Automatic Differentiation Variational Inference

Alp Kucukelbir  
Data Science Institute and Department of Computer Science  
Columbia University  
New York, NY 10027, USA

Dustin Tran  
Department of Statistics  
Harvard University  
Cambridge, MA 02138, USA

Rajesh Ranganath  
Department of Computer Science  
Princeton University  
Princeton, NJ 08540, USA

Andrew Gelman  
Data Science Institute, Departments of Political Science and Statistics  
Columbia University  
New York, NY 10027, USA

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

Editor: Lorem Ipsum

Abstract

Modern data analysis requires an iterative cycle: in the probabilistic modeling framework, a simple model is fit to the data, and it is refined as we gather more knowledge about the data’s hidden structure. However, fitting complex models to large datasets is mathematically and computationally challenging. We develop an automated tool called automatic differentiation variational inference (ADVI). The scientist only provides a probabilistic model and a dataset; nothing else. ADVI automatically derives an efficient algorithm that handles both complex models and large datasets. No conjugacy assumptions are required, and a broad class of models is supported. We study ADVI across ten different models and apply it to a dataset with millions of observations. ADVI is integrated into Stan, a probabilistic programming system; this makes ADVI available for immediate use.

1. Introduction

Scientists query the processes that govern our world. They analyze data, noisy measurements of the output of such processes. Consider how investigating drug trials reveals potential side-effects of medications. Or how studying taxi trips unravels traffic patterns in a city. These types of data analyses are insightful. Sadly, they are not perfect; in fact, they are inherently uncertain.
To quantify this uncertainty, we turn to probabilistic modeling. This framework analyzes data using the language of probabilities. We first propose a probabilistic model that describes a process. We then analyze how well it describes data from the same process. This second step is called *inference*, where we “fit the model to the data.” As the analysis provides insight, we revisit the model and repeat the process until satisfied.

Looping over this modeling-inference pipeline is not easy in practice. The processes we study are often complex. We want to propose rich probabilistic models and analyze large amounts of data. The bottleneck is computation: inference can be prohibitively expensive.

Variational inference (VI) offers one option for fitting probabilistic models to large data. VI converts the task of inference into an optimization problem. Decades of optimization research provides theory and algorithms that handle large data. This makes VI scalable; however, it does not necessarily make it easy to use.

The classical VI approach requires a custom algorithm for every probabilistic model. A small adjustment to the model may lead to arbitrarily complicated changes to the algorithm. To circumvent this, classical VI algorithms limit themselves to a small class of probabilistic models. This stifles scientists and breaks the crucial iterative workflow that probabilistic modeling is meant to provide.

What we want is to easily develop new models and study large data without worrying about inference. This is the dream of modern data analysis. VI can scale to large data; how can we automate its application?

**Main idea.** We propose a VI algorithm that decouples the inference from the probabilistic model. The idea is to transform a large class of models into a common space. Solving the VI optimization problem in this common space solves VI for all models in this large class.

Our method follows a simple recipe. First, consider a large class of models where the latent variables are differentiable. Second, transform all latent variables into the real coordinate space. Third, pose the variational optimization problem in this space. Finally, solve the optimization problem using automatic computational techniques.

We call our method automatic differentiation variational inference (ADVI). We implement and deploy ADVI as part of Stan, a probabilistic programming system. This provides a fully automated solution to VI; the only inputs are a probabilistic model and a dataset, nothing else. We leverage the probabilistic model compiler and transformation library in Stan to automate the entire process. ADVI enables a modern data analysis workflow. Scientists can easily write down a model and fit it to a large dataset. They can then study the adequacy of the fit and revise the model accordingly. Section 4 presents examples, some of which have hundreds of thousands to millions of data observations.

**Technical summary.** ADVI is an automated method. To avoid any user input, the algorithm makes its own decisions. In this paper, we explore and justify these decisions. We extend the initial recipe provided in Kucukelbir et al. (2015); what follows is a technical summary of its ingredients.

**Class of models.** We support a large class of probabilistic models. Two properties define this class. First, the latent variables must be differentiable. Second, the support of the posterior density of latent variables given data must match the support of the prior density. These are mild conditions; many commonly-used probability models satisfy these constraints (Table 1). Since we avoid any conjugacy requirements, extensions to these common models typically remain within this class.

**Automatic transformations.** To decouple variational inference from the model, we transform all models into a common space. Specifically, we transform the support of latent variables into the real coordinate space. For example, we use the logarithm to map from $\mathbb{R}_{>0}$ to $\mathbb{R}$. We leverage a library
of such transformations and use Stan’s probabilistic model compiler to automatically transform any
model in our class into the real coordinate space.

Fixed variational problem. \(\nu_1\) approximates the posterior density of latent variables given data
using a tractable family of probability densities. After the transformations, all latent variables live
in the real coordinate space. Here, we consider multivariate Gaussian approximations with diagonal
and full-rank covariances. The inverse transformations map back to a non-Gaussian density over the
original support of the latent variables. Thus, solving the fixed variational problem in real coordinate
space solves it for any model in our class.

Automatic scalable optimization. In spite of using a fixed Gaussian approximation, the variational
optimization problem in real coordinate space still involves calculations that depend on the
probabilistic model. In particular, we need to compute gradients and expectations of the model. We
automate these using automatic differentiation and Monte Carlo (MC) sampling. This produces noisy
gradients of the variational objective for any model in our class. We solve the optimization problem
using stochastic gradient ascent (SGA), which elegantly scales to large data.

Related work. ADVI automates variational inference; we deploy ADVI within the Stan probabilistic
programming system (Stan Development Team, 2015). This draws on two major themes.

The first is a body of work that aims to generalize \(\nu_1\). Ranganath et al. (2014) and Salimans and
Knowles (2014) propose a black-box technique that only requires computing gradients of the approx-
imating family. Kingma and Welling (2013) 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 small class of models. Here we build on and extend these ideas to
automate variational inference; we highlight technical connections as we develop the method.

The second theme is probabilistic programming. Wingate and Weber (2013) study \(\nu_1\) in general
probabilistic programs, as supported by languages like Church (Goodman et al., 2008), Figaro (Pf-
effer, 2009), Venture (Mansinghka et al., 2014), and Anglican (Wood et al., 2014). Another probabil-
istic programming system is infer.NET, which implements variational message passing (Winn and
Bishop, 2005), an efficient algorithm for conditionally conjugate graphical models. Stan sup-
ports a more comprehensive class of nonconjugate models with differentiable latent variables; see
Section 2.1.

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 Markov chain Monte Carlo
(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 automates variational inference for differentiable probability models. ADVI offers a recipe for
automating the computations involved in variational inference. The strategy is simple: transform a
user-specified model into a common space, choose a variational approximation in the common space,
and use generic computational techniques to solve the variational problem. We begin by defining
the class of probability models we support.
Consider a dataset $X = x_{1:N}$ with $N$ observations. Each $x_n$ is a discrete or continuous random variable. The likelihood $p(X \mid \theta)$ relates the observations to a set of latent random variables $\theta$. Bayesian analysis posits a prior density $p(\theta)$ on the latent variables. Combining the likelihood with the prior gives the joint density $p(X, \theta) = p(X \mid \theta) p(\theta)$.

We focus on inference for differentiable probability models. These models have continuous latent variables $\theta$. They also have a gradient of the log-joint with respect to the latent variables $\nabla_\theta \log p(X, \theta)$. The gradient is valid within the support of the prior $\text{supp}(p(\theta)) = \{ \theta \mid \theta \in \mathbb{R}^K \text{ and } p(\theta) > 0 \} \subseteq \mathbb{R}^K$.

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

Consider a model that contains a Poisson likelihood with unknown rate, $p(x \mid \theta)$. The observed variable $x$ is discrete; the latent rate $\theta$ is continuous and positive. Place a Weibull prior on $\theta$, defined over the positive real numbers. The resulting joint density describes a nonconjugate differentiable probability model. Its partial derivative $\partial / \partial \theta p(x, \theta)$ is valid within the support of the Weibull distribution, $\text{supp}(p(\theta)) = \mathbb{R}_{>0} \subseteq \mathbb{R}$. Since this model is nonconjugate, the posterior is not a Weibull distribution. This presents a challenge for classical variational inference, but not for ADVI, as we shall see.

Many machine learning models are differentiable. For example: linear and logistic regression, matrix factorization with continuous or discrete measurements, linear dynamical systems, and Gaussian processes. Mixture models, hidden Markov models, and topic models have discrete latent variables. Marginalizing out these discrete variables renders these models differentiable. (See Table 1.) However, marginalization is not tractable for all models, such as the Ising model, sigmoid belief network, and (untruncated) Bayesian nonparametric models, such as Dirichlet process mixtures (Blei et al., 2006).

Figure 1: Specifying a simple nonconjugate probability model in Stan.
2.2 Variational Inference

In Bayesian inference, we seek the posterior density \( p(\theta \mid X) \), which describes how the latent variables vary, conditioned on a dataset of observations \( X \). Many posterior densities are intractable because their normalizing constants lack analytic (closed-form) solutions. Thus, we seek to approximate the posterior.

Consider an approximating density \( q(\theta ; \phi) \) parameterized by parameters \( \phi \in \Phi \). We make no assumptions about the shape or support of \( q \); likewise, the parameters \( \phi \) live in some arbitrary set \( \Phi \). We want to find the parameters of \( q(\theta ; \phi) \) to best match the posterior according to some loss function. Variational inference (VI) minimizes the Kullback-Leibler (KL) divergence,

\[
\phi^* = \arg \min_{\phi \in \Phi} \text{KL}(q(\theta ; \phi) \| p(\theta \mid X)),
\]

from the approximation to the posterior (Wainwright and Jordan, 2008). Typically the KL divergence also lacks an analytic form. Instead we maximize a proxy to the KL divergence, the evidence lower bound (ELBO)

\[
\mathcal{L}(\phi) = \mathbb{E}_{q(\theta)}\left[ \log p(X, \theta) \right] - \mathbb{E}_{q(\theta)}\left[ \log q(\theta ; \phi) \right].
\]

The first term is an expectation of the joint density under the approximation, and the second is the entropy of the variational density. Maximizing the ELBO minimizes the KL divergence (Jordan et al., 1999; Bishop, 2006).

The minimization problem from Equation (1) becomes

\[
\phi^* = \arg \max_{\phi \in \Phi} \mathcal{L}(\phi) \quad \text{such that} \quad \text{supp}(q(\theta ; \phi)) \subseteq \text{supp}(p(\theta \mid X)),
\]

where we explicitly specify the support matching constraint implied in the KL divergence.\(^1\) We highlight this constraint, as we do not specify the form of the variational approximation; thus we must ensure that \( q(\theta ; \phi) \) stays within the support of the posterior. This is the variational problem we seek to solve.

However, the support of the posterior may also be unknown. We assume that the support of the posterior equals that of the prior, \( \text{supp}(p(\theta \mid X)) = \text{supp}(p(\theta)) \). This is true when the likelihood does not constrain the prior; i.e., the likelihood must be positive over the sample space for any \( \theta \) drawn from the prior.

What makes VI difficult to automate? The traditional way of solving Equation (3) is as follows. We begin by choosing a variational family \( q(\theta ; \phi) \) that, by definition, satisfies the support matching

---

\(^1\) If \( \text{supp}(q) \not\subseteq \text{supp}(p) \) then outside the support of \( p \) we have \( \text{KL}(q \parallel p) = \mathbb{E}_q[\log q] - \mathbb{E}_q[\log p] = -\infty \).
constraint. We compute the expectations in the ELBO, either analytically or through some sort of approximation. We then decide on a strategy to maximize the ELBO. For instance, we might follow gradients of the ELBO with respect to $\phi$ while staying within $\Phi$. Finally, we implement, test, and debug software that performs the above.

Each step requires expert thought and analysis. The first two steps are intertwined: the classical approach constrains the model to a small class of models (conditionally conjugate) where the optimal choice for $q$ is well-defined and the ELBO has an analytic closed-form solution (Bishop, 2006). The following steps are also connected: tedious mathematics are necessary to derive an optimization procedure. The last step describes the inevitable cost of implementing an algorithm specific to a particular model.

Our recipe for automating VI. We allow the user to define any differentiable probability model. We then automatically transform the support of the latent variables $\theta$ to the real coordinate space.

In this space we can choose from a variety of variational distributions $q$ without worrying about the support matching constraint. We then compute the ELBO for any model using MC integration, which only requires being able to sample from the variational distribution. We employ stochastic gradient ascent to maximize the ELBO and use automatic differentiation to compute gradients without any user input. With these tools, we can develop a truly generic method that automatically solves the variational optimization problem for a large class of models.

The first idea is our transformation-based approach. We begin by transforming the support of the latent variables in our model to the real coordinate space. Then, we posit a variational density with support on the real coordinate space. The inverse of our transform induces a variational approximation in the original variable space. The transformation guarantees that the approximation stays within the support of the posterior, thus satisfying the support matching constraint by construction.

2.3 Automatic Transformation of Constrained Variables

Begin by transforming the support of the latent variables $\theta$ such that they live in the real coordinate space $\mathbb{R}^K$. Define a one-to-one differentiable function

$$T : \text{supp}(p(\theta)) \rightarrow \mathbb{R}^K,$$

and identify the transformed variables as $\xi = T(\theta)$. The transformed joint density $g(X, \xi)$ is a function of $\xi$; it has the representation

$$g(X, \xi) = p(X, T^{-1}(\xi)) \left| \det J_{T^{-1}}(\xi) \right|,$$

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

Consider again our running example. The latent variable $\theta$ 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, $| \det J_{T^{-1}}(\xi) | = \exp(\xi)$. The transformed density is

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

Figures 2a and 2b depict this transformation.
As we describe in the introduction, we implement our algorithm in Stan to enable generic inference. Stan implements a model compiler that automatically handles transformations. It works by maintaining a library of transformations and their corresponding Jacobians. The compiler then builds in the necessary transformations to the joint model density. This process transforms the joint density of any differentiable probability model to the real coordinate space. (See Figure 1.)

There are many ways to differentially transform the support a model to the real coordinate space. The form of the transformation directly affects the shape of the variational approximation. In Section 3.3 we study sensitivity to the choice of transformation. In any case, after this automatic transformation step, we can choose a variational distribution independently from the model.

![Figure 2: Transforming the model to real coordinate space. The purple line is the posterior. The green line is the approximation.](image)

2.4 Variational Approximations in Real Coordinate Space

After the transformation, the latent variables \( \xi \) have support in the real coordinate space, \( \mathbb{R}^K \). We have a choice of variational approximations in this space.

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

\[
q(\xi \mid \phi) = \mathcal{N}(\xi \mid \mu, \sigma^2) = \prod_{k=1}^{K} \mathcal{N}(\xi_k \mid \mu_k, \sigma_k^2),
\]

where the vector \( \phi = (\mu_1, \ldots, \mu_K, \sigma_1^2, \ldots, \sigma_K^2) \) concatenates the mean and variance of each Gaussian factor. Since all variance terms must be positive, the variational parameters live in the set \( \Phi = \{\mathbb{R}^K, \mathbb{R}^{K+1}_{>0}\} \).

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 2a) is non-Gaussian, because of the inverse transformation \( T^{-1} \) and its Jacobian.

---

2. Stan provides various transformations for upper and lower bounds, simplex and ordered vectors, and structured matrices such as covariance matrices and Cholesky factors (Stan Development Team, 2015).
Full-rank Gaussian. Another option is to posit a full-rank Gaussian variational approximation
\[ q(\zeta : \phi) = \mathcal{N}(\zeta ; \mu, \Sigma), \]
where the vector \( \phi = (\mu, \Sigma) \) concatenates the mean vector \( \mu \) and covariance matrix \( \Sigma \). The variational parameters live in the set \( \Phi = \{ \mathbb{R}^K, \mathbb{S}^K_+ \} \) where \( \mathbb{S}^K_+ \) is the set of \( K \times K \) positive semidefinite matrices \( \{ \Sigma | \Sigma = \Sigma^T \in \mathbb{R}^{K \times K} \text{ and } u^T \Sigma u \geq 0, \forall u \in \mathbb{R}^K \} \).

This generalizes the mean-field Gaussian approximation. The off-diagonal terms in the covariance matrix capture posterior correlations across latent random variables. This leads to a more accurate posterior approximation than the mean-field Gaussian; however, it comes at a severe 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).

2.5 The Variational Problem in Real Coordinate Space

We write the variational objective function, the elbo, in real coordinate space as
\[ \mathcal{L}(\phi) = \mathbb{E}_{q(\zeta ; \phi)} \left[ \log p(X, T^{-1}(\zeta)) + \log | \det J_{T^{-1}}(\zeta) | \right] + \mathcal{H}[q(\zeta ; \phi)]. \] (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 \( \phi \) and the entropy \( \mathcal{H} \), both of which depend on the chosen variational approximation. (Derivation in Appendix B.)

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 \( q(T(\theta) ; \phi) | \det J_T(\theta) | \). The transformation ensures that the support of this approximation is always bounded by that of the true posterior in the original latent variable space.

Thus 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,
\[ \phi^* = \arg \max_{\phi \in \Phi} \mathcal{L}(\phi). \] (6)

However, one final constraint remains: the variational parameters must live within \( \Phi \). Luckily, this is easy to handle with simple re-parameterizations of the variational families. Converting the above to a fully unconstrained problem pays off when we turn to automatically maximizing the objective function.

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

Re-parameterized full-rank Gaussian. The covariance matrix must be positive semidefinite. Re-parameterize the full-rank covariance matrix using a Cholesky factorization, \( \Sigma = LL^T \). We use the non-unique definition of the Cholesky factorization where the diagonal elements of \( L \) need
not be positively constrained (Pinheiro and Bates, 1996). Therefore $L$ lives in the unconstrained space of lower-triangular matrices with $K(K + 1)/2$ real-valued entries. The full-rank Gaussian becomes $q(\xi : \phi) = \mathcal{N}(\xi ; \mu, LL^T)$, and the variational parameters $\phi = (\mu, L)$ are unconstrained in $\mathbb{R}^{K + K(K + 1)/2}$.

**Unconstrained variational problem.** With these re-parameterized variational families, the optimization problem from Equation (6) becomes,

$$
\phi^* = \arg \max_{\phi} \mathcal{L}(\phi) \tag{7}
$$

where the parameter vector $\phi$ lives in some appropriately dimensioned real coordinate space. This is now a fully unconstrained optimization problem. We discuss how to automatically solve it next.

### 2.6 Automatically Solving the Variational Problem

We can solve the unconstrained optimization problem of Equation (7) using gradient ascent. Traditionally, this would require manual computation of gradients. Instead, we develop a stochastic gradient ascent algorithm that uses automatic differentiation to compute gradients and MC integration to approximate expectations.

We cannot directly use automatic differentiation on the ELBO. This is because the ELBO is an expectation that depends on the model $p$ and transformation $T$, both of which are arbitrary. In general, we do not have an analytic form for the ELBO; thus, we cannot represent it as a simple computer program that can be automatically differentiated. Thus, we employ one final transformation: elliptical standardization\(^3\) (Härdle and Simar, 2012).

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

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

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

as shown in Figures 3a and 3b.

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

$$
\phi^* = \arg \max_{\phi} \mathbb{E}_{\mathcal{N}(\eta ; 0, I)} \left[ \log p(X, T^{-1}(S^{-1}_\phi(\eta))) + \log | \det J_{T^{-1}}(S^{-1}_\phi(\eta)) | \right] 
+ \mathbb{H}[q(\xi : \phi)].
$$

The expectation is now in terms of a standard Gaussian density. There is no explicit Jacobian adjustment here. This is because the Gaussian distribution is a member of the location-scale family: standardizing a Gaussian gives another Gaussian distribution.

\(3.\) 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, 2013).
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.

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

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

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

\[
\nabla_\omega \mathcal{L} = \mathbb{E}_{\mathcal{N}(\eta)} \left[ (\nabla_\theta \log p(X, \theta) \nabla_\xi T^{-1}(\xi) + \nabla_\xi \log |\det J_{T^{-1}}(\xi)|) \eta^T \text{diag}(\exp(\omega)) \right] + 1
\]

\[
\nabla_L \mathcal{L} = \mathbb{E}_{\mathcal{N}(\eta)} \left[ (\nabla_\theta \log p(X, \theta) \nabla_\xi T^{-1}(\xi) + \nabla_\xi \log |\det J_{T^{-1}}(\xi)|) \eta^T \right] + (L^{-1})^T
\]

(Derivations in Appendix C.)

We can now compute the gradients inside the expectation with automatic differentiation. The only thing left is the expectation. MC integration provides a simple approximation: draw \( M \) samples from the standard Gaussian and evaluate the empirical mean of the gradients within the expectation (Robert and Casella, 1999). (See Appendix D).

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

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

Adaptive step-size sequence. Adaptive step-size sequences retain (possibly infinite) memory about past gradients and adapt to the high-dimensional curvature of the ELBO optimization space (Duchi

\[4\] This is also called a learning rate or schedule in the machine learning community.
Automatic Differentiation Variational Inference

et al., 2011; Ranganath et al., 2013; Kingma and Ba, 2014) analogous to natural gradients (Amari, 1998). These sequences enjoy theoretical bounds on their convergence rates. However, in practice, they can be slow to converge. The empirically justified RMSProp sequence (Tieleman and Hinton, 2012), which only retains finite memory of past gradients, converges quickly in practice but lacks any convergence guarantees. We propose a new step-size sequence which effectively combines both approaches.

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

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

where we apply the following recursive update

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

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 $\eta$ by performing a grid search over 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. Without the previous decaying term, this would lead to possibly large oscillations around a local optimum of the ELBO. The additional perturbation $\tau > 0$ prevents division by zero and down-weights early iterations. In practice the step-size is not very sensitive to this value (Hoffman et al., 2013), so we set $\tau = 1$.

Complexity and data subsampling. ADVI has complexity $\Theta(NMK)$ 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 $\Theta(NK)$ per pass over the dataset. The added complexity of automatic differentiation over analytic gradients is roughly constant, in the order of 10 (Carpenter 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 $\Theta(2BMK)$.

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

3. Properties of ADVI

Automatic differentiation variational inference (ADVI) extends classical variational inference techniques in a few directions. In this section, we empirically study three aspects of ADVI: accuracy of the mean-field and full-rank approximations, variance of the ADVI gradient estimator, and sensitivity to the transformation $T$.
Algorithm 1: Automatic differentiation variational inference (ADVI)

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

while change in ELBO is above some threshold do
    Draw $M$ samples $\eta_m \sim N(0, I)$ from the standard multivariate Gaussian.
    Approximate $\nabla_\mu \mathcal{L}$ using mc integration (Equation (8)).
    Approximate $\nabla_\omega \mathcal{L}$ or $\nabla_L \mathcal{L}$ using mc integration (Equations (9) and (10)).
    Calculate step-size $\rho^{(i)}$ (Equation (11)).
    Update $\mu^{(i+1)} \leftarrow \mu^{(i)} + \text{diag}(\rho^{(i)}) \nabla_\mu \mathcal{L}$.
    Update $\omega^{(i+1)} \leftarrow \omega^{(i)} + \text{diag}(\rho^{(i)}) \nabla_\omega \mathcal{L}$ or $L^{(i+1)} \leftarrow L^{(i)} + \text{diag}(\rho^{(i)}) \nabla_L \mathcal{L}$.
    Increment iteration counter.
end

Return $\mu^* \leftarrow \mu^{(i)}$.
Return $\omega^* \leftarrow \omega^{(i)}$ or $L^* \leftarrow L^{(i)}$.

3.1 Accuracy

We study accuracy across three models that highlight different aspects of ADVI.

Two-dimensional Gaussian. Consider a multivariate Gaussian likelihood $\mathcal{N}(y \mid \mu, \Sigma)$ with fixed
covariance $\Sigma$. The mean $\mu$ is the latent variable we seek to estimate. If we posit another multivariate
Gaussian as the prior on $\mu$, then the posterior density is also a multivariate Gaussian that we can
calculate analytically (Young and Smith, 2005).

Figure 4 compares mean-field and full-rank ADVI to the exact posterior. We draw 1000 datapoints
from the model and run both variants of ADVI until convergence. In this model, both variational
approximations have the correct distribution: there are no transformations at play and the analytic
posterior is also Gaussian.

Both procedures correctly identify the mean of the analytic posterior. However, the shape of the
mean-field approximation is incorrect. This is because the mean-field approximation explicitly ig-
nores off-diagonal terms of the Gaussian covariance. The variational problem seeks to minimize the
KL divergence from the approximation to the exact posterior; this leads to a systemic underestimation
of marginal variances (Bishop, 2006).

Logistic regression. Consider a logistic regression model with design matrix $X$ and likelihood
$\text{Bern}(y \mid \text{logit}^{-1}(X\beta))$. The vector of regression coefficients $\beta$ is the latent variable we seek to
estimate. We posit an independent Gaussian prior for each regression coefficient.
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.

<table>
<thead>
<tr>
<th></th>
<th>Analytic</th>
<th>Full-rank</th>
<th>Mean-field</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variance along $x_1$</td>
<td>0.28</td>
<td>0.28</td>
<td>0.13</td>
</tr>
<tr>
<td>Variance along $x_2$</td>
<td>0.31</td>
<td>0.31</td>
<td>0.14</td>
</tr>
</tbody>
</table>

Figure 5: Comparison of marginal posterior densities for a logistic regression model. Each plot shows the true (simulated) value for each regression coefficient, along with posterior density estimates. Mean-field ADVI underestimates variances for most of the coefficients.

Figure 5 compares mean-field and full-rank ADVI to an estimate of the posterior obtained through MCMC sampling. We simulate a design matrix with 9 random features (plus intercept) and draw 1000 datapoints. The plots show marginal posterior densities estimated from 1000 samples drawn from all three posteriors at convergence.

All methods perform similarly in terms of estimating the posterior means of the regression coefficients. The mean-field approximation, as expected, underestimates marginal posterior variances on most of the coefficients. The full-rank approximation, once again, better matches the posterior.

Stochastic volatility time-series model. Consider a stochastic volatility model that describes the variance of an economic asset across discrete time measurements (Kim et al., 1998). An autoregressive process models latent volatility; thus we expect posterior estimates of volatility to be correlated, especially when the volatility deviates from the mean.

The likelihood exhibits the latent volatility as part of the variance of a Gaussian

$$y_t \sim \mathcal{N}(0, \exp(h_t/2))$$
where the volatility itself follows an auto-regressive process

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

We posit the following priors for 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 draw 500 time-steps from the generative model above. Figure 6 plots the posterior mean of the volatility \( h_t \) as a function of time. Mean-field ADVI struggles to describe the mean of the posterior, particularly when the volatility drifts far away from \( \mu \). In contrast, full-rank ADVI matches the estimates obtained from sampling.

We further investigate this by drawing 1000 posterior samples of the latent volatility. Figure 7 shows the empirical posterior covariance matrix 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). The regions where the local correlation is strongest correspond to the regions where mean-field underestimates the volatility in Figure 6. All empirical 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 \( \mu \). Full-rank ADVI matches the accuracy of sampling.

![Posterior mean of \( h_t \)](image)

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.
**Recommendations.** Scientists interested in posterior inference should use the full-rank approximation. Full-rank ADVI captures posterior correlations, in turn producing more accurate marginal variance estimates. For large models, however, full-rank ADVI can be prohibitively slow to use.

Scientists interested in prediction should use 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; underestimating marginal variances matters less.

### 3.2 Variance of Gradient Estimators

In ADVI, we compute gradients of the ELBO in real coordinate space, where the variational approximation is a Gaussian distribution. We employ elliptical standardization to push the gradients inside the expectation. Automatic differentiation and MC integration define the form of the ADVI gradient estimators (Equations (8) to (10)). However, this is not the only way to automatically compute gradients 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

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

is equivalent to the ADVI gradient estimator in Equation (8). The BBVI approach is more general, as it does not require the gradient of the model. However, MC estimation of the BBVI gradient suffers from high variance.

**Figure 8** empirically compares the variance of both estimators for two models. Figure 8a plots the variance of the gradient estimators for a simple univariate model across an increasing number of MC samples. Figure 8b shows the same for a 100-dimensional nonlinear regression model. Because this is a multivariate example, we also show the BBVI gradient estimator with the control variate variance reduction scheme described in (Ranganath et al., 2014). In both cases, the ADVI gradient estimators exhibit lower variance.
3.3 Sensitivity to Transformations

We now study how $T$ affects the non-Gaussian posterior approximation in the original latent variable space. Consider a posterior density in the Gamma family, with support over $\mathbb{R}_{>0}$. Figure 9 shows three configurations of the Gamma, ranging from Gamma(1, 2), which places most of its mass close to $\theta = 0$, to Gamma(10, 10), which is centered at $\theta = 1$.

Consider two transformations $T_1$ and $T_2$

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

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

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 $\theta$. 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. The optimal transformation is then

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

where $P$ is the cumulative density function of the posterior and $\Phi^{-1}$ is the inverse cumulative density function of the standard Gaussian. $P$ maps the posterior to a uniform distribution and $\Phi^{-1}$ maps the uniform distribution to the standard Gaussian. The optimal choice of transformation enables the Gaussian variational approximation to be exact.

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, $P_{\text{Gamma}}$ is difficult to compute.) Challis and Barber (2012) study Fourier transform techniques for location-scale variational approximations beyond the Gaussian. Another option is to automatically learn good transformations. We discuss recent approaches in this direction in Section 5.

<table>
<thead>
<tr>
<th></th>
<th>Gamma(1, 2)</th>
<th>Gamma(2.5, 4.2)</th>
<th>Gamma(10, 10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>KL$(q \parallel p)$ with $T_1$</td>
<td>$8.1 \times 10^{-2}$</td>
<td>$3.3 \times 10^{-2}$</td>
<td>$8.5 \times 10^{-3}$</td>
</tr>
<tr>
<td>KL$(q \parallel p)$ with $T_2$</td>
<td>$1.6 \times 10^{-2}$</td>
<td>$3.6 \times 10^{-3}$</td>
<td>$7.7 \times 10^{-4}$</td>
</tr>
</tbody>
</table>

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

---

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

16
4. ADVI in Practice

We now apply ADVI to an array of nonconjugate probability models. We study hierarchical regression, non-negative matrix factorization, mixture models, and probabilistic principal component analysis. We compare mean-field ADVI to two MCMC sampling algorithms: Hamiltonian Monte Carlo (HMC) (Girolami and Calderhead, 2011) and the no-U-turn sampler (NUTS)\(^6\) (Hoffman and Gelman, 2014). We assess ADVI convergence by tracking the ELBO. To place ADVI and MCMC on a common scale, we report predictive likelihood on held-out data as a function of time. Specifically, we estimate the predictive likelihood

\[
p(X_{\text{held-out}} \mid X) = \int p(X_{\text{held-out}} \mid \theta) p(\theta \mid X) \, d\theta
\]

using MC sampling. For MCMC, we plug in posterior samples. For ADVI, we draw samples from the posterior approximation during the optimization. We initialize ADVI with a draw from a standard Gaussian.

We conclude with a case study: an exploratory analysis of millions of taxi rides. Here we show how a data 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 sparse linear regression model with a hierarchical prior structure. This model is conditionally conjugate; thus, we can hand-derive a classical VI algorithm. Drugowitsch (2013) derive such an algorithm; it takes multiple pages. (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.

---

\(^6\) NUTS is an adaptive extension of HMC. It is the default sampler in Stan.
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 also is the fastest. (We set $M = 1$ from here on.)

![Graph](attachment:image.png)

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

**Figure 10:** Hierarchical generalized linear models.

4.2 Non-negative Matrix Factorization

We continue by exploring two nonconjugate non-negative matrix factorization models: a constrained Gamma Poisson model (Canny, 2004) and a Dirichlet Exponential Poisson model. Here, we show how easy it is to explore new models using ADVI. In both models, we use the Frey Face dataset, which contains 1956 frames ($28 \times 20$ 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 matrices 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 the plots.

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.

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.
**Automatic Differentiation Variational Inference**

![Graph](image1)

**Figure 11**: Non-negative matrix factorization of the Frey Faces dataset.

![Graph](image2)

**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).

In Figure 12a we randomly select 1000 images and train a model with 10 mixture components. NUTS struggles to find an adequate solution and HMC fails altogether. This is likely due to label switching, which can affect HMC-based techniques in mixture models (Stan Development Team, 2015).

Figure 12b shows ADVI results on the full dataset. Here we use ADVI with stochastic subsampling of minibatches from the dataset (Hoffman et al., 2013). We increase the number of mixture components to 30. 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 new models? To answer these questions, we apply ADVI to a modern exploratory data analysis task: analyzing traffic patterns.

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 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 arbitrary 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 interpolate all trajectories to 50 coordinate pairs. Vectorizing the pairs converts each trajectory into a point in \(\mathbb{R}^{100}\).

The trajectories are structured; major roads and highways appear frequently. This motivates a spectral clustering approach. Spectral clustering first identifies an intrinsic subspace (or manifold) within the data. Projecting each trajectory into this subspace yields a lower-dimensional representation of the data. Clustering the trajectories in this subspace is easier than in the original data space; the properties of the subspace determine the quality of the clustering.

We begin with a simple subspace identification technique: probabilistic principal component analysis (PPCA) (Bishop, 2006). This is a Bayesian generalization of classical principal component analysis, which makes it 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 which principal components are most effective at explaining variation in the data. The strategy is similar to that in Section 4.1. First, impose a hierarchical prior that ranks principal components; then use this ranking to identify a lower dimensional subspace. (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.](image)

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 this subspace. We then use the GMM
from Section 4.3 with $K = 30$ components to cluster the trajectories. ADVI takes less than half an hour to converge.

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

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

Interpolating each trajectory to an equal length 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 the trajectories. 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 we use 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 that connect highways that circumscribe the city.

Exploratory data analysis is an iterative effort: we want to rapidly evaluate models and modify them based on what we learn. An automatic and fast inference procedure enables effective exploration of massive datasets. This is where ADVI shines.
5. Discussion

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

There are several avenues for research.

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

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

Follow with practical heuristics. Two things affect ADVI convergence: initialization and step-size scaling. We initialize ADVI in the real coordinate space with a draw from a standard Gaussian. A better heuristic could adapt to the model and dataset. We adaptively tune the scale of the step-size sequence using a grid search. A better heuristic could avoid this additional computation.

End with probabilistic programming. We designed and deployed ADVI with Stan in mind. Thus, we focused on the class of differentiable probability models. We can think of extensions to ADVI that, for instance, work with discrete latent variables. One approach would adapt ADVI to use the score
function 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.) In a similar vein, we may see modified versions of ADVI being useful in more general probabilistic programming systems, such as Church (Goodman et al., 2008), Figaro (Pfeffer, 2009), Venture (Mansinghka et al., 2014), and Anglican (Wood et al., 2014).

### Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADVI</td>
<td>automatic differentiation variational inference</td>
</tr>
<tr>
<td>ARD</td>
<td>automatic relevance determination</td>
</tr>
<tr>
<td>BBVI</td>
<td>black-box variational inference</td>
</tr>
<tr>
<td>ELBO</td>
<td>evidence lower bound</td>
</tr>
<tr>
<td>GMM</td>
<td>Gaussian mixture model</td>
</tr>
<tr>
<td>HMC</td>
<td>Hamiltonian Monte Carlo</td>
</tr>
<tr>
<td>KL</td>
<td>Kullback-Leibler</td>
</tr>
<tr>
<td>MC</td>
<td>Monte Carlo</td>
</tr>
<tr>
<td>MCMC</td>
<td>Markov chain Monte Carlo</td>
</tr>
<tr>
<td>NUTS</td>
<td>no-U-turn sampler</td>
</tr>
<tr>
<td>PPCA</td>
<td>probabilistic principal component analysis</td>
</tr>
<tr>
<td>SGA</td>
<td>stochastic gradient ascent</td>
</tr>
<tr>
<td>SUP-PPCA</td>
<td>supervised probabilistic principal component analysis</td>
</tr>
<tr>
<td>VI</td>
<td>variational inference</td>
</tr>
</tbody>
</table>

### Acknowledgments

The authors acknowledge their amazing colleagues and funding sources here.
Appendix A. Transformations of Continuous Probability Densities

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

Consider a univariate (scalar) 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 \( X \) to \( Y \), then \( Y \) has probability density function

\[
f_Y(y) = f_X(T^{-1}(y)) \left| \frac{dT^{-1}(y)}{dy} \right|.
\]

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

\[
J_{T^{-1}}(Y) = \begin{pmatrix}
\frac{\partial T^{-1}_1}{\partial y_1} & \cdots & \frac{\partial T^{-1}_K}{\partial y_1} \\
\vdots & \ddots & \vdots \\
\frac{\partial T^{-1}_K}{\partial y_1} & \cdots & \frac{\partial T^{-1}_K}{\partial y_K}
\end{pmatrix}.
\]

Intuitively, the Jacobian describes how a transformation warps unit volumes across spaces. This matters for transformations of random variables, since probability density functions must always integrate to one. If the transformation is linear, then we can drop the Jacobian adjustment; it evaluates to one. Similarly, affine transformations, like elliptical standardizations, do not require Jacobian adjustments; they preserve unit volumes.

Appendix B. Transformation of the Evidence Lower Bound

Recall that \( \xi = T(\theta) \) and that the variational approximation in the real coordinate space is \( q(\xi; \phi) \).

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

\[
\mathcal{L}(\phi) = \int q(\theta) \log \left( \frac{p(\mathbf{X}, \theta)}{q(\theta)} \right) \, d\theta
\]

\[
= \int q(\xi; \phi) \log \left( \frac{p(\mathbf{X}, T^{-1}(\xi)) | \det J_{T^{-1}(\xi)} |}{q(\xi; \phi)} \right) \, d\xi
\]

\[
= \int q(\xi; \phi) \log \left( \frac{p(\mathbf{X}, T^{-1}(\xi)) | \det J_{T^{-1}(\xi)} |}{q(\xi; \phi)} \right) \, d\xi - \int q(\xi; \phi) \log [q(\xi; \phi)] \, d\xi
\]

\[
= \mathbb{E}_{q(\xi; \phi)} \left[ \log p(\mathbf{X}, T^{-1}(\xi)) + \log | \det J_{T^{-1}(\xi)} | - \log q(\xi; \phi) \right]
\]

\[
= \mathbb{E}_{q(\xi; \phi)} \left[ \log p(\mathbf{X}, T^{-1}(\xi)) + \log | \det J_{T^{-1}(\xi)} | - \log [q(\xi; \phi)] \right] + \mathbb{H}[q(\xi; \phi)].
\]
Appendix C. Gradients of the Evidence Lower Bound

First, consider the gradient with respect to the $\mu$ parameter of the standardization. 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.

\[
\nabla_{\mu} \mathcal{L} = \nabla_{\mu} \left\{ \mathbb{E}_{\mathcal{N}(\eta; 0, 1)} \left[ \log p \left( \mathbf{X}, T^{-1}(S^{-1}(\eta)) \right) + \log \left| \det J_{T^{-1}} \left( S^{-1}(\eta) \right) \right| \right] + \mathbb{E}[q(\xi; \phi)] \right\} \\
= \mathbb{E}_{\mathcal{N}(\eta; 0, 1)} \left\{ \nabla_{\mu} \left[ \log p \left( \mathbf{X}, T^{-1}(S^{-1}(\eta)) \right) + \log \left| \det J_{T^{-1}} \left( S^{-1}(\eta) \right) \right| \right] \right\} \\
= \mathbb{E}_{\mathcal{N}(\eta; 0, 1)} \left[ (\nabla_{\theta} \log p(\mathbf{X}, \theta) \nabla_{\xi} T^{-1}(\xi) + \nabla_{\xi} \log \left| \det J_{T^{-1}}(\xi) \right|) \nabla_{\mu} S^{-1}(\eta) \right] \\
= \mathbb{E}_{\mathcal{N}(\eta; 0, 1)} \left[ \nabla_{\theta} \log p(\mathbf{X}, \theta) \nabla_{\xi} T^{-1}(\xi) + \nabla_{\xi} \log \left| \det J_{T^{-1}}(\xi) \right| \right].
\]

Then, consider the gradient with respect to the mean-field $\omega$ parameter of the standardization.

\[
\nabla_{\omega} \mathcal{L} = \nabla_{\omega} \left\{ \mathbb{E}_{\mathcal{N}(\eta; 0, 1)} \left[ \log p \left( \mathbf{X}, T^{-1}(S^{-1}(\eta)) \right) + \log \left| \det J_{T^{-1}} \left( S^{-1}(\eta) \right) \right| \right] \right\} \\
+ \frac{K}{2} (1 + \log(2\pi)) + \sum_{k=1}^{K} \log(\exp(\omega_k)) \\
= \mathbb{E}_{\mathcal{N}(\eta; 0, 1)} \left\{ \nabla_{\omega} \left[ \log p \left( \mathbf{X}, T^{-1}(S^{-1}(\eta)) \right) + \log \left| \det J_{T^{-1}} \left( S^{-1}(\eta) \right) \right| \right] \right\} + 1 \\
= \mathbb{E}_{\mathcal{N}(\eta; 0, 1)} \left[ (\nabla_{\theta} \log p(\mathbf{X}, \theta) \nabla_{\xi} T^{-1}(\xi) + \nabla_{\xi} \log \left| \det J_{T^{-1}}(\xi) \right|) \nabla_{\omega} S^{-1}(\eta) \right] + 1 \\
= \mathbb{E}_{\mathcal{N}(\eta; 0, 1)} \left[ \nabla_{\theta} \log p(\mathbf{X}, \theta) \nabla_{\xi} T^{-1}(\xi) + \nabla_{\xi} \log \left| \det J_{T^{-1}}(\xi) \right| \eta^{\top} \text{diag}(\exp(\omega)) \right] + 1.
\]

Finally, consider the gradient with respect to the full-rank $L$ parameter of the standardization.

\[
\nabla_{L} \mathcal{L} = \nabla_{L} \left\{ \mathbb{E}_{\mathcal{N}(\eta; 0, 1)} \left[ \log p \left( \mathbf{X}, T^{-1}(S^{-1}(\eta)) \right) + \log \left| \det J_{T^{-1}} \left( S^{-1}(\eta) \right) \right| \right] \right\} \\
+ \frac{K}{2} (1 + \log(2\pi)) + 1 \log \left| \det(LL^{\top}) \right| \\
= \mathbb{E}_{\mathcal{N}(\eta; 0, 1)} \left\{ \nabla_{L} \left[ \log p \left( \mathbf{X}, T^{-1}(S^{-1}(\eta)) \right) + \log \left| \det J_{T^{-1}} \left( S^{-1}(\eta) \right) \right| \right] \right\} \\
+ \nabla_{L} \frac{1}{2} \log \left| \det(LL^{\top}) \right| \\
= \mathbb{E}_{\mathcal{N}(\eta; 0, 1)} \left[ (\nabla_{\theta} \log p(\mathbf{X}, \theta) \nabla_{\xi} T^{-1}(\xi) + \nabla_{\xi} \log \left| \det J_{T^{-1}}(\xi) \right|) \nabla_{L} S^{-1}(\eta) \right] + (L^{-1})^{\top} \\
= \mathbb{E}_{\mathcal{N}(\eta; 0, 1)} \left[ (\nabla_{\theta} \log p(\mathbf{X}, \theta) \nabla_{\xi} T^{-1}(\xi) + \nabla_{\xi} \log \left| \det J_{T^{-1}}(\xi) \right|) \eta^{\top} \right] + (L^{-1})^{\top}.
\]

Appendix D. Automating Expectations: Monte Carlo Integration

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

\[
\mathbb{E}_{q(\eta)}[f(\eta)] = \int f(\eta)q(\eta) \, d\eta \approx \frac{1}{S} \sum_{s=1}^{S} f(\eta_s), \text{ where } \eta_s \sim q(\eta).
\]
MC integration provides noisy, yet unbiased, estimates of the integral. The standard deviation of the estimates are of order $1/\sqrt{S}$.

**Appendix E. Running ADVI in Stan**

Visit [http://mc-stan.org/](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. The syntax is where `myData.data.R` is the dataset stored in the R language Rdump format.

```plaintext
./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**

`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). We describe the model below. Stan code is in Figure 17.

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

$$p(y | X, w, \sigma) = \prod_{n=1}^{N} \mathcal{N}(y_n | w^T x_n, \sigma)$$

describes measurements corrupted by iid Gaussian noise with unknown standard deviation $\sigma$.

The ARD prior and hyper-prior structure is as follows

$$p(w, \sigma, \alpha) = p(w, \sigma | \alpha) p(\alpha)$$

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

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

We simulate data such that only half the regressions have predictive power. The results in Figure 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 models structured datasets in an intuitive way. We study a model of voting preferences from the 1988 United States presidential election. Chapter 14.1 of (Gelman and
Hill, 2006) motivates the model and explains the dataset. We also describe the model below. Stan code is in Figure 18, based on (Stan Development Team, 2015).

\[
\Pr(y_n = 1) = \text{sigmoid}\left(\beta^0 + \beta^\text{female} \cdot \text{female}_n + \beta^\text{black} \cdot \text{black}_n + \beta^\text{female.black} \cdot \text{female.black}_n \\
+ \alpha^\text{age}_k[n] + \alpha^\text{edu}_l[n] + \alpha^\text{age.edu}_k[l][n] + \alpha^\text{state}_j[n]\right)
\]

\[
\alpha^\text{state}_j \sim \mathcal{N}\left(\alpha^\text{region}_m[j] + \beta_v^\text{prev} \cdot \text{v.prev}_j, \sigma^\text{state}_j\right).
\]

The hierarchical variables are

\[
\alpha^\text{age}_k \sim \mathcal{N}(0, \sigma^\text{age}) \text{ for } k = 1, \ldots, K
\]
\[
\alpha^\text{edu}_l \sim \mathcal{N}(0, \sigma^\text{edu}) \text{ for } l = 1, \ldots, L
\]
\[
\alpha^\text{age.edu}_k[l] \sim \mathcal{N}(0, \sigma^\text{age.edu}) \text{ for } k = 1, \ldots, K, l = 1, \ldots, L
\]
\[
\alpha^\text{region}_m \sim \mathcal{N}(0, \sigma^\text{region}) \text{ for } m = 1, \ldots, M.
\]

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

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

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

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

1. For each user \( u \) in \( \{1, \cdots, U\} \):
   - For each component \( k \), draw \( \theta_{uk} \sim \text{Gam}(a_0, b_0) \).

2. For each item \( i \) in \( \{1, \cdots, I\} \):
   - For each component \( k \), draw \( \beta_{ik} \sim \text{Gam}(c_0, d_0) \).

3. For each user and item:
   - Draw the observation \( y_{ui} \sim \text{Poisson}(\theta_u^T \beta_i) \).

A potential downfall of this model is that it is not uniquely identifiable: swapping rows and columns of \( \theta \) and \( \beta \) 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 \( \theta_u \) vector in our model in this fashion. Stan code is in Figure 19. We set \( K = 10 \) and all the Gamma hyper-parameters to 1 in our experiments.
F.4 Non-negative Matrix Factorization: Dirichlet Exponential Poisson Model

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

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

1. For each user $u$ in $\{1, \cdots, U\}$:
   - Draw the $K$-vector $\theta_u \sim \text{Dir}(\alpha_0)$.

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 $y_{ui} \sim \text{Poisson}(\theta_u^T \beta_i)$.

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

F.5 Gaussian Mixture Model

The GMM is a celebrated probability model. 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.

The images are in $Y = y_{1:N}$ where each $y_n$ is $D$-dimensional and there are $N$ observations. The likelihood for the images is

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

with a Dirichlet prior for the mixture proportions

$$p(\theta) = \text{Dir}(\theta \mid \alpha_0).$$

a Gaussian prior for the mixture means

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

and a lognormal prior for the mixture standard deviations

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

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 $\alpha_0$ to a small value encourages the model to use fewer components to explain the data. Larger values of $\alpha_0$ encourage the model to use all $K$ components. We set $\alpha_0 = 1000$ in our experiments.

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

F.6 Probabilistic Principal Component Analysis with Automatic Relevance Determination

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

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

$$p(Z) = \prod_{n=1}^{N} \mathcal{N}(z_n : 0, I).$$

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

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

Finally define the likelihood through an inner product as

$$p(X | W, Z, \sigma) = \prod_{n=1}^{N} \mathcal{N}(x_n : Wz_n, \sigma I).$$

The standard deviation $\sigma$ 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 $\alpha$ which chooses which principal components to retain. ($M < D$ now represents the maximum number of principal components to consider.) The extended extends the above by

$$p(\alpha) = \prod_{m=1}^{M} \text{InvGamma}(\alpha_m : 1, 1)$$

$$p(W | \alpha) = \prod_{d=1}^{D} \mathcal{N}(w_d : 0, \sigma \text{diag}(\alpha))$$

$$p(X | W, Z, \sigma) = \prod_{n=1}^{N} \mathcal{N}(x_n : Wz_n, \sigma I).$$

ADVI code is in Figure 23.
F.7 Supervised Probabilistic Principal Component Analysis with Automatic Relevance Determination

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

\[
p(Z) = \prod_{n=1}^{N} \mathcal{N}(z_n; 0, I)
\]

\[
p(\sigma) = \text{logNormal}(\sigma; 0, 1)
\]

\[
p(\alpha) = \prod_{m=1}^{M} \text{InvGamma}(\alpha_m; 1, 1)
\]

\[
p(W_X | \alpha) = \prod_{d=1}^{D} \mathcal{N}(w_d; 0, \sigma \text{diag}(\alpha))
\]

\[
p(w_y | \alpha) = \mathcal{N}(w_y; 0, \sigma \text{diag}(\alpha))
\]

\[
p(X | W_X, Z, \sigma) = \prod_{n=1}^{N} \mathcal{N}(x_n; W_X z_n, \sigma I)
\]

\[
p(y | w_y, Z, \sigma) = \prod_{n=1}^{N} \mathcal{N}(y_n; w_y z_n, \sigma).
\]

ADVI code is in Figure 24.
data {
  int<lower=0> N; // number of data items
  int<lower=0> D; // dimension of input features
  matrix[N, D] x; // input matrix
  vector[N] y; // output vector

  // hyperparameters for Gamma priors
  real<lower=0> a0;
  real<lower=0> b0;
  real<lower=0> c0;
  real<lower=0> D0;
}

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

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

model {
  // alpha: hyper-prior on weights
  alpha ~ gamma(c0, D0);

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

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

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

Figure 17: Stan code for Linear Regression with Automatic Relevance Determination.
```
data {
  int<lower=0> N;
  int<lower=0> n_age;
  int<lower=0> n_age_edu;
  int<lower=0> n_edu;
  int<lower=0> n_region_full;
  int<lower=0> n_state;
  int<lower=0,upper=n_age> age[N];
  int<lower=0,upper=n_age_edu> age_edu[N];
  vector<lower=0,upper=1>[N] black;
  int<lower=0,upper=n_edu> edu[N];
  vector<lower=0,upper=1>[N] female;
  int<lower=0,upper=n_region_full> region_full[N];
  int<lower=0,upper=n_state> state[N];
  vector[N] v_prev_full;
  int<lower=0,upper=1> y[N];
}

parameters {
  vector[n_age] a;
  vector[n_edu] b;
  vector[n_age_edu] c;
  vector[n_state] d;
  vector[n_region_full] e;
  vector[5] beta;
  real<lower=0,upper=100> sigma_a;
  real<lower=0,upper=100> sigma_b;
  real<lower=0,upper=100> sigma_c;
  real<lower=0,upper=100> sigma_d;
  real<lower=0,upper=100> sigma_e;
}

transformed parameters {
  vector[N] y_hat;

  for (i in 1:N)
                  + beta[3] * female[i]
                  + beta[4] * v_prev_full[i]
                  + a[age[i]]
                  + b[edu[i]]
                  + c[age_edu[i]]
                  + d[state[i]]
                  + e[region_full[i]];
}

model {
  a ~ normal (0, sigma_a);
  b ~ normal (0, sigma_b);
  c ~ normal (0, sigma_c);
  d ~ normal (0, sigma_d);
  e ~ normal (0, sigma_e);
  beta ~ normal(0, 100);
  y ~ bernoulli_logit(y_hat);
}
```

Figure 18: Stan code for Hierarchical Logistic Regression, from (Stan Development Team, 2015).
data {
    int<lower=0> U;
    int<lower=0> I;
    int<lower=0> K;
    int<lower=0> y[U, I];
    real<lower=0> a;
    real<lower=0> b;
    real<lower=0> c;
    real<lower=0> D;
}

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

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

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

Figure 19: Stan code for Gamma Poisson non-negative matrix factorization model.
data {
  int<lower=0> U;
  int<lower=0> I;
  int<lower=0> K;
  int<lower=0> y[U, I];
  real<lower=0> lambda0;
  real<lower=0> alpha0;
}

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

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

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

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

**Figure 20:** Stan code for 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(0.0, 1.0);
    sigma[k] ~ lognormal(0.0, 1.0);
  }

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

Figure 21: ADVI Stan code for the GMM example.
functions {  
    real divide_promote_real(int x, int y) {  
        real x_real;  
        x_real <- x;  
        return x_real / y;  
    }  
}  

data {  
    int<lower=0> NFULL;  // total number of datapoints in dataset  
    int<lower=0> N; // number of data points in minibatch  
    int<lower=0> K; // number of mixture components  
    int<lower=0> D; // dimension  
    vector[D] yFULL[NFULL]; // dataset  
    vector[D] y[N]; // minibatch  
    real<lower=0> alpha0; // dirichlet hyper-prior parameter  
}  

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

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

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

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

Figure 22: ADVI Stan code for the GMM example, with stochastic subsampling of the dataset.
Figure 23: ADV1 Stan code for the PPCA with ARD.
data {
  int<lower=0> N;       // number of data points in dataset
  int<lower=0> D;       // dimension
  int<lower=0> M;       // maximum dimension of latent space to consider
  vector[D] x[N];
  vector[N] y;
}

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

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

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

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

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

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

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

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


