoulli}(p_i), \quad p_i = \frac{e^{x_i^\top \beta}}{1 + e^{x_i^\top \beta}}$$
$$\pi(\beta): \quad \beta \sim N(0, \tau^2 I_p).$$

We use the same simulation set-up for β and $\boldsymbol{x}$ as of Section 3.3 on linear regression. Here we use RW-MH instead of Gibbs sampling because we no longer have closed form complete conditionals. Maintaining consistency with previous experiments, we use trace and autocorrelation plots to decide on convergence.[5] We initialize the RW-MH algorithm with the maximum likelihood estimates for β. We use plots and metrics as in 3.3, replacing $\sqrt{\text{MSE}}$ by Accuracy.

Results. Density plots for β when $n = 100, p = 2$ are presented in Figure 9. From the contour plot we see that MF-VI does not capture the elliptical structure of the joint distribution of β which the other two methods display. For higher dimensions, kernel density plots of β SSE, $||\beta - \beta_0||_2^2$ when $\beta \sim \pi(\beta|y_{1:n})$ are presented in Figure 10. Similar to the trend displayed by Gaussian linear regression, we see that kernel density plots for MF-VI seem to center on a lower SSE. The FAVI and RW-MH algorithms perform similarly with respect to uncertainty quantification as dimension p increases but MF-VI has lower aggregate posterior variance for β (See Table 4). All three methods display identical model predictive Accuracy given by

[5] *For most cases the autocorrelation after thinning is between* 0.0 − 0.2, *however when* $p = 100$ *we allow autocorrelation of 0.4 given computational considerations.*

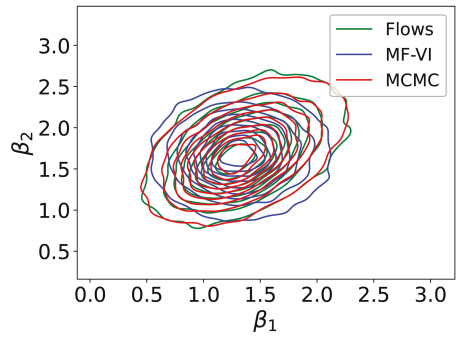
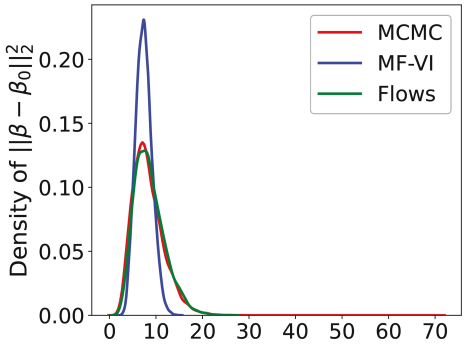
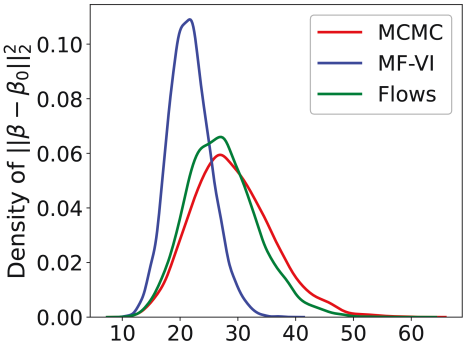


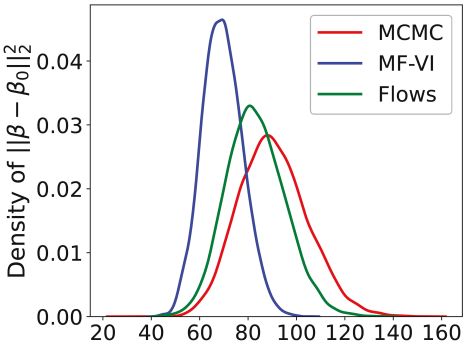
Figure 9. Logistic regression: $\pi(\beta_1, \beta_2 | y_{1:n})$ when $n = 100$, $p = 2$.



(a) $n = 100$, $p = 20$.



(b) $n = 100$, $p = 50$.



(c) $n = 100$, $p = 100$.

Figure 10. Logistic regression: Density plots of $||\beta - \beta_0||_2^2$ where β is sampled from $\sim \pi(\beta | y_{1:n})$.

	Accuracy			Avg. β sd. $\overline{s_\beta}$		
(n, p)	MH	FAVI	MF-VI	MH	FAVI	MF-VI
(50, 2)	0.80	0.80	0.80	0.41	0.40	0.39
(50, 20)	0.90	0.90	0.90	0.56	0.55	0.47
(50, 50)	0.60	0.60	0.60	0.77	0.73	0.61
(50, 100)	0.50	0.50	0.50	0.88	0.85	0.74
(100, 2)	0.60	0.60	0.60	0.36	0.36	0.32
(100, 20)	0.65	0.65	0.65	0.40	0.39	0.34
(100, 50)	0.95	0.95	0.95	0.59	0.55	0.44
(100, 100)	0.70	0.70	0.70	0.77	0.71	0.56
(200, 2)	0.85	0.85	0.85	0.25	0.25	0.25
(200, 20)	0.85	0.85	0.85	0.25	0.25	0.25
(200, 50)	0.55	0.55	0.55	0.40	0.38	0.31
(200, 100)	0.78	0.78	0.78	0.61	0.54	0.42

Table 4. Logistic regression: Model accuracy (larger values are better). | Avg. s.d. of β samples. Here RW-MH is abbreviated as MH.

$\sum_{i=1}^{n_{\text{test}}} \mathbb{I}\{\hat{y}_i = y_i\}/n_{\text{test}}$. The plot of average run-time across 5 trials against dimension p presents an interesting contrast (Figure 11). As expected, MF-VI scales the best, since it has run-time of approximately the same order regardless of dimension. RW-MH algorithm scales poorly ranging from 40s for $p = 2$ to approximately 1000s when $p = 100$. FAVI on the other hand, only has a run-time in the 400s when $p = 100$, less than half the time of the RW-MH algorithm. Thus, FAVI is scalable relative to the RW-MH algorithm and performs much better than MF-VI at approximating densities in their entirety, beyond just measures of central tendency.

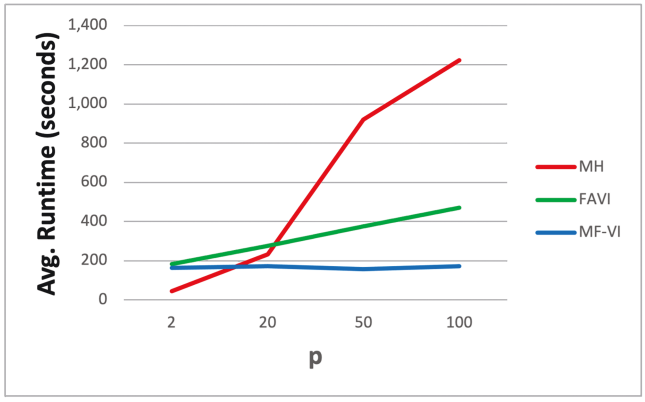


Figure 11. Logistic regression: Average run-time vs. β dimension when $n = 200$.

4. Looking Ahead

This article discusses how normalizing flows are a useful tool in probabilistic modeling. The examples we covered

in Section 3 confirm that FAVI lies somewhere between basic MCMC methods (RW-MH, Gibbs sampling) and MF-VI with respect to accuracy and scalability. They can approximate multimodal densities and come reasonably close to MCMC for uncertainty quantification while retaining some scalability. An exciting feature of FAVI is a high degree of flexibility over the desired levels of expressivity and scalability for the probabilistic model. This is due to our ability to select the flow depth, type of transformation, and the number of flow parameters ϕ.

There are still many challenges remaining in this area. There is no rigorous study on the scalability vs. expressivity trade-off in the literature. Many normalizing flow families such as NAF have universal approximation properties which allow them to approximate any distribution arbitrarily well, given enough flow depth. This does not, however, provide answers on the flow depth required to achieve desired levels of accuracy. In contrast, MCMC methods have effective sample size measures, which indicate the minimum amount of samples needed to obtain the required quality in posterior samples.

Another research direction would be improving FAVI's scalability without a significant accuracy loss. We observe that FAVI can be computationally expensive in Bayesian inference if the likelihood $p(\boldsymbol{x}|\boldsymbol{z})$ is difficult to evaluate. There are already efforts in this direction. [WLS22] replaces the likelihood with a surrogate likelihood that we can learn while training the flow transformations. In addition, we see in Section 3.2, that FAVI can be highly sensitive to the initialization of flow parameters ϕ. Thus FAVI can take longer to converge to the minima with bad initializations. It follows that a beneficial avenue of research would be going beyond naive initializations for FAVI.

As discussed in Section 3, we have used the RW-MH and Gibbs sampling algorithms for our experiments. However, there exists a range of other MCMC algorithms in the literature. Among the most popular is the HMC algorithm [Nea11], and its self tuning variant, the No-U-Turn sampler (NUTS) [HG14]. These methods have achieved considerable success by leveraging gradient information to make jumps through the state space for the target distributions. In fact, HMC scales at $O(d^{1/4})$ iterations to achieve 2 nearly independent samples in comparison to $O(d)$ time for RW-MH. Thus, HMC is considered to be the gold standard for unimodal high-dimensional regimes, if computationally feasible. [WJGC19] contains a comparison of FAVI and HMC on 13 Bayesian linear regression models. Their results indicate that FAVI is competitive with HMC, having a lower MSE in 5 of 13 models. [MPS18] show that for highly multimodal distributions the above scaling regime need not hold. Specifically, HMC and the RW-MH algorithm behave the same way, with their spectral gaps decaying at the same rate. Thus, FAVI has the potential to compete with HMC for multimodal densities. A more rigorous, wide scale exploration of how FAVI compares to gradient based MCMC methods is essential.

Normalizing flows can also be used to aid MCMC sampling. We expand on some existing ideas to do this. For multimodal target distributions, the MCMC chains may converge slowly, and samples are highly correlated.[6] The MH algorithm, uses a "proposal density" $p(\boldsymbol{z})$ from which we generate candidates for posterior samples. The proposal density is often selected to be easy to sample from, for example, the Gaussian distribution. Unfortunately, when the target density has a complicated geometry, this results in slow exploration of the sample space. To address this, we can use normalizing flows to shift the proposal density space to a "distorted" space. This is possible because normalizing flows are nothing but a reshaping of one density into another. The MH algorithm is then able to move faster through this "distorted" space. Recently, inverse autoregressive flows have been used to aid HMC sampling [HSD+19].

Until now, we have discussed how normalizing flows can be used for learning densities on continuous support. What happens for discrete probability distributions? There is an equivalent change of variable formula for flows on discrete distributions:

$$p_Z(\boldsymbol{z}) = p_U(T^{-1}(\boldsymbol{z})).$$

$T : \mathcal{U} \to \mathcal{Z}$ is a bijection between two discrete spaces $\mathcal{U}, \mathcal{Z}$. However, some issues exist with using the above for learning discrete distributions. For one, we rely heavily on the base distribution for expressivity in discrete flow models. We need to incorporate dependencies across variables into the base distribution itself. Further, there is no research on modeling joint discrete-continuous distributions. We expect this to be a popular avenue of research in the near future.

Normalizing flows are a significant advancement for probabilistic modeling, particularly for VI. This area is in the nascent stages, and many issues need to be tackled. We hope to see future collaborations between computer scientists and statisticians to address some of these issues, thus enabling wider adoption of normalizing flows, especially for Bayesian inference.

ACKNOWLEDGMENTS. The authors thank the editors and anonymous referees for their insightful comments and suggestions that improved the presentation of the work. This work is partially supported by the grants NSF-1924724, NSF-1952856, and NSF-2124605.

[6] *We see this occur for the mixture Gaussian density U4 in section 3.2*

References

[BKM17] David M. Blei, Alp Kucukelbir, and Jon D. McAuliffe, *Variational inference: a review for statisticians*, J. Amer. Statist. Assoc. **112** (2017), no. 518, 859–877, DOI 10.1080/01621459.2017.1285773. MR3671776

[Bot98] Léon Bottou, *Online algorithms and stochastic approximations*, Online Learning and Neural Networks, 1998. Revised, 2012.

[CG95] Siddhartha Chib and Edward Greenberg, *Understanding the Metropolis-Hastings algorithm*, The American Statistician **49** (1995), no. 4, 327–335.

[HLCK18] Chin-Wei Huang, Alexandre Lacoste, Aaron Courville, and David Krueger, *Neural autoregressive flows*, Proceedings of the 35th International Conference on Machine Learning (ICML-2018), 2018.

[GR92] Andrew Gelman and Donald B. Rubin, *Inference from iterative simulation using multiple sequences*, Statistical Science **7** (1992), no. 4, 457–472.

[CG92] George Casella and Edward I. George, *Explaining the Gibbs sampler*, Amer. Statist. **46** (1992), no. 3, 167–174, DOI 10.2307/2685208. MR1183069

[GGML15] Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle, *Made: Masked autoencoder for distribution estimation*, Proceedings of the 32nd International Conference on Machine Learning, 2015, pp. 881–889.

[HG14] Matthew D. Hoffman and Andrew Gelman, *The no-U-turn sampler: adaptively setting path lengths in Hamiltonian Monte Carlo*, J. Mach. Learn. Res. **15** (2014), 1593–1623. MR3214779

[HSD+19] Matthew D. Hoffman, Pavel Sountsov, Joshua V. Dillon, Ian Langmore, Dustin Tran, and Srinivas Vasudevan, *Neutra-lizing bad geometry in Hamiltonian Monte Carlo using neural transport*, arXiv: Computation **abs/1903.03704** (2019).

[KB14] Diederik P. Kingma and Jimmy Ba, *Adam: A method for stochastic optimization*, CoRR **abs/1412.6980** (2014).

[KPB2005] Ivan Kobyzev, Simon Prince, and Marcus Brubaker, *Normalizing flows: An introduction and review of current methods*, IEEE Transactions on Pattern Analysis and Machine Intelligence **PP** (202005), 1–1.

[LT16] Yingzhen Li and Richard E. Turner, *Rényi divergence variational inference*, Nips, 2016.

[MPS18] Oren Mangoubi, Natesh S. Pillai, and Aaron Smith, *Does Hamiltonian Monte Carlo mix faster than a random walk on multimodal densities?*, arXiv (2018).

[Nea11] Radford M. Neal, *MCMC using Hamiltonian dynamics*, Handbook of Markov chain Monte Carlo, Chapman & Hall/CRC Handb. Mod. Stat. Methods, CRC Press, Boca Raton, FL, 2011, pp. 113–162. MR2858447

[PNR+21] George Papamakarios, Eric Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Lakshminarayanan, *Normalizing flows for probabilistic modeling and inference*, J. Mach. Learn. Res. **22** (2021), Paper No. 57, 64. MR4253750

[RM15] Danilo Jimenez Rezende and Shakir Mohamed, *Variational inference with normalizing flows*, International Conference on Machine Learning, 2015.

[RM51] Herbert Robbins and Sutton Monro, *A stochastic approximation method*, Ann. Math. Statistics **22** (1951), 400–407, DOI 10.1214/aoms/1177729586. MR42668

[WJGC19] Stefan Webb, Martin Jankowiak, Noah Goodman, and Jonathan P. Chen, *Improving automated variational inference with normalizing flows*, ICML workshop on automated machine learning, 2019.

[WLS22] Yu Wang, Fang Liu, and Daniele E. Schiavazzi, *Variational inference with NoFAS: normalizing flow with adaptive surrogate for computationally expensive models*, J. Comput. Phys. **467** (2022), Paper No. 111454, 21, DOI 10.1016/j.jcp.2022.111454. MR4454270

Sumegha Premchandar

Bhattacharya Shrijita

Maiti Tapabrata

Credits

Opening image is courtesy of NicoElNino via Getty.

Figures 1–11 are courtesy of the authors.

Photo of Sumegha Premchandar is courtesy of Michigan State University/Harley Seeley.

Photo of Bhattacharya Shrijita is courtesy of Bhattacharya Shrijita.

Photo of Maiti Tapabrata is courtesy of Michigan State University/Harley Seeley.