A Guide to Black Box Variational Inference for Gamma Distributions

Allison J.B. Chaney

October 27, 2015

Black box variational inference (BBVI) (Ranganath et al., 2014) is a promising approach to statistical inference. It allows practitioners to avoid long derivations of updates of traditional variational inference (Wainwright and Jordan, 2008), and it can be made stochastic (Hoffman et al., 2013) to scale to millions, if not more, observed data points. For models that are non-conjugate, there are variational inference approaches that work for certain classes of models (Wang and Blei, 2013), but BBVI works for the broadest set of exponential family models.

Despite its promise, BBVI can be difficult to implement for variables with certain kinds of distributions. One example of of a difficult distribution is the gamma distribution, because of its "spike and slab" nature—the gamma often has a density with high probability near zero (the spike), and low probability for higher numerical values (the slab).

This guide will work through the theory of BBVI for a simple model that involves gamma distributed variables. It will cover practical tips so that the reader should be able to implement BBVI for this simple model. This should hopefully help readers apply BBVI to more complicated models.

Thanks to Rajesh Ranganath for his advice, which made this document and my own understanding possible.

1 A Simple Gamma Model

We would like the simplest model that has a gamma-distributed latent variable (as opposed to a gamma-distributed observed variable).

Since we would like to ensure that BBVI works for multiple values that parameterize that gamma, we incorporate a variety of values into the model itself. We can imagine generating data as follows.

- For $k = 1, \dots, K$:
  - Draw mean parameter $\mu_k \sim \text{Gamma}(\alpha_0, \mu_0/\alpha_0)$
- For $n = 1, \dots, N$:
  - For $k = 1, \dots, K$:
    - Draw observation $x_{nk} \sim \mathcal{N}(\mu_k, 1)$

Data can be generated with the following Python script. I generated data using $K = 12$ and $N = 1000$ using prior sparsity $\alpha_0 = 0.1$ and mean $\mu_0 = 5$.

```
import numpy as np

np.random.seed(11)

alpha_0 = 0.1
mu_0 = 5
K = 12
N = 1000

mu = np.random.gamma(alpha_0, mu_0 / alpha_0, K)

x = np.random.normal(mu, 1, (N,K))
```

This brings us to our first trick: how to specify gamma models to be used with BBVI.

> **Gamma parameterization**. While the shape/rate parameterization is useful in other contexts, the shape/scale parameterization should be used for gamma distributed variables being fit with BBVI as it makes the variables much easier to fit. Additionally, the scale parameter should be set to the mean over the shape (μ/α). This has the added benefit of being more interpretable than either of the two standard parameterizations: α controls for sparsity and μ is the mean.

Moving forward, x will be our observations, and we will used BBVI to determine the values of α_k and μ_k for all k.

2 BBVI for the Simple Model

Variational inference approaches the problem of posterior inference by minimizing the KL divergence from an approximating distribution q to the true posterior p. This is equivalent to maximizing the ELBO:

$$\mathscr{L}(q) = \mathbb{E}_{q(\mu)}[\log p(x,\mu) - \log q(\mu)].$$

We define the approximating distribution q using the mean field assumption:

$$q(\mu) = \prod_k q(\mu_k).$$

The variational distributions $q(\mu_k)$ are gamma-distributed with free variational parameters sparsity λ_k^α and mean λ_k^μ for each respective index k.

> **Constrain free parameters**. We use the soft-plus function $\mathscr{P}(x) = \log(1+\exp(x))$ to constrain the free parameters to be positive, so they do not violate the requirements of the gamma distribution.[1]

The expectations under q, which are needed to maximize the ELBO, do not have a simple analytic form, so here is where black box techniques enter the picture. For each variable, we can write the log probability of all terms containing that variable, giving us

$$\log p_k^\mu(x,\mu) \triangleq \log p(\mu_k \mid \alpha_0, \mu_0) + \sum_{n=1}^{N} \log p(x_{nk} \mid \mu_k).$$

Then we can write the gradients with respect to the variational parameters as:

$$\nabla_{\lambda_k^\alpha} \mathscr{L} = \mathbb{E}_q\left[\nabla_{\lambda_i^\alpha} \log q(\mu_k \mid \lambda_k^\alpha, \lambda_k^\mu)\big(\log p_k^\mu(x,\mu) - \log q(\mu_k \mid \lambda_k^\alpha, \lambda_k^\mu)\big)\right],$$

[1] An alternative to this would be to just to exponentiate the free parameters. This seems appealing in its simplicity, but it doesn't work as well, so is not recommended.

and
$$\nabla_{\lambda_k^\mu}\mathscr{L} = \mathbb{E}_q\left[\nabla_{\lambda_i^\mu}\log q(\mu_k \mid \lambda_k^\alpha, \lambda_k^\mu)\big(\log p_k^\mu(x,\mu) - \log q(\mu_k \mid \lambda_k^\alpha, \lambda_k^\mu)\big)\right].$$

Using this framework, we construct our black box algorithm below in Algorithm 1.

Algorithm 1: Inference for Simple Gamma Model

```
Input: observations $x$
Output: estimates of latent parameters $\mu$
Initialize $\lambda^\alpha$ and $\lambda^\mu$
Initialize iteration count $t = 0$
while change in validation likelihood $< \delta$ do
    for each sample $s = 1, \dots, S$ do
        for each component $k = 1, \dots, K$ do
            draw sample mean $\mu_k[s] \sim \text{Gamma}(\mathscr{P}(\lambda_k^\alpha), \mathscr{P}(\lambda_k^\mu)/\mathscr{P}(\lambda_k^\alpha))$
            set $p_k[s] = \log p(\mu_k[s] \mid \alpha_0, \mu_0)$                                  // see Eqn. 2
            set $q_k[s] = \log q(\mu_k[s] \mid \lambda_k^\alpha, \lambda_k^\mu)$                   // see Eqn. 3
            set $g_k^\alpha[s] = \nabla_{\lambda_k^\alpha} \log q(\mu_k[s] \mid \lambda_k^\alpha, \lambda_k^\mu)$   // see Eqn. 7
            set $g_k^\mu[s] = \nabla_{\lambda_k^\mu} \log q(\mu_k[s] \mid \lambda_k^\alpha, \lambda_k^\mu)$       // see Eqn. 8
        end
        for each observation $n = 1, \dots, N$ do
            for each component $k = 1, \dots, K$ do
                update $p_k[s] \mathrel{+}= \log p(x_{nk} \mid \mu_k[s])$                          // see Eqn. 9
            end
        end
    end
    set $\rho = (t + \tau)^\kappa$
    for each component $k = 1, \dots, K$ do
        set $\hat{\nabla}_{\lambda_k^\alpha}\mathscr{L} \triangleq \frac{1}{S}\sum_{s=1}^S g_k^\alpha[s](p_k[s] - q_k[s])$
        set $\hat{\nabla}_{\lambda_k^\mu}\mathscr{L} \triangleq \frac{1}{S}\sum_{s=1}^S g_k^\mu[s](p_k[s] - q_k[s])$
        set $\lambda^\alpha \mathrel{+}= \rho\hat{\nabla}_{\lambda_k^\alpha}\mathscr{L}$
        set $\lambda^\mu \mathrel{+}= \rho\hat{\nabla}_{\lambda_k^\mu}\mathscr{L}$
    end
end
for each component $k = 1, \dots, K$ do
    set $\mathbb{E}[\mu_k] = \lambda_k^\mu$
end
return $\mathbb{E}[\mu]$
```

Just for convenience of implementation, we write some of the values to be computed in their full forms. We begin with the general log form of the gamma distribution with our atypical parameterization.

$$\log \text{Gamma}(x \mid a, m) = a\log a - a\log m - \log\Gamma(a) + (a-1)\log x - \frac{ax}{m} \tag{1}$$

Next we define the q and p values for each sample, using Eq. 1.

$$\log p(\mu_k[s] \mid \alpha_0, \mu_0) = \log \text{Gamma}(\mu_k[s] \mid \alpha_0, \mu_0) \tag{2}$$
$$\log q(\mu_k[s] \mid \lambda_k^\alpha, \lambda_k^\mu) = \log \text{Gamma}(\mu_k[s] \mid \mathscr{P}(\lambda_k^\alpha), \mathscr{P}(\lambda_k^\mu)) \tag{3}$$

We can also define the general form of the log gamma gradients.

$$\nabla_a \log \text{Gamma}(x \mid a, m) = \log a + 1 - \log m - \Psi(a) + \log x - \frac{x}{m} \tag{4}$$

$$\nabla_m \log \text{Gamma}(x \mid a, m) = -\frac{a}{m} - \frac{ax}{m^2} \tag{5}$$

We also need the general form for the derivative of the soft-plus function.

$$\mathscr{P}'(x) = \frac{e^x}{1+e^x} \tag{6}$$

Now we can write the gradients with respect the the free variational parameters. Eq. 7 relies on Eqs. 4 and 6; Eq. 8 relies on Eqs. 5 and 6.

$$\nabla_{\lambda_k^\alpha} \log q(\mu_k[s] \mid \lambda_k^\alpha, \lambda_k^\mu) = \mathscr{P}'(\lambda_k^\alpha) \nabla_{\lambda_k^\alpha} \log \text{Gamma}(\mu_k[s] \mid \mathscr{P}(\lambda_k^\alpha), \mathscr{P}(\lambda_k^\mu)) \tag{7}$$

$$\nabla_{\lambda_k^\alpha} \log q(\mu_k[s] \mid \lambda_k^\alpha, \lambda_k^\mu) = \mathscr{P}'(\lambda_m^\alpha) \nabla_{\lambda_k^\mu} \log \text{Gamma}(\mu_k[s] \mid \mathscr{P}(\lambda_k^\alpha), \mathscr{P}(\lambda_k^\mu)) \tag{8}$$

Finally, we need the log probability of seeing a data point given the sampled parameters, which is just the log of the Gaussian distribution with unit variance.

$$\log p(x_{nk} \mid \mu_k[s]) = -\frac{1}{2}(x_{nk} - \mu_k[s])^2 - \log \sqrt{2\pi} \tag{9}$$

This algorithm implemented in Python is found below.

```python
import numpy as np
import sys
from scipy.special import gammaln, digamma
np.random.seed(11)

# load or generate data (as shown above) here

## helper functions
# soft-plus
def SP(x):
    return np.log(1. + np.exp(x))

# derivative of soft-plus
def dSP(x):
    return np.exp(x) / (1. + np.exp(x))

# inverse of soft-plus
def iSP(x):
    return np.log(np.exp(x) - 1.)

# log probability of a gamma given sparsity a and mean m
def Gamma(x, a, m):
    p = a * np.log(a) - a * np.log(m) - gammaln(a) + \
        (a-1.) * np.log(x) - a * x / m
    return p

# derivative of above log gamma with respect to sparsity a
def dGamma_alpha(x, a, m):
    p = np.log(a) + 1. - np.log(m) - digamma(a) + \
        np.log(x) - (x / m)
    return p

# derivative of above log gamma with respect to mean m
def dGamma_mu(x, a, m):
    return - (a / m) + ((a * x) / m**2)

# log probabilty of a Normal distribution with unit
# variance and mean m
def Normal(x, m):
    return - 0.5 * (x - m)**2 - np.log(np.sqrt(2 * np.pi))

# covariance where each column is draws for a variable
# (returns a row of covariances)
def cov(a, b):
    v = (a - sum(a)/a.shape[0]) * (b - sum(b)/b.shape[0])
    return sum(v)/v.shape[0]

# variance with same structure as covariance above
def var(a):
    return cov(a,a)

## set up for inference
# the number of samples to draw for each parameter
S = 1024

# initialize; randomness not needed here, but may be for
# other models
lambda_alpha = 0.1 * np.ones(K)
lambda_mu = 0.01 * np.ones(K)
# lambda_mu = 0.01 * np.exp(np.random.normal(0, 1, K)))

iteration = 0

# used for RMSprop
MS_mu = np.zeros(K)
MS_alpha = np.zeros(K)

# set up log file
logf = open("log.tsv", 'w+')
logf.write("iteration\tcomponent\tmean\tsparsity\n")

# log the truth used to simulate the data
for k in range(K):
    logf.write("-1\t%d\t%f\t%f\n" % (k, mu[k], 0))

# log the initial conditions
for k in range(K):
    logf.write("%d\t%d\t%f\t%f\n" % (iteration, k, \
        SP(lambda_mu)[k], SP(lambda_alpha)[k]))

# run BBVI for a fixed number of iterations
while (iteration < 100):
    sample_mus = np.random.gamma(SP(lambda_alpha), \
        SP(lambda_mu) / SP(lambda_alpha), (S,K))

    # truncate samples (don't sample zero)
    sample_mus[sample_mus < 1e-300] = 1e-300

    # probability of samples given prior
    p = Gamma(sample_mus, alpha_0, mu_0)

    # probability of samples given variational parameters
    q = Gamma(sample_mus, SP(lambda_alpha), SP(lambda_mu))

    # gradients of variational parameters sparsity (alpha)
    # and mean (mu) given samples
    g_alpha = dSP(lambda_alpha) * \
        dGamma_alpha(sample_mus, SP(lambda_alpha), \
        SP(lambda_mu))
    g_mu = dSP(lambda_mu) * dGamma_mu(sample_mus, \
        SP(lambda_alpha), SP(lambda_mu))

    # probability of observations given samples
    for i in xrange(N):
        p += Normal(x[i], sample_mus)

    # control variates to decrease variance of gradient;
    # one for each variational parameter
    cv_alpha = cov(g_alpha * (p - q), g_alpha) / \
        var(g_alpha)
    cv_mu = cov(g_mu * (p - q), g_mu) / var(g_mu)

    # RMSprop: keep running average of gradient magnitudes
    # (the gradient will be divided by sqrt of this later)
    if MS_mu.all() == 0:
        MS_mu = (g_mu**2).sum(0)
        MS_alpha = (g_alpha**2).sum(0)
    else:
        MS_mu = 0.9 * MS_mu + 0.1 * (g_mu**2).sum(0)
        MS_alpha = 0.9 * MS_alpha + \
            0.1 * (g_alpha**2).sum(0)

    # Robbins-Monro sequence for step size
    rho = (iteration + 1024) ** -0.7

    # update each variational parameter with smaple average
    lambda_mu += rho * (1. / S) * \
        (g_mu / np.sqrt(MS_mu) * (p - q - cv_mu)).sum(0)
    lambda_alpha += rho * (1. / S) * \
        (g_alpha / np.sqrt(MS_alpha) * \
        (p - q - cv_alpha)).sum(0)

    # truncate variational parameters
    lambda_alpha[lambda_alpha < iSP(0.005)] = iSP(0.005)
    lambda_mu[lambda_mu < iSP(1e-5)] = iSP(1e-5)
    lambda_alpha[lambda_alpha > \
        iSP(np.log(sys.float_info.max))] = \
138             iSP(np.log(sys.float_info.max))
139         lambda_mu[lambda_mu > \
140             iSP(np.log(sys.float_info.max))] = \
141             iSP(np.log(sys.float_info.max))
142 
143         iteration += 1
144 
145         # log this iteration
146         for k in range(K):
147             logf.write("%d\t%d\t%f\t%f\n" % (iteration, k, \
148                 SP(lambda_mu)[k], SP(lambda_alpha)[k]))
```

There are a few implementation tricks worth highlighting.

> **Initialization**. It is recommended that the variational parameters be initialized in unconstrained space. The sparsity parameter should be set around 0.1 (line 58) and the mean parameter should be around 0.01 (line 59). This model does not require random noise in initialization because it is not a membership model of any kind, but commented out line 60 shows how to do a random initialization for models that require it. For membership models, the sparsity parameters may need to be held fixed until the mean parameters settle a little.

> **Truncation**. There are two recommended truncations: one for sampled variables and another for free parameters. For sampled parameters (e.g., line 87), the objective is to avoid sampling too close to zero—values that are too small can be cast as zero due to underflow, which causes problems as zero is not within the support of the Gamma.
>
> The second application of truncation is after the free parameters are updated (lines 134–141). Here, there are two objectives: the first is to prevent the values from becoming too low (lines 134 and 135) and the second is to prevent overflow when constraining the free parameters (lines 136–141). For the first objective, if the sparsity becomes too low, the variance of samples can become incredibly high; this is why the truncation is higher for the sparsity than for the mean. If the mean becomes too low, we care less about the actual value and usually just interpret the parameter as being zero.

> **RMSprop**. To help control the size of the gradient, we can use RMSprop,[2] which means dividing the gradient for a parameter by a running average of the magnitudes of recent gradients for that parameter. See lines 65–66, 115–121, 128, and 130 to see how RMSprop is implemented for this model.

3 Results

Here we show the results on the simulated data with the settings described previously; each estimate converges to the true value as shown in Figure 1. The script to generate this plot is shown at the end of the section.

There is much more that can be explored now, including the showing that ELBO increases and converges. It would also be useful to have an exploration of how the number of observed data points or number of samples impacts convergence. The stochastic variant should also be explored—the basic idea is to subsample data and scale it's probability of being observed (line 105) by the number of true data points divided by the number of sampled data points. We will not cover these concepts here, but hopefully the reader has the tools and understanding to explore them on his or her own, and apply BBVI to gamma-distributed variables in more complex and interesting models.

[2]http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf

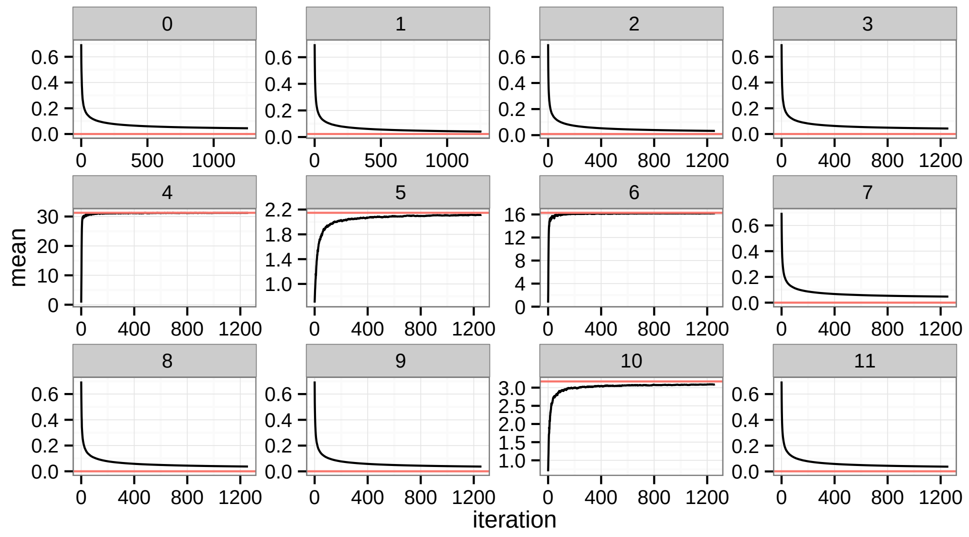

Figure 1: The convergence of the estimated parameters to the true values for twelve gamma-distributed variables.

```
library(ggplot2)
# setwd('insert your working dir here, if needed')

dat <- read.csv('log.tsv', sep='\t')

# separate truth and estimates
truth <- dat[dat$iteration==-1,]
dat <- dat[dat$iteration!=-1,]

# set up plot
p <- ggplot(dat, aes(x=iteration, y=mean))
p <- p + facet_wrap(~ component, scales="free")
p <- p + geom_line()
p <- p + geom_hline(data=truth,
            aes(yintercept=mean,color="red"))
p + theme_bw()
```

References

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

Ranganath, R., Gerrish, S., and Blei, D. M. (2014). Black box variational inference. In *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*, AISTATS '14, pages 814–822.

Wainwright, M. J. and Jordan, M. I. (2008). Graphical models, exponential families, and variational inference. *Found. Trends Mach. Learn.*, 1(1-2):1–305.

Wang, C. and Blei, D. M. (2013). Variational inference in nonconjugate models. *JMLR*.