Journal of Machine Learning Research 18 (2017) 1-45 Submitted 3/16; Revised 8/16; Published 1/17

Automatic Differentiation Variational Inference

Alp Kucukelbir ALP@CS.COLUMBIA.EDU
Data Science Institute, Department of Computer Science
Columbia University
New York, NY 10027, USA

Dustin Tran DUSTIN@CS.COLUMBIA.EDU
Department of Computer Science
Columbia University
New York, NY 10027, USA

Rajesh Ranganath RAJESHR@CS.PRINCETON.EDU
Department of Computer Science
Princeton University
Princeton, NJ 08540, USA

Andrew Gelman GELMAN@STAT.COLUMBIA.EDU
Data Science Institute, Departments of Political Science and Statistics
Columbia University
New York, NY 10027, USA

David M. Blei DAVID.BLEI@COLUMBIA.EDU
Data Science Institute, Departments of Computer Science and Statistics
Columbia University
New York, NY 10027, USA

Editor: David Barber

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 modern probabilistic models and apply it to a dataset with millions of observations. We deploy ADVI as part of Stan, a probabilistic programming system.

Keywords: Bayesian inference, approximate inference, probabilistic programming

1. Introduction

We develop an automatic method that derives variational inference algorithms for complex probabilistic models. We implement our method in 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 automatic inference with large datasets on a practical class of modern probabilistic models.

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.

Figure 14 presents 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. With our method, we write a probabilistic program and compile it; we can then fit the model in minutes and analyze the results with ease.

Probabilistic modeling leads to a natural research cycle. First, we use our domain knowledge to posit a simple model that includes latent variables; then, we use an inference algorithm to infer those variables from our data; next, we analyze our results and identify where the model works and where it falls short; last, we refine the model and repeat 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. The efforts involved in deriving, programming, debugging, and scaling each model precludes us from taking full advantage of the probabilistic modeling cycle.

It is this problem that motivates the 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 can be too slow for many real-world applications.

We approach the problem through variational inference (Jordan et al., 1999; Wainwright and Jordan, 2008), a faster alternative to MCMC that has been used in many large-scale problems (Hoffman et al., 2013; 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 variational inference to leverage some of the capabilities 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.[1,2] We implement and deploy ADVI as part of Stan, a probabilistic programming system (Stan Development Team, 2016).

ADVI resolves the computational bottleneck of the probabilistic modeling cycle. We 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 direct applications of ADVI to modern probabilistic modeling examples, including an iterative analysis of 1.7 million taxi trajectories.

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

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 problem. 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 $p(\mathbf{x}, \boldsymbol{\theta})$ to $p(\mathbf{x}, \boldsymbol{\zeta})$, 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 $\mathrm{KL}\left(q(\boldsymbol{\zeta}) \parallel p(\boldsymbol{\zeta} \mid \mathbf{x})\right)$. 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 $\nabla_{\boldsymbol{\theta}} \log p(\mathbf{x}, \boldsymbol{\theta})$. Expressing the gradient as an expectation opens the door to Monte Carlo methods for approximating it (Robert and Casella, 1999; Ranganath et al., 2014).

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.

1. This paper extends the method presented in Kucukelbir et al. (2015).
2. Automatic differentation has enjoyed a recent resurgence in machine learning; see Section 2.7.

Each step above is carefully designed to make ADVI work "out of the box" for a practical class of modern probabilistic models. This focus on developing an automated inference algorithm differentiates ADVI from other "black box" variational methods (Ranganath et al., 2014; Ruiz et al., 2016b).

We deploy ADVI in the Stan probabilistic programming 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 $\nabla_{\boldsymbol{\theta}} \log p(\mathbf{x}, \boldsymbol{\theta})$ (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

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 $\mathbf{x} = x_{1:N}$ with N observations. Each x_n is a realization of a discrete or continuous (multivariate) random variable. The likelihood $p(\mathbf{x} \mid \boldsymbol{\theta})$ 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 $p(\mathbf{x}, \boldsymbol{\theta}) = p(\mathbf{x} \mid \boldsymbol{\theta})\, p(\boldsymbol{\theta})$. The goal of inference is to compute the posterior density $p(\boldsymbol{\theta} \mid \mathbf{x})$, which describes how the latent variables vary, conditioned on data.

Many posterior densities are not tractable; 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 $\nabla_{\boldsymbol{\theta}} \log p(\mathbf{x}, \boldsymbol{\theta})$. 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 and Ylvisaker, 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 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 differentiable random variables may seem to leave out other common machine learning models, such as mixture models and topic models. However, marginalizing out the discrete variables in the likelihoods of these models renders them differentiable.

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)*
Gaussian process models	*(e.g., regression / classification)*

Table 1: Popular differentiable probability models in machine learning.

Marginalization is not tractable for all models, such as the Ising model, sigmoid belief networks, and (untruncated) Bayesian nonparametric models, such as Dirichlet process mixtures (Antoniak, 1974). These are not differentiable probability models.

2.2 Variational Inference

Variational inference (VI) 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 $\boldsymbol{\phi} \in \boldsymbol{\Phi}$. VI finds the parameters that minimize the KL divergence to the posterior,

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

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)

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

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 $\log p(\mathbf{x})$. 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 $p(\boldsymbol{\theta} \mid \mathbf{x})$.[3] With this constraint made explicit, the optimization

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

problem from Equation (1) becomes

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

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, $\text{supp}(p(\boldsymbol{\theta} \mid \mathbf{x})) = \text{supp}(p(\boldsymbol{\theta}))$. 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 $\boldsymbol{\Phi}$. 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 a user to define any differentiable probability model for which 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 transform the joint density, we can choose the variational approximation independent of the model.

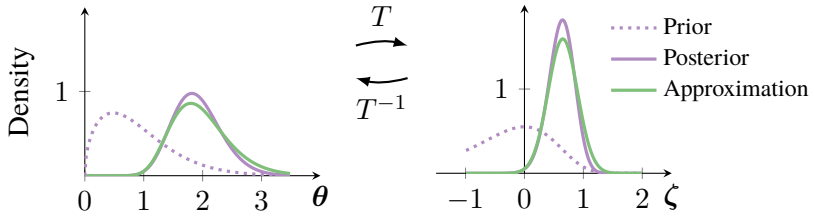
Define a one-to-one differentiable function

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

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

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

where $p(\mathbf{x}, \boldsymbol{\theta} = T^{-1}(\boldsymbol{\zeta}))$ is the joint density in the original latent variable space, and $J_{T^{-1}}(\boldsymbol{\zeta})$ 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.)



(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.

Consider again our 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

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

Figures 1a and 1b depict this transformation.

As we describe in the introduction, we implement our algorithm in Stan (Stan Development Team, 2016). Stan maintains a library of transformations and their corresponding Jacobians. Specifically, it provides various transformations for upper and lower bounds, simplex and ordered vectors, and structured matrices such as covariance matrices and Cholesky factors. With Stan, we can automatically transform the joint density of any differentiable probability model to one with real-valued latent variables. (See Figure 2.)

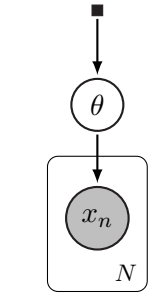
2.4 Variational Approximations in Real Coordinate Space

After the transformation, the latent variables $\boldsymbol{\zeta}$ 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 (Figure 1b); these implicitly induce non-Gaussian variational distributions in the original latent variable space (Figure 1a).

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

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

where the vector $\boldsymbol{\phi} = (\mu_1, \cdots, \mu_K, \sigma_1^2, \cdots, \sigma_K^2)$ concatenates the mean and variance of each Gaussian factor. Since the variance parameters must always be positive, the variational parameters live in the set $\boldsymbol{\Phi} = \{\mathbb{R}^K, \mathbb{R}^K_{>0}\}$. Re-parameterizing the mean-field Gaussian removes this constraint. Consider the 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 $q(\boldsymbol{\zeta}\,;\,\boldsymbol{\phi}) = \text{Normal}\left(\boldsymbol{\zeta} \mid \boldsymbol{\mu}, \text{diag}(\exp(\boldsymbol{\omega})^2)\right)$, 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 $\mathbb{R}^{2K}$.




```
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.

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

$$q(\boldsymbol{\zeta}\,;\,\boldsymbol{\phi}) = \text{Normal}\left(\boldsymbol{\zeta} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}\right),$$

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, $\boldsymbol{\Sigma} = \mathbf{L}\mathbf{L}^\top$. We use the non-unique definition of the Cholesky factorization where the diagonal elements of $\mathbf{L}$ need not be positively constrained (Pinheiro and Bates, 1996). Therefore $\mathbf{L}$ lives in the unconstrained space of lower-triangular matrices with $K(K+1)/2$ real-valued entries. The full-rank Gaussian becomes $q(\boldsymbol{\zeta}\,;\,\boldsymbol{\phi}) = \text{Normal}\left(\boldsymbol{\zeta} \mid \boldsymbol{\mu}, \mathbf{L}\mathbf{L}^\top\right)$, where the variational parameters $\boldsymbol{\phi} = (\boldsymbol{\mu}, \mathbf{L})$ 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.[4] 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 (Figure 1a) 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.

4. 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 $p(\mathbf{x}, \boldsymbol{\theta})$. 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

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

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

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

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 intractable expectations.

We cannot directly use automatic differentiation on the ELBO. This is because the ELBO involves an intractable 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[5] (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 $\boldsymbol{\eta} = S_{\boldsymbol{\phi}}(\boldsymbol{\zeta}) = \text{diag}\left(\exp\left(\boldsymbol{\omega}\right)\right)^{-1}\left(\boldsymbol{\zeta} - \boldsymbol{\mu}\right)$. In the full-rank Gaussian, the standardization is $\boldsymbol{\eta} = S_{\boldsymbol{\phi}}(\boldsymbol{\zeta}) = \mathbf{L}^{-1}(\boldsymbol{\zeta} - \boldsymbol{\mu})$.

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

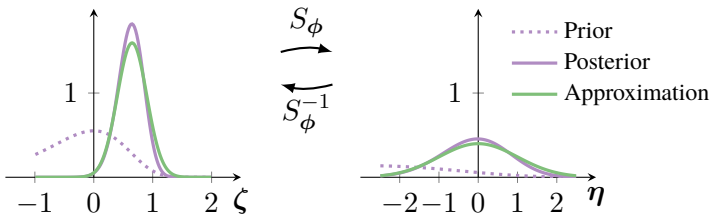
$$q(\boldsymbol{\eta}) = \text{Normal}\left(\boldsymbol{\eta} \mid \mathbf{0}, \mathbf{I}\right) = \prod_{k=1}^{K} \text{Normal}\left(\eta_k \mid 0, 1\right),$$

as shown in Figures 3a and 3b.

The standardization transforms the variational problem from Equation (5) into

$$\boldsymbol{\phi}^* = \arg\max_{\boldsymbol{\phi}} \mathbb{E}_{\text{N}(\boldsymbol{\eta}\,;\,\mathbf{0},\mathbf{I})}\left[\log p\left(\mathbf{x}, T^{-1}(S_{\boldsymbol{\phi}}^{-1}(\boldsymbol{\eta}))\right) + \log\left|\det J_{T^{-1}}\left(S_{\boldsymbol{\phi}}^{-1}(\boldsymbol{\eta})\right)\right|\right] + \mathbb{H}\big[q(\boldsymbol{\zeta}\,;\,\boldsymbol{\phi})\big].$$

5. 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.

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.

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

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

We obtain gradients with respect to $\boldsymbol{\omega}$ (mean-field) and $\mathbf{L}$ (full-rank) in a similar fashion

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

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

(Derivations in Appendix C.)

We can now compute the gradients inside the expectation with automatic differentiation. The only thing left is the intractable 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 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.[6] Stochastic gradient ascent falls

6. This is also called a *learning rate* or *schedule* in the machine learning community.

Algorithm 1: Automatic differentiation variational inference (ADVI)

Input: Dataset $\mathbf{x} = x_{1:N}$, model $p(\mathbf{x}, \boldsymbol{\theta})$.
Set iteration counter $i = 1$.
Initialize $\boldsymbol{\mu}^{(1)} = \mathbf{0}$.
Initialize $\boldsymbol{\omega}^{(1)} = \mathbf{0}$ (mean-field) or $\mathbf{L}^{(1)} = \mathbf{I}$ (full-rank).
Determine η via a search over finite values.

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

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

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

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

Increment iteration counter.

end

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

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

under the class of stochastic approximations, where Robbins and Monro (1951) established a pair of conditions that ensure convergence. 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) 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 $\mathbf{g}^{(i)}$ 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 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 hundreds of thousands to millions of observations.

2.7 Related Work

ADVI is an automatic variational inference algorithm, implemented within the Stan probabilistic programming system. This draws on two major themes.

Probabilistic programming. 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, 2003), and Stan (Stan Development Team, 2016). Another class of systems allows the user to directly specify probabilistic programs that may not admit a closed form probability distribution. Some examples are Church (Goodman et al., 2008), Figaro (Pfeffer, 2009), Venture (Mansinghka et al., 2014), Anglican (Wood et al., 2014), and WebPPL (Goodman and Stuhlmüller, 2014). Both classes primarily rely on various forms of MCMC sampling for inference.

Variational inference. 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 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), Ranganath et al. (2016) and Tran et al. (2016b) 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) and Minka et al. (2014) extend this to graphical models with non-conjugate relationships by either using custom approximations or sampling. 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.

Automatic differentiation. Automatic differentiation and machine learning enjoy a colorful and intertwined history (Baydin et al., 2015). For example, the backpropagation algorithm, rediscovered independently many times, is a form of automatic differentiation for neural network weights (Widrow and Lehr, 1990). Similarly, researchers have applied automatic differentiation to specific models, such as extended Kalman filters (Meyer et al., 2003) and computer vision models (Pock et al., 2007). Automatic differentiation also appears in recent variational inference research. For instance, the Bayes-by-backprop algorithm is a specific application of automatic differentiation to variational inference in Bayesian neural networks (Blundell et al., 2015). Many of the methods described above could, if applicable, use automatic differentiation to compute gradients of the model and the variational approximating families.

Software. ADVI can also be implemented in other general-purpose software frameworks, such as autograd (Maclaurin et al., 2015), Theano (Theano Development Team, 2016) and TensorFlow (Abadi et al., 2016). These frameworks offer features such as symbolic or automatic differentiation and abstractions for parallel computation. Two other implementations of ADVI are available, at the time of publication. The first is in PyMC3 (Salvatier et al., 2016), a probabilistic programming package, that implements ADVI in Python using Theano. The second is in Edward (Tran et al., 2016a), a Python library for probabilistic modeling, inference, and criticism, that implements ADVI in Python using TensorFlow.

3. Properties of Automatic Differentiation Variational Inference

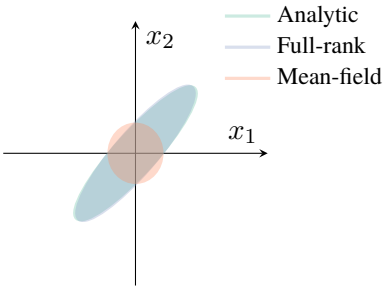
Automatic differentiation variational inference (ADVI) extends classical variational inference techniques in a few directions. In this section, we use simulated data to study three aspects of ADVI: the accuracy of mean-field and full-rank approximations, the variance of the ADVI gradient estimator, and 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. Consider a multivariate Gaussian likelihood $\text{Normal}(\mathbf{y} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma})$ with fixed, yet highly correlated, covariance $\boldsymbol{\Sigma}$; our goal is to estimate the mean $\boldsymbol{\mu}$. If we place a multivariate Gaussian prior on $\boldsymbol{\mu}$ 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 cor-



	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.

rectly 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 $\mathbf{x}$, and likelihood $\text{Bern}(y \mid \text{logit}^{-1}(\mathbf{x}^\top \boldsymbol{\beta}))$; our goal is to estimate the coefficients $\boldsymbol{\beta}$. 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.

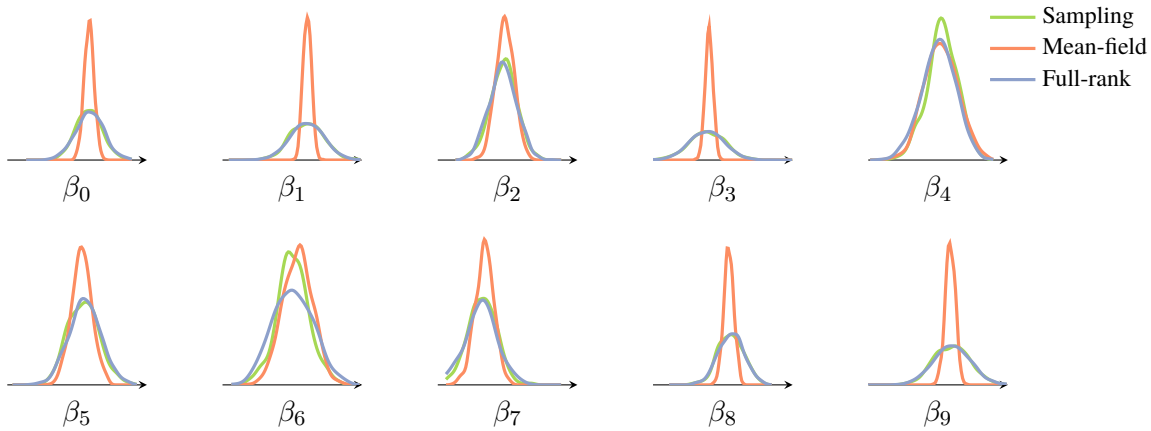


Figure 5: Comparison of marginal posterior densities for a logistic regression model. Each plot shows kernel density estimates for the posterior of each coefficient using 1000 samples. Mean-field ADVI underestimates variances for most of the coefficients.

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

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

where the log volatility follows an auto-regressive process

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

We place the following priors on the latent variables

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

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. This is expected behavior for a mean-field approximation to a time-series model (Turner and Sahani, 2008). 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 $\{\mathbf{h}^{(s)}\}_1^S$. Figure 7 shows the empirical posterior covariance matrix, $1/S-1 \sum_s (\mathbf{h}^{(s)} - \overline{\mathbf{h}})(\mathbf{h}^{(s)} - \overline{\mathbf{h}})^\top$ 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.

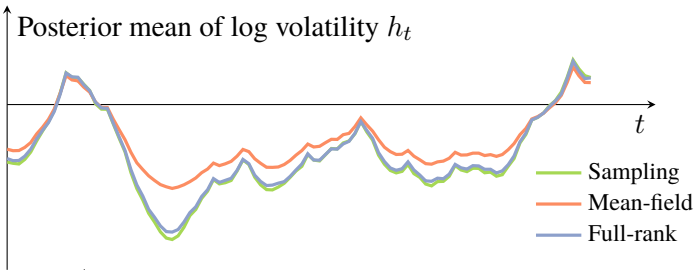
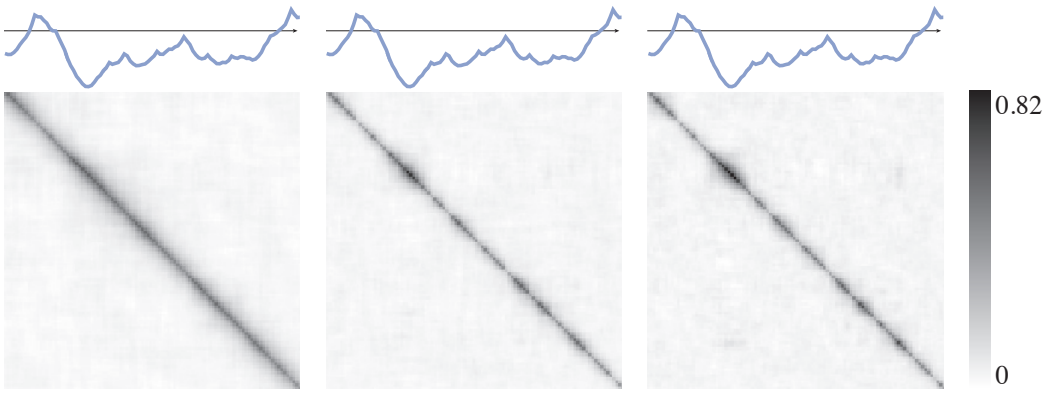


Figure 6: Comparison of posterior mean estimates of volatility h_t. Mean-field ADVI underestimates h_t, especially when it moves far away from its mean μ. Full-rank ADVI matches the accuracy of sampling.

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.



(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.

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

3.2 Variance of the Stochastic Gradients

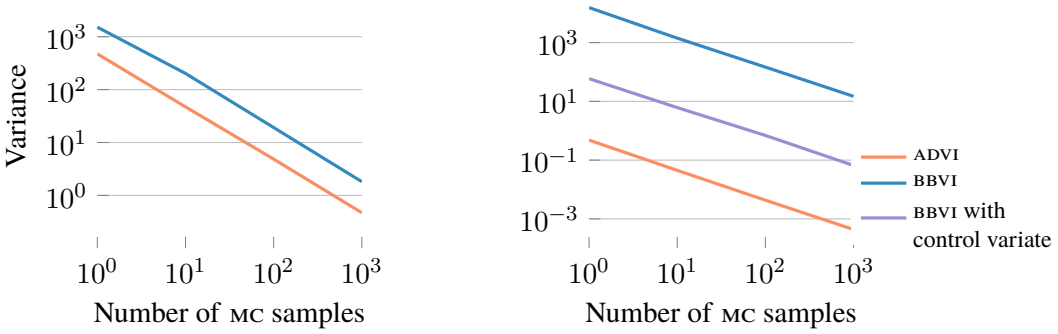
ADVI uses Monte Carlo integration to approximate gradients of the ELBO, and then uses these gradients in a stochastic optimization algorithm (Section 2). The speed of ADVI hinges on the variance of the gradient estimates. When a stochastic optimization algorithm suffers from high-variance gradients, 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

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

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 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.)



(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.

Figure 8b shows the same calculation for a 100-dimensional nonlinear regression model with likelihood $\text{Normal}(\mathbf{y} \mid \tanh(\mathbf{x}^\top \boldsymbol{\beta}), \mathbf{I})$ 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 gradient is more sample efficient.

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 $\text{Gamma}(1, 2)$, which places most of its mass close to $\theta = 0$, to $\text{Gamma}(10, 10)$, which is centered at $\theta = 1$. Consider two transformations T_1 and T_2

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

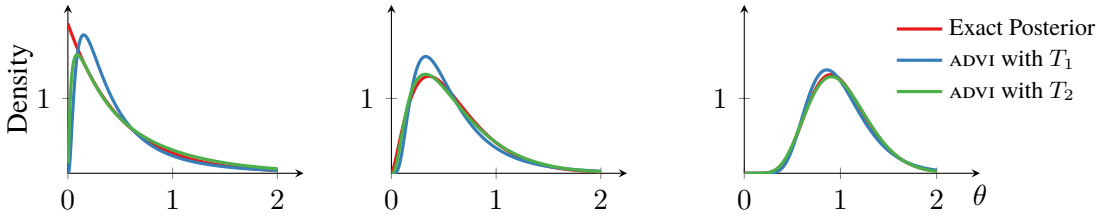
both of which map $\mathbb{R}_{>0}$ to $\mathbb{R}$. ADVI can use either transformation to approximate the Gamma posterior. Which one is better?

Figures 9a to 9c 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.

Is there an optimal transformation? Without loss of generality, we consider fixing a standard Gaussian distribution in the real coordinate space.[7] The optimal transformation is then

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

7. 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 $T_1(\theta) = T_3(T_2(\theta))$.



(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.

	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 for two transformations.

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 $P(\boldsymbol{\theta} \mid \mathbf{x})$; this is just as hard as the original goal of estimating the posterior density $p(\boldsymbol{\theta} \mid \mathbf{x})$.

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. ADVI in Practice

We now apply 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) (Neal, 2011) and the no-U-turn sampler (NUTS) (Hoffman and Gelman, 2014). NUTS is an adaptive extension of HMC and the default sampler in Stan.

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

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

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 over a million taxi rides. Here we show how a scientist might use ADVI in practice.

4.1 Hierarchical Regression Models

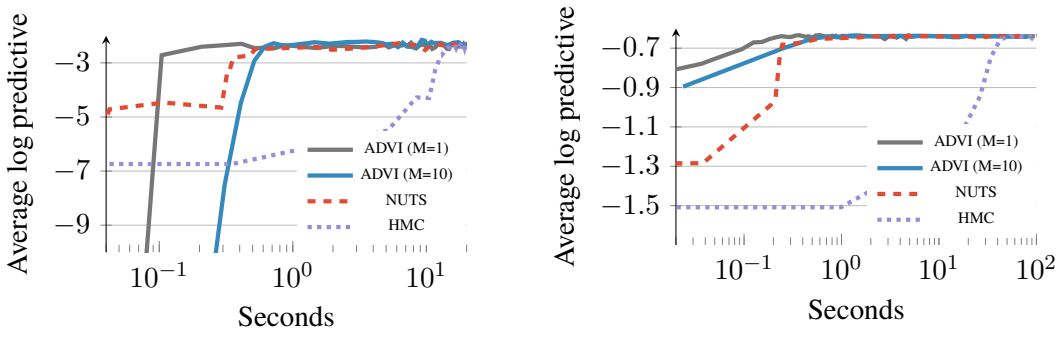
We begin with two nonconjugate regression models: linear regression with automatic relevance determination (ARD) (Bishop, 2006) and hierarchical logistic regression (Gelman and Hill, 2006).

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

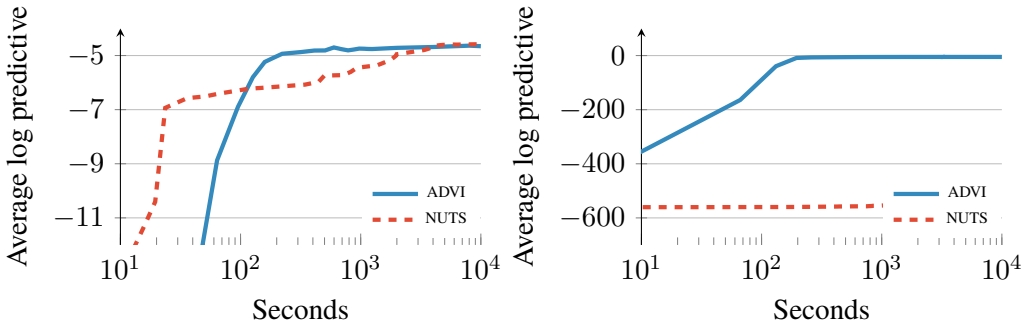
Frey Faces dataset, which contains 1956 frames (28×20 pixels) of facial expressions extracted from a video sequence.

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.)

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 (Figure 11a). NUTS struggles with the Dirichlet Exponential model (Figure 11b). 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 (Figure 11c) appears to pick significant frames out of the dataset. The Dirichlet Exponential factors (Figure 11d) 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

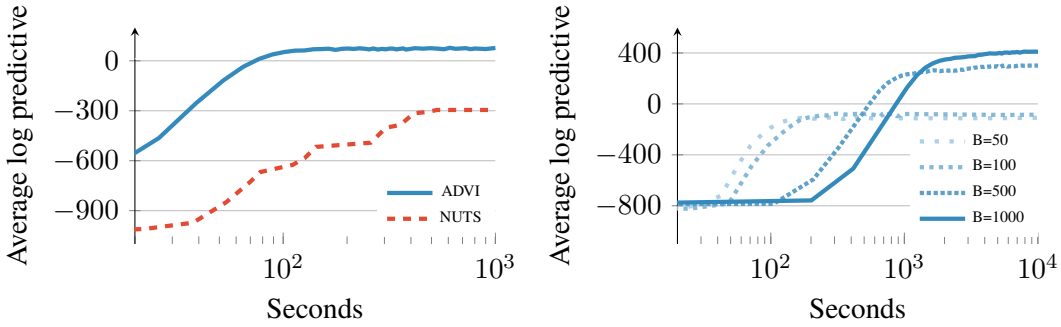
(c) Gamma Poisson factors **(d)** Dirichlet exponential factors

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



(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).

(not shown). This is likely due to label switching, which can affect HMC-based algorithms in mixture models (Stan Development Team, 2016).

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.

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.

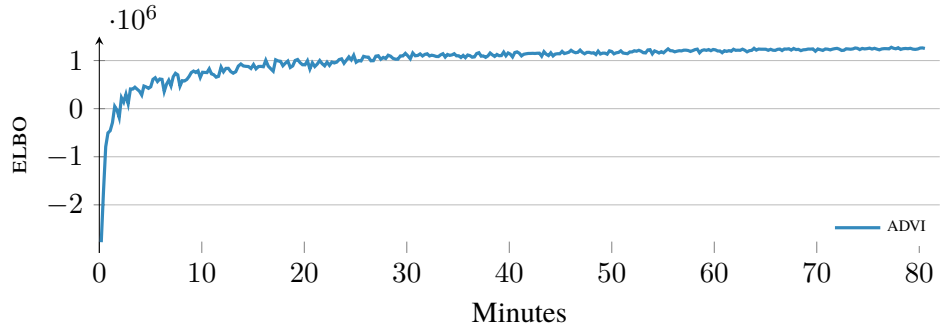


Figure 13: ELBO of PPCA model with ARD. ADVI converges in approximately an hour.

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.

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.

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 connecting highways that circumscribe the city.

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

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

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

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 algorithm that works "out of the box" for a large class of modern 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 and deployed it as part of Stan, a probabilistic programming system.

There are several avenues for research.

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. (2016b) place a Gaussian process to learn the optimal transformation and prove its expressiveness as a universal approximator. Hierarchical variational models (Ranganath et al., 2016) develop rich approximations for non-differentiable latent variable models.

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). ADVI works with unbiased gradient estimators; introducing some bias to reduce variance could also improve convergence speed (Ruiz et al., 2016a,b). Optimization using line search could also improve convergence robustness (Mahsereci and Hennig, 2015), as well as natural gradient approaches for nonconjugate models (Khan et al., 2015).

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.

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), Anglican (Wood et al., 2014), WebPPL (Goodman and Stuhlmüller, 2014), and Edward (Tran et al., 2016a).

Before we conclude, we offer some general advice. ADVI, like all of variational inference, is an approximate inference technique. As such, we recommend carefully studying its accuracy for new models. While Section 3.1 indicates a potential shortcoming, the quality of ADVI's posterior approximation will differ from model to model. We recommend validating ADVI for new models by running "fake data" checks (Cook et al., 2006). One of the advantages of ADVI is that its speed of convergence gives more opportunity for such checking in practice.

In summary, ADVI is a first step towards an automated variational inference algorithm that works well for a large class of practical models on modern real-world datasets. Each step of the recipe for ADVI highlights key design decisions: automating transformation of latent variables using a compiler, choosing a variational family that leads to low-variance gradient estimators of the variational objective, and developing an adaptive stochastic optimization step-size sequence that works not only in theory, but also in practice. ADVI enables scientists to easily build, explore, and revise complex probabilistic models with large data.

Acknowledgments

We thank Bruno Jacobs, Bob Carpenter, Daniel Lee, and the reviewers for their helpful comments. This work is supported by NSF IIS-0745520, IIS-1247664, IIS-1009542, SES-1424962, ONR N00014-11-1-0651, N00014-15-1-2541, DARPA FA8750-14-2-0009, N66001-15-C-4032, Sloan G-2015-13987, IES DE R305D140059, NDSEG, Facebook, Adobe, Amazon, and the Siebel Scholar and John Templeton Foundations.

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

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

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.

Appendix 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.

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

Appendix 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.

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

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

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

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

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

Appendix D. Automating Expectations: Monte Carlo Integration

Expectations of continuous random variables 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 decrease as $\mathcal{O}(1/\sqrt{S})$.

Appendix 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.

```
./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`.

Here, `myData.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.

Appendix 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). This sort of regression model is sometimes referred to as an *hierarchical* or *multilevel* regression model. We describe the model below. The Stan program is in Figure 17.

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

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

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

The ARD prior and hyper-prior structure is as follows

$$\begin{aligned} p(\mathbf{w}, \sigma, \boldsymbol{\alpha}) &= p(\mathbf{w}, \sigma \mid \boldsymbol{\alpha}) p(\boldsymbol{\alpha}) \\ &= \text{Normal}\left(\mathbf{w} \mid 0 \,,\, \sigma \left(\text{diag}\sqrt{\boldsymbol{\alpha}}\right)^{-1}\right) \text{InvGamma}(\sigma \mid a_0, b_0) \prod_{i=1}^{D} \text{Gamma}(\alpha_i \mid c_0, d_0) \end{aligned}$$

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 regressors have predictive power. The results in Figure 10a 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 is an intuitive way to model structured classification problems. We study a model of voting preferences, republican or democrat, from the 1988 United States presidential election. Chapter 14.1 of (Gelman and Hill, 2006) motivates the model and explains the dataset in

detail. We also briefly describe the model below. The Stan program is in Figure 18, based on (Stan Development Team, 2016).

$$\begin{aligned}\Pr(y_n = 1) = \text{sigmoid}\Big(&\beta^0 + \beta^{\text{female}} \cdot \text{female}_n + \beta^{\text{black}} \cdot \text{black}_n + \beta^{\text{female.black}} \cdot \text{female.black}_n \\ &+ \alpha^{\text{age}}_{k[n]} + \alpha^{\text{edu}}_{l[n]} + \alpha^{\text{age.edu}}_{k[n],l[n]} + \alpha^{\text{state}}_{j[n]}\Big) \\ \alpha^{\text{state}}_j \sim \text{Normal}&\left(\alpha^{\text{region}}_{m[j]} + \beta^{\text{v.prev}} \cdot \text{v.prev}_j\,,\, \sigma_{\text{state}}\right).\end{aligned}$$

The hierarchical variables are

$$\begin{aligned}\alpha^{\text{age}}_k &\sim \text{Normal}\left(0\,,\, \sigma_{\text{age}}\right) \text{ for } k = 1, \ldots, K \\ \alpha^{\text{edu}}_l &\sim \text{Normal}\left(0\,,\, \sigma_{\text{edu}}\right) \text{ for } l = 1, \ldots, L \\ \alpha^{\text{age.edu}}_{k,l} &\sim \text{Normal}\left(0\,,\, \sigma_{\text{age.edu}}\right) \text{ for } k = 1, \ldots, K, l = 1, \ldots, L \\ \alpha^{\text{region}}_m &\sim \text{Normal}\left(0\,,\, \sigma_{\text{region}}\right) \text{ for } m = 1, \ldots, M.\end{aligned}$$

The regression coefficient β has a Normal(0, 10) prior and all standard deviation latent variables have half Normal(0, 10) priors.

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 $\theta_{uk} \sim \text{Gamma}(a_0, b_0)$.
2. For each item i in $\{1, \cdots, I\}$:
    - For each component k, draw $\beta_{ik} \sim \text{Gamma}(c_0, d_0)$.
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. The Stan program 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 before.

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 $\boldsymbol{\theta}_u \sim \text{Dirichlet}(\boldsymbol{\alpha}_0)$.
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 $y_{ui} \sim \text{Poisson}(\boldsymbol{\theta}_u^\top \boldsymbol{\beta}_i)$.

The Stan program 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 versatile probability model (Bishop, 2006), often used for density estimation and clustering. Here 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 $\mathbf{y} = y_{1:N}$ where each y_n is D-dimensional and there are N observations. The likelihood for the images is

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

with a Dirichlet prior for the mixture proportions

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

a Gaussian prior for the mixture means

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

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 $\alpha_0 = 1\,000$ in our experiments.

The Stan program 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). Consider a dataset of $\mathbf{x} = x_{1:N}$ where each x_n is D-dimensional. Let $M < D$ be the dimension of the subspace we will use for analysis.

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

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

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

$$p(\mathbf{w}) = \prod_{d=1}^{D} \text{Normal}(w_d \,;\, \mathbf{0}, \mathbf{I}).$$

Finally define the likelihood through an inner product as

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

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

$$p(\sigma) = \text{Lognormal}(\sigma \,;\, 0, 1).$$

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.) This extends the above by

$$\begin{aligned}
p(\boldsymbol{\alpha}) &= \prod_{m=1}^{M} \text{InvGamma}(\alpha_m \,;\, 1, 1) \\
p(\mathbf{w} \mid \boldsymbol{\alpha}) &= \prod_{d=1}^{D} \text{Normal}(w_d \,;\, \mathbf{0}, \sigma\text{diag}(\boldsymbol{\alpha})) \\
p(\mathbf{x} \mid \mathbf{w}, \mathbf{z}, \sigma) &= \prod_{n=1}^{N} \text{Normal}(x_n \,;\, \mathbf{w}z_n, \sigma\mathbf{I}).
\end{aligned}$$

The Stan program that implements PPCA 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 $\mathbf{x}$, but to also predict y. The complete model is

$$
\begin{aligned}
p(\mathbf{z}) &= \prod_{n=1}^{N} \text{Normal}(z_n \,;\, \mathbf{0}, \mathbf{I}) \\
p(\sigma) &= \text{Lognormal}(\sigma \,;\, 0, 1) \\
p(\boldsymbol{\alpha}) &= \prod_{m=1}^{M} \text{InvGamma}(\alpha_m \,;\, 1, 1) \\
p(\mathbf{w}_{\mathbf{x}} \mid \boldsymbol{\alpha}) &= \prod_{d=1}^{D} \text{Normal}(w_d \,;\, \mathbf{0}, \sigma \text{diag}(\boldsymbol{\alpha})) \\
p(w_y \mid \boldsymbol{\alpha}) &= \text{Normal}(w_y \,;\, \mathbf{0}, \sigma \text{diag}(\boldsymbol{\alpha})) \\
p(\mathbf{x} \mid \mathbf{w}_{\mathbf{x}}, \mathbf{z}, \sigma) &= \prod_{n=1}^{N} \text{Normal}(x_n \,;\, \mathbf{w}_{\mathbf{x}} z_n, \sigma \mathbf{I}) \\
p(y \mid w_y, \mathbf{z}, \sigma) &= \prod_{n=1}^{N} \text{Normal}(y_n \,;\, w_y z_n, \sigma).
\end{aligned}
$$

The Stan program that implements SUP-PPCA 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 (d in 1:D)
    one_over_sqrt_alpha[d] = 1 / sqrt(alpha[d]);
}

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 program for Linear Regression with Automatic Relevance Determination.

```
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> sigma_a;
  real<lower=0> sigma_b;
  real<lower=0> sigma_c;
  real<lower=0> sigma_d;
  real<lower=0> 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, 10);
  sigma_a ~ normal(0, 10);
  sigma_b ~ normal(0, 10);
  sigma_c ~ normal(0, 10);
  sigma_d ~ normal(0, 10);
  sigma_e ~ normal(0, 10);
  y ~ bernoulli_logit(y_hat);
}
```

Figure 18: Stan program for Hierarchical Logistic Regression, adapted from (Stan Development Team, 2016).

```
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 program 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 program for the Dirichlet Exponential non-negative matrix factorization model.

```
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, 1);
      sigma[k] ~ lognormal(0, 1);
  }

  // likelihood
  for (n in 1:N) {
    real ps[K];
    for (k in 1:K) {
      ps[k] = log(theta[k]) + normal_lpdf(y[n] | mu[k], sigma[k]);
    }
    target += log_sum_exp(ps);
  }
}
```

Figure 21: Stan program 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, 1);
      sigma[k] ~ lognormal(0, 1);
  }

  // likelihood
  for (n in 1:N) {
    real ps[K];
    for (k in 1:K) {
      ps[k] = log(theta[k]) + normal_lpdf(y[n] | mu[k], sigma[k]);
    }
    target += minibatch_factor * log_sum_exp(ps);
  }
}
```

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

```
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 {
  matrix[M, N] z;            // latent variable
  matrix[D, M] w;            // weights parameters
  real<lower=0> sigma;       // standard deviation parameter
  vector<lower=0>[M] alpha;  // hyper-parameters on weights
}

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: Stan program for 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 consider

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

parameters {
  matrix[M, N] z;             // latent variable
  matrix[D, M] w_x;           // weight parameters for x
  vector[M] w_y;              // weight parameters for y
  real<lower=0> sigma;        // standard deviation parameter
  vector<lower=0>[M] alpha;   // hyper-parameters on weights
}

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: Stan program for SUP-PPCA with ARD.

References

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek Gordon Murray, Benoit Steiner, Paul A. Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zhang. Tensorflow: a system for large-scale machine learning. *arXiv:1605.08695*, 2016.

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

Charles E Antoniak. Mixtures of Dirichlet processes with applications to Bayesian nonparametric problems. *The Annals of Statistics*, 2(6):1152–1174, 1974.

David Barber. *Bayesian Reasoning and Machine Learning*. Cambridge University Press, 2012.

Atilim Gunes Baydin, Barak A Pearlmutter, and Alexey Andreyevich Radul. Automatic differentiation in machine learning: a survey. *arXiv:1502.05767*, 2015.

José M Bernardo and Adrian FM Smith. *Bayesian Theory*. John Wiley & Sons, 2009.

Christopher M Bishop. *Pattern Recognition and Machine Learning*. Springer New York, 2006.

Christopher M Bishop, David Spiegelhalter, and John Winn. VIBES: a variational inference engine for Bayesian networks. In *Neural Information Processing Systems*, 2002.

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

David M Blei, Alp Kucukelbir, and Jon D McAuliffe. Variational inference: a review for statisticians. *arXiv:1601.00670*, 2016.

Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural networks. In *Proceedings of The 32nd International Conference on Machine Learning*, pages 1613–1622, 2015.

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

John Canny. GaP: a factor model for discrete data. In *SIGIR Conference on Research and Development in Information Retrieval*, 2004.

Bob Carpenter, Matthew D Hoffman, Marcus Brubaker, Daniel Lee, Peter Li, and Michael Betancourt. The Stan math library: reverse-mode automatic differentiation in C++. *arXiv:1509.07164*, 2015.

Ali Taylan Cemgil. Bayesian inference for nonnegative matrix factorisation models. *Computational Intelligence and Neuroscience*, 2009.

Edward Challis and David Barber. Affine independent variational inference. In *Neural Information Processing Systems*, 2012.

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

Erhan Çınlar. *Probability and Stochastics*. Springer, 2011.

Samantha R Cook, Andrew Gelman, and Donald B Rubin. Validation of software for Bayesian models using posterior quantiles. *Journal of Computational and Graphical Statistics*, pages 675–692, 2006.

Persi Diaconis and Donald Ylvisaker. Conjugate priors for exponential families. *The Annals of Statistics*, 7(2):269–281, 1979.

Laurent Dinh, David Krueger, and Yoshua Bengio. NICE: non-linear independent components estimation. *arXiv:1410.8516*, 2014.

Jan Drugowitsch. Variational Bayesian inference for linear and logistic regression. *arXiv:1310.5438*, 2013.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *The Journal of Machine Learning Research*, 12:2121–2159, 2011.

Kai Fan, Ziteng Wang, Jeff Beck, James Kwok, and Katherine A Heller. Fast second order stochastic backpropagation for variational inference. In *Neural Information Processing Systems*, 2015.

Andrew Gelman and Jennifer Hill. *Data Analysis using Regression and Multilevel/Hierarchical Models*. Cambridge University Press, 2006.

Andrew Gelman, John B Carlin, Hal S Stern, David B Dunson, Aki Vehtari, and Donald B Rubin. *Bayesian Data Analysis*. CRC Press, 2013.

Noah D Goodman and Andreas Stuhlmüller. The Design and Implementation of Probabilistic Programming Languages, 2014.

Noah D Goodman, Vikash K Mansinghka, Daniel Roy, Keith Bonawitz, and Joshua B Tenenbaum. Church: a language for generative models. In *Uncertainty in Artificial Intelligence*, 2008.

Wolfgang Härdle and Léopold Simar. *Applied Multivariate Statistical Analysis*. Springer, 2012.

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

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

Michael I Jordan, Zoubin Ghahramani, Tommi S Jaakkola, and Lawrence K Saul. An introduction to variational methods for graphical models. *Machine Learning*, 37(2):183–233, 1999.

Mohammad E Khan, Pierre Baqué, François Fleuret, and Pascal Fua. Kullback-Leibler proximal variational inference. In *Neural Information Processing Systems*, 2015.

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

Diederik Kingma and Max Welling. Auto-encoding variational Bayes. In *International Conference on Learning Representations*, 2014.

Diederik P Kingma and Jimmy Ba Adam. A method for stochastic optimization. In *International Conference on Learning Representation*, 2015.

David A Knowles. Stochastic gradient variational Bayes for Gamma approximating distributions. *arXiv:1509.01631*, 2015.

Alp Kucukelbir, Rajesh Ranganath, Andrew Gelman, and David Blei. Automatic variational inference in Stan. In *Neural Information Processing Systems*, 2015.

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

Dougal Maclaurin, David Duvenaud, and Ryan P Adams. Autograd: reverse-mode differentiation of native Python. In *ICML Workshop on Automatic Machine Learning*, 2015.

Maren Mahsereci and Philipp Hennig. Probabilistic line searches for stochastic optimization. In *Neural Information Processing Systems*, 2015.

Vikash Mansinghka, Daniel Selsam, and Yura Perov. Venture: a higher-order probabilistic programming platform with programmable inference. *arXiv:1404.0099*, 2014.

Renate Meyer, David A Fournier, and Andreas Berg. Stochastic volatility: Bayesian computation using automatic differentiation and the extended Kalman filter. *The Econometrics Journal*, 6(2): 408–420, 2003.

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

Kevin P Murphy. *Machine Learning: a Probabilistic Perspective*. MIT Press, 2012.

Radford M Neal. MCMC using Hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo*, 2: 113–162, 2011.

David J Olive. *Statistical Theory and Inference*. Springer, 2014.

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

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

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

Martyn Plummer. JAGS: a program for analysis of Bayesian graphical models using Gibbs sampling. In *International Workshop on Distributed Statistical Computing*, 2003.

Thomas Pock, Michael Pock, and Horst Bischof. Algorithmic differentiation: application to variational problems in computer vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(7):1180–1193, 2007.

Rajesh Ranganath, Chong Wang, Blei David, and Eric Xing. An adaptive learning rate for stochastic variational inference. In *International Conference on Machine Learning*, 2013.

Rajesh Ranganath, Sean Gerrish, and David M Blei. Black box variational inference. In *Artificial Intelligence and Statistics*, 2014.

Rajesh Ranganath, Dustin Tran, and David M Blei. Hierarchical variational models. In *International Conference on Machine Learning*, 2016.

Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *International Conference on Machine Learning*, 2015.

Danilo J Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *International Conference on Machine Learning*, 2014.

Herbert Robbins and Sutton Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, 1951.

Christian P Robert and George Casella. *Monte Carlo Statistical Methods*. Springer, 1999.

Francisco JR Ruiz, Michalis K Titsias, and David M Blei. The generalized reparameterization gradient. In *Neural Information Processing Systems*, 2016a.

Francisco JR Ruiz, Michalis K Titsias, and David M Blei. Overdispersed black-box variational inference. In *Uncertainty in Artificial Intelligence*, 2016b.

Tim Salimans and David Knowles. On using control variates with stochastic approximation for variational Bayes. *arXiv:1401.1022*, 2014.

John Salvatier, Thomas V Wiecki, and Christopher Fonnesbeck. Probabilistic programming in Python using PyMC3. *PeerJ Computer Science*, 2:e55, 2016.

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

David J Spiegelhalter, Andrew Thomas, Nicky G Best, and Wally R Gilks. BUGS: Bayesian inference using Gibbs sampling, version 0.50. *MRC Biostatistics Unit, Cambridge*, 1995.

Stan Development Team. *Stan Modeling Language Users Guide and Reference Manual*, 2016.

Theano Development Team. Theano: a Python framework for fast computation of mathematical expressions. *arXiv:1605.02688*, 2016.

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

Michalis Titsias and Miguel Lázaro-Gredilla. Doubly stochastic variational Bayes for non-conjugate inference. In *International Conference on Machine Learning*, 2014.

Dustin Tran, Alp Kucukelbir, Adji B Dieng, Maja Rudolph, Dawen Liang, and David M Blei. Edward: a library for probabilistic modeling, inference, and criticism. *arXiv:1610.09787*, 2016a.

Dustin Tran, Rajesh Ranganath, and David M Blei. The variational Gaussian process. In *International Conference on Learning Representations*, 2016b.

Richard E Turner and Maneesh Sahani. Two problems with variational expectation maximisation for time-series models. In *Workshop on Inference and Estimation in Probabilistic Time-Series Models*, 2008.

Mauricio Villegas, Roberto Paredes, and Bart Thomee. Overview of the ImageCLEF 2013 Scalable Concept Image Annotation Subtask. In *CLEF Evaluation Labs and Workshop*, 2013.

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

Bernard Widrow and Michael A Lehr. 30 years of adaptive neural networks: perceptron, madaline, and backpropagation. *Proceedings of the IEEE*, 78(9):1415–1442, 1990.

David Wingate and Theophane Weber. Automated variational inference in probabilistic programming. *arXiv:1301.1299*, 2013.

John Winn and Christopher M Bishop. Variational message passing. *Journal of Machine Learning Research*, 6:661–694, 2005.

Frank Wood, Jan Willem van de Meent, and Vikash Mansinghka. A new approach to probabilistic programming inference. In *Artificial Intelligence and Statistics*, 2014.