Automatic Differentiation Variational Inference

Alp Kucukelbir
Data Science Institute, Department of Computer Science
Columbia University

Dustin Tran
Department of Computer Science
Columbia University

Rajesh Ranganath
Department of Computer Science
Princeton University

Andrew Gelman
Data Science Institute, Departments of Political Science and Statistics
Columbia University

David M. Blei
Data Science Institute, Departments of Computer Science and Statistics
Columbia University

March 3, 2016

Abstract

Probabilistic modeling is iterative. A scientist posits a simple model, fits it to her data, refines it according to her analysis, and repeats. However, fitting complex models to large data is a bottleneck in this process. Deriving algorithms for new models can be both mathematically and computationally challenging, which makes it difficult to efficiently cycle through the steps. To this end, we develop automatic differentiation variational inference (ADVI). Using our method, the scientist only provides a probabilistic model and a dataset, nothing else. ADVI automatically derives an efficient variational inference algorithm, freeing the scientist to refine and explore many models. ADVI supports a broad class of models—no conjugacy assumptions are required. We study ADVI across ten different models and apply it to a dataset with millions of observations. ADVI is integrated into Stan, a probabilistic programming system; it is available for immediate use.

1   Introduction

We develop an automatic method that derives variational inference algorithms for complex probabilistic models. We implement our method in Stan, a probabilistic programming system that lets a user specify a model in an intuitive programming language and then compiles that model into an inference executable. Our method enables fast inference with large datasets on an expansive class of probabilistic models.

Figure 14 gives an example. Say we want to analyze how people navigate a city by car. We have a dataset

of all the taxi rides taken over the course of a year: 1.7 million trajectories. To explore patterns in this data, we propose a mixture model with an unknown number of components. This is a non-conjugate model that we seek to fit to a large dataset. Previously, we would have to manually derive an inference algorithm that scales to large data. In our method, we write a Stan program and compile it; we can then fit the model in minutes and analyze the results with ease.

The context of this research is the field of probabilistic modeling, which has emerged as a powerful language for customized data analysis. Probabilistic modeling lets us express our assumptions about data in a formal mathematical way, and then derive algorithms that use those assumptions to compute about an observed dataset. It has had an impact on myriad applications in both statistics and machine learning, including natural language processing, speech recognition, computer vision, population genetics, and computational neuroscience.

Probabilistic modeling leads to a natural research cycle. A scientist first uses her domain knowledge to posit a simple model that includes latent variables; then, she uses an inference algorithm to infer those variables from her data; next, she analyzes her results and identifies where the model works and where it falls short; last, she refines the model and repeats the process. When we cycle through these steps, we find expressive, interpretable, and useful models (Gelman et al., 2013; Blei, 2014). One of the broad goals of machine learning is to make this process easy.

Looping around this cycle, however, is not easy. The data we study are often large and complex; accordingly, we want to propose rich probabilistic models and scale them up. But using such models requires complex algorithms that are difficult to derive, implement, and scale. The bottleneck is this computation that precludes the scientist from taking full advantage of the probabilistic modeling cycle.

This problem motivates the important ideas of probabilistic programming and automated inference. Probabilistic programming allows a user to write a probability model as a computer program and then compile that program into an efficient inference executable. Automated inference is the backbone of such a system—it inputs a probability model, expressed as a program, and outputs an efficient algorithm for computing with it. Previous approaches to automatic inference have mainly relied on Markov chain Monte Carlo (MCMC) algorithms. The results have been successful, but automated MCMC is too slow for many real-world applications.

We approach the problem through variational inference, a faster alternative to MCMC that has been used in many large-scale problems (Blei et al., 2016). Though it is a promising method, developing a variational inference algorithm still requires tedious model-specific derivations and implementation; it has not seen widespread use in probabilistic programming. Here we automate the process of deriving scalable variational inference algorithms. We build on recent ideas in so-called "black-box" variational inference to leverage strengths of probabilistic programming systems, namely the ability to transform the space of latent variables and to automate derivatives of the joint distribution. The result, called automatic differentiation variational inference (ADVI), provides an automated solution to variational inference: the inputs are a probabilistic model and a dataset; the outputs are posterior inferences about the model's latent variables.[1] We implemented and deployed ADVI as part of Stan, a probabilistic programming system (Stan Development Team, 2015).

ADVI in Stan resolves the computational bottleneck of the probabilistic modeling cycle. A scientist can easily propose a probabilistic model, analyze a large dataset, and revise the model, without worrying about computation. ADVI enables this cycle by providing automated and scalable variational inference for an expansive class of models. Sections 3 and 4 present ten probabilistic modeling examples, including a progressive analysis of 1.7 million taxi trajectories.

Technical summary. Formally, a probabilistic model defines a joint distribution of observations x and latent variables θ, $p(x, \theta)$. The inference problem is to compute the posterior, the conditional distribution of the latent variables given the observations $p(\theta \mid x)$. The posterior can reveal patterns in the data and form predictions through the posterior predictive distribution. The problem is that, for many models, the posterior is not tractable to compute.

---

[1]This paper extends the method presented in (Kucukelbir et al., 2015).

Variational inference turns the task of computing a posterior into an optimization problem. We posit a parameterized family of distributions $q(\boldsymbol{\theta}) \in \mathcal{Q}$ and then find the member of that family that minimizes the Kullback-Leibler (KL) divergence to the exact posterior. Traditionally, using a variational inference algorithm requires the painstaking work of developing and implementing a custom optimization routine: specifying a variational family appropriate to the model, computing the corresponding objective function, taking derivatives, and running a gradient-based or coordinate-ascent optimization.

ADVI solves this problem automatically. The user specifies the model, expressed as a program, and ADVI automatically generates a corresponding variational algorithm. The idea is to first automatically transform the inference problem into a common space and then to solve the variational optimization. Solving the problem in this common space solves variational inference for all models in a large class. In more detail, ADVI follows these steps.

1. ADVI transforms the model into one with unconstrained real-valued latent variables. Specifically, it transforms $p(\boldsymbol{x}, \boldsymbol{\theta})$ to $p(\boldsymbol{x}, \boldsymbol{\zeta})$, where the mapping from $\boldsymbol{\theta}$ to $\boldsymbol{\zeta}$ is built into the joint distribution. This removes all original constraints on the latent variables $\boldsymbol{\theta}$. ADVI then defines the corresponding variational problem on the transformed variables, that is, to minimize $\mathrm{KL}\big(q(\boldsymbol{\zeta}) \,\|\, p(\boldsymbol{\zeta} \mid \boldsymbol{x})\big)$. With this transformation, all latent variables are defined on the same space. ADVI can now use a single variational family for all models.

2. ADVI recasts the gradient of the variational objective function as an expectation over q. This involves the gradient of the log joint with respect to the latent variables $\nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{x}, \boldsymbol{\theta})$. Expressing the gradient as an expectation opens the door to Monte Carlo methods for approximating it (Robert and Casella, 1999).

3. ADVI further reparameterizes the gradient in terms of a standard Gaussian. To do this, it uses another transformation, this time within the variational family. This second transformation enables ADVI to efficiently compute Monte Carlo approximations—it needs only to sample from a standard Gaussian (Kingma and Welling, 2014; Rezende et al., 2014).

4. ADVI uses noisy gradients to optimize the variational distribution (Robbins and Monro, 1951). An adaptively tuned step-size sequence provides good convergence in practice (Bottou, 2012).

We developed ADVI in the Stan system, which gives us two important types of automatic computation around probabilistic models. First, Stan provides a library of transformations—ways to convert a variety of constrained latent variables (e.g., positive reals) to be unconstrained, without changing the underlying joint distribution. Stan's library of transformations helps us with step 1 above. Second, Stan implements automatic differentiation to calculate $\nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{x}, \boldsymbol{\theta})$ (Carpenter et al., 2015; Baydin et al., 2015). These derivatives are crucial in step 2, when computing the gradient of the ADVI objective.

Organization of paper. Section 2 develops the recipe that makes ADVI. We expose the details of each of the steps above and present a concrete algorithm. Section 3 studies the properties of ADVI. We explore its accuracy, its stochastic nature, and its sensitivity to transformations. Section 4 applies ADVI to an array of probability models. We compare its speed to MCMC sampling techniques and present a case study using a dataset with millions of observations. Section 5 concludes the paper with a discussion.

2 Automatic Differentiation Variational Inference

Automatic differentiation variational inference (ADVI) offers a recipe for automating the computations involved in variational inference. The strategy is as follows: transform the latent variables of the model into a common space, choose a variational approximation in the common space, and use generic computational techniques to solve the variational problem.

2.1 Differentiable Probability Models

We begin by defining the class of probability models that ADVI supports. Consider a dataset $\boldsymbol{x} = x_{1:N}$ with N observations. Each x_n is a realization of a discrete or continuous (multivariate) random variable. The likelihood $p(\boldsymbol{x} \mid \boldsymbol{\theta})$ relates the observations to a set of latent random variables $\boldsymbol{\theta}$. A Bayesian model posits a prior density $p(\boldsymbol{\theta})$ on the latent variables. Combining the likelihood with the prior gives the joint density $p(\boldsymbol{x}, \boldsymbol{\theta}) = p(\boldsymbol{x} \mid \boldsymbol{\theta}) p(\boldsymbol{\theta})$. The goal of inference is to compute the posterior density $p(\boldsymbol{\theta} \mid \boldsymbol{x})$, which describes how the latent variables vary, conditioned on data.

Many posterior densities are not tractable to compute; their normalizing constants lack analytic (closed-form) solutions. Thus we often seek to approximate the posterior. ADVI approximates the posterior of differentiable probability models. Members of this class of models have continuous latent variables $\boldsymbol{\theta}$ and a gradient of the log-joint with respect to them, $\nabla_{\boldsymbol{\theta}} \log p(\boldsymbol{x}, \boldsymbol{\theta})$. The gradient is valid within the support of the prior

$$\text{supp}(p(\boldsymbol{\theta})) = \left\{ \boldsymbol{\theta} \mid \boldsymbol{\theta} \in \mathbb{R}^K \text{ and } p(\boldsymbol{\theta}) > 0 \right\} \subseteq \mathbb{R}^K,$$

where K is the dimension of the latent variable space. This support set is important: it will play a role later in the paper. We make no assumptions about conjugacy, either full (Diaconis et al., 1979) or conditional (Hoffman et al., 2013).

Consider a model that contains a Poisson likelihood with unknown rate, $p(x \mid \theta)$. The observed variable x is discrete; the latent rate θ is continuous and positive. Place a Weibull prior on θ, defined over the positive real numbers. The resulting joint density describes a nonconjugate differentiable probability model; the posterior distribution of θ is not in the same class as the prior. (The conjugate prior would be a Gamma.) However, it is in the class of differentiable models. Its partial derivative $\partial/\partial\theta \log p(x, \theta)$ is valid within the support of the Weibull distribution, $\text{supp}(p(\theta)) = \mathbb{R}_{>0} \subset \mathbb{R}$. While this model would be a challenge for classical variational inference, it is not for ADVI.

Many machine learning models are differentiable. For example: linear and logistic regression, matrix factorization with continuous or discrete observations, linear dynamical systems, and Gaussian processes. (See Table 1.) At first blush, the restriction to continuous random variables may seem to leave out other common machine learning models, such as mixture models, hidden Markov models, and topic models. However, marginalizing out the discrete variables in the likelihoods of these models renders them differentiable.[2]

Generalized linear models	*(e.g., linear / logistic / probit)*
Mixture models	*(e.g., mixture of Gaussians)*
Deep exponential families	*(e.g., deep latent Gaussian models)*
Topic models	*(e.g., latent Dirichlet allocation)*
Linear dynamical systems	*(e.g., state space models, hidden Markov models)*
Gaussian process models	*(e.g., regression / classification)*

Table 1: Popular differentiable probability models in machine learning.

2.2 Variational Inference

Variational inference turns approximate posterior inference into an optimization problem (Wainwright and Jordan, 2008; Blei et al., 2016). Consider a family of approximating densities of the latent variables $q(\boldsymbol{\theta} ; \boldsymbol{\phi})$, parameterized by a vector $\boldsymbol{\phi} \in \Phi$. Variational inference (VI) finds the parameters that minimize the KL divergence to the posterior,

$$\boldsymbol{\phi}^* = \underset{\boldsymbol{\phi} \in \Phi}{\arg\min} \, \text{KL}\left(q(\boldsymbol{\theta} ; \boldsymbol{\phi}) \,\|\, p(\boldsymbol{\theta} \mid \boldsymbol{x})\right). \tag{1}$$

---

[2] Marginalization is not tractable for all models, such as the Ising model, sigmoid belief network, and (untruncated) Bayesian nonparametric models, such as Dirichlet process mixtures (Blei et al., 2006). These are not differentiable probability models.

The optimized $q(\boldsymbol{\theta} \,;\, \boldsymbol{\phi}^*)$ then serves as an approximation to the posterior.

The KL divergence lacks an analytic form because it involves the posterior. Instead we maximize the evidence lower bound (ELBO)

$$\mathcal{L}(\boldsymbol{\phi}) = \mathbb{E}_{q(\boldsymbol{\theta})}\big[\log p(\boldsymbol{x}, \boldsymbol{\theta})\big] - \mathbb{E}_{q(\boldsymbol{\theta})}\big[\log q(\boldsymbol{\theta} \,;\, \boldsymbol{\phi})\big]. \tag{2}$$

The first term is an expectation of the joint density under the approximation, and the second is the entropy of the variational density. The ELBO is equal to the negative KL divergence up to the constant $\log p(\boldsymbol{x})$. Maximizing the ELBO minimizes the KL divergence (Jordan et al., 1999; Bishop, 2006).

Optimizing the KL divergence implies a constraint that the support of the approximation $q(\boldsymbol{\theta} \,;\, \boldsymbol{\phi})$ lie within the support of the posterior $p(\boldsymbol{\theta} \mid \boldsymbol{x})$.[3] With this constraint made explicit, the optimization problem from Equation (1) becomes

$$\boldsymbol{\phi}^* = \underset{\boldsymbol{\phi} \in \Phi}{\arg\max}\, \mathcal{L}(\boldsymbol{\phi}) \quad \text{such that} \quad \operatorname{supp}(q(\boldsymbol{\theta} \,;\, \boldsymbol{\phi})) \subseteq \operatorname{supp}(p(\boldsymbol{\theta} \mid \boldsymbol{x})). \tag{3}$$

We explicitly include this constraint because we have not specified the form of the variational approximation; we must ensure that $q(\boldsymbol{\theta} \,;\, \boldsymbol{\phi})$ stays within the support of the posterior.

The support of the posterior, however, may also be unknown. So, we further assume that the support of the posterior equals that of the prior, $\operatorname{supp}(p(\boldsymbol{\theta} \mid \boldsymbol{x})) = \operatorname{supp}(p(\boldsymbol{\theta}))$. This is a benign assumption, which holds for most models considered in machine learning. In detail, it holds when the likelihood does not constrain the prior; i.e., the likelihood must be positive over the sample space for any $\boldsymbol{\theta}$ drawn from the prior.

Our recipe for automating VI. The traditional way of solving Equation (3) is difficult. We begin by choosing a variational family $q(\boldsymbol{\theta} \,;\, \boldsymbol{\phi})$ that, by definition, satisfies the support matching constraint. We compute the expectations in the ELBO, either analytically or through approximation. We then decide on a strategy to maximize the ELBO. For instance, we might use coordinate ascent by iteratively updating the components of $\boldsymbol{\phi}$. Or, we might follow gradients of the ELBO with respect to $\boldsymbol{\phi}$ while staying within Φ. Finally, we implement, test, and debug software that performs the above. Each step requires expert thought and analysis in the service of a single algorithm for a single model.

In contrast, our approach allows the scientist to define any differentiable probability model, and we automate the process of developing a corresponding VI algorithm. Our recipe for automating VI has three ingredients. First, we automatically transform the support of the latent variables $\boldsymbol{\theta}$ to the real coordinate space (Section 2.3); this lets us choose from a variety of variational distributions q without worrying about the support matching constraint (Section 2.4). Second, we compute the ELBO for any model using Monte Carlo (MC) integration, which only requires being able to sample from the variational distribution (Section 2.5). Third, we employ stochastic gradient ascent to maximize the ELBO and use automatic differentiation to compute gradients without any user input (Section 2.6). With these tools, we can develop a generic method that automatically solves the variational optimization problem for a large class of models.

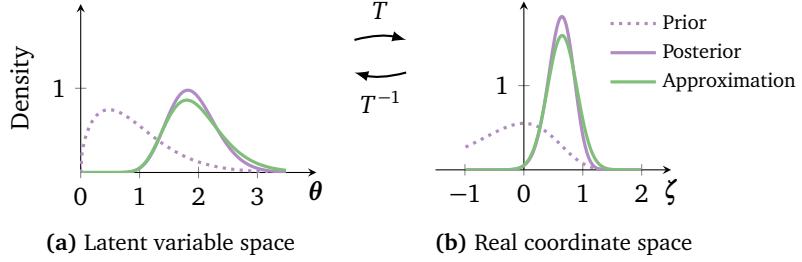## 2.3 Automatic Transformation of Constrained Variables

We begin by transforming the support of the latent variables $\boldsymbol{\theta}$ such that they live in the real coordinate space $\mathbb{R}^K$. Once we have transformed the density, we can choose the variational approximation independent of the model.

Define a one-to-one differentiable function

$$T : \operatorname{supp}(p(\boldsymbol{\theta})) \to \mathbb{R}^K, \tag{4}$$

---

[3] If $\operatorname{supp}(q) \not\subseteq \operatorname{supp}(p)$ then outside the support of p we have KL $(q \parallel p) = \mathbb{E}_q[\log q] - \mathbb{E}_q[\log p] = -\infty$.

(a) Latent variable space   **(b)** Real coordinate space

Figure 1: Transforming the latent variable to real coordinate space. The purple line is the posterior. The green line is the approximation. **(a)** The latent variable space is $\mathbb{R}_{>0}$. **(a→b)** T transforms the latent variable space to $\mathbb{R}$. **(b)** The variational approximation is a Gaussian in real coordinate space.

and identify the transformed variables as $\zeta = T(\theta)$. The transformed joint density $p(x, \zeta)$ is a function of ζ; it has the representation

$$p(x, \zeta) = p\left(x, T^{-1}(\zeta)\right)\left|\det J_{T^{-1}}(\zeta)\right|,$$

where $p(x, \theta = T^{-1}(\zeta))$ is the joint density in the original latent variable space, and $J_{T^{-1}}(\zeta)$ is the Jacobian of the inverse of T. Transformations of continuous probability densities require a Jacobian; it accounts for how the transformation warps unit volumes and ensures that the transformed density integrates to one (Olive, 2014). (See Appendix A.)

Consider again our running Weibull-Poisson example from Section 2.1. The latent variable θ lives in $\mathbb{R}_{>0}$. The logarithm $\zeta = T(\theta) = \log(\theta)$ transforms $\mathbb{R}_{>0}$ to the real line $\mathbb{R}$. Its Jacobian adjustment is the derivative of the inverse of the logarithm $|\det J_{T^{-1}(\zeta)}| = \exp(\zeta)$. The transformed density is

$$p(x, \zeta) = \text{Poisson}(x \mid \exp(\zeta)) \times \text{Weibull}(\exp(\zeta); 1.5, 1) \times \exp(\zeta).$$

Figure 1 depicts this transformation.

As we describe in the introduction, we implement our algorithm in Stan (Stan Development Team, 2015). Stan maintains a library of transformations and their corresponding Jacobians.[4] With Stan, we can automatically transforms the joint density of any differentiable probability model to one with real-valued latent variables. (See Figure 2.)

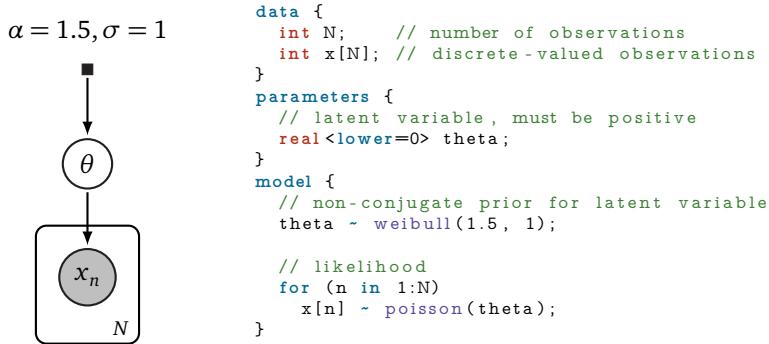## 2.4 Variational Approximations in Real Coordinate Space

After the transformation, the latent variables ζ have support in the real coordinate space $\mathbb{R}^K$. We have a choice of variational approximations in this space. Here, we consider Gaussian distributions; these implicitly induce non-Gaussian variational distributions in the original latent variable space.

Mean-field Gaussian. One option is to posit a factorized (mean-field) Gaussian variational approximation

$$q(\zeta; \phi) = \mathcal{N}\left(\zeta; \mu, \text{diag}(\sigma^2)\right) = \prod_{k=1}^{K} \mathcal{N}\left(\zeta_k; \mu_k, \sigma_k^2\right),$$

where the vector $\phi = (\mu_1, \cdots, \mu_K, \sigma_1^2, \cdots, \sigma_K^2)$ concatenates the mean and variance of each Gaussian factor. Since the variance parameters must always be positive, the variational parameters live in the set $\Phi = \{\mathbb{R}^K, \mathbb{R}_{>0}^K\}$. Re-parameterizing the mean-field Gaussian removes this constraint. Consider the

---

[4]Stan provides various transformations for upper and lower bounds, simplex and ordered vectors, and structured matrices such as covariance matrices and Cholesky factors.

$\alpha = 1.5, \sigma = 1$

```stan
data {
  int N;    // number of observations
  int x[N]; // discrete-valued observations
}
parameters {
  // latent variable, must be positive
  real<lower=0> theta;
}
model {
  // non-conjugate prior for latent variable
  theta ~ weibull(1.5, 1);

  // likelihood
  for (n in 1:N)
    x[n] ~ poisson(theta);
}
```

Figure 2: Specifying a simple nonconjugate probability model in Stan.

logarithm of the standard deviations, $\boldsymbol{\omega} = \log(\boldsymbol{\sigma})$, applied element-wise. The support of $\boldsymbol{\omega}$ is now the real coordinate space and $\boldsymbol{\sigma}$ is always positive. The mean-field Gaussian becomes $q(\boldsymbol{\zeta}\,;\boldsymbol{\phi}) = \mathcal{N}\left(\boldsymbol{\zeta}\,;\boldsymbol{\mu}, \text{diag}(\exp(\boldsymbol{\omega})^2)\right)$, where the vector $\boldsymbol{\phi} = (\mu_1, \cdots, \mu_K, \omega_1, \cdots, \omega_K)$ concatenates the mean and logarithm of the standard deviation of each factor. Now, the variational parameters are unconstrained in $\mathbb{R}^{2K}$.

Full-rank Gaussian. Another option is to posit a full-rank Gaussian variational approximation

$$q(\boldsymbol{\zeta}\,;\boldsymbol{\phi}) = \mathcal{N}\left(\boldsymbol{\zeta}\,;\boldsymbol{\mu}, \boldsymbol{\Sigma}\right),$$

where the vector $\boldsymbol{\phi} = (\boldsymbol{\mu}, \boldsymbol{\Sigma})$ concatenates the mean vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$. To ensure that $\boldsymbol{\Sigma}$ always remains positive semidefinite, we re-parameterize the covariance matrix using a Cholesky factorization, $\boldsymbol{\Sigma} = \boldsymbol{L}\boldsymbol{L}^\top$. We use the non-unique definition of the Cholesky factorization where the diagonal elements of $\boldsymbol{L}$ need not be positively constrained (Pinheiro and Bates, 1996). Therefore $\boldsymbol{L}$ lives in the unconstrained space of lower-triangular matrices with $K(K+1)/2$ real-valued entries. The full-rank Gaussian becomes $q(\boldsymbol{\zeta}\,;\boldsymbol{\phi}) = \mathcal{N}\left(\boldsymbol{\zeta}\,;\boldsymbol{\mu}, \boldsymbol{L}\boldsymbol{L}^\top\right)$, where the variational parameters $\boldsymbol{\phi} = (\boldsymbol{\mu}, \boldsymbol{L})$ are unconstrained in $\mathbb{R}^{K+K(K+1)/2}$.

The full-rank Gaussian generalizes the mean-field Gaussian approximation. The off-diagonal terms in the covariance matrix $\boldsymbol{\Sigma}$ capture posterior correlations across latent random variables.[5] This leads to a more accurate posterior approximation than the mean-field Gaussian; however, it comes at a computational cost. Various low-rank approximations to the covariance matrix reduce this cost, yet limit its ability to model complex posterior correlations (Seeger, 2010; Challis and Barber, 2013).

The choice of a Gaussian. Choosing a Gaussian distribution may call to mind the Laplace approximation technique, where a second-order Taylor expansion around the maximum-a-posteriori estimate gives a Gaussian approximation to the posterior. However, using a Gaussian variational approximation is not equivalent to the Laplace approximation (Opper and Archambeau, 2009). Our approach is distinct in another way: the posterior approximation in the original latent variable space is non-Gaussian.

The implicit variational density. The transformation T from Equation (4) maps the support of the latent variables to the real coordinate space. Thus, its inverse T^{-1} maps back to the support of the latent variables. This implicitly defines the variational approximation in the original latent variable space as $q\left(T(\boldsymbol{\theta})\,;\boldsymbol{\phi}\right)\left|\det J_T(\boldsymbol{\theta})\right|$. The transformation ensures that the support of this approximation is always bounded by that of the posterior in the original latent variable space.

Sensitivity to T. There are many ways to transform the support a variable to the real coordinate space. The form of the transformation directly affects the shape of the variational approximation in the original latent variable space. We study sensitivity to the choice of transformation in Section 3.3.

---

[5] This is a form of structured mean-field variational inference (Wainwright and Jordan, 2008; Barber, 2012).

2.5 The Variational Problem in Real Coordinate Space

Here is the story so far. We began with a differentiable probability model $p(\boldsymbol{x}, \boldsymbol{\theta})$. We transformed the latent variables into $\boldsymbol{\zeta}$, which live in the real coordinate space. We defined variational approximations in the transformed space. Now, we consider the variational optimization problem.

Write the variational objective function, the ELBO, in real coordinate space as

$$\mathcal{L}(\boldsymbol{\phi}) = \mathbb{E}_{q(\boldsymbol{\zeta};\boldsymbol{\phi})} \left[\log p\left(\boldsymbol{x}, T^{-1}(\boldsymbol{\zeta})\right) + \log \left| \det J_{T^{-1}}(\boldsymbol{\zeta}) \right| \right] + \mathbb{H}\left[q(\boldsymbol{\zeta};\boldsymbol{\phi}) \right]. \tag{5}$$

The inverse of the transformation T^{-1} appears in the joint model, along with the determinant of the Jacobian adjustment. The ELBO is a function of the variational parameters $\boldsymbol{\phi}$ and the entropy $\mathbb{H}$, both of which depend on the variational approximation. (Derivation in Appendix B.)

Now, we can freely optimize the ELBO in the real coordinate space without worrying about the support matching constraint. The optimization problem from Equation (3) becomes

$$\boldsymbol{\phi}^* = \underset{\boldsymbol{\phi}}{\arg\max}\, \mathcal{L}(\boldsymbol{\phi}) \tag{6}$$

where the parameter vector $\boldsymbol{\phi}$ lives in some appropriately dimensioned real coordinate space. This is an unconstrained optimization problem that we can solve using gradient ascent. Traditionally, this would require manual computation of gradients. Instead, we develop a stochastic gradient ascent algorithm that uses automatic differentiation to compute gradients and MC integration to approximate expectations.

We cannot directly use automatic differentiation on the ELBO. This is because the ELBO involves an unknown expectation. However, we can automatically differentiate the functions inside the expectation. (The model p and transformation T are both easy to represent as computer functions (Baydin et al., 2015).) To apply automatic differentiation, we want to push the gradient operation inside the expectation. To this end, we employ one final transformation: elliptical standardization[6] (Härdle and Simar, 2012).

Elliptical standardization. Consider a transformation $S_{\boldsymbol{\phi}}$ that absorbs the variational parameters $\boldsymbol{\phi}$; this converts the Gaussian variational approximation into a standard Gaussian. In the mean-field case, the standardization is $\boldsymbol{\eta} = S_{\boldsymbol{\phi}}(\boldsymbol{\zeta}) = \mathrm{diag}\left(\exp(\boldsymbol{\omega})\right)^{-1}(\boldsymbol{\zeta} - \boldsymbol{\mu})$. In the full-rank Gaussian, the standardization is $\boldsymbol{\eta} = S_{\boldsymbol{\phi}}(\boldsymbol{\zeta}) = L^{-1}(\boldsymbol{\zeta} - \boldsymbol{\mu})$.

In both cases, the standardization encapsulates the variational parameters; in return it gives a fixed variational density

$$q(\boldsymbol{\eta}) = \mathcal{N}\left(\boldsymbol{\eta}\,;\, \mathbf{0}, I\right) = \prod_{k=1}^{K} \mathcal{N}\left(\eta_k\,;\, 0, 1\right),$$

as shown in Figure 3.

The standardization transforms the variational problem from Equation (5) into
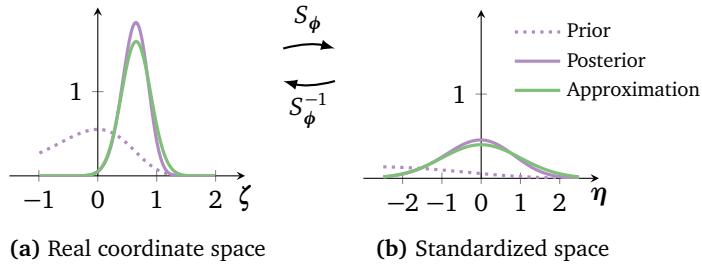
$$\boldsymbol{\phi}^* = \underset{\boldsymbol{\phi}}{\arg\max}\, \mathbb{E}_{\mathcal{N}(\boldsymbol{\eta};\mathbf{0},I)} \left[\log p\left(\boldsymbol{x}, T^{-1}(S_{\boldsymbol{\phi}}^{-1}(\boldsymbol{\eta}))\right) + \log \left| \det J_{T^{-1}}\left(S_{\boldsymbol{\phi}}^{-1}(\boldsymbol{\eta})\right) \right| \right] + \mathbb{H}\left[q(\boldsymbol{\zeta};\boldsymbol{\phi}) \right].$$

The expectation is now in terms of a standard Gaussian density. The Jacobian of elliptical standardization evaluates to one, because the Gaussian distribution is a member of the location-scale family: standardizing a Gaussian gives another Gaussian distribution. (See Appendix A.)

We do not need to transform the entropy term as it does not depend on the model or the transformation; we have a simple analytic form for the entropy of a Gaussian and its gradient. We implement these once and reuse for all models.

---

[6]Also known as a "coordinate transformation" (Rezende et al., 2014), an "invertible transformation" (Titsias and Lázaro-Gredilla, 2014), and the "re-parameterization trick" (Kingma and Welling, 2014).

(a) Real coordinate space   **(b)** Standardized space

Figure 3: Elliptical standardization. The purple line is the posterior. The green line is the approximation. **(a)** The variational approximation is a Gaussian in real coordinate space. **(a→b)** S_ϕ absorbs the parameters of the Gaussian. **(b)** We maximize the ELBO in the standardized space, with a fixed approximation. The green line is a standard Gaussian.

2.6 Stochastic Optimization

We now reach the final step: stochastic optimization of the variational objective function.

Computing gradients. Since the expectation is no longer dependent on ϕ, we can directly calculate its gradient. Push the gradient inside the expectation and apply the chain rule to get

$$\nabla_{\mu}\mathcal{L} = \mathbb{E}_{\mathcal{N}(\eta)}\left[\nabla_{\theta}\log p(x,\theta)\nabla_{\zeta}T^{-1}(\zeta) + \nabla_{\zeta}\log\left|\det J_{T^{-1}}(\zeta)\right|\right]. \tag{7}$$

We obtain gradients with respect to ω (mean-field) and L (full-rank) in a similar fashion

$$\nabla_{\omega}\mathcal{L} = \mathbb{E}_{\mathcal{N}(\eta)}\left[\left(\nabla_{\theta}\log p(x,\theta)\nabla_{\zeta}T^{-1}(\zeta) + \nabla_{\zeta}\log\left|\det J_{T^{-1}}(\zeta)\right|\right)\eta^{\top}\mathrm{diag}(\exp(\omega))\right] + \mathbf{1} \tag{8}$$

$$\nabla_{L}\mathcal{L} = \mathbb{E}_{\mathcal{N}(\eta)}\left[\left(\nabla_{\theta}\log p(x,\theta)\nabla_{\zeta}T^{-1}(\zeta) + \nabla_{\zeta}\log\left|\det J_{T^{-1}}(\zeta)\right|\right)\eta^{\top}\right] + (L^{-1})^{\top}. \tag{9}$$

(Derivations in Appendix C.)

We can now compute the gradients inside the expectation with automatic differentiation. The only thing left is the expectation. MC integration provides a simple approximation: draw samples from the standard Gaussian and evaluate the empirical mean of the gradients within the expectation (Appendix D). In practice a single sample suffices. (We study this in detail in Section 3.2 and in the experiments in Section 4.)

This gives noisy unbiased gradients of the ELBO for any differentiable probability model. We can now use these gradients in a stochastic optimization routine to automate variational inference.

Stochastic gradient ascent. Equipped with noisy unbiased gradients of the ELBO, ADVI implements stochastic gradient ascent (Algorithm 1). This algorithm is guaranteed to converge to a local maximum of the ELBO under certain conditions on the step-size sequence.[7] Stochastic gradient ascent falls under the class of stochastic approximations, where Robbins and Monro (1951) established a pair of conditions that ensure convergence: most prominently, the step-size sequence must decay sufficiently quickly. Many sequences satisfy these criteria, but their specific forms impact the success of stochastic gradient ascent in practice. We describe an adaptive step-size sequence for ADVI below.

Adaptive step-size sequence. Adaptive step-size sequences retain (possibly infinite) memory about past gradients and adapt to the high-dimensional curvature of the ELBO optimization space (Amari, 1998; Duchi et al., 2011; Ranganath et al., 2013; Kingma and Adam, 2015). These sequences enjoy theoretical bounds on their convergence rates. However, in practice, they can be slow to converge. The empirically justified RMSPROP sequence (Tieleman and Hinton, 2012), which only retains finite memory

---

[7]This is also called a *learning rate* or *schedule* in the machine learning community.

of past gradients, converges quickly in practice but lacks any convergence guarantees. We propose a new step-size sequence which effectively combines both approaches.

Consider the step-size $\boldsymbol{\rho}^{(i)}$ and a gradient vector $\boldsymbol{g}^{(i)}$ at iteration i. We define the kth element of $\boldsymbol{\rho}^{(i)}$ as

$$\rho_k^{(i)} = \eta \times i^{-1/2+\epsilon} \times \left(\tau + \sqrt{s_k^{(i)}} \right)^{-1}, \tag{10}$$

where we apply the following recursive update

$$s_k^{(i)} = \alpha g_k^{2(i)} + (1-\alpha)s_k^{(i-1)}, \tag{11}$$

with an initialization of $s_k^{(1)} = g_k^{2(1)}$.

The first factor $\eta \in \mathbb{R}_{>0}$ controls the scale of the step-size sequence. It mainly affects the beginning of the optimization. We adaptively tune η by searching over $\eta \in \{0.01, 0.1, 1, 10, 100\}$ using a subset of the data and selecting the value that leads to the fastest convergence (Bottou, 2012).

The middle term $i^{-1/2+\epsilon}$ decays as a function of the iteration i. We set $\epsilon = 10^{-16}$, a small value that guarantees that the step-size sequence satisfies the Robbins and Monro (1951) conditions.

The last term adapts to the curvature of the ELBO optimization space. Memory about past gradients are processed in Equation (11). The weighting factor $\alpha \in (0, 1)$ defines a compromise of old and new gradient information, which we set to 0.1. The quantity s_k converges to a non-zero constant. Without the previous decaying term, this would lead to possibly large oscillations around a local optimum of the ELBO. The additional perturbation $\tau > 0$ prevents division by zero and down-weights early iterations. In practice the step-size is not very sensitive to this value (Hoffman et al., 2013), so we set $\tau = 1$.

Complexity and data subsampling. ADVI has complexity $\mathcal{O}(NMK)$ per iteration, where N is the number of data points, M is the number of MC samples (typically between 1 and 10), and K is the number of latent variables. Classical VI which hand-derives a coordinate ascent algorithm has complexity $\mathcal{O}(NK)$ per pass over the dataset. The added complexity of automatic differentiation over analytic gradients is roughly constant (Carpenter et al., 2015; Baydin et al., 2015).

We scale ADVI to large datasets using stochastic optimization with data subsampling (Hoffman et al., 2013; Titsias and Lázaro-Gredilla, 2014). The adjustment to Algorithm 1 is simple: sample a minibatch of size $B \ll N$ from the dataset and scale the likelihood of the model by N/B (Hoffman et al., 2013). The stochastic extension of ADVI has a per-iteration complexity $\mathcal{O}(BMK)$.

In Sections 4.3 and 4.4, we apply this stochastic extension to analyze datasets with millions of observations.

2.7 Related Work

ADVI automates variational inference within the Stan probabilistic programming system. This draws on two major themes.

The first theme is probabilistic programming. One class of systems focuses on probabilistic models where the user specifies a joint probability distribution. Some examples are BUGS (Spiegelhalter et al., 1995), JAGS (Plummer et al., 2003), and Stan (Stan Development Team, 2015). Another class of systems allows the user to directly specify more general probabilistic programs. Some examples are Church (Goodman et al., 2008), Figaro (Pfeffer, 2009), Venture (Mansinghka et al., 2014), and Anglican (Wood et al., 2014). Both classes primarily rely on various forms of MCMC techniques for inference; they typically cannot scale to very large data.

The second is a body of work that generalizes variational inference. Ranganath et al. (2014) and Salimans and Knowles (2014) propose a black-box technique that only requires computing gradients of

---

Algorithm 1: Automatic Differentiation Variational Inference

Input: Dataset $\boldsymbol{x} = x_{1:N}$, model $p(\boldsymbol{x}, \boldsymbol{\theta})$.

Set iteration counter $i = 1$.

Initialize $\boldsymbol{\mu}^{(1)} = \boldsymbol{0}$.

Initialize $\boldsymbol{\omega}^{(1)} = \boldsymbol{0}$ (mean-field) or $\boldsymbol{L}^{(1)} = \boldsymbol{I}$ (full-rank).

Determine η via a search over finite values.

while *change in* ELBO *is above some threshold* **do**

    Draw M samples $\boldsymbol{\eta}_m \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ from the standard multivariate Gaussian.

    Approximate $\nabla_{\boldsymbol{\mu}}\mathcal{L}$ using MC integration (Equation (7)).

    Approximate $\nabla_{\boldsymbol{\omega}}\mathcal{L}$ or $\nabla_L\mathcal{L}$ using MC integration (Equations (8) and (9)).

    Calculate step-size $\boldsymbol{\rho}^{(i)}$ (Equation (10)).

    Update $\boldsymbol{\mu}^{(i+1)} \longleftarrow \boldsymbol{\mu}^{(i)} + \text{diag}(\boldsymbol{\rho}^{(i)})\nabla_{\boldsymbol{\mu}}\mathcal{L}$.

    Update $\boldsymbol{\omega}^{(i+1)} \longleftarrow \boldsymbol{\omega}^{(i)} + \text{diag}(\boldsymbol{\rho}^{(i)})\nabla_{\boldsymbol{\omega}}\mathcal{L}$ or $\boldsymbol{L}^{(i+1)} \longleftarrow \boldsymbol{L}^{(i)} + \text{diag}(\boldsymbol{\rho}^{(i)})\nabla_L\mathcal{L}$.

    Increment iteration counter.

end

Return $\boldsymbol{\mu}^* \longleftarrow \boldsymbol{\mu}^{(i)}$.

Return $\boldsymbol{\omega}^* \longleftarrow \boldsymbol{\omega}^{(i)}$ or $\boldsymbol{L}^* \longleftarrow \boldsymbol{L}^{(i)}$.

---

the variational approximating family. Kingma and Welling (2014) and Rezende et al. (2014) describe a reparameterization of the variational problem that simplifies optimization. Titsias and Lázaro-Gredilla (2014) leverage the gradient of the model for a class of real-valued models. Rezende and Mohamed (2015) and Tran et al. (2016) improve the accuracy of black-box variational approximations. Here we build on and extend these ideas to automate variational inference; we highlight technical connections as we study the properties of ADVI in Section 3.

Some notable work crosses both themes. Bishop et al. (2002) present an automated variational algorithm for graphical models with conjugate exponential relationships between all parent-child pairs. Winn and Bishop (2005); Minka et al. (2014) extend this to graphical models with non-conjugate relationships by either using custom approximations or an expensive sampling approach. ADVI automatically supports a more comprehensive class of nonconjugate models; see Section 2.1. Wingate and Weber (2013) study a more general setting, where the variational approximation itself is a probabilistic program.
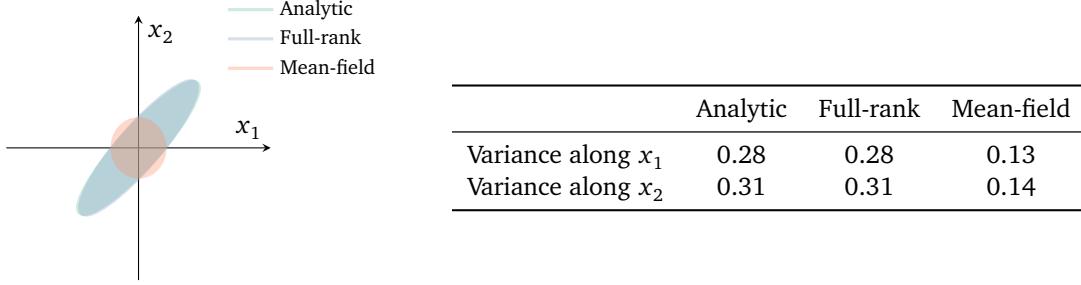
3 Properties of Automatic Differentiation Variational Inference

Automatic differentiation variational inference (ADVI) extends classical variational inference techniques in a few directions. In this section, we use simulated data to study three aspects of ADVI: the accuracy of mean-field and full-rank approximations, the variance of the ADVI gradient estimator, and the sensitivity to the transformation T.

3.1 Accuracy

We begin by considering three models that expose how the mean-field approximation affects the accuracy of ADVI.

Two-dimensional Gaussian. We first study a simple model that does not require approximate inference.

	Analytic	Full-rank	Mean-field
Variance along x_1	0.28	0.28	0.13
Variance along x_2	0.31	0.31	0.14

Figure 4: Comparison of mean-field and full-rank ADVI on a two-dimensional Gaussian model. The figure shows the accuracy of the full-rank approximation. Ellipses correspond to two-sigma level sets of the Gaussian. The table quantifies the underestimation of marginal variances by the mean-field approximation.

Consider a multivariate Gaussian likelihood $\mathcal{N}(y \mid \mu, \Sigma)$ with fixed, yet highly correlated, covariance Σ; our goal is to estimate the mean μ. If we place a multivariate Gaussian prior on μ then the posterior is also a Gaussian that we can compute analytically (Bernardo and Smith, 2009).

We draw 1000 datapoints from the model and run both variants of ADVI, mean-field and full-rank, until convergence. Figure 4 compares the ADVI methods to the exact posterior. Both procedures correctly identify the mean of the analytic posterior. However, the shape of the mean-field approximation is incorrect. This is because the mean-field approximation ignores off-diagonal terms of the Gaussian covariance. ADVI minimizes the KL divergence from the approximation to the exact posterior; this leads to a systemic underestimation of marginal variances (Bishop, 2006).

Logistic regression. We now study a model for which we need approximate inference. Consider logistic regression, a generalized linear model with a binary response y, covariates x, and likelihood $\text{Bern}(y \mid \text{logit}^{-1}(x^\top \beta))$; our goal is to estimate the coefficients β. We place an independent Gaussian prior on each regression coefficient.

We simulated 9 random covariates from the prior distribution (plus a constant intercept) and drew 1000 datapoints from the likelihood. We estimated the posterior of the coefficients with ADVI and Stan's default MCMC technique, the no-U-turn sampler (NUTS) (Hoffman and Gelman, 2014). Figure 5 shows the marginal posterior densities obtained from each approximation. MCMC and ADVI perform similarly in their estimates of the posterior mean. The mean-field approximation, as expected, underestimates marginal posterior variances on most of the coefficients. The full-rank approximation, once again, better matches the posterior.

Stochastic volatility time-series model. Finally, we study a model where the data are not exchangeable. Consider an autoregressive process to model how the latent volatility (i.e., variance) of an economic asset changes over time (Kim et al., 1998); our goal is to estimate the sequence of volatilities. We expect these posterior estimates to be correlated, especially when the volatilities trend away from their mean value.

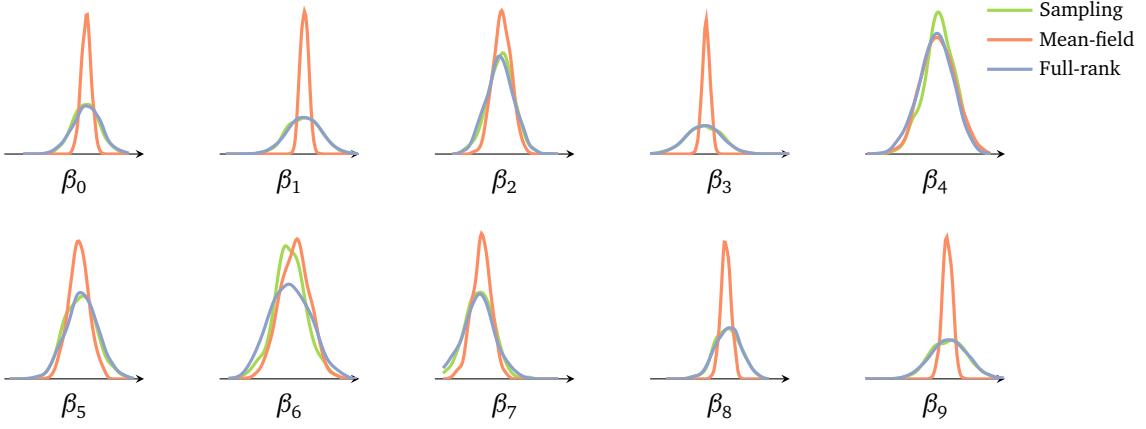In detail, the price data exhibit latent volatility as part of the variance of a zero-mean Gaussian

$$y_t \sim \mathcal{N}\left(0, \exp(h_t/2)\right)$$

where the log volatility follows an auto-regressive process

$$h_t \sim \mathcal{N}\left(\mu + \phi(h_{t-1} - \mu), \sigma\right) \quad \text{with initialization} \quad h_1 \sim \mathcal{N}\left(\mu, \frac{\sigma}{\sqrt{1-\phi^2}}\right).$$

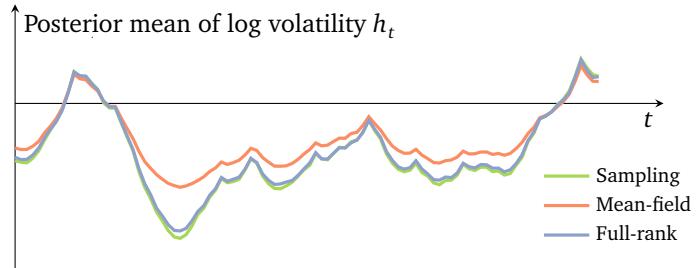We place the following priors on the latent variables

$$\mu \sim \text{Cauchy}(0, 10), \quad \phi \sim \text{Unif}(-1, 1), \quad \text{and} \quad \sigma \sim \text{LogNormal}(0, 10).$$

Figure 5: Comparison of marginal posterior densities for a logistic regression model. Each plot shows kernel density estimates for the posterior of each coefficient using 1000 samples. Mean-field ADVI underestimates variances for most of the coefficients.

We set $\mu = -1.025$, $\phi = 0.9$ and $\sigma = 0.6$, and simulate a dataset of 500 time-steps from the generative model above. Figure 6 plots the posterior mean of the log volatility h_t as a function of time. Mean-field ADVI struggles to describe the mean of the posterior, particularly when the log volatility drifts far away from μ. In contrast, full-rank ADVI matches the estimates obtained from sampling.

We further investigate this by studying posterior correlations of the log volatility sequence. We draw $S = 1000$ samples of 500-dimensional log volatility sequences $\{\boldsymbol{h}^{(s)}\}_1^S$. Figure 7 shows the empirical posterior covariance matrix, $1/s-1\sum_s(\boldsymbol{h}^{(s)} - \overline{\boldsymbol{h}})(\boldsymbol{h}^{(s)} - \overline{\boldsymbol{h}})^\top$ for each method. The mean-field covariance (fig. 7a) fails to capture the locally correlated structure of the full-rank and sampling covariance matrices (figs. 7b and 7c). All covariance matrices exhibit a blurry spread due to finite sample size.
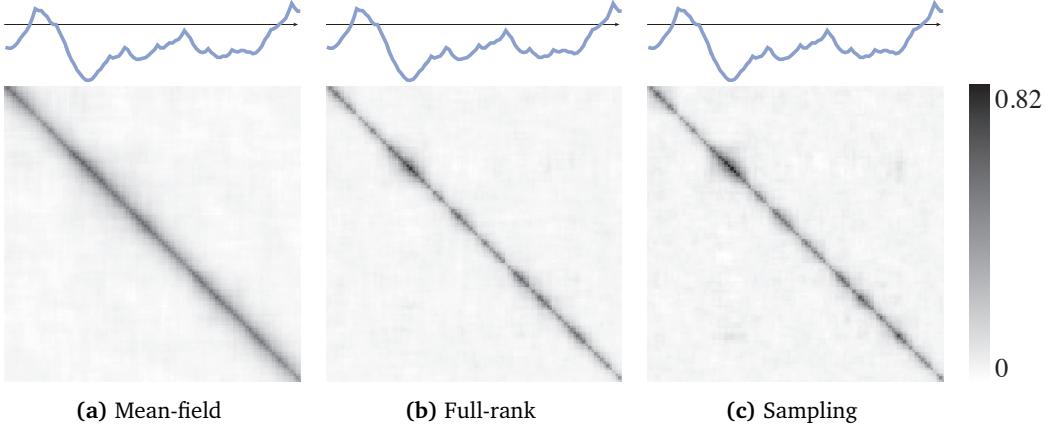

Figure 6: Comparison of posterior mean estimates of volatility h_t. Mean-field ADVI underestimates h_t, especially when it moves far away from its mean μ. Full-rank ADVI matches the accuracy of sampling.

The regions where the local correlation is strongest correspond to the regions where mean-field underestimates the log volatility. To help identify these regions, we overlay the sampling mean log volatility estimate from Figure 6 above each matrix. Both full-rank ADVI and sampling results exhibit correlation where the log volatility trends away from its mean value.

Recommendations. How to choose between full-rank and mean-field ADVI? Scientists interested in posterior variances and covariances should use the full-rank approximation. Full-rank ADVI captures posterior correlations, in turn producing more accurate marginal variance estimates. For large data, however, full-rank ADVI can be prohibitively slow.

Scientists interested in prediction should initially rely on the mean-field approximation. Mean-field

(a) Mean-field **(b)** Full-rank **(c)** Sampling

Figure 7: Comparison of empirical posterior covariance matrices. The mean-field ADVI covariance matrix fails to capture the local correlation structure seen in the full-rank ADVI and sampling results. All covariance matrices exhibit a blurry spread due to finite sample size.

ADVI offers a fast algorithm for approximating the posterior mean. In practice, accurate posterior mean estimates dominate predictive accuracy; underestimating marginal variances matters less.

3.2 Variance of the Stochastic Gradients

ADVI uses Monte Carlo integration to approximate gradients of the ELBO, and then uses these gradients in a stochastic optimization algorithm (Section 2). The speed of ADVI hinges on the variance of the gradient estimates. When a stochastic optimization algorithm suffers from high-variance gradients, it must repeatedly recover from poor parameter estimates.
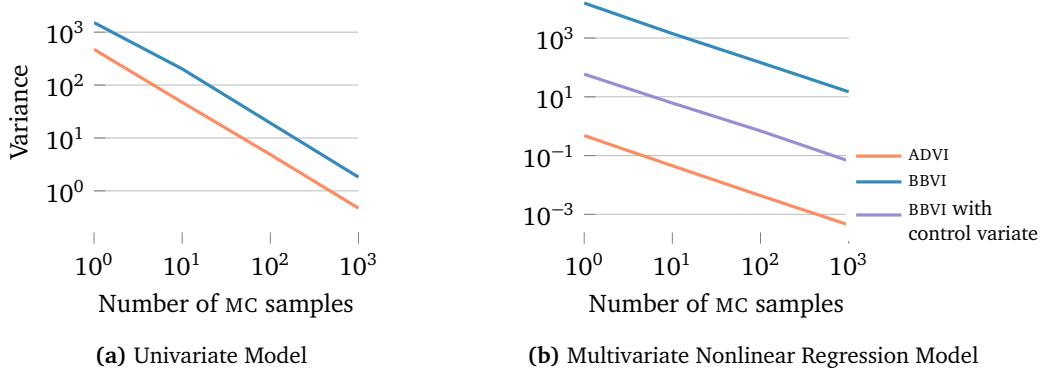
ADVI is not the only way to compute Monte Carlo approximations of the gradient of the ELBO. Black box variational inference (BBVI) takes a different approach (Ranganath et al., 2014). The BBVI gradient estimator uses the gradient of the variational approximation and avoids using the gradient of the model. For example, the following BBVI estimator

$$\nabla_{\boldsymbol{\mu}}^{\text{BBVI}} \mathcal{L} = \mathbb{E}_{q(\boldsymbol{\zeta}\,;\,\boldsymbol{\phi})} \left[\nabla_{\boldsymbol{\mu}} \log q(\boldsymbol{\zeta}\,;\,\boldsymbol{\phi}) \left\{ \log p\left(\boldsymbol{x}, T^{-1}(\boldsymbol{\zeta})\right) + \log \left| \det J_{T^{-1}}(\boldsymbol{\zeta}) \right| - \log q(\boldsymbol{\zeta}\,;\,\boldsymbol{\phi}) \right\} \right]$$

and the ADVI gradient estimator in Equation (7) both lead to unbiased estimates of the exact gradient. While BBVI is more general—it does not require the gradient of the model and thus applies to more settings—its gradients can suffer from high variance.

Figure 8 empirically compares the variance of both estimators for two models. Figure 8a shows the variance of both gradient estimators for a simple univariate model, where the posterior is a $\text{Gamma}(10, 10)$. We estimate the variance using ten thousand re-calculations of the gradient $\nabla_{\boldsymbol{\phi}} \mathcal{L}$, across an increasing number of MC samples M. The ADVI gradient has lower variance; in practice, a single sample suffices. (See the experiments in Section 4.)

Figure 8b shows the same calculation for a 100-dimensional nonlinear regression model with likelihood $\mathcal{N}(\boldsymbol{y} \mid \tanh(\boldsymbol{x}^{\top}\boldsymbol{\beta}), \boldsymbol{I})$ and a Gaussian prior on the regression coefficients $\boldsymbol{\beta}$. Because this is a multivariate example, we also show the BBVI gradient with a variance reduction scheme using control variates described in Ranganath et al. (2014). In both cases, the ADVI gradients are statistically more efficient.

(a) Univariate Model **(b)** Multivariate Nonlinear Regression Model

Figure 8: Comparison of gradient estimator variances. The ADVI gradient estimator exhibits lower variance than the BBVI estimator. Moreover, it does not require control variate variance reduction, which is not available in univariate situations.
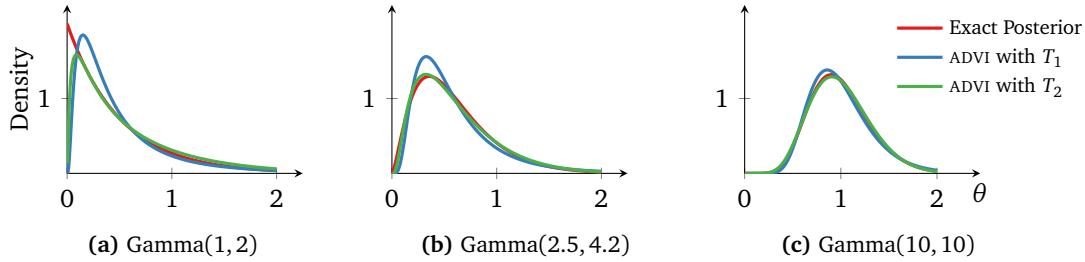
3.3   Sensitivity to Transformations

ADVI uses a transformation T from the unconstrained space to the constrained space. We now study how the choice of this transformation affects the non-Gaussian posterior approximation in the original latent variable space.

Consider a posterior density in the Gamma family, with support over $\mathbb{R}_{>0}$. Figure 9 shows three configurations of the Gamma, ranging from Gamma$(1,2)$, which places most of its mass close to $\theta = 0$, to Gamma$(10,10)$, which is centered at $\theta = 1$. Consider two transformations T_1 and T_2

$$T_1 : \theta \mapsto \log(\theta) \quad \text{and} \quad T_2 : \theta \mapsto \log(\exp(\theta) - 1),$$

both of which map $\mathbb{R}_{>0}$ to $\mathbb{R}$. ADVI can use either transformation to approximate the Gamma posterior. Which one is better?

Figure 9 show the ADVI approximation under both transformations. Table 2 reports the corresponding KL divergences. Both graphical and numerical results prefer T_2 over T_1. A quick analysis corroborates this. T_1 is the logarithm, which flattens out for large values. However, T_2 is almost linear for large values of θ. Since both the Gamma (the posterior) and the Gaussian (the ADVI approximation) densities are light-tailed, T_2 is the preferable transformation.

(a) Gamma$(1,2)$ **(b)** Gamma$(2.5,4.2)$ **(c)** Gamma$(10,10)$

Figure 9: ADVI approximations to Gamma densities under two different transformations.

Is there an optimal transformation? Without loss of generality, we consider fixing a standard Gaussian

	Gamma$(1, 2)$	Gamma$(2.5, 4.2)$	Gamma$(10, 10)$
KL $(q \parallel p)$ with T_1	8.1×10^{-2}	3.3×10^{-2}	8.5×10^{-3}
KL $(q \parallel p)$ with T_2	1.6×10^{-2}	3.6×10^{-3}	7.7×10^{-4}

Table 2: KL divergence of ADVI approximations to Gamma densities under two different transformations.

distribution in the real coordinate space.[8] The optimal transformation is then

$$T^* = \Phi^{-1} \circ P(\boldsymbol{\theta} \mid \boldsymbol{x})$$

where P is the cumulative density function of the posterior and Φ^{-1} is the inverse cumulative density function of the standard Gaussian. P maps the posterior to a uniform distribution and Φ^{-1} maps the uniform distribution to the standard Gaussian. The optimal choice of transformation enables the Gaussian variational approximation to be exact. Sadly, estimating the optimal transformation requires estimating the cumulative density function of the posterior $P(\boldsymbol{\theta} \mid \boldsymbol{x})$; this is just as hard as the original goal of estimating the posterior density $p(\boldsymbol{\theta} \mid \boldsymbol{x})$.

This observation motivates pairing transformations with Gaussian variational approximations; there is no need for more complex variational families. ADVI takes the approach of using a library and a model compiler. This is not the only option. For example, Knowles (2015) posits a factorized Gamma density for positively constrained latent variables. In theory, this is equivalent to a mean-field Gaussian density paired with the transformation $T = P_{\text{Gamma}}$, the cumulative density function of the Gamma. (In practice, P_{Gamma} is difficult to compute.) Challis and Barber (2012) study Fourier transform techniques for location-scale variational approximations beyond the Gaussian. Another option is to learn the transformation during optimization. We discuss recent approaches in this direction in Section 5.

4   Automatic Differentiation Variational Inference in Practice

We now apply automatic differentiation variational inference (ADVI) to an array of nonconjugate probability models. With simulated and real data, we study linear regression with automatic relevance determination, hierarchical logistic regression, several variants of non-negative matrix factorization, mixture models, and probabilistic principal component analysis. We compare mean-field ADVI to two MCMC sampling algorithms: Hamiltonian Monte Carlo (HMC) (Girolami and Calderhead, 2011) and NUTS, which is an adaptive extension of HMC[9] (Hoffman and Gelman, 2014).

To place ADVI and MCMC on a common scale, we report predictive likelihood on held-out data as a function of time. Specifically, we estimate the predictive likelihood

$$p(\boldsymbol{x}_{\text{held-out}} \mid \boldsymbol{x}) = \int p(\boldsymbol{x}_{\text{held-out}} \mid \boldsymbol{\theta}) p(\boldsymbol{\theta} \mid \boldsymbol{x}) \, \mathrm{d}\boldsymbol{\theta}$$

using Monte Carlo estimation. With MCMC, we run the chain and plug in each sample to estimate the integral above; with ADVI, we draw a sample from the variational approximation at every iteration.

We conclude with a case study: an exploratory analysis of millions of taxi rides. Here we show how a scientist might use ADVI in practice.

---

[8]For two transformations T_1 and T_2 from latent variable space to real coordinate space, there always exists a transformation T_3 within the real coordinate space such that $T_1(\theta) = T_3(T_2(\theta))$.

[9]It is the default sampler in Stan.
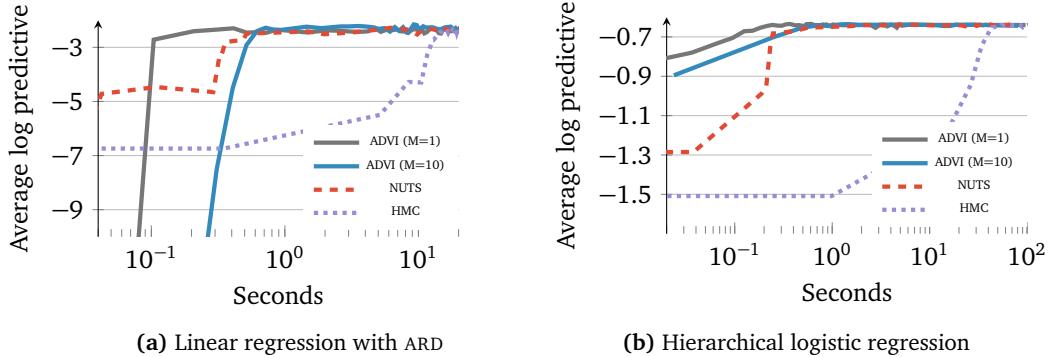
4.1 Hierarchical Regression Models

We begin with two nonconjugate regression models: linear regression with automatic relevance determination (ARD) (Bishop, 2006) and hierarchical logistic regression (Gelman and Hill, 2006).

Linear regression with ARD. This is a linear regression model with a hierarchical prior structure that leads to sparse estimates of the coefficients. (Details in Appendix F.1.) We simulate a dataset with 250 regressors such that half of the regressors have no predictive power. We use 10 000 data points for training and withhold 1000 for evaluation.

Logistic regression with a spatial hierarchical prior. This is a hierarchical logistic regression model from political science. The prior captures dependencies, such as states and regions, in a polling dataset from the United States 1988 presidential election (Gelman and Hill, 2006). The model is nonconjugate and would require some form of approximation to derive a classical VI algorithm. (Details in Appendix F.2.)

The dataset includes 145 regressors, with age, education, and state and region indicators. We use 10 000 data points for training and withhold 1536 for evaluation.

Results. Figure 10 plots average log predictive accuracy as a function of time. For these simple models, all methods reach the same predictive accuracy. We study ADVI with two settings of M, the number of MC samples used to estimate gradients. A single sample per iteration is sufficient; it is also the fastest. (We set $M = 1$ from here on.)

(a) Linear regression with ARD   **(b)** Hierarchical logistic regression

Figure 10: Held-out predictive accuracy results | hierarchical generalized linear models on simulated and real data.
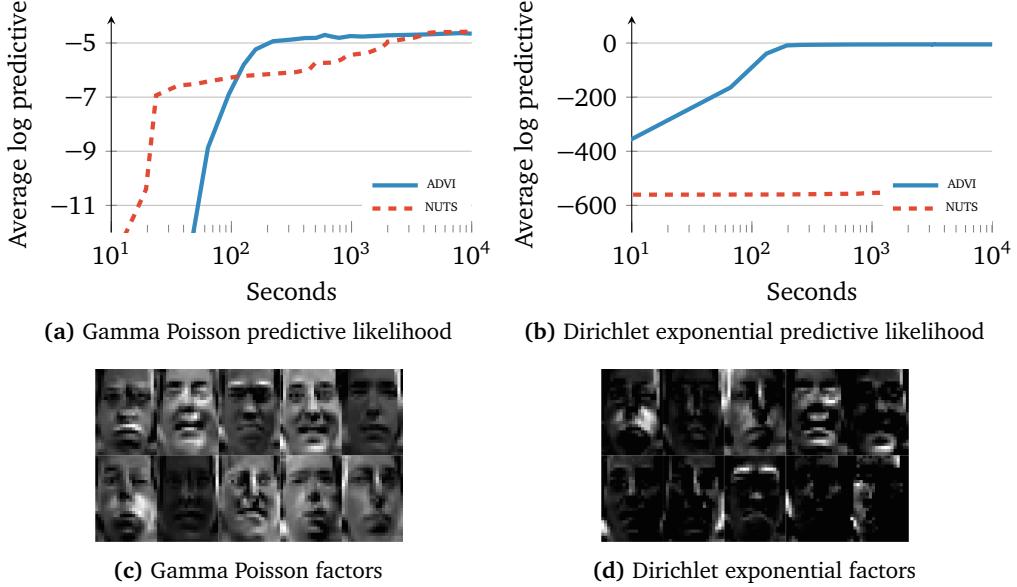
4.2 Non-negative Matrix Factorization

We continue by exploring two nonconjugate non-negative matrix factorization models (Lee and Seung, 1999): a constrained Gamma Poisson model (Canny, 2004) and a Dirichlet Exponential Poisson model. Here, we show how easy it is to explore new models using ADVI. In both models, we use the Frey Face dataset, which contains 1956 frames (28×20 pixels) of facial expressions extracted from a video sequence.

Constrained Gamma Poisson. This is a Gamma Poisson matrix factorization model with an ordering constraint: each row of one of the Gamma factors goes from small to large values. (Details in Appendix F.3.)

Dirichlet Exponential Poisson. This is a nonconjugate Dirichlet Exponential factorization model with a Poisson likelihood. (Details in Appendix F.4.)

17

Results. Figure 11 shows average log predictive accuracy as well as ten factors recovered from both models. ADVI provides an order of magnitude speed improvement over NUTS. NUTS struggles with the Dirichlet Exponential model. In both cases, HMC does not produce any useful samples within a budget of one hour; we omit HMC from here on.

The Gamma Poisson model appears to pick significant frames out of the dataset. The Dirichlet Exponential factors are sparse and indicate components of the face that move, such as eyebrows, cheeks, and the mouth.
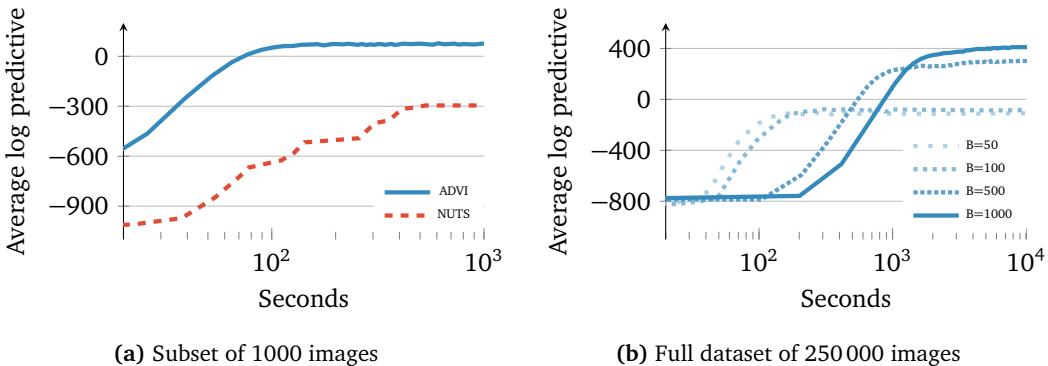

(a) Gamma Poisson predictive likelihood

(b) Dirichlet exponential predictive likelihood

(c) Gamma Poisson factors

(d) Dirichlet exponential factors

Figure 11: Held-out predictive accuracy results | two non-negative matrix factorization models applied to the Frey Faces dataset.

4.3 Gaussian Mixture Model

This is a nonconjugate Gaussian mixture model (GMM) applied to color image histograms. We place a Dirichlet prior on the mixture proportions, a Gaussian prior on the component means, and a lognormal prior on the standard deviations. (Details in Appendix F.5.) We explore the imageCLEF dataset, which has 250 000 images (Villegas et al., 2013). We withhold 10 000 images for evaluation.

In Figure 12a we randomly select 1000 images and train a model with 10 mixture components. ADVI quickly finds a good solution. NUTS struggles to find an adequate solution and HMC fails altogether (not shown). This is likely due to label switching, which can affect HMC-based algorithms in mixture models (Stan Development Team, 2015).

Figure 12b shows ADVI results on the full dataset. We increase the number of mixture components to 30. Here we use ADVI, with additional stochastic subsampling of minibatches from the data (Hoffman et al., 2013). With a minibatch size of 500 or larger, ADVI reaches high predictive accuracy. Smaller minibatch sizes lead to suboptimal solutions, an effect also observed in Hoffman et al. (2013). ADVI converges in about two hours; NUTS cannot handle such large datasets.

(a) Subset of 1000 images   **(b)** Full dataset of 250 000 images

Figure 12: Held-out predictive accuracy results | GMM of the imageCLEF image histogram dataset. **(a)** ADVI outperforms NUTS (Hoffman and Gelman, 2014). **(b)** ADVI scales to large datasets by subsampling minibatches of size B from the dataset at each iteration (Hoffman et al., 2013).

4.4 A Case Study: Exploring Millions of Taxi Trajectories

How might a scientist use ADVI in practice? How easy is it to develop and revise new models? To answer these questions, we apply ADVI to a modern exploratory data analysis task: analyzing traffic patterns. In this section, we demonstrate how ADVI enables a scientist to quickly develop and revise complex hierarchical models.

The city of Porto has a centralized taxi system of 442 cars. When serving customers, each taxi reports its spatial location at 15 second intervals; this sequence of (x, y) coordinates describes the trajectory and duration of each trip. A dataset of trajectories is publicly available: it contains all 1.7 million taxi rides taken during the year 2014 (European Conference of Machine Learning, 2015).
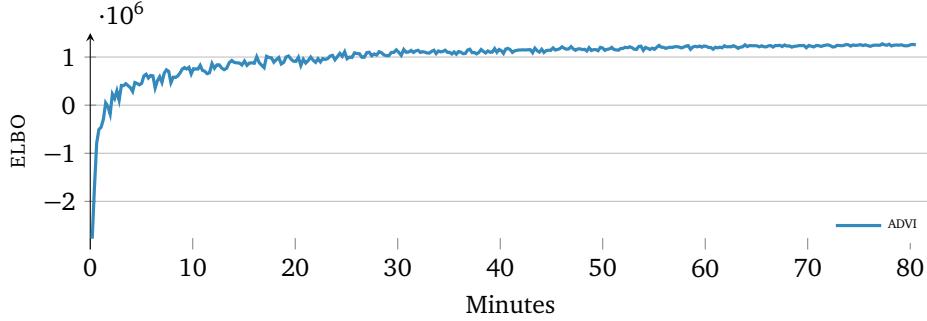
To gain insight into this dataset, we wish to cluster the trajectories. The first task is to process the raw data. Each trajectory has a different length: shorter trips contain fewer (x, y) coordinates than longer ones. The average trip is approximately 13 minutes long, which corresponds to 50 coordinates. We want to cluster independent of length, so we interpolate all trajectories to 50 coordinate pairs. This converts each trajectory into a point in $\mathbb{R}^{100}$.

The trajectories have structure; for example, major roads and highways appear frequently. This motivates an approach where we first identify a lower-dimensional representation of the data to capture aggregate features, and then we cluster the trajectories in this representation. This is easier than clustering them in the original data space.

We begin with simple dimension reduction: probabilistic principal component analysis (PPCA) (Bishop, 2006). This is a Bayesian generalization of classical principal component analysis, which is easy to write in Stan. However, like its classical counterpart, PPCA does not identify how many principal components to use for the subspace. To address this, we propose an extension: PPCA with automatic relevance determination (ARD).

PPCA with ARD identifies the latent dimensions that are most effective at explaining variation in the data. The strategy is similar to that in Section 4.1. We assume that there are 100 latent dimensions (i.e., the same dimension as the data) and impose a hierarchical prior that encourages sparsity. Consequently, the model only uses a subset of the latent dimensions to describe the data. (Details in Appendix F.6.)
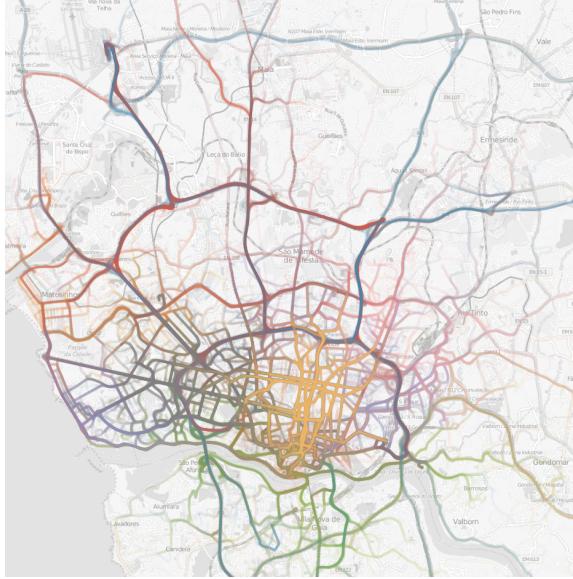
We randomly subsample ten thousand trajectories and use ADVI to infer a subspace. Figure 13 plots the progression of the ELBO. ADVI converges in approximately an hour and finds an eleven-dimensional subspace. We omit sampling results as both HMC and NUTS struggle with the model; neither produce useful samples within an hour.

Figure 13: ELBO of PPCA model with ARD. ADVI converges in approximately an hour.

Equipped with this eleven-dimensional subspace, we turn to analyzing the full dataset of 1.7 million taxi trajectories. We first project all trajectories into the subspace. We then use the GMM from Section 4.3 ($K = 30$) components to cluster the trajectories. ADVI takes less than half an hour to converge.

Figure 14 shows a visualization of fifty thousand randomly sampled trajectories. Each color represents the set of trajectories that associate with a particular Gaussian mixture. The clustering is geographical: taxi trajectories that are close to each other are bundled together. The clusters identify frequently taken taxi trajectories.
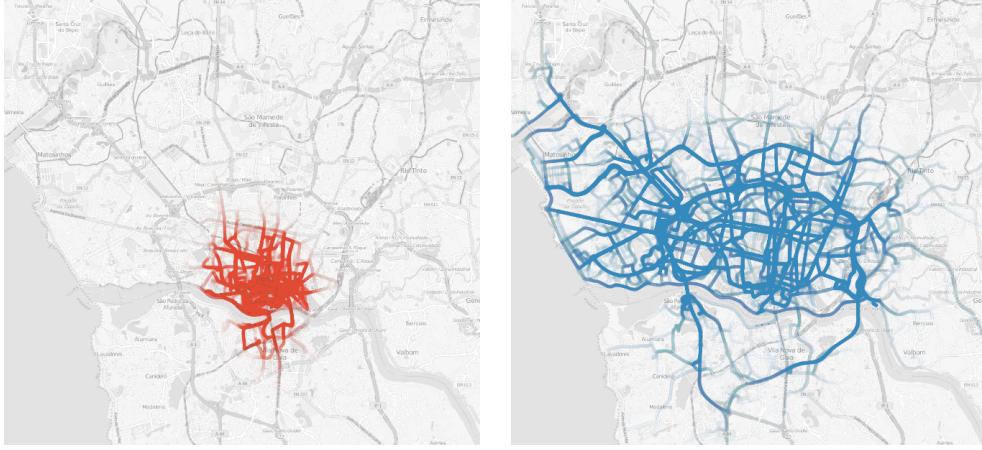

Figure 14: A visualization of fifty thousand randomly sampled taxi trajectories. The colors represent thirty Gaussian mixtures and the trajectories associated with each.

When we processed the raw data, we interpolated each trajectory to an equal length. This discards all duration information. What if some roads are particularly prone to traffic? Do these roads lead to longer trips?

Supervised probabilistic principal component analysis (SUP-PPCA) is one way to model this. The idea is to regress the durations of each trip onto a subspace that also explains variation in a response variable, in this case, the duration. SUP-PPCA is a simple extension of PPCA (Murphy, 2012). We further extend it using the same ARD prior as before. (Details in Appendix F.7.)

ADVI enables a quick repeat of the above analysis, this time with SUP-PPCA. With ADVI, we find another

set of GMM clusters in less than two hours. These clusters, however, are more informative.

(a) Trajectories that take the inner bridges.   **(b)** Trajectories that take the outer bridges.

Figure 15: Two clusters using SUP-PPCA subspace clustering.

Figure 15 shows two clusters that identify particularly busy roads: the bridges of Porto that cross the Duoro river. Figure 15a shows a group of short trajectories that use the two old bridges near the city center. Figure 15b show a group of longer trajectories that use the two newer bridges that connect highways that circumscribe the city.

Analyzing these taxi trajectories illustrates how exploratory data analysis is an iterative effort: we want to rapidly evaluate models and modify them based on what we learn. ADVI, which provides automatic and fast inference, enables effective exploration of massive datasets.

5   Discussion

We presented automatic differentiation variational inference (ADVI), a variational inference tool that works for a large class of probabilistic models. The main idea is to transform the latent variables into a common space. Solving the variational inference problem in this common space solves it for all models in the class. We studied ADVI using ten different probability models; this showcases how easy it is to use ADVI in practice. We also developed and deployed ADVI as part of Stan, a probabilistic programming system; this makes ADVI available to everyone.

There are several avenues for research.

Begin with accuracy. As we showed in Section 3.3, ADVI can be sensitive to the transformations that map the constrained parameter space to the real coordinate space. Dinh et al. (2014) and Rezende and Mohamed (2015) use a cascade of simple transformations to improve accuracy. Tran et al. (2016) place a Gaussian process to learn the optimal transformation and prove its expressiveness as a universal approximator. A class of hierarchical variational models (Ranganath et al., 2015) extend these complex distributions to discrete latent variable models.

Continue with optimization. ADVI uses first-order automatic differentiation to implement stochastic gradient ascent. Higher-order gradients may enable faster convergence; however computing higher-order gradients comes at a computational cost (Fan et al., 2015). Optimization using line search could also improve convergence speed and robustness (Mahsereci and Hennig, 2015), as well as natural gradient approaches for nonconjugate models (Khan et al., 2015).

Follow with practical heuristics. Two things affect ADVI convergence: initialization and step-size scaling. We initialize ADVI in the real coordinate space as a standard Gaussian. A better heuristic could adapt

to the model and dataset based on moment matching. We adaptively tune the scale of the step-size sequence using a finite search. A better heuristic could avoid this additional computation.

End with probabilistic programming. We designed and deployed ADVI with Stan in mind. Thus, we focused on the class of differentiable probability models. How can we extend ADVI to discrete latent variables? One approach would be to adapt ADVI to use the black box gradient estimator for these variables (Ranganath et al., 2014). This requires some care as these gradients will exhibit higher variance than the gradients with respect to the differentiable latent variables. (See Section 3.2.) With support for discrete latent variables, modified versions of ADVI could be extended to more general probabilistic programming systems, such as Church (Goodman et al., 2008), Figaro (Pfeffer, 2009), Venture (Mansinghka et al., 2014), and Anglican (Wood et al., 2014).

A Transformations of Continuous Probability Densities

We present a brief summary of transformations, largely based on (Olive, 2014).

Consider a scalar (univariate) random variable X with probability density function $f_X(x)$. Let $\mathcal{X} = \text{supp}(f_X(x))$ be the support of X. Now consider another random variable Y defined as $Y = T(X)$. Let $\mathcal{Y} = \text{supp}(f_Y(y))$ be the support of Y.

If T is a one-to-one and differentiable function from $\mathcal{X}$ to $\mathcal{Y}$, then Y has probability density function

$$f_Y(y) = f_X\left(T^{-1}(y)\right) \left| \frac{\mathrm{d} T^{-1}(y)}{\mathrm{d} y} \right|.$$

Let us sketch a proof. Consider the cumulative density function Y. If the transformation T is increasing, we directly apply its inverse to the cdf of Y. If the transformation T is decreasing, we apply its inverse to one minus the cdf of Y. The probability density function is the derivative of the cumulative density function. These things combined give the absolute value of the derivative above.

The extension to multivariate variables $\boldsymbol{X}$ and $\boldsymbol{Y}$ requires a multivariate version of the absolute value of the derivative of the inverse transformation. This is the absolute determinant of the Jacobian, $|\det J_{T^{-1}}(\boldsymbol{Y})|$ where the Jacobian is

$$J_{T^{-1}}(\boldsymbol{Y}) = \begin{pmatrix} \frac{\partial T_1^{-1}}{\partial y_1} & \cdots & \frac{\partial T_1^{-1}}{\partial y_K} \\ \vdots & & \vdots \\ \frac{\partial T_K^{-1}}{\partial y_1} & \cdots & \frac{\partial T_K^{-1}}{\partial y_K} \end{pmatrix}.$$

Intuitively, the Jacobian describes how a transformation warps unit volumes across spaces. This matters for transformations of random variables, since probability density functions must always integrate to one. If the transformation is linear, then we can drop the Jacobian adjustment; it evaluates to one. Similarly, affine transformations, like elliptical standardizations, also have Jacobians that evaluate to one; they preserve unit volumes.

B Transformation of the Evidence Lower Bound

Recall that $\boldsymbol{\zeta} = T(\boldsymbol{\theta})$ and that the variational approximation in the real coordinate space is $q(\boldsymbol{\zeta}; \boldsymbol{\phi})$.

We begin with the ELBO in the original latent variable space. We then transform the latent variable space to the real coordinate space.

$$\begin{aligned} \mathcal{L}(\boldsymbol{\phi}) &= \int q(\boldsymbol{\theta}) \log\left[\frac{p(\boldsymbol{x}, \boldsymbol{\theta})}{q(\boldsymbol{\theta})} \right] \mathrm{d}\boldsymbol{\theta} \\ &= \int q(\boldsymbol{\zeta}; \boldsymbol{\phi}) \log\left[\frac{p\left(\boldsymbol{x}, T^{-1}(\boldsymbol{\zeta})\right) \left| \det J_{T^{-1}}(\boldsymbol{\zeta}) \right|}{q(\boldsymbol{\zeta}; \boldsymbol{\phi})} \right] \mathrm{d}\boldsymbol{\zeta} \\ &= \int q(\boldsymbol{\zeta}; \boldsymbol{\phi}) \log\left[p\left(\boldsymbol{x}, T^{-1}(\boldsymbol{\zeta})\right) \left| \det J_{T^{-1}}(\boldsymbol{\zeta}) \right| \right] \mathrm{d}\boldsymbol{\zeta} - \int q(\boldsymbol{\zeta}; \boldsymbol{\phi}) \log\left[q(\boldsymbol{\zeta}; \boldsymbol{\phi}) \right] \mathrm{d}\boldsymbol{\zeta} \\ &= \mathbb{E}_{q(\boldsymbol{\zeta}; \boldsymbol{\phi})}\left[\log p\left(\boldsymbol{x}, T^{-1}(\boldsymbol{\zeta})\right) + \log\left| \det J_{T^{-1}}(\boldsymbol{\zeta}) \right| \right] - \mathbb{E}_{q(\boldsymbol{\zeta}; \boldsymbol{\phi})}\left[\log q(\boldsymbol{\zeta}; \boldsymbol{\phi}) \right] \\ &= \mathbb{E}_{q(\boldsymbol{\zeta}; \boldsymbol{\phi})}\left[\log p\left(\boldsymbol{x}, T^{-1}(\boldsymbol{\zeta})\right) + \log\left| \det J_{T^{-1}}(\boldsymbol{\zeta}) \right| \right] + \mathbb{H}\left[q(\boldsymbol{\zeta}; \boldsymbol{\phi}) \right]. \end{aligned}$$

C   Gradients of the Evidence Lower Bound

First, consider the gradient with respect to the $\boldsymbol{\mu}$ parameter. We exchange the order of the gradient and the integration through the dominated convergence theorem (Çınlar, 2011). The rest is the chain rule for differentiation.

$$
\begin{aligned}
\nabla_{\boldsymbol{\mu}}\mathscr{L} &= \nabla_{\boldsymbol{\mu}}\Big\{\mathbb{E}_{\mathcal{N}(\boldsymbol{\eta};\mathbf{0},I)}\Big[\log p\left(\boldsymbol{x},T^{-1}(S_{\boldsymbol{\phi}}^{-1}(\boldsymbol{\eta}))\right)+\log\big|\det J_{T^{-1}}\left(S_{\boldsymbol{\phi}}^{-1}(\boldsymbol{\eta})\right)\big|\Big]+\mathbb{H}[q(\boldsymbol{\zeta};\boldsymbol{\phi})]\Big\} \\
&= \mathbb{E}_{\mathcal{N}(\boldsymbol{\eta};\mathbf{0},I)}\Big[\nabla_{\boldsymbol{\mu}}\big\{\log p\left(\boldsymbol{x},T^{-1}(S^{-1}(\boldsymbol{\eta}))\right)+\log\big|\det J_{T^{-1}}\left(S^{-1}(\boldsymbol{\eta})\right)\big|\big\}\Big] \\
&= \mathbb{E}_{\mathcal{N}(\boldsymbol{\eta};\mathbf{0},I)}\Big[\left(\nabla_{\boldsymbol{\theta}}\log p(\boldsymbol{x},\boldsymbol{\theta})\nabla_{\boldsymbol{\zeta}}T^{-1}(\boldsymbol{\zeta})+\nabla_{\boldsymbol{\zeta}}\log\big|\det J_{T^{-1}}(\boldsymbol{\zeta})\big|\right)\nabla_{\boldsymbol{\mu}}S_{\boldsymbol{\phi}}^{-1}(\boldsymbol{\eta})\Big] \\
&= \mathbb{E}_{\mathcal{N}(\boldsymbol{\eta};\mathbf{0},I)}\Big[\nabla_{\boldsymbol{\theta}}\log p(\boldsymbol{x},\boldsymbol{\theta})\nabla_{\boldsymbol{\zeta}}T^{-1}(\boldsymbol{\zeta})+\nabla_{\boldsymbol{\zeta}}\log\big|\det J_{T^{-1}}(\boldsymbol{\zeta})\big|\Big]
\end{aligned}
$$

Then, consider the gradient with respect to the mean-field $\boldsymbol{\omega}$ parameter.

$$
\begin{aligned}
\nabla_{\boldsymbol{\omega}}\mathscr{L} &= \nabla_{\boldsymbol{\omega}}\Big\{\mathbb{E}_{\mathcal{N}(\boldsymbol{\eta};\mathbf{0},I)}\Big[\log p\left(\boldsymbol{x},T^{-1}(S_{\boldsymbol{\phi}}^{-1}(\boldsymbol{\eta}))\right)+\log\big|\det J_{T^{-1}}\left(S_{\boldsymbol{\phi}}^{-1}(\boldsymbol{\eta})\right)\big|\Big] \\
&\qquad\quad +\frac{K}{2}(1+\log(2\pi))+\sum_{k=1}^{K}\log(\exp(\omega_k))\Big\} \\
&= \mathbb{E}_{\mathcal{N}(\boldsymbol{\eta};\mathbf{0},I)}\Big[\nabla_{\boldsymbol{\omega}}\big\{\log p\left(\boldsymbol{x},T^{-1}(S_{\boldsymbol{\phi}}^{-1}(\boldsymbol{\eta}))\right)+\log\big|\det J_{T^{-1}}\left(S_{\boldsymbol{\phi}}^{-1}(\boldsymbol{\eta})\right)\big|\big\}\Big]+\mathbf{1} \\
&= \mathbb{E}_{\mathcal{N}(\boldsymbol{\eta};\mathbf{0},I)}\Big[\left(\nabla_{\boldsymbol{\theta}}\log p(\boldsymbol{x},\boldsymbol{\theta})\nabla_{\boldsymbol{\zeta}}T^{-1}(\boldsymbol{\zeta})+\nabla_{\boldsymbol{\zeta}}\log\big|\det J_{T^{-1}}(\boldsymbol{\zeta})\big|\right)\nabla_{\boldsymbol{\omega}}S_{\boldsymbol{\phi}}^{-1}(\boldsymbol{\eta}))\Big]+\mathbf{1} \\
&= \mathbb{E}_{\mathcal{N}(\boldsymbol{\eta};\mathbf{0},I)}\Big[\left(\nabla_{\boldsymbol{\theta}}\log p(\boldsymbol{x},\boldsymbol{\theta})\nabla_{\boldsymbol{\zeta}}T^{-1}(\boldsymbol{\zeta})+\nabla_{\boldsymbol{\zeta}}\log\big|\det J_{T^{-1}}(\boldsymbol{\zeta})\big|\right)\boldsymbol{\eta}^{\top}\mathrm{diag}(\exp(\boldsymbol{\omega}))\Big]+\mathbf{1}.
\end{aligned}
$$

Finally, consider the gradient with respect to the full-rank $\boldsymbol{L}$ parameter.

$$
\begin{aligned}
\nabla_{\boldsymbol{L}}\mathscr{L} &= \nabla_{\boldsymbol{L}}\Big\{\mathbb{E}_{\mathcal{N}(\boldsymbol{\eta};\mathbf{0},I)}\Big[\log p\left(\boldsymbol{x},T^{-1}(S_{\boldsymbol{\phi}}^{-1}(\boldsymbol{\eta}))\right)+\log\big|\det J_{T^{-1}}\left(S_{\boldsymbol{\phi}}^{-1}(\boldsymbol{\eta})\right)\big|\Big] \\
&\qquad\quad +\frac{K}{2}(1+\log(2\pi))+\frac{1}{2}\log\big|\det(\boldsymbol{L}\boldsymbol{L}^{\top})\big|\Big\} \\
&= \mathbb{E}_{\mathcal{N}(\boldsymbol{\eta};\mathbf{0},I)}\Big[\nabla_{\boldsymbol{L}}\big\{\log p\left(\boldsymbol{x},T^{-1}(S_{\boldsymbol{\phi}}^{-1}(\boldsymbol{\eta}))\right)+\log\big|\det J_{T^{-1}}\left(S_{\boldsymbol{\phi}}^{-1}(\boldsymbol{\eta})\right)\big|\big\}\Big] \\
&\qquad\quad +\nabla_{\boldsymbol{L}}\frac{1}{2}\log\big|\det(\boldsymbol{L}\boldsymbol{L}^{\top})\big| \\
&= \mathbb{E}_{\mathcal{N}(\boldsymbol{\eta};\mathbf{0},I)}\Big[\left(\nabla_{\boldsymbol{\theta}}\log p(\boldsymbol{x},\boldsymbol{\theta})\nabla_{\boldsymbol{\zeta}}T^{-1}(\boldsymbol{\zeta})+\nabla_{\boldsymbol{\zeta}}\log\big|\det J_{T^{-1}}(\boldsymbol{\zeta})\big|\right)\nabla_{\boldsymbol{L}}S_{\boldsymbol{\phi}}^{-1}(\boldsymbol{\eta}))\Big]+(\boldsymbol{L}^{-1})^{\top} \\
&= \mathbb{E}_{\mathcal{N}(\boldsymbol{\eta};\mathbf{0},I)}\Big[\left(\nabla_{\boldsymbol{\theta}}\log p(\boldsymbol{x},\boldsymbol{\theta})\nabla_{\boldsymbol{\zeta}}T^{-1}(\boldsymbol{\zeta})+\nabla_{\boldsymbol{\zeta}}\log\big|\det J_{T^{-1}}(\boldsymbol{\zeta})\big|\right)\boldsymbol{\eta}^{\top}\Big]+(\boldsymbol{L}^{-1})^{\top}
\end{aligned}
$$

D   Automating Expectations: Monte Carlo Integration

Expectations are integrals. We can use MC integration to approximate them (Robert and Casella, 1999). All we need are samples from q.

$$
\mathbb{E}_{q(\boldsymbol{\eta})}\big[f(\boldsymbol{\eta})\big]=\int f(\boldsymbol{\eta})q(\boldsymbol{\eta})\,\mathrm{d}\boldsymbol{\eta}\approx\frac{1}{S}\sum_{s=1}^{S}f(\boldsymbol{\eta}_s)\text{ where }\boldsymbol{\eta}_s\sim q(\boldsymbol{\eta}).
$$

MC integration provides noisy, yet unbiased, estimates of the integral. The standard deviation of the estimates are of order $1/\sqrt{S}$.

E   Running ADVI in Stan

Visit `http://mc-stan.org/` to download the latest version of Stan. Follow instructions on how to install Stan. You are then ready to use ADVI.

Stan offers multiple interfaces. We describe the command line interface (`cmdStan`) below, where my-

```
./myModel   variational
            grad_samples=M               ( M = 1 default )
            data file=myData.data.R
            output file=output_advi.csv
            diagnostic_file=elbo_advi.csv
```

Figure 16: Syntax for using ADVI via `cmdStan`.

`Data.data.R` is the dataset stored in the R language `Rdump` format. `output_advi.csv` contains samples from the posterior and `elbo_advi.csv` reports the ELBO.

F   Details of Studied Models

F.1   Linear Regression with Automatic Relevance Determination

Linear regression with ARD is a high-dimensional sparse regression model (Bishop, 2006; Drugowitsch, 2013). We describe the model below. Stan code is in Figure 17.

The inputs are $\boldsymbol{x} = x_{1:N}$ where each x_n is D-dimensional. The outputs are $\boldsymbol{y} = y_{1:N}$ where each y_n is 1-dimensional. The weights vector $\boldsymbol{w}$ is D-dimensional. The likelihood

$$p(\boldsymbol{y} \mid \boldsymbol{x}, \boldsymbol{w}, \sigma) = \prod_{n=1}^{N} \mathcal{N}\left(y_n \mid \boldsymbol{w}^{\top} \boldsymbol{x}_n, \sigma\right)$$

describes measurements corrupted by iid Gaussian noise with unknown standard deviation σ.

The ARD prior and hyper-prior structure is as follows

$$p(\boldsymbol{w}, \sigma, \boldsymbol{\alpha}) = p(\boldsymbol{w}, \sigma \mid \boldsymbol{\alpha}) p(\boldsymbol{\alpha})$$

$$= \mathcal{N}\left(\boldsymbol{w} \mid 0, \sigma \left(\operatorname{diag}\sqrt{\boldsymbol{\alpha}}\right)^{-1}\right) \operatorname{InvGam}(\sigma \mid a_0, b_0) \prod_{i=1}^{D} \operatorname{Gam}(\alpha_i \mid c_0, d_0)$$

where $\boldsymbol{\alpha}$ is a D-dimensional hyper-prior on the weights, where each component gets its own independent Gamma prior.

We simulate data such that only half the regressions have predictive power. The results in Figure 10 use $a_0 = b_0 = c_0 = d_0 = 1$ as hyper-parameters for the Gamma priors.

F.2   Hierarchical Logistic Regression

Hierarchical logistic regression models structured datasets in an intuitive way. We study a model of voting preferences from the 1988 United States presidential election. Chapter 14.1 of (Gelman and Hill, 2006) motivates the model and explains the dataset. We also describe the model below. Stan code is in

Figure 18, based on (Stan Development Team, 2015).

$$\Pr(y_n = 1) = \text{sigmoid}\left(\beta^0 + \beta^{\text{female}} \cdot \text{female}_n + \beta^{\text{black}} \cdot \text{black}_n + \beta^{\text{female.black}} \cdot \text{female.black}_n \right.$$
$$\left. + \alpha_{k[n]}^{\text{age}} + \alpha_{l[n]}^{\text{edu}} + \alpha_{k[n],l[n]}^{\text{age.edu}} + \alpha_{j[n]}^{\text{state}} \right)$$
$$\alpha_j^{\text{state}} \sim \mathcal{N}\left(\alpha_{m[j]}^{\text{region}} + \beta^{\text{v.prev}} \cdot \text{v.prev}_j , \sigma_{\text{state}} \right).$$

The hierarchical variables are

$$\alpha_k^{\text{age}} \sim \mathcal{N}\left(0, \sigma_{\text{age}} \right) \text{ for } k = 1, \ldots, K$$
$$\alpha_l^{\text{edu}} \sim \mathcal{N}\left(0, \sigma_{\text{edu}} \right) \text{ for } l = 1, \ldots, L$$
$$\alpha_{k,l}^{\text{age.edu}} \sim \mathcal{N}\left(0, \sigma_{\text{age.edu}} \right) \text{ for } k = 1, \ldots, K, l = 1, \ldots, L$$
$$\alpha_m^{\text{region}} \sim \mathcal{N}\left(0, \sigma_{\text{region}} \right) \text{ for } m = 1, \ldots, M.$$

The standard deviation terms all have uniform hyper-priors, constrained between 0 and 100.

F.3 Non-negative Matrix Factorization: Constrained Gamma Poisson Model

The Gamma Poisson factorization model describes discrete data matrices (Canny, 2004; Cemgil, 2009).

Consider a $U \times I$ matrix of observations. We find it helpful to think of $u = \{1, \cdots, U\}$ as users and $i = \{1, \cdots, I\}$ as items, as in a recommendation system setting. The generative process for a Gamma Poisson model with K factors is

1. For each user u in $\{1, \cdots, U\}$:
   - For each component k, draw $\theta_{uk} \sim \text{Gam}(a_0, b_0)$.
2. For each item i in $\{1, \cdots, I\}$:
   - For each component k, draw $\beta_{ik} \sim \text{Gam}(c_0, d_0)$.
3. For each user and item:
   - Draw the observation $y_{ui} \sim \text{Poisson}(\boldsymbol{\theta}_u^\top \boldsymbol{\beta}_i)$.

A potential downfall of this model is that it is not uniquely identifiable: swapping rows and columns of $\boldsymbol{\theta}$ and $\boldsymbol{\beta}$ give the same inner product. One way to contend with this is to constrain either vector to be an ordered vector during inference. We constrain each $\boldsymbol{\theta}_u$ vector in our model in this fashion. Stan code is in Figure 19. We set $K = 10$ and all the Gamma hyper-parameters to 1 in our experiments.

F.4 Non-negative Matrix Factorization: Dirichlet Exponential Poisson Model

Another model for discrete data is a Dirichlet Exponential model. The Dirichlet enforces uniqueness while the exponential promotes sparsity. This is a non-conjugate model that does not appear to have been studied in the literature.

The generative process for a Dirichlet Exponential model with K factors is

1. For each user u in $\{1, \cdots, U\}$:
   - Draw the K-vector $\boldsymbol{\theta}_u \sim \text{Dir}(\boldsymbol{\alpha}_0)$.
2. For each item i in $\{1, \cdots, I\}$:

- For each component k, draw $\beta_{ik} \sim \text{Exponential}(\lambda_0)$.

3. For each user and item:

   - Draw the observation $y_{ui} \sim \text{Poisson}(\boldsymbol{\theta}_u^\top \boldsymbol{\beta}_i)$.

Stan code is in Figure 20. We set $K = 10$, $\alpha_0 = 1000$ for each component, and $\lambda_0 = 0.1$. With this configuration of hyper-parameters, the factors $\boldsymbol{\beta}_i$ appear sparse.

F.5 Gaussian Mixture Model

The Gaussian mixture model (GMM) is a celebrated probability model (Bishop, 2006). We use it to group a dataset of natural images based on their color histograms. We build a high-dimensional GMM with a Gaussian prior for the mixture means, a lognormal prior for the mixture standard deviations, and a Dirichlet prior for the mixture components.

Represent the images as $\boldsymbol{y} = y_{1:N}$ where each y_n is D-dimensional and there are N observations. The likelihood for the images is

$$p(\boldsymbol{y} \mid \boldsymbol{\theta}, \boldsymbol{\mu}, \boldsymbol{\sigma}) = \prod_{n=1}^{N} \sum_{k=1}^{K} \theta_k \prod_{d=1}^{D} \mathcal{N}(y_{nd} \mid \mu_{kd}, \sigma_{kd})$$

with a Dirichlet prior for the mixture proportions

$$p(\boldsymbol{\theta}) = \text{Dir}(\boldsymbol{\theta} ; \boldsymbol{\alpha}_0),$$

a Gaussian prior for the mixture means

$$p(\boldsymbol{\mu}) = \prod_{k=1}^{D} \prod_{d=1}^{D} \mathcal{N}(\mu_{kd} ; 0, 1)$$

and a lognormal prior for the mixture standard deviations

$$p(\boldsymbol{\sigma}) = \prod_{k=1}^{D} \prod_{d=1}^{D} \text{logNormal}(\sigma_{kd} ; 0, 1)$$

The dimension of the color histograms in the imageCLEF dataset is $D = 576$. This is a concatenation of three 192-length histograms, one for each color channel (red, green, blue) of the images.

We scale the image histograms to have zero mean and unit variance. Setting α_0 to a small value encourages the model to use fewer components to explain the data. Larger values of α_0 encourage the model to use all K components. We set $\alpha_0 = 1000$ in our experiments.

ADVI code is in Figure 21. The stochastic data subsampling version of the code is in Figure 22.

F.6 Probabilistic Principal Component Analysis with Automatic Relevance Determination

Probabilistic principal component analysis (PPCA) is a Bayesian extension of classical principal component analysis (Bishop, 2006). The generative process is straightforward. Consider a dataset of $\boldsymbol{x} = x_{1:N}$ where each x_n is D-dimensional. Let M be the dimension of the subspace we seek.

First define a set of latent variables $\boldsymbol{z} = z_{1:N}$ where each z_n is M-dimensional. Draw each z_n from a standard normal

$$p(\boldsymbol{z}) = \prod_{n=1}^{N} \mathcal{N}(z_n ; \boldsymbol{0}, \boldsymbol{I}).$$

Then define a set of principal components $\boldsymbol{w} = w_{1:D}$ where each w_d is M-dimensional. Similarly, draw the principal components from a standard normal

$$p(\boldsymbol{w}) = \prod_{d=1}^{D} \mathcal{N}(w_d \,;\, \boldsymbol{0}, \boldsymbol{I}).$$

Finally define the likelihood through an inner product as

$$p(\boldsymbol{x} \mid \boldsymbol{w}, \boldsymbol{z}, \sigma) = \prod_{n=1}^{N} \mathcal{N}(x_n \,;\, \boldsymbol{w} z_n, \sigma \boldsymbol{I}).$$

The standard deviation σ is also a latent variable. Place a lognormal prior on it as

$$p(\sigma) = \mathrm{logNormal}(\sigma \,;\, 0, 1).$$

We extend PPCA by adding an ARD hierarchical prior. The extended model introduces a M-dimensional vector $\boldsymbol{\alpha}$ which chooses which principal components to retain. ($M < D$ now represents the maximum number of principal components to consider.) The extended extends the above by

$$p(\boldsymbol{\alpha}) = \prod_{m=1}^{M} \mathrm{InvGamma}(\alpha_m \,;\, 1, 1)$$

$$p(\boldsymbol{w} \mid \boldsymbol{\alpha}) = \prod_{d=1}^{D} \mathcal{N}(w_d \,;\, \boldsymbol{0}, \sigma \mathrm{diag}(\boldsymbol{\alpha}))$$

$$p(\boldsymbol{x} \mid \boldsymbol{w}, \boldsymbol{z}, \sigma) = \prod_{n=1}^{N} \mathcal{N}(x_n \,;\, \boldsymbol{w} z_n, \sigma \boldsymbol{I}).$$

ADVI code is in Figure 23.

F.7 Supervised Probabilistic Principal Component Analysis with Automatic Relevance Determination

Supervised probabilistic principal component analysis (SUP-PPCA) augments PPCA by regressing a vector of observed random variables y onto the principal component subspace. The idea is to not only find a set of principal components that describe variation in the dataset $\boldsymbol{x}$, but to also predict y. The complete model is

$$p(\boldsymbol{z}) = \prod_{n=1}^{N} \mathcal{N}(z_n \,;\, \boldsymbol{0}, \boldsymbol{I})$$

$$p(\sigma) = \mathrm{logNormal}(\sigma \,;\, 0, 1)$$

$$p(\boldsymbol{\alpha}) = \prod_{m=1}^{M} \mathrm{InvGamma}(\alpha_m \,;\, 1, 1)$$

$$p(\boldsymbol{w_x} \mid \boldsymbol{\alpha}) = \prod_{d=1}^{D} \mathcal{N}(w_d \,;\, \boldsymbol{0}, \sigma \mathrm{diag}(\boldsymbol{\alpha}))$$

$$p(\boldsymbol{w_y} \mid \boldsymbol{\alpha}) = \mathcal{N}(w_y \,;\, \boldsymbol{0}, \sigma \mathrm{diag}(\boldsymbol{\alpha}))$$

$$p(\boldsymbol{x} \mid \boldsymbol{w_x}, \boldsymbol{z}, \sigma) = \prod_{n=1}^{N} \mathcal{N}(x_n \,;\, \boldsymbol{w_x} z_n, \sigma \boldsymbol{I})$$

$$p(y \mid \boldsymbol{w_y}, \boldsymbol{z}, \sigma) = \prod_{n=1}^{N} \mathcal{N}(y_n \,;\, w_y z_n, \sigma).$$

ADVI code is in Figure 24.

```
data {
  int<lower=0> N;    // number of data items
  int<lower=0> D;    // dimension of input features
  matrix[N,D]  x;    // input matrix
  vector[N]    y;    // output vector

  // hyperparameters for Gamma priors
  real<lower=0> a0;
  real<lower=0> b0;
  real<lower=0> c0;
  real<lower=0> d0;
}

parameters {
  vector[D] w;                 // weights (coefficients) vector
  real<lower=0> sigma;         // standard deviation
  vector<lower=0>[D] alpha;    // hierarchical latent variables
}

transformed parameters {
  vector[D] one_over_sqrt_alpha;
  for (i in 1:D) {
    one_over_sqrt_alpha[i] <- 1 / sqrt(alpha[i]);
  }
}

model {
  // alpha: hyper-prior on weights
  alpha ~ gamma(c0, d0);

  // sigma: prior on standard deviation
  sigma ~ inv_gamma(a0, b0);

  // w: prior on weights
  w ~ normal(0, sigma * one_over_sqrt_alpha);

  // y: likelihood
  y ~ normal(x * w, sigma);
}
```

Figure 17: Stan code for Linear Regression with Automatic Relevance Determination.

```stan
data {
  int<lower=0> N;
  int<lower=0> n_age;
  int<lower=0> n_age_edu;
  int<lower=0> n_edu;
  int<lower=0> n_region_full;
  int<lower=0> n_state;
  int<lower=0,upper=n_age> age[N];
  int<lower=0,upper=n_age_edu> age_edu[N];
  vector<lower=0,upper=1>[N] black;
  int<lower=0,upper=n_edu> edu[N];
  vector<lower=0,upper=1>[N] female;
  int<lower=0,upper=n_region_full> region_full[N];
  int<lower=0,upper=n_state> state[N];
  vector[N] v_prev_full;
  int<lower=0,upper=1> y[N];
}
parameters {
  vector[n_age] a;
  vector[n_edu] b;
  vector[n_age_edu] c;
  vector[n_state] d;
  vector[n_region_full] e;
  vector[5] beta;
  real<lower=0,upper=100> sigma_a;
  real<lower=0,upper=100> sigma_b;
  real<lower=0,upper=100> sigma_c;
  real<lower=0,upper=100> sigma_d;
  real<lower=0,upper=100> sigma_e;
}
transformed parameters {
  vector[N] y_hat;

  for (i in 1:N)
    y_hat[i] <- beta[1]
                + beta[2] * black[i]
                + beta[3] * female[i]
                + beta[5] * female[i] * black[i]
                + beta[4] * v_prev_full[i]
                + a[age[i]]
                + b[edu[i]]
                + c[age_edu[i]]
                + d[state[i]]
                + e[region_full[i]];
}
model {
  a ~ normal (0, sigma_a);
  b ~ normal (0, sigma_b);
  c ~ normal (0, sigma_c);
  d ~ normal (0, sigma_d);
  e ~ normal (0, sigma_e);
  beta ~ normal(0, 100);
  y ~ bernoulli_logit(y_hat);
}
```

Figure 18: Stan code for Hierarchical Logistic Regression, from (Stan Development Team, 2015).

```
data {
  int<lower=0> U;
  int<lower=0> I;
  int<lower=0> K;
  int<lower=0> y[U,I];
  real<lower=0> a;
  real<lower=0> b;
  real<lower=0> c;
  real<lower=0> d;
}

parameters {
  positive_ordered[K] theta[U]; // user preference
  vector<lower=0>[K] beta[I];   // item attributes
}

model {
  for (u in 1:U)
    theta[u] ~ gamma(a, b); // componentwise gamma
  for (i in 1:I)
    beta[i] ~ gamma(c, d);  // componentwise gamma

  for (u in 1:U) {
    for (i in 1:I) {
      y[u,i] ~ poisson(theta[u]'*beta[i]);
    }
  }
}
```

Figure 19: Stan code for the Gamma Poisson non-negative matrix factorization model.

```
data {
  int<lower=0> U;
  int<lower=0> I;
  int<lower=0> K;
  int<lower=0> y[U,I];
  real<lower=0> lambda0;
  real<lower=0> alpha0;
}

transformed data {
  vector<lower=0>[K] alpha0_vec;
  for (k in 1:K) {
    alpha0_vec[k] <- alpha0;
  }
}

parameters {
  simplex[K] theta[U];            // user preference
  vector<lower=0>[K] beta[I];   // item attributes
}

model {
  for (u in 1:U)
    theta[u] ~ dirichlet(alpha0_vec); // componentwise dirichlet
  for (i in 1:I)
    beta[i] ~ exponential(lambda0);   // componentwise
     exponential

  for (u in 1:U) {
    for (i in 1:I) {
      y[u,i] ~ poisson(theta[u]'*beta[i]);
    }
  }
}
```

Figure 20: Stan code for the Dirichlet Exponential non-negative matrix factorization model.

```stan
data {
  int<lower=0> N;        // number of data points in entire dataset
  int<lower=0> K;        // number of mixture components
  int<lower=0> D;        // dimension
  vector[D] y[N];        // observations

  real<lower=0> alpha0;  // dirichlet prior
}

transformed data {
  vector<lower=0>[K] alpha0_vec;
  for (k in 1:K)
    alpha0_vec[k] <- alpha0;
}

parameters {
  simplex[K] theta;                 // mixing proportions
  vector[D] mu[K];                  // locations of mixture components
  vector<lower=0>[D] sigma[K];      // standard deviations of mixture components
}

model {
  // priors
  theta ~ dirichlet(alpha0_vec);
  for (k in 1:K) {
      mu[k] ~ normal(0.0, 1.0);
      sigma[k] ~ lognormal(0.0, 1.0);
  }

  // likelihood
  for (n in 1:N) {
    real ps[K];
    for (k in 1:K) {
      ps[k] <- log(theta[k]) + normal_log(y[n], mu[k], sigma[k]);
    }
    increment_log_prob(log_sum_exp(ps));
  }
}
```

Figure 21: ADVI Stan code for the GMM example.

```
functions {
  real divide_promote_real(int x, int y) {
    real x_real;
    x_real <- x;
    return x_real / y;
  }
}

data {
  int<lower=0> NFULL;      // total number of datapoints in dataset
  int<lower=0> N;          // number of data points in minibatch

  int<lower=0> K;          // number of mixture components
  int<lower=0> D;          // dimension

  vector[D] yFULL[NFULL];  // dataset
  vector[D] y[N];          // minibatch

  real<lower=0> alpha0;    // dirichlet hyper-prior parameter
}

transformed data {
  real minibatch_factor;
  vector<lower=0>[K] alpha0_vec;
  for (k in 1:K) {
    alpha0_vec[k] <- alpha0 / K;
  }
  minibatch_factor <- divide_promote_real(N, NFULL);
}

parameters {
  simplex[K] theta;                 // mixing proportions
  vector[D] mu[K];                  // locations of mixture components
  vector<lower=0>[D] sigma[K];      // standard deviations of mixture components
}

model {
  // priors
  theta ~ dirichlet(alpha0_vec);
  for (k in 1:K) {
      mu[k] ~ normal(0.0, 1.0);
      sigma[k] ~ lognormal(0.0, 1.0);
  }

  // likelihood
  for (n in 1:N) {
    real ps[K];
    for (k in 1:K) {
      ps[k] <- log(theta[k]) + normal_log(y[n], mu[k], sigma[k]);
    }
    increment_log_prob(log_sum_exp(ps));
  }
  increment_log_prob(log(minibatch_factor));
}
```

Figure 22: ADVI Stan code for the GMM example, with stochastic subsampling of the dataset.

```stan
data {
  int<lower=0> N;          // number of data points in dataset
  int<lower=0> D;          // dimension
  int<lower=0> M;          // maximum dimension of latent space to consider

  vector[D] x[N];          // data
}

parameters {
  // latent variable
  matrix[M,N] z;

  // weights parameters
  matrix[D,M] w;

  // variance parameter
  real<lower=0> sigma;

  // hyper-parameters on weights
  vector<lower=0>[M] alpha;
}

model {
  // priors
  to_vector(z) ~ normal(0,1);
  for (d in 1:D)
    w[d] ~ normal(0, sigma * alpha);
  sigma ~ lognormal(0,1);
  alpha ~ inv_gamma(1,1);

  // likelihood
  for (n in 1:N)
    x[n] ~ normal(w * col(z, n), sigma);
}
```

Figure 23: ADVI Stan code for the PPCA with ARD.

```
data {
  int<lower=0> N;          // number of data points in dataset
  int<lower=0> D;          // dimension
  int<lower=0> M;          // maximum dimension of latent space to cinsider

  vector[D] x[N];
  vector[N] y;
}

parameters {
  // latent variable
  matrix[M,N] z;

  // weights parameters
  matrix[D,M] w_x;
  vector[M]   w_y;

  // variance parameter
  real<lower=0> sigma;

  // hyper-parameters on weights
  vector<lower=0>[M] alpha;
}

model {
  // priors
  to_vector(z) ~ normal(0,1);
  for (d in 1:D)
    w_x[d] ~ normal(0, sigma * alpha);
  w_y ~ normal(0, sigma * alpha);

  sigma ~ lognormal(0,1);
  alpha ~ inv_gamma(1,1);

  // likelihood
  for (n in 1:N) {
    x[n] ~ normal(w_x * col(z, n), sigma);
    y[n] ~ normal(w_y' * col(z, n), sigma);
  }
}
```

Figure 24: ADVI Stan code for the SUP-PPCA with ARD.

References

Amari, S.-I. (1998). Natural gradient works efficiently in learning. *Neural Computation*, 10(2):251–276.

Barber, D. (2012). *Bayesian Reasoning and Machine Learning*. Cambridge University Press.

Baydin, A. G., Pearlmutter, B. A., and Radul, A. A. (2015). Automatic differentiation in machine learning: a survey. *arXiv preprint arXiv:1502.05767*.

Bernardo, J. M. and Smith, A. F. (2009). *Bayesian Theory*, volume 405. John Wiley & Sons.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer New York.

Bishop, C. M., Spiegelhalter, D., and Winn, J. (2002). VIBES: a variational inference engine for Bayesian networks. In *Neural Information Processing Systems*, pages 777–784.

Blei, D. M. (2014). Build, compute, critique, repeat: data analysis with latent variable models. *Annual Review of Statistics and Its Application*, 1:203–232.

Blei, D. M., Jordan, M. I., et al. (2006). Variational inference for Dirichlet process mixtures. *Bayesian analysis*, 1(1):121–143.

Blei, D. M., Kucukelbir, A., and McAuliffe, J. D. (2016). Variational inference: a review for statisticians. *arXiv preprint arXiv:1601.00670*.

Bottou, L. (2012). Stochastic gradient descent tricks. In *Neural Networks: Tricks of the Trade*, pages 421–436. Springer.

Canny, J. (2004). GaP: a factor model for discrete data. In *SIGIR Conference on Research and Development in Information Retrieval*, pages 122–129. ACM.

Carpenter, B., Hoffman, M. D., Brubaker, M., Lee, D., Li, P., and Betancourt, M. (2015). The Stan math library: reverse-mode automatic differentiation in C++. *arXiv preprint arXiv:1509.07164*.

Cemgil, A. T. (2009). Bayesian inference for nonnegative matrix factorisation models. *Computational Intelligence and Neuroscience*, 2009.

Challis, E. and Barber, D. (2012). Affine independent variational inference. In *Neural Information Processing Systems*, pages 2186–2194.

Challis, E. and Barber, D. (2013). Gaussian Kullback-Leibler approximate inference. *The Journal of Machine Learning Research*, 14(1):2239–2286.

Çınlar, E. (2011). *Probability and Stochastics*. Springer.

Diaconis, P., Ylvisaker, D., et al. (1979). Conjugate priors for exponential families. *The Annals of Statistics*, 7(2):269–281.

Dinh, L., Krueger, D., and Bengio, Y. (2014). NICE: non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*.

Drugowitsch, J. (2013). Variational Bayesian inference for linear and logistic regression. *arXiv preprint arXiv:1310.5438*.

Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, 12:2121–2159.

Fan, K., Wang, Z., Beck, J., Kwok, J., and Heller, K. A. (2015). Fast second order stochastic backpropagation for variational inference. In *Neural Information Processing Systems*, pages 1387–1395.

Gelman, A., Carlin, J. B., Stern, H. S., Dunson, D. B., Vehtari, A., and Rubin, D. B. (2013). *Bayesian Data Analysis*. CRC Press.

Gelman, A. and Hill, J. (2006). *Data Analysis using Regression and Multilevel/Hierarchical Models*. Cambridge University Press.

Girolami, M. and Calderhead, B. (2011). Riemann manifold Langevin and Hamiltonian Monte Carlo methods. *Journal of the Royal Statistical Society: Series B*, 73(2):123–214.

Goodman, N. D., Mansinghka, V. K., Roy, D., Bonawitz, K., and Tenenbaum, J. B. (2008). Church: a language for generative models. In *UAI*, pages 220–229.

Härdle, W. and Simar, L. (2012). *Applied Multivariate Statistical Analysis*. Springer.

Hoffman, M. D., Blei, D. M., Wang, C., and Paisley, J. (2013). Stochastic variational inference. *The Journal of Machine Learning Research*, 14(1):1303–1347.

Hoffman, M. D. and Gelman, A. (2014). The No-U-Turn sampler. *The Journal of Machine Learning Research*, 15(1):1593–1623.

Jordan, M. I., Ghahramani, Z., Jaakkola, T. S., and Saul, L. K. (1999). An introduction to variational methods for graphical models. *Machine Learning*, 37(2):183–233.

Khan, M. E., Baqué, P., Fleuret, F., and Fua, P. (2015). Kullback-Leibler proximal variational inference. In *Neural Information Processing Systems*, pages 3384–3392.

Kim, S., Shephard, N., and Chib, S. (1998). Stochastic volatility: likelihood inference and comparison with ARCH models. *The Review of Economic Studies*, 65(3):361–393.

Kingma, D. and Welling, M. (2014). Auto-encoding variational Bayes. In *International Conference on Learning Representations*.

Kingma, D. P. and Adam, J. B. (2015). A method for stochastic optimization. In *International Conference on Learning Representation*.

Knowles, D. A. (2015). Stochastic gradient variational Bayes for Gamma approximating distributions. *arXiv preprint arXiv:1509.01631*.

Kucukelbir, A., Ranganath, R., Gelman, A., and Blei, D. (2015). Automatic variational inference in Stan. In *Neural Information Processing Systems*, pages 568–576.

Lee, D. and Seung, H. (1999). Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755):788–791.

Mahsereci, M. and Hennig, P. (2015). Probabilistic line searches for stochastic optimization. In *Neural Information Processing Systems*, pages 181–189.

Mansinghka, V., Selsam, D., and Perov, Y. (2014). Venture: a higher-order probabilistic programming platform with programmable inference. *arXiv preprint arXiv:1404.0099*.

Minka, T., Winn, J., Guiver, J., Webster, S., Zaykov, Y., Yangel, B., Spengler, A., and Bronskill, J. (2014). Infer.NET 2.6. Microsoft Research Cambridge. http://research.microsoft.com/infernet.

Murphy, K. P. (2012). *Machine Learning: a Probabilistic Perspective*. MIT Press.

Olive, D. J. (2014). *Statistical Theory and Inference*. Springer.

Opper, M. and Archambeau, C. (2009). The variational Gaussian approximation revisited. *Neural Computation*, 21(3):786–792.

Pfeffer, A. (2009). Figaro: an object-oriented probabilistic programming language. *Charles River Analytics Technical Report*, 137.

Pinheiro, J. C. and Bates, D. M. (1996). Unconstrained parametrizations for variance-covariance matrices. *Statistics and Computing*, 6(3):289–296.

Plummer, M. et al. (2003). JAGS: a program for analysis of Bayesian graphical models using Gibbs sampling. In *International Workshop on Distributed Statistical Computing*, volume 124, page 125.

Ranganath, R., Gerrish, S., and Blei, D. M. (2014). Black box variational inference. In *Artificial Intelligence and Statistics*.

Ranganath, R., Tran, D., and Blei, D. M. (2015). Hierarchical variational models. *arXiv preprint arXiv:1511.02386*.

Ranganath, R., Wang, C., David, B., and Xing, E. (2013). An adaptive learning rate for stochastic variational inference. In *International Conference on Machine Learning*, pages 298–306.

Rezende, D. and Mohamed, S. (2015). Variational inference with normalizing flows. In *International Conference on Machine Learning*, pages 1530–1538.

Rezende, D. J., Mohamed, S., and Wierstra, D. (2014). Stochastic backpropagation and approximate inference in deep generative models. In *International Conference on Machine Learning*, pages 1278–1286.

Robbins, H. and Monro, S. (1951). A stochastic approximation method. *The Annals of Mathematical Statistics*.

Robert, C. P. and Casella, G. (1999). *Monte Carlo Statistical Methods*. Springer.

Salimans, T. and Knowles, D. (2014). On using control variates with stochastic approximation for variational Bayes. *arXiv preprint arXiv:1401.1022*.

Seeger, M. (2010). Gaussian covariance and scalable variational inference. In *International Conference on Machine Learning*.

Spiegelhalter, D. J., Thomas, A., Best, N. G., and Gilks, W. R. (1995). BUGS: Bayesian inference using Gibbs sampling, version 0.50. *MRC Biostatistics Unit, Cambridge*.

Stan Development Team (2015). *Stan Modeling Language Users Guide and Reference Manual*.

Tieleman, T. and Hinton, G. (2012). Lecture 6.5-rmsprop: divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*, 4.

Titsias, M. and Lázaro-Gredilla, M. (2014). Doubly stochastic variational Bayes for non-conjugate inference. In *International Conference on Machine Learning*, pages 1971–1979.

Tran, D., Ranganath, R., and Blei, D. M. (2016). Variational Gaussian process. In *International Conference on Learning Representations*.

Villegas, M., Paredes, R., and Thomee, B. (2013). Overview of the ImageCLEF 2013 Scalable Concept Image Annotation Subtask. In *CLEF Evaluation Labs and Workshop*.

Wainwright, M. J. and Jordan, M. I. (2008). Graphical models, exponential families, and variational inference. *Foundations and Trends in Machine Learning*, 1(1-2):1–305.

Wingate, D. and Weber, T. (2013). Automated variational inference in probabilistic programming. *arXiv preprint arXiv:1301.1299*.

Winn, J. M. and Bishop, C. M. (2005). Variational message passing. In *Journal of Machine Learning Research*, pages 661–694.

Wood, F., van de Meent, J. W., and Mansinghka, V. (2014). A new approach to probabilistic programming inference. In *Artificial Intelligence and Statistics*, pages 2–46.