A practical tutorial on Variational Bayes

Minh-Ngoc Tran[*]
University of Sydney Business School

Abstract

This tutorial gives a quick introduction to Variational Bayes (VB), also called Variational Inference or Variational Approximation, from a practical point of view. The aim is that the reader can quickly derive and implement their first VB algorithm for Bayesian inference with their data analysis problem. The Matlab code and the data used in the examples can be found at `https://github.com/VBayesLab/Tutorial-on-VB`.

1 Introduction

The writing of this tutorial started when an honours student of mine, who didn't have a strong background in computational statistics, asked for some materials to help them quickly learn about Variational Bayes and be able to implement VB to fit their model. There were many excellent tutorials and review papers on VB, however, most of them were either too abstract or tangential to the statistics readership, and didn't offer much hands-on experience. So I decided to write my own note for the student, and this present tutorial is based on that note.

Let y be the data and $p(y|\theta)$ the likelihood function based on a postulated model, with $\theta \in \Theta$ the vector of model parameters to be estimated. Let $p(\theta)$ be the prior. Bayesian inference encodes all the available information about the model parameter θ in its posterior distribution with density

$$p(\theta|y) = \frac{p(y,\theta)}{p(y)} = \frac{p(\theta)p(y|\theta)}{p(y)} \propto p(\theta)p(y|\theta),$$

where $p(y) = \int_\Theta p(\theta)p(y|\theta)d\theta$, called the *marginal likelihood* or *evidence*. Here, the notation '$\propto$' means proportional up to the normalizing constant that is independent of the parameter (θ). In most of Bayesian derivations, such a constant can be safely ignored. Bayesian inference typically requires computing expectations with respect to the posterior distribution. However, it is often difficult to compute such expectations, partly because the density $p(\theta|y)$ itself is intractable as the normalizing constant $p(y)$ is often unknown. For many applications, Bayesian inference is performed using Markov Chain Monte Carlo (MCMC), which estimates expectations w.r.t. $p(\theta|y)$ by sampling from it. For other applications where θ is high dimensional or fast computation is of primary interest, VB is an attractive alternative to MCMC. VB approximates the posterior $p(\theta|y)$ by a probability distribution with density $q(\theta)$ belonging to some tractable family of distributions $\mathcal{Q}$

[*]minh-ngoc.tran@sydney.edu.au

such as Gaussians. The best VB approximation $q^* \in \mathcal{Q}$ is found by minimizing the Kullback-Leibler (KL) divergence to $p(\theta|y)$ from $q(\theta)$

$$q^* = \arg\min_{q\in\mathcal{Q}} \left\{ \text{KL}\big(q\|p(\cdot|y)\big) := \int q(\theta)\log\frac{q(\theta)}{p(\theta|y)}d\theta \right\}. \tag{1}$$

Then, Bayesian inference is performed with the intractable posterior $p(\theta|y)$ replaced by the tractable VB approximation $q^*(\theta)$. It is easy to see that

$$\text{KL}(q\|p(\cdot|y)) = -\int q(\theta)\log\frac{p(\theta)p(y|\theta)}{q(\theta)}d\theta + \log p(y),$$

thus minimizing KL is equivalent to maximizing the lower bound on $\log p(y)$

$$\text{LB}(q) := \int q(\theta)\log\frac{p(\theta)p(y|\theta)}{q(\theta)}d\theta. \tag{2}$$

Without any constraint on $\mathcal{Q}$, the solution to (1) is $q^*(\theta)=p(\theta|y)$; of course this solution is useless as it is itself intractable. Depending on the constraint imposed on the class $\mathcal{Q}$, VB algorithms can be categorized into two classes: Mean Field VB and Fixed Form VB which are presented in Section 2 and Section 3, respectively. These two sections can be read completely separately depending on the reader's interest.

2 Mean Field Variational Bayes

Let's write θ as $\theta=(\theta_1^\top,\theta_2^\top)^\top$. Here $a^\top$ denotes the transpose of vector a; and all vectors in this note are column vectors. MFVB assumes the following factorization form for q

$$q(\theta) = q_1(\theta_1)q_2(\theta_2),$$

i.e., we ignore the posterior dependence between θ_1, θ_2 and attempt to approximate $p(\theta_1,\theta_2|y)$ by $q(\theta)=q_1(\theta_1)q_2(\theta_2)$. This is the only assumption/restriction we put on the class $\mathcal{Q}$. The lower bound in (2) is

$$\begin{aligned}
\text{LB}(q_1,q_2) &= \int q_1(\theta_1)q_2(\theta_2)\log\frac{p(\theta,y)}{q_1(\theta_1)q_2(\theta_2)}d\theta_1 d\theta_2 \\
&= \int q_1(\theta_1)q_2(\theta_2)\log p(\theta,y)d\theta_1 d\theta_2 \\
&\quad - \int q_1(\theta_1)\log q_1(\theta_1)d\theta_1 - \int q_2(\theta_2)\log q_2(\theta_2)d\theta_2 \\
&= \int q_1(\theta_1)\mathbb{E}_{-\theta_1}[\log p(y,\theta)]d\theta_1 - \int q_1(\theta_1)\log q_1(\theta_1)d\theta_1 + C_2(q_2)
\end{aligned}$$

where $\mathbb{E}_{-\theta_1}[\log p(y,\theta)] := \mathbb{E}_{q_2(\theta_2)}[\log p(y,\theta)] = \int q_2(\theta_2)\log p(y,\theta)d\theta_2$ and $C_2(q_2)$ is the term independent of q_1. Hence,

$$\begin{aligned}
\mathrm{LB}(q_1,q_2) &= \int q_1(\theta_1)\log\frac{\exp\left(\mathbb{E}_{-\theta_1}[\log p(y,\theta)]\right)}{q_1(\theta_1)}d\theta_1 + C_2(q_2)\\
&= \int q_1(\theta_1)\log\frac{\widetilde{q}_1(\theta_1)}{q_1(\theta_1)}d\theta_1 + C_2(q_2) + \log\widetilde{C}_2(q_2)\\
&= -\mathrm{KL}(q_1\|\widetilde{q}_1) + C(q_2) + \log\widetilde{C}_2(q_2), \qquad (3)
\end{aligned}$$

where $\widetilde{q}_1(\theta_1)$ is the probability density function determined by

$$\widetilde{q}_1(\theta_1) := \frac{\exp(\mathbb{E}_{-\theta_1}[\log p(y,\theta)])}{\widetilde{C}_2(q_2)} \propto \exp(\mathbb{E}_{-\theta_1}[\log p(y,\theta)]), \text{ with } \widetilde{C}_2(q_2) := \int \exp(\mathbb{E}_{-\theta_1}[\log p(y,\theta)])d\theta_1.$$

Similarly,

$$\mathrm{LB}(q_1,q_2) = -\mathrm{KL}(q_2\|\widetilde{q}_2) + \text{ constant dependent only on } q_1, \qquad (4)$$

where $\widetilde{q}_2(\theta_2) \propto \exp(\mathbb{E}_{-\theta_2}[\log p(y,\theta)])$ with $\mathbb{E}_{-\theta_2}[\log p(y,\theta)] := \int q_1(\theta_1)\log p(y,\theta)d\theta_1$. The expressions in (3)-(4) suggest a coordinate ascent optimization procedure for maximizing the lower bound: given q_2, we minimize $\mathrm{KL}(q_1\|\widetilde{q}_1)$ to find q_1, and given q_1 we minimize $\mathrm{KL}(q_2\|\widetilde{q}_2)$ to find q_2. The hope is that solving the optimization problems

$$\min_{q_1}\left\{\mathrm{KL}(q_1\|\widetilde{q}_1)\right\} \quad \text{and} \quad \min_{q_2}\left\{\mathrm{KL}(q_2\|\widetilde{q}_2)\right\} \qquad (5)$$

is easier than minimizing the original KL divergence between $q(\theta_1,\theta_2)$ and $p(\theta_1,\theta_2|y)$. The most useful scenario is the case of *conjugate prior*: the prior $p(\theta_1)$ belongs to a density family $\mathcal{F}_1$, then $\widetilde{q}_1(\theta_1)$ also belongs to $\mathcal{F}_1$. Similarly, the prior $p(\theta_2)$ belongs to a density family $\mathcal{F}_2$, then $\widetilde{q}_2(\theta_2)$ also belongs to $\mathcal{F}_2$. Then the solutions to (5) are

$$q_1(\theta_1) = \widetilde{q}_1(\theta_1) \text{ and } q_2(\theta_2) = \widetilde{q}_2(\theta_2).$$

Computing the parameter in q_1 requires q_2 and vice versa, which suggests the following coordinate ascent-type algorithm for maximizing the lower bound:

Algorithm 1 (Mean Field Variational Bayes). *1. Initialize the parameter of* $q_1(\theta_1)$

2. Given $q_1(\theta_1)$*, update the parameter of* $q_2(\theta_2)$ *using*

$$q_2(\theta_2) \propto \exp\left(\mathbb{E}_{-\theta_2}[\log p(y,\theta)]\right) = \exp\Big(\int q_1(\theta_1)\log p(y,\theta_1,\theta_2)d\theta_1\Big)$$

3. Given $q_2(\theta_2)$*, update the parameter of* $q_1(\theta_1)$ *using*

$$q_1(\theta_1) \propto \exp\left(\mathbb{E}_{-\theta_1}[\log p(y,\theta)]\right) = \exp\Big(\int q_2(\theta_2)\log p(y,\theta_1,\theta_2)d\theta_2\Big)$$

4. Repeat Steps 2 and 3 until the stopping condition is met.

A common stopping rule is to terminate the update if the change in the parameters of the VB posterior $q(\theta) = q_1(\theta_1)q_2(\theta_2)$ is less than some threshold ϵ. In the case the lower bound $\mathrm{LB}(q_1,q_2)$ can be computed in closed-form, one can stop the algorithm if the increase in the lower bound is less than some threshold ϵ. Note that $\mathrm{LB}(q)$ increases after each iteration.

Example 1

Let $\boldsymbol{y}=(11;12;8;10;9;8;9;10;13;7)$ be iid samples from $\mathcal{N}(\mu,\sigma^2)$, the normal distribution with mean μ and variance σ^2. Suppose that we use the prior $\mathcal{N}(\mu_0,\sigma_0^2)$ for μ and Inverse-Gamma(α_0,β_0) for σ^2, with hyperparameters $\mu_0=0,\sigma_0=10,\alpha_0=1$ and $\beta_0=1$. Assume the VB factorization $q(\mu,\sigma^2)=q(\mu)q(\sigma^2)$. Let's derive the MFVB procedure for approximating the posterior $p(\mu,\sigma^2|\boldsymbol{y})\propto p(\mu)p(\sigma^2)p(\boldsymbol{y}|\mu,\sigma^2)$.

The optimal VB posterior for σ^2 is

$$\begin{aligned}
q(\sigma^2) &\propto \exp\Big(\mathbb{E}_{-\sigma^2}[\log p(\boldsymbol{y},\mu,\sigma^2)]\Big)=\exp\Big(\mathbb{E}_{q(\mu)}[\log p(\boldsymbol{y},\mu,\sigma^2)]\Big)\\
&\propto \exp\Big(\mathbb{E}_{q(\mu)}[\log p(\sigma^2)+\log p(\boldsymbol{y}|\mu,\sigma^2)]\Big)\\
&\propto \exp\Big(-(\alpha_0+\frac{n}{2}+1)\log\sigma^2-\big(\beta_0+\frac{1}{2}\mathbb{E}_{q(\mu)}[\sum(y_i-\mu)^2]\big)/\sigma^2\Big).
\end{aligned}$$

It follows that $q(\sigma^2)$ is inverse-Gamma with parameters

$$\alpha_q=\alpha_0+\frac{n}{2},\;\beta_q=\beta_0+\frac{1}{2}\mathbb{E}_{q(\mu)}\Big[\sum(y_i-\mu)^2\Big].$$

The optimal VB posterior for μ is

$$\begin{aligned}
q(\mu) &\propto \exp\Big(\mathbb{E}_{q(\sigma^2)}[\log p(y,\mu,\sigma^2)]\Big)\\
&\propto \exp\Big(\mathbb{E}_{q(\sigma^2)}[\log p(\mu)+\log p(\boldsymbol{y}|\mu,\sigma^2)]\Big)\\
&\propto \exp\Big(-\frac{1}{2\sigma_0^2}(\mu^2-2\mu_0\mu)-\frac{n}{2}\mathbb{E}_{q(\sigma^2)}[\frac{1}{\sigma^2}](-2\bar{y}\mu+\mu^2)\Big)\\
&\propto \exp\Big(-\frac{1}{2}\underbrace{\big(\frac{1}{\sigma_0^2}+n\mathbb{E}_{q(\sigma^2)}[\frac{1}{\sigma^2}]\big)}_{A}\mu^2+\mu\underbrace{\big(\frac{\mu_0}{\sigma_0^2}+n\bar{y}\mathbb{E}_{q(\sigma^2)}[\frac{1}{\sigma^2}]\big)}_{B}\Big)\\
&= \exp\Big(-\frac{1}{2}A\mu^2+B\mu\Big)\\
&\propto \exp\Big(-\frac{1}{2}\frac{(\mu-B/A)^2}{1/A}\Big).
\end{aligned}$$

It follows that $q(\mu)$ is Gaussian with mean μ_q and variance σ_q^2

$$\mu_q=\frac{\frac{\mu_0}{\sigma_0^2}+n\bar{y}\mathbb{E}_{q(\sigma^2)}[\frac{1}{\sigma^2}]}{\frac{1}{\sigma_0^2}+n\mathbb{E}_{q(\sigma^2)}[\frac{1}{\sigma^2}]},\;\sigma_q^2=\Big(\frac{1}{\sigma_0^2}+n\mathbb{E}_{q(\sigma^2)}[\frac{1}{\sigma^2}]\Big)^{-1}.$$

Note that

$$\begin{aligned}
\beta_q &= \beta_0+\frac{1}{2}\mathbb{E}_{q(\mu)}\big[\sum(y_i-\mu)^2\big]\\
&= \beta_0+\frac{1}{2}\Big(\sum y_i^2-2n\bar{y}\mathbb{E}_{q(\mu)}[\mu]+n\mathbb{E}_{q(\mu)}[\mu^2]\Big)\\
&= \beta_0+\frac{1}{2}\sum y_i^2-n\bar{y}\mu_q+\frac{n}{2}(\mu_q^2+\sigma_q^2).
\end{aligned}$$

As $q(\sigma^2)\sim$ Inverse-Gamma(α_q,β_q), $\mathbb{E}(1/\sigma^2)=\alpha_q/\beta_q$. Hence,

$$\mu_q=\Big(\frac{\mu_0}{\sigma_0^2}+n\bar{y}\frac{\alpha_q}{\beta_q}\Big)/\Big(\frac{1}{\sigma_0^2}+n\frac{\alpha_q}{\beta_q}\Big),\ \text{and}\ \sigma_q^2=\Big(\frac{1}{\sigma_0^2}+n\frac{\alpha_q}{\beta_q}\Big)^{-1}.$$

We arrive at the following updating procedure

- Initialize μ_q,σ_q^2
- Update the following recursively

$$\begin{aligned}\alpha_q &\leftarrow \alpha_0+\frac{n}{2}\\ \beta_q &\leftarrow \beta_0+\frac{1}{2}\sum y_i^2-n\bar{y}\mu_q+\frac{n}{2}(\mu_q^2+\sigma_q^2)\\ \mu_q &\leftarrow \Big(\frac{\mu_0}{\sigma_0^2}+n\bar{y}\frac{\alpha_q}{\beta_q}\Big)/\Big(\frac{1}{\sigma_0^2}+n\frac{\alpha_q}{\beta_q}\Big)\\ \sigma_q^2 &\leftarrow \Big(\frac{1}{\sigma_0^2}+n\frac{\alpha_q}{\beta_q}\Big)^{-1}\end{aligned}$$

  until convergence.

We can stop the iterative scheme when the change of the ℓ_2-norm of the vector $\lambda=(\alpha_q,\beta_q,\mu_q,\sigma_q^2)^\top$ is smaller than some ϵ, $\epsilon=10^{-5}$ for example. We can also initialize α_q,β_q and then update the variational parameters recursively in the order of μ_q, σ_q^2, α_q and β_q. However, it's often a better idea to initialize μ_q,σ_q^2 as it is easier to guess the values related to location parameters than the scale parameters. Figure 1 plots the posterior densities estimated by the MFVB algorithm derived above, and by the Gibbs sampling.

It is straightforward to extend the MFVB procedure in Algorithm 1 to the general case where θ is divided into k blocks $\theta=(\theta_1^\top,\theta_2^\top,...,\theta_k^\top)^\top$, and where we want to approximate the posterior $p(\theta_1,\theta_2,...,\theta_k|y)$ by $q(\theta)=q_1(\theta_1)q_2(\theta_2)...q_k(\theta_k)$. The optimal $q_j(\theta_j)$ that maximizes LB(q), when $q_1,...,q_{j-1},q_{j+1},...,q_k$ are fixed, is

$$q_j(\theta_j)\propto\exp\big(\mathbb{E}_{-\theta_j}[\log p(y,\theta)]\big),\ j=1,...,k.$$

Here $\mathbb{E}_{-\theta_j}(\cdot)$ denotes the expectation w.r.t. q_1,..., q_{j-1}, q_{j+1},..., q_k, i.e.,

$$\mathbb{E}_{-\theta_j}\big[\log p(y,\theta)\big]:=\int q_1(\theta_1)...q_{j-1}(\theta_{j-1})q_{j+1}(\theta_{j+1})...q_k(\theta_k)\log p(y,\theta)d\theta_1....d\theta_{j-1}d\theta_{j+1}...d\theta_k.$$

A similar procedure to Algorithm 1 can be developed, in which we first initialize the parameters in the $k-1$ factors $q_1,...,q_{k-1}$, then update q_k and other factors recursively.

3 Fixed Form Variational Bayes

FFVB assumes a fixed form for the VB approximation density q, i.e. $q=q_\lambda$ belongs to some class of distributions $\mathcal{Q}$ indexed by a variational parameter λ. For example, q_λ is a Gaussian distribution

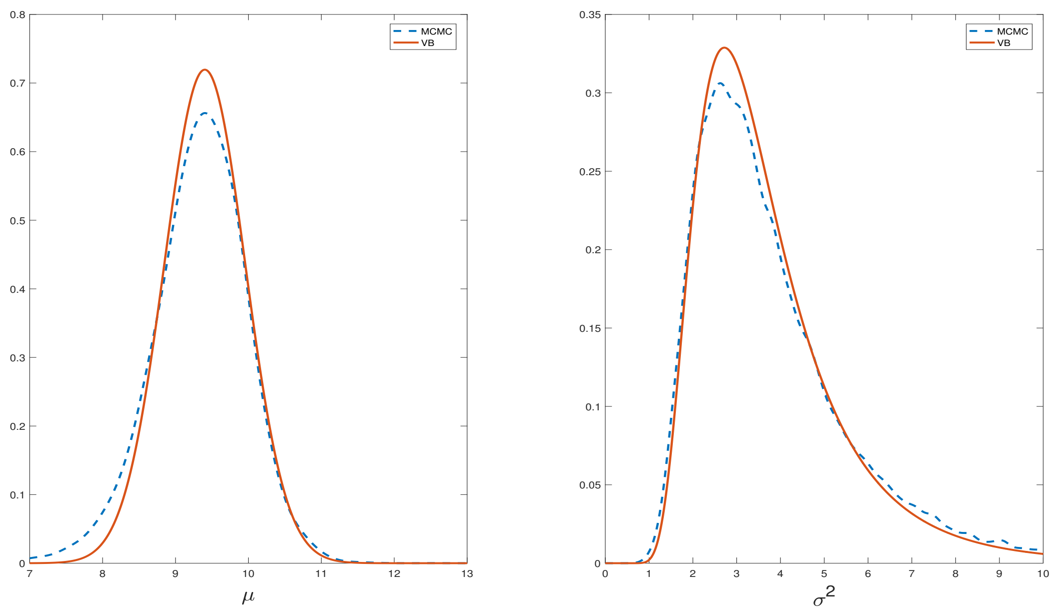

Figure 1: Example 1: Posterior density for μ and σ^2 estimated by MFVB and Gibbs sampling.

with mean μ and covariance matrix Σ. FFVB finds the best q_λ in the class $\mathcal{Q}$ by optimizing the lower bound

$$\text{LB}(\lambda) = \mathbb{E}_{q_\lambda}\left[\log \frac{p(\theta)p(y|\theta)}{q_\lambda(\theta)}\right]. \tag{6}$$

Except for a few trivial cases where the LB can be computed analytically and optimized using classical optimization routines, stochastic optimization is often used to optimize LB(λ). The gradient of LB is

$$\begin{aligned}
\nabla_\lambda \text{LB}(\lambda) &= \int_\Theta \nabla_\lambda q_\lambda(\theta) \log \frac{p(\theta)p(y|\theta)}{q_\lambda(\theta)} d\theta - \int_\Theta q_\lambda(\theta)\nabla_\lambda \log q_\lambda(\theta) d\theta \\
&= \int_\Theta q_\lambda(\theta)\nabla_\lambda \log q_\lambda(\theta) \log \frac{p(\theta)p(y|\theta)}{q_\lambda(\theta)} d\theta - \int_\Theta \nabla_\lambda q_\lambda(\theta) d\theta \\
&= \int_\Theta q_\lambda(\theta)\nabla_\lambda \log q_\lambda(\theta) \log \frac{p(\theta)p(y|\theta)}{q_\lambda(\theta)} d\theta - \nabla_\lambda \int_\Theta q_\lambda(\theta) d\theta \\
&= \mathbb{E}_{q_\lambda}\left[\nabla_\lambda \log q_\lambda(\theta) \times \log \frac{p(\theta)p(y|\theta)}{q_\lambda(\theta)}\right] \\
&= \mathbb{E}_{q_\lambda}\left[\nabla_\lambda \log q_\lambda(\theta) \times h_\lambda(\theta)\right], \quad h_\lambda(\theta) := \log\big(p(\theta)p(y|\theta)/q_\lambda(\theta)\big).
\end{aligned}$$

Later we also use $h(\theta)$, without the subscript, to denote the model-specific function $\log\big(p(\theta)p(y|\theta)\big)$. It follows that, by generating $\theta \sim q_\lambda(\theta)$[1], it is straightforward to obtain an unbiased estimator

[1]In Monte Carlo simulation, by $\theta \sim q_\lambda(\theta)$ we mean that we draw a random variable or random vector θ from the probability distribution with density $q_\lambda(\theta)$. That notation also means θ is a random variable/vector whose probability density function is $q_\lambda(\theta)$.

$\widehat{\nabla_\lambda \text{LB}}(\lambda)$ of the gradient $\nabla_\lambda \text{LB}(\lambda)$, i.e., $\mathbb{E}\big[\widehat{\nabla_\lambda \text{LB}}(\lambda)\big] = \nabla_\lambda \text{LB}(\lambda)$. Therefore, we can use stochastic optimization to optimize $\text{LB}(\lambda)$[2]. The basic algorithm is as follows:

Algorithm 2 (Basic FFVB algorithm)**.**

- *Initialize $\lambda^{(0)}$ and stop the following iteration if the stopping criterion is met.*
- *For $t=0,1,...$*
    - *Generate $\theta_s \sim q_{\lambda^{(t)}}(\theta)$, $s=1,...,S$*
    - *Compute the unbiased estimate of the LB gradient*

$$\widehat{\nabla_\lambda LB}(\lambda^{(t)}) := \frac{1}{S}\sum_{s=1}^{S} \nabla_\lambda \log q_\lambda(\theta_s) \times h_\lambda(\theta_s)|_{\lambda=\lambda^{(t)}}.$$

    - *Update*

$$\lambda^{(t+1)} = \lambda^{(t)} + a_t \widehat{\nabla_\lambda LB}(\lambda^{(t)}). \tag{7}$$

The sequence of learning rates $\{a_t\}$ should satisfy $a_t > 0$, $\sum_t a_t = \infty$ and $\sum_t a_t^2 < \infty$. However, this basic VB algorithm hardly works in practice and requires some refinements to make it work. Much of the rest of this tutorial focuses on presenting and explaining those refinements.

3.1 Stopping criterion

Let us first discuss on the stopping rule. An easy-to-implement stopping rule is to terminate the updating procedure if the change between $\lambda^{(t+1)}$ and $\lambda^{(t)}$, e.g. in terms of the Euclidean distance, is less than some threshold ϵ. However, it is difficult to select ϵ as such a distance depends on the scales and the length of the vector λ. Denote by $\widehat{\text{LB}}(\lambda)$ an estimate of $\text{LB}(\lambda)$ by sampling from $q_\lambda(\theta)$. Although $\text{LB}(\lambda)$ is non-decreasing over iterations, its sample estimate $\widehat{\text{LB}}(\lambda)$ might not be. To account for this, one can stop the updating procedure if the change in an averaged value of the lower bounds over a window of t_W iterations, $\overline{\text{LB}}(\lambda^{(t)}) = (1/t_W)\sum_{k=1}^{t_W} \widehat{\text{LB}}(\lambda^{(t-k+1)})$, is less than some threshold ϵ. At convergence, the values $\text{LB}(\lambda^{(t)})$ stay roughly the same, therefore $\overline{\text{LB}}(\lambda^{(t)})$ will average out the noise in $\widehat{\text{LB}}(\lambda^{(t)})$ and is stable. Typical choice for t_W and ϵ is $t_W = 20$ or $t_W = 50$ and $\epsilon = 10^{-5}$. Another stopping rule that is widely used in machine learning is to stop training if the moving averaged lower bound does not improve after P iterations; and P is sometime fancily referred to as the *patience* parameter. Typical choice is $P = 20$ or $P = 50$.

3.2 Adaptive learning rate and natural gradient

Let's write the update in (7) as

$$\begin{cases} \lambda_1^{(t+1)} = \lambda_1^{(t)} + a_t \widehat{\nabla_{\lambda_1} \text{LB}}(\lambda^{(t)}) \\ ... \\ \lambda_{d_\lambda}^{(t+1)} = \lambda_{d_\lambda}^{(t)} + a_t \widehat{\nabla_{\lambda_{d_\lambda}} \text{LB}}(\lambda^{(t)}), \end{cases}$$

[2]Unbiased estimate of the gradient of the target function is theoretically required in stochastic optimization.

with d_λ the size of vector λ. Intuitively, each coordinate of vector $\lambda^{(t+1)}$ might need a different learning rate that can take into account the scale of that coordinate or the geometry of the space λ living in. It turns out that the basic Algorithm 2 rarely works in practice without a method for selecting the learning rate adaptively.

3.2.1 Adaptive learning rate

For a coordinate i with a large variance $\mathbb{V}(\widehat{\nabla_{\lambda_i}\text{LB}}(\lambda^{(t)}))$, its learning rate $a_{t,i}$ should be small, otherwise the new update $\lambda_i^{(t+1)}$ jumps all over the place and destroys everything the process has learned so far. Denote $g_t := \widehat{\nabla_\lambda\text{LB}}(\lambda^{(t)})$ and $v_t := (g_t)^2$ (this is a coordinate-wise operator). The commonly used adaptive learning rate methods such as ADAM and AdaGrad work by scaling the coordinates of g_t by their corresponding variances. These variances are estimated by moving average. The algorithm below is a basic version of this class of adaptive learning methods:

1) Initialize $\lambda^{(0)}$, g_0 and v_0 and set $\bar{g}=g_0$, $\bar{v}=v_0$. Let $\beta_1,\beta_2\in(0,1)$.

2) For $t=0,1,...$, update

$$\begin{aligned}\bar{g} &= \beta_1\bar{g}+(1-\beta_1)g_t\\ \bar{v} &= \beta_2\bar{v}+(1-\beta_2)v_t\\ \lambda^{(t+1)} &= \lambda^{(t)}+\alpha_t\bar{g}/\sqrt{\bar{v}},\end{aligned}$$

   with α_t a scalar step size, also called learning rate. Here $\bar{g}/\sqrt{\bar{v}}$ should be understood component wise.

Note that the LB gradients g_t have also been smoothened out using moving average. This helps to accelerate the convergence - a method known as the momentum method in the stochastic optimization literature. Typical choice of the scalar α_t is

$$\alpha_t=\min\Big(\epsilon_0,\epsilon_0\frac{\tau}{t}\Big)=\begin{cases}\epsilon_0, & t\le\tau\\ \epsilon_0\frac{\tau}{t}, & t>\tau\end{cases}$$

for some small ϵ_0 (e.g. 0.1 or 0.01) and some threshold τ (e.g., 1000). In the first τ iterations, the training procedure explores the learning space with a fixed learning rate ϵ_0, then this exploration is settled down by reducing the step size after τ iterations.

3.2.2 Natural gradient

Natural gradient can be considered as an adaptive learning method that exploits the geometry of the λ space. The ordinary gradient $\nabla_\lambda\text{LB}(\lambda)$ does not adequately capture the geometry of the approximating family $q_\lambda(\theta)$. A small Euclidean distance between λ and λ' does not necessarily mean a small KL divergence between $q_\lambda(\theta)$ and $q_{\lambda'}(\theta)$. Statisticians and machine learning researchers have long realized the importance of information on the geometry of the manifold of a statistical model, and that the steepest direction for optimizing the objective function $\text{LB}(\lambda)$ on the manifold formed by the family $q_\lambda(\theta)$ is directed by the so-called natural gradient which is defined by pre-multiplying the ordinary gradient with the inverse of the Fisher information matrix

$$\nabla_\lambda\text{LB}(\lambda)^{\text{nat}} := I_F^{-1}(\lambda)\nabla_\lambda\text{LB}(\lambda), \tag{8}$$

with $I_F(\lambda)=\text{cov}_{q_\lambda}(\nabla_\lambda \log q_\lambda(\theta))$ the Fisher information matrix.

The main difficulty of using the natural gradient is the computation of $I_F(\lambda)$, and the solution of the linear systems required to compute (8). The problem is more severe in high dimensional models because this matrix has a large size. An efficient method for computing $I_F(\lambda)^{-1}\nabla_\lambda\text{LB}(\lambda)$ is using iterative conjugate gradient methods which compute (8) by solving the linear system $I_F(\lambda)x=\nabla_\lambda\text{LB}(\lambda)$ for x using only matrix-vector products involving $I_F(\lambda)$. In some cases this matrix vector product can be done efficiently both in terms of computation time and memory requirements by exploiting the structure of the Fisher matrix $I_F(\lambda)$. See Section 3.5.2 for a special case where the natural gradient is computed efficiently in high dimensional problems.

As mentioned before, the gradient momentum method is often useful in the stochastic optimization literature that helps accelerate and stabilize the optimization procedure. The momentum update rule is

$$\begin{aligned}\overline{\nabla_\lambda\text{LB}} &= \alpha_\text{m}\overline{\nabla_\lambda\text{LB}}+(1-\alpha_\text{m})\widehat{\nabla_\lambda\text{LB}}(\lambda^{(t)})^{\text{nat}},\\ \lambda^{(t+1)} &= \lambda^{(t)}+\alpha_t\overline{\nabla_\lambda\text{LB}},\end{aligned}$$

where $\alpha_\text{m}\in[0,1]$ is the momentum weight; α_m around 0.6-0.9 is a typical choice. The use of the moving average gradient $\overline{\nabla_\lambda\text{LB}}$ helps remove some of the noise inherent in the estimated gradients of the lower bound. Note that the momentum method is already embedded in the moving-average-based adaptive learning rate methods in Section 3.2.1.

3.3 Control variate

As is typical of stochastic optimization algorithms, the performance of Algorithm 2 depends greatly on the variance of the noisy gradient. Variance reduction for the noisy gradient is a key ingredient in FFVB algorithms. This section describes a control variate technique for variance reduction, another technique known as *reparameterization trick* is presented in Section 3.4.

Let $\theta_s\sim q_\lambda(\theta)$, $s=1,...,S$, be S samples from the variational distribution $q_\lambda(\theta)$. A naive estimator of the ith element of $\nabla_\lambda\text{LB}(\lambda)$ is

$$\widehat{\nabla_{\lambda_i}\text{LB}}(\lambda)^{\text{naive}}=\frac{1}{S}\sum_{s=1}^S\nabla_{\lambda_i}[\log q_\lambda(\theta_s)]\times h_\lambda(\theta_s), \tag{9}$$

whose variance is often too large to be useful. For any number c_i, consider

$$\widehat{\nabla_{\lambda_i}\text{LB}}(\lambda)=\frac{1}{S}\sum_{s=1}^S\nabla_{\lambda_i}[\log q_\lambda(\theta_s)](h_\lambda(\theta_s)-c_i), \tag{10}$$

which is still an unbiased estimator of $\nabla_{\lambda_i}\text{LB}(\lambda)$ since $\mathbb{E}(\nabla_\lambda[\log q_\lambda(\theta)])=0$, whose variance can be greatly reduced by an appropriate choice of control variate c_i. The variance of $\widehat{\nabla_{\lambda_i}\text{LB}}(\lambda)$ is

$$\frac{1}{S}\mathbb{V}\Big(\nabla_{\lambda_i}[\log q_\lambda(\theta)]h_\lambda(\theta)\Big)+\frac{c_i^2}{S}\mathbb{V}\Big(\nabla_{\lambda_i}[\log q_\lambda(\theta)]\Big)-\frac{2c_i}{S}\text{cov}\Big(\nabla_{\lambda_i}[\log q_\lambda(\theta)]h_\lambda(\theta),\nabla_{\lambda_i}[\log q_\lambda(\theta)]\Big).$$

The optimal c_i that minimizes this variance is

$$c_i=\text{cov}\Big(\nabla_{\lambda_i}[\log q_\lambda(\theta)]h_\lambda(\theta),\nabla_{\lambda_i}[\log q_\lambda(\theta)]\Big)\Big/\mathbb{V}\Big(\nabla_{\lambda_i}[\log q_\lambda(\theta)]\Big). \tag{11}$$

Then $\mathbb{V}(\widehat{\nabla_{\lambda_i}\text{LB}}(\lambda))=\mathbb{V}(\widehat{\nabla_{\lambda_i}\text{LB}}(\lambda)^{\text{naive}})(1-\rho_i^2)\le\mathbb{V}(\widehat{\nabla_{\lambda_i}\text{LB}}(\lambda)^{\text{naive}})$, where ρ_i is the correlation between $\nabla_{\lambda_i}[\log q_\lambda(\theta)]h_\lambda(\theta$ and $\nabla_{\lambda_i}[\log q_\lambda(\theta)]$. Often, ρ_i^2 is very close to 1.

One can estimate the numbers c_i in (11) using samples $\theta_s\sim q_\lambda(\theta)$. In order to ensure the unbiasedness of the gradient estimator, the samples used to estimate c_i must be independent of the samples used to estimate the gradient. In practice, the c_i can be updated sequentially as follows. At iteration t, we use the c_i computed in the previous iteration $t-1$, i.e. based on the samples from $q_{\lambda^{(t-1)}}(\theta)$, to estimate the gradient $\widehat{\nabla_\lambda\text{LB}}(\lambda^{(t)})$, which is computed using new samples from $q_{\lambda^{(t)}}(\theta)$. We then update the c_i using this new set of samples. By doing so, the unbiasedness is guaranteed while no extra samples are needed in updating the control variates c_i.

Algorithm 3 provides a detailed pseudo-code implementation of the FFVB approach that uses the control variate for variation reduction and moving average adaptive learning, and Algorithm 4 implements the FFVB approach that uses the control variate and natural gradient.

Algorithm 3 (Fixed form VB with control variate and adaptive learning)**. Input***: Initial $\lambda^{(0)}$, adaptive learning weights $\beta_1,\beta_2\in(0,1)$, fixed learning rate ϵ_0, learning rate threshold τ, rolling window size t_W and maximum patience P.* **Model-specific requirement***: function $h(\theta):=\log\big(p(\theta)p(y|\theta)\big)$.*

```
• Initialization
    – Generate $\theta_s\sim q_{\lambda^{(0)}}(\theta)$, $s=1,...,S$.
    – Compute the unbiased estimate of the LB gradient
          $$\widehat{\nabla_\lambda LB}(\lambda^{(0)}):=\frac{1}{S}\sum_{s=1}^{S}\nabla_\lambda\log q_\lambda(\theta_s)\times h_\lambda(\theta_s)|_{\lambda=\lambda^{(0)}}.$$
    – Set $g_0:=\widehat{\nabla_\lambda LB}(\lambda^{(0)})$, $v_0:=(g_0)^2$, $\bar g:=g_0$, $\bar v:=v_0$.
    – Estimate control variate $c$ as in (11) using the samples $\{\theta_s,s=1,...,S\}$.
    – Set $t=0$, patience$=0$ and stop=false.
• While stop=false:
    – Generate $\theta_s\sim q_{\lambda^{(t)}}(\theta)$, $s=1,...,S$
    – Compute the unbiased estimate of the LB gradient
          $$g_t:=\widehat{\nabla_\lambda LB}(\lambda^{(t)})=\frac{1}{S}\sum_{s=1}^{S}\nabla_\lambda\log q_\lambda(\theta_s)\times\big(h_\lambda(\theta_s)-c\big)|_{\lambda=\lambda^{(t)}}.$$
    – Estimate the new control variate $c$ as in (11) using the samples $\{\theta_s,s=1,...,S\}$.
    – Compute $v_t=(g_t)^2$ and
          $$\bar g=\beta_1\bar g+(1-\beta_1)g_t,\ \bar v=\beta_2\bar v+(1-\beta_2)v_t.$$
    – Compute $\alpha_t=\min(\epsilon_0,\epsilon_0\frac{\tau}{t})$ and update
          $$\lambda^{(t+1)}=\lambda^{(t)}+\alpha_t\bar g/\sqrt{\bar v}$$
```

– *Compute the lower bound estimate*

$$\widehat{LB}(\lambda^{(t)}) := \frac{1}{S}\sum_{s=1}^{S} h_{\lambda^{(t)}}(\theta_s).$$

– *If* $t \geq t_W$*: compute the moving averaged lower bound*

$$\overline{LB}_{t-t_W+1} = \frac{1}{t_W}\sum_{k=1}^{t_W}\widehat{LB}(\lambda^{(t-k+1)}),$$

and if $\overline{LB}_{t-t_W+1} \geq \max(\overline{\mathrm{LB}})$ *patience = 0; else patience:=patience*$+1$.

– *If patience*$\geq P$, **`stop=true`**.

– *Set* $t := t+1$.

Example 2

With the model and data in Example 1, let's derive a FFVB procedure for approximating the posterior $p(\mu,\sigma^2|\boldsymbol{y}) \propto p(\mu)p(\sigma^2)p(\boldsymbol{y}|\mu,\sigma^2)$ using Algorithm 3. Suppose that the VB approximation is $q_\lambda(\mu,\sigma^2) = q(\mu)q(\sigma^2)$ with $q(\mu) = \mathcal{N}(\mu_\mu,\sigma^2_\mu)$ and $q(\sigma^2) = \text{Inverse-Gamma}(\alpha_{\sigma^2},\beta_{\sigma^2})$.

The model parameter is $\theta = (\mu,\sigma^2)^\top$ and the variational parameter $\lambda = (\mu_\mu,\sigma^2_\mu,\alpha_{\sigma^2},\beta_{\sigma^2})^\top$. In order to implement Algorithm 3, we need $h_\lambda(\theta) = h(\theta) - \log q_\lambda(\theta)$ with

$$\begin{aligned}
h(\theta) &= \log\big(p(\mu)p(\sigma^2)p(y|\mu,\sigma^2)\big)\\
&= -\frac{n+1}{2}\log(2\pi) - \frac{1}{2}\log(\sigma_0^2) - \frac{(\mu-\mu_0)^2}{2\sigma_0^2} + \alpha_0\log(\beta_0) - \log\Gamma(\alpha_0) - (\frac{n}{2}+\alpha_0+1)\log(\sigma^2)\\
&\qquad - \frac{\beta_0}{\sigma^2} - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i-\mu)^2,\\
\log q_\lambda(\theta) &= \alpha_{\sigma^2}\log\beta_{\sigma^2} - \log\Gamma(\alpha_{\sigma^2}) - (\alpha_{\sigma^2}+1)\log\sigma^2 - \frac{\beta_{\sigma^2}}{\sigma^2} - \frac{1}{2}\log(2\pi) - \frac{1}{2}\log(\sigma^2_\mu) - \frac{(\mu-\mu_\mu)^2}{2\sigma^2_\mu},
\end{aligned}$$

and

$$\nabla_\lambda \log q_\lambda(\theta) = \Big(-\frac{\mu-\mu_\mu}{\sigma^2_\mu}, -\frac{1}{2\sigma^2_\mu} + \frac{(\mu-\mu_\mu)^2}{2\sigma^4_\mu}, \log\beta_{\sigma^2} - \frac{\Gamma'(\alpha_{\sigma^2})}{\Gamma(\alpha_{\sigma^2})} - \log\sigma^2, \frac{\alpha_{\sigma^2}}{\beta_{\sigma^2}} - \frac{1}{\sigma^2}\Big)^\top.$$

We are now ready to implement Algorithm 3. Figure 2 plots the estimate of the posterior densities together with the lower bound.

Algorithm 4 (Fixed form VB with control variate and natural gradient). **Input***: Initial* $\lambda^{(0)}$*, momentum weight* α_m*, fixed learning rate* ϵ_0*, learning rate threshold* τ*, rolling window size* t_W *and maximum patience* P. **Model-specific requirement***: function* $h(\theta) := \log\big(p(\theta)p(y|\theta)\big)$.

- *Initialization*

  – *Generate* $\theta_s \sim q_{\lambda^{(0)}}(\theta)$, $s = 1,...,S$.

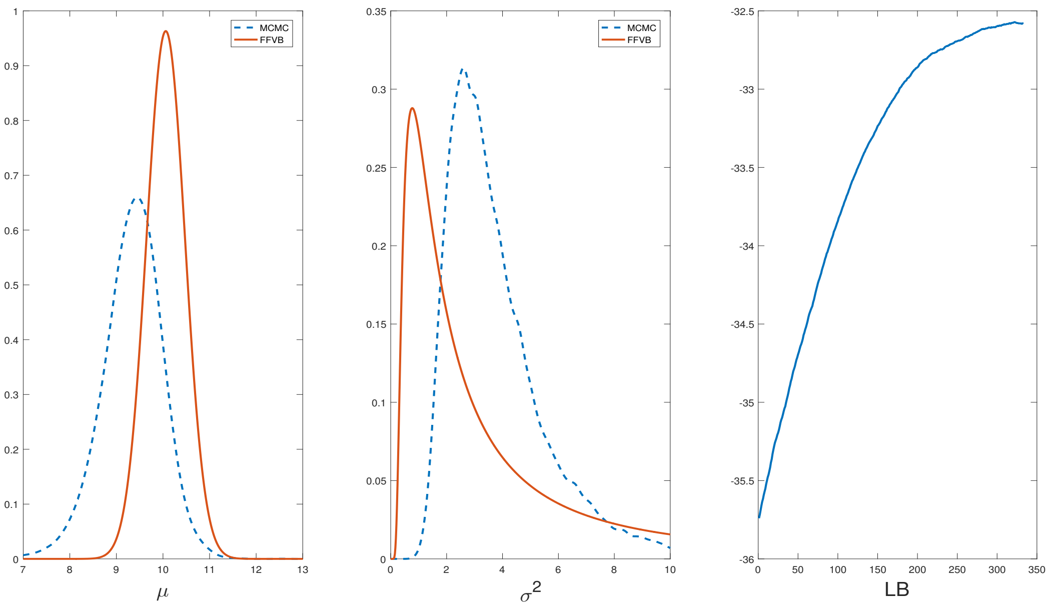

Figure 2: Example 2: Posterior density for μ and σ^2 estimated by FFVB with control variate and adaptive learning and Gibbs sampling. The last panel shows the averaged lower bounds $\overline{\text{LB}}_t$.

  - *Compute the unbiased estimate of the LB gradient*

$$\widehat{\nabla_\lambda LB}(\lambda^{(0)}):=\frac{1}{S}\sum_{s=1}^{S}\nabla_\lambda \log q_\lambda(\theta_s)\times h_\lambda(\theta_s)|_{\lambda=\lambda^{(0)}}$$

    and the natural gradient

$$\widehat{\nabla_\lambda \text{LB}}(\lambda^{(0)})^{nat}:=I_F^{-1}(\lambda^{(0)})\widehat{\nabla_\lambda \text{LB}}(\lambda^{(0)}).$$

  - *Set momentum gradient* $\overline{\nabla_\lambda \text{LB}}:=\widehat{\nabla_\lambda \text{LB}}(\lambda^{(0)})^{nat}$.
  - *Estimate control variate* c *as in* (11) *using the samples* $\{\theta_s, s=1,...,S\}$.
  - *Set* $t=0$, *patience*$=0$ *and* `stop=false`.

- *While* `stop=false`*:*
  - *Generate* $\theta_s\sim q_{\lambda^{(t)}}(\theta)$, $s=1,...,S$
  - *Compute the unbiased estimate of the LB gradient*

$$\widehat{\nabla_\lambda LB}(\lambda^{(t)})=\frac{1}{S}\sum_{s=1}^{S}\nabla_\lambda \log q_\lambda(\theta_s)\times\big(h_\lambda(\theta_s)-c\big)|_{\lambda=\lambda^{(t)}}$$

    and the natural gradient

$$\widehat{\nabla_\lambda \text{LB}}(\lambda^{(t)})^{nat}=I_F^{-1}(\lambda^{(t)})\widehat{\nabla_\lambda \text{LB}}(\lambda^{(t)}).$$

- *Estimate the new control variate* c *as in* (11) *using the samples* $\{\theta_s, s=1,...,S\}$.
- *Compute the momentum gradient*

$$\overline{\nabla_\lambda \text{LB}} = \alpha_m \overline{\nabla_\lambda \text{LB}} + (1-\alpha_m)\widehat{\nabla_\lambda \text{LB}}(\lambda^{(t)})^{nat}.$$

- *Compute* $\alpha_t = \min(\epsilon_0, \epsilon_0 \frac{\tau}{t})$ *and update*

$$\lambda^{(t+1)} = \lambda^{(t)} + \alpha_t \overline{\nabla_\lambda \text{LB}}.$$

- *Compute the lower bound estimate*

$$\widehat{LB}(\lambda^{(t)}) := \frac{1}{S}\sum_{s=1}^{S} h_{\lambda^{(t)}}(\theta_s).$$

- *If* $t \geq t_W$*: compute the moving average lower bound*

$$\overline{LB}_{t-t_W+1} = \frac{1}{t_W}\sum_{k=1}^{t_W}\widehat{LB}(\lambda^{(t-k+1)}),$$

  and if $\overline{LB}_{t-t_W+1} \geq \max(\overline{\text{LB}})$ *patience = 0; else patience*:=*patience*+1.
- *If patience* $\geq P$, `stop=true`.
- *Set* $t := t+1$.

Example 3

With the model and data in Example 1, let's derive a FFVB procedure for approximating the posterior $p(\mu,\sigma^2|\boldsymbol{y}) \propto p(\mu)p(\sigma^2)p(\boldsymbol{y}|\mu,\sigma^2)$ using Algorithm 4. In order to implement Algorithm 4, apart from $h_\lambda(\theta)$ and $\nabla_\lambda \log q_\lambda(\theta)$ as in Example 2, we need the Fisher information matrix I_F. It can be seen that this is a diagonal block matrix with two main blocks

$$\begin{pmatrix} \frac{1}{\sigma_\mu^2} & 0 \\ 0 & \frac{1}{2\sigma_\mu^4} \end{pmatrix}, \text{ and } \begin{pmatrix} \frac{\partial^2 \log\Gamma(\alpha_{\sigma^2})}{\partial\alpha_{\sigma^2}\partial\alpha_{\sigma^2}} & -\frac{1}{\beta_{\sigma^2}} \\ -\frac{1}{\beta_{\sigma^2}} & \frac{\alpha_{\sigma^2}}{\beta_{\sigma^2}^2} \end{pmatrix}.$$

Figure 3 shows the estimated densities together with the lower bound estimates. In this example, Algorithm 4 appears to work more stably and produce more accurate approximation (compared to the Gibbs sampling estimate) than Algorithm 3.

3.4 Reparameterization trick

The reparameterization trick is an attractive alternative to the control variate in Section 3.3. Suppose that for $\theta \sim q_\lambda(\cdot)$, there exists a deterministic function $g(\lambda,\varepsilon)$ such that $\theta = g(\lambda,\varepsilon) \sim q_\lambda(\cdot)$ where $\varepsilon \sim p_\varepsilon(\cdot)$, which is independent of λ. For example, if $q_\lambda(\theta) = \mathcal{N}(\theta;\mu,\Sigma)$ then $\theta = \mu + \Sigma^{1/2}\varepsilon$ with $\varepsilon \sim \mathcal{N}(0,I)$ and I is the identity matrix. Writing LB(λ) as an expectation with respect to $p_\varepsilon(\cdot)$

$$\text{LB}(\lambda) = \mathbb{E}_{\varepsilon\sim p_\varepsilon}\Big(h_\lambda(g(\varepsilon,\lambda))\Big),$$

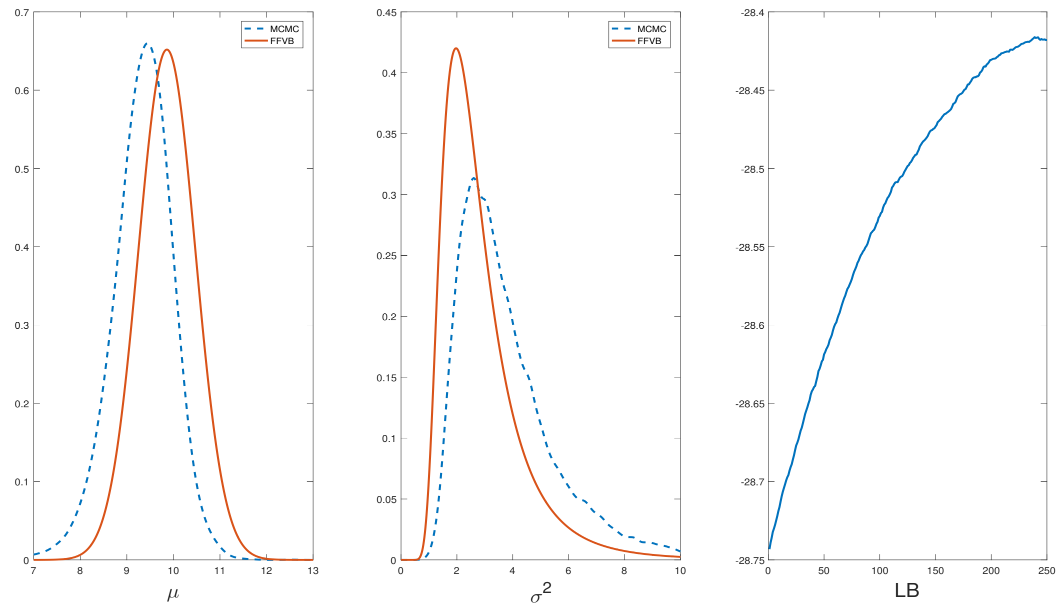

Figure 3: Example 3: Posterior density for μ and σ^2 estimated by FFVB control variate and natural gradient and Gibbs sampling. The last panel shows the averaged lower bounds $\overline{\text{LB}}_t$.

where $\mathbb{E}_{\varepsilon\sim p_\varepsilon}(\cdot)$ denotes expectation with respect to $p_\varepsilon(\cdot)$, and differentiating under the integral sign gives

$$\nabla_\lambda \text{LB}(\lambda) = \mathbb{E}_{\varepsilon\sim p_\varepsilon}\Big(\nabla_\lambda g(\lambda,\varepsilon)\nabla_\theta h_\lambda(\theta)\Big) + \mathbb{E}_{\varepsilon\sim p_\varepsilon}\Big(\nabla_\lambda h_\lambda(\theta)\Big)$$

where the θ within $h_\lambda(\theta)$ is understood as $\theta = g(\varepsilon,\lambda)$ with λ fixed. In particular, the gradient $\nabla_\lambda h_\lambda(\theta)$ is taken when θ is not considered as a function of λ. Note that

$$\begin{aligned}\mathbb{E}_{\varepsilon\sim p_\varepsilon}\Big(\nabla_\lambda h_\lambda(\theta)\Big) = \mathbb{E}_{\varepsilon\sim q_\varepsilon}\Big(\nabla_\lambda h_\lambda\big(\theta = g(\varepsilon,\lambda)\big)\Big) &= -\mathbb{E}_{\varepsilon\sim q_\varepsilon}\Big(\nabla_\lambda \log q_\lambda\big(\theta = g(\varepsilon,\lambda)\big)\Big)\\ &= -\mathbb{E}_{\theta\sim q_\lambda}\Big(\nabla_\lambda \log q_\lambda(\theta)\Big) = 0,\end{aligned}$$

hence

$$\nabla_\lambda \text{LB}(\lambda) = \mathbb{E}_{\varepsilon\sim q_\varepsilon}\Big(\nabla_\lambda g(\lambda,\varepsilon)\nabla_\theta h_\lambda(\theta)\Big). \tag{12}$$

The gradient (12) can be estimated unbiasedly using i.i.d samples $\varepsilon_s \sim p_\varepsilon(\cdot)$, $s = 1,...,S$, as

$$\widehat{\nabla_\lambda \text{LB}}(\lambda) = \frac{1}{S}\sum_{s=1}^{S}\nabla_\lambda g(\lambda,\varepsilon_s)\nabla_\theta\big\{h_\lambda(g(\lambda,\varepsilon_s))\big\}. \tag{13}$$

The reparametrized gradient estimator (13) is often more efficient than alternative approaches to estimating the lower bound gradient, partly because it takes into account the information from the gradient $\nabla_\theta h_\lambda(\theta)$. Algorithm 5 provides a detailed implementation of the FFVB approach that uses the reparameterization trick and adaptive learning. A small modification of Algorithm 5 (not presented) gives the implementation of the FFVB approach that uses the reparameterization trick and natural gradient.

Algorithm 5 (FFVB with reparameterization trick and adaptive learning)**. Input***: Initial* $\lambda^{(0)}$*, adaptive learning weights* $\beta_1,\beta_2\in(0,1)$*, fixed learning rate* ϵ_0*, learning rate threshold* τ*, rolling window size* t_W *and maximum patience* P*.* **Model-specific requirement***: function* $h(\theta):=\log\big(p(\theta)p(y|\theta)\big)$ *and its gradient* $\nabla_\theta h(\theta)$*.*

- *Initialization*
  - *Generate* $\varepsilon_s\sim p_\varepsilon(\cdot)$*,* $s=1,...,S$*.*
  - *Compute the unbiased estimate of the LB gradient*

$$\widehat{\nabla_\lambda LB}(\lambda^{(0)}):=\frac{1}{S}\sum_{s=1}^{S}\nabla_\lambda g(\lambda,\varepsilon_s)\nabla_\theta\big\{h_\lambda(g(\lambda,\varepsilon_s))\big\}\big|_{\lambda=\lambda^{(0)}}.$$

  - *Set* $g_0:=\widehat{\nabla_\lambda LB}(\lambda^{(0)})$*,* $v_0:=(g_0)^2$*,* $\bar g:=g_0$*,* $\bar v:=v_0$*.*
  - *Set* $t=0$*, patience*$=0$ *and* **`stop=false`***.*

- *While* **`stop=false`***:*
  - *Generate* $\varepsilon_s\sim p_\varepsilon(\cdot)$*,* $s=1,...,S$
  - *Compute the unbiased estimate of the LB gradient*

$$g_t:=\widehat{\nabla_\lambda LB}(\lambda^{(t)})=\frac{1}{S}\sum_{s=1}^{S}\nabla_\lambda g(\lambda,\varepsilon_s)\nabla_\theta\big\{h_\lambda(g(\lambda,\varepsilon_s))\big\}\big|_{\lambda=\lambda^{(t)}}.$$

  - *Compute* $v_t=(g_t)^2$ *and*

$$\bar g=\beta_1\bar g+(1-\beta_1)g_t,\ \bar v=\beta_2\bar v+(1-\beta_2)v_t.$$

  - *Compute* $\alpha_t=\min(\varepsilon_0,\varepsilon_0\frac{\tau}{t})$ *and update*

$$\lambda^{(t+1)}=\lambda^{(t)}+\alpha_t\bar g/\sqrt{\bar v}$$

  - *Compute the lower bound estimate*

$$\widehat{LB}(\lambda^{(t)}):=\frac{1}{S}\sum_{s=1}^{S}h_{\lambda^{(t)}}(\theta_s),\ \theta_s=g(\lambda^{(t)},\varepsilon_s).$$

  - *If* $t\ge t_W$*: compute the moving average lower bound*

$$\overline{LB}_{t-t_W+1}=\frac{1}{t_W}\sum_{k=1}^{t_W}\widehat{LB}(\lambda^{(t-k+1)}),$$

    and if $\overline{LB}_{t-t_W+1}\ge\max(\overline{\mathrm{LB}})$ *patience = 0; else patience:=patience+1.*
  - *If patience*$\ge P$*,* **`stop=true`***.*
  - *Set* $t:=t+1$*.*

3.5 Gaussian Variational Bayes

The most popular VB approaches are probably Gaussian VB where the approximation $q_\lambda(\theta)$ is a Gaussian distribution with mean μ and covariance matrix Σ. This section presents several implementation versions of this GVB approach.

3.5.1 GVB with Cholesky decomposed covariance

This GVB method uses the Cholesky decomposition for the covariance matrix Σ, $\Sigma = LL^\top$ with L a lower triangular matrix[3]. A sample $\theta \sim q_\lambda(\theta)$ can be written as $\theta = \mu + L\varepsilon$ with $\varepsilon \sim \mathcal{N}_d(0, I_d)$, and d the dimension of θ. The variational parameter λ includes μ and L. By noting that $\mathbb{E}_{\theta\sim q_\lambda}\big[(\theta-\mu)^\top\Sigma^{-1}(\theta-\mu)\big] = d$, the lower bound in (6) is

$$\begin{aligned}
\text{LB}(\lambda) &= \mathbb{E}_{q_\lambda}\big[\log\big(p(\theta)p(y|\theta)\big)\big] - \mathbb{E}_{q_\lambda}\big[\log q_\lambda(\theta)\big] \\
&= \mathbb{E}_{q_\lambda}\big[h(\theta)\big] + \frac{1}{2}\log|\Sigma| + \frac{d}{2} + \frac{d}{2}\log(2\pi), \quad h(\theta) := \log\big(p(\theta)p(y|\theta)\big) \\
&= \mathbb{E}_\varepsilon\big[h(\mu + L\varepsilon)\big] + \frac{1}{2}\log|\Sigma| + \frac{d}{2} + \frac{d}{2}\log(2\pi).
\end{aligned}$$

The gradient of LB w.r.t. μ is

$$\nabla_\mu \text{LB}(\lambda) = \mathbb{E}_\varepsilon\big[\nabla_\theta h(\theta)\big], \text{ with } \theta = \mu + L\varepsilon.$$

To compute the gradient w.r.t. L, we first need some notations. For a $d\times d$ matrix A, denote by vec(A) the d^2-vector obtained by stacking the columns of A one underneath the other, by vech(A) the $\frac{1}{2}d(d+1)$-vector obtained by stacking the columns of the lower triangular part of A, and by $A\otimes B$ the Kronecker product of matrices A and B. For any matrices A, B and X of suitable sizes, we shall use the fact that $\text{vec}(AXB) = (B^\top \otimes A)\text{vec}(X)$. Then, $L\varepsilon = \text{vec}(I_d L\varepsilon) = (\varepsilon^\top \otimes I_d)\text{vec}(L)$ and hence

$$\begin{aligned}
\nabla_{\text{vec}(L)}\mathbb{E}_\varepsilon\big[h(\mu + L\varepsilon)\big] &= \mathbb{E}_\varepsilon\Big[\nabla_{\text{vec}(L)} h\big(\mu + (\varepsilon^\top \otimes I_d)\text{vec}(L)\big)\Big] \\
&= \mathbb{E}_\varepsilon\Big[(\varepsilon \otimes I_d)\nabla_\theta h\big(\mu + L\varepsilon\big)\Big] \\
&= \mathbb{E}_\varepsilon\Big[\text{vec}\big(\nabla_\theta h(\theta)\varepsilon^\top\big)\Big], \quad \text{with} \quad \theta = \mu + L\varepsilon.
\end{aligned}$$

This implies that

$$\nabla_{\text{vech}(L)}\mathbb{E}_\varepsilon\big[h(\mu + L\varepsilon)\big] = \mathbb{E}_\varepsilon\big[\text{vech}\big(\nabla_\theta h(\theta)\varepsilon^\top\big)\big], \quad \text{with} \quad \theta = \mu + L\varepsilon. \tag{14}$$

Also, as $\log|\Sigma| = \sum_{i=1}^d \log L_{ii}^2$, $\nabla_{\text{vech}(L)}\log|\Sigma| = 2\text{vech}(\text{diag}(1/L_{11}, ..., 1/L_{dd}))$, we have that

$$\nabla_{\text{vech}(L)}\text{LB}(\lambda) = \mathbb{E}_\varepsilon\big[\text{vech}\big(\nabla_\theta h(\theta)\varepsilon^\top\big)\big] + \text{vech}(\text{diag}(1/L_{11}, ..., 1/L_{dd})). \tag{15}$$

Algorithm 6 (GVB with reparameterization trick and adaptive learning)**. Input***: Initial* $\mu^{(0)}$, $L^{(0)}$ *and* $\lambda^{(0)} := (\mu^{(0)\top}, \text{vech}(L^{(0)})^\top)^\top$*, adaptive learning weights* $\beta_1, \beta_2 \in (0,1)$*, fixed learning rate* ϵ_0*, learning rate threshold* τ*, rolling window size* t_W *and maximum patience* P*.* **Model-specific requirement***: function* $h(\theta)$ *and* $\nabla_\theta h(\theta)$*.*

[3]For the Cholesky decomposition of Σ to be unique, one needs the constraint that the diagonal entries of L to be strictly positive. For simplicity, however, we do not impose this constraint here.

- *Initialization*
  - *Generate* $\varepsilon_s \sim N_d(0,I)$, $s=1,...,S$.
  - *Compute the LB gradient estimate* $\widehat{\nabla}_\lambda \text{LB}(\lambda^{(0)}) = (\widehat{\nabla}_\mu \text{LB}(\lambda^{(0)})^\top, \widehat{\nabla}_{\text{vech}(L)} \text{LB}(\lambda^{(0)})^\top)^\top$ *where*
$$\begin{aligned}\widehat{\nabla}_\mu \text{LB}(\lambda^{(0)}) &:= \frac{1}{S}\sum_{s=1}^S \nabla_\theta h(\theta_s),\\ \widehat{\nabla}_{\text{vech}(L)} \text{LB}(\lambda^{(0)}) &:= \frac{1}{S}\sum_{s=1}^S \text{vech}\big(\nabla_\theta h(\theta_s)\varepsilon_s^\top\big) + \text{vech}\big(\text{diag}(1/L_{11}^{(0)},...,L_{dd}^{(0)})\big),\end{aligned}$$
    with $\theta_s = \mu^{(0)} + L^{(0)}\varepsilon_s$.
  - *Set* $g_0 := \widehat{\nabla}_\lambda LB(\lambda^{(0)})$, $v_0 := (g_0)^2$, $\bar{g} := g_0$, $\bar{v} := v_0$.
  - *Set* $t=0$, *patience*$=0$ *and* `stop=false`.
- *While* `stop=false`*:*
  - *Generate* $\varepsilon_s \sim p_\varepsilon(\cdot)$, $s=1,...,S$. *Recalculate* $\mu^{(t)}$ *and* $L^{(t)}$ *from* $\lambda^{(t)}$.
  - *Compute the LB gradient estimate* $g_t := \widehat{\nabla}_\lambda \text{LB}(\lambda^{(t)}) = (\widehat{\nabla}_\mu \text{LB}(\lambda^{(t)})^\top, \widehat{\nabla}_{\text{vech}(L)} \text{LB}(\lambda^{(t)})^\top)^\top$ *where*
$$\begin{aligned}\widehat{\nabla}_\mu \text{LB}(\lambda^{(t)}) &:= \frac{1}{S}\sum_{s=1}^S \nabla_\theta h(\theta_s),\\ \widehat{\nabla}_{\text{vech}(L)} \text{LB}(\lambda^{(t)}) &:= \frac{1}{S}\sum_{s=1}^S \text{vech}\big(\nabla_\theta h(\theta_s)\varepsilon_s^\top\big) + \text{vech}\big(\text{diag}(1/L_{11}^{(t)},...,L_{dd}^{(t)})\big),\end{aligned}$$
    with $\theta_s = \mu^{(t)} + L^{(t)}\varepsilon_s$.
  - *Compute* $v_t = (g_t)^2$ *and*
$$\bar{g} = \beta_1\bar{g} + (1-\beta_1)g_t,\ \bar{v} = \beta_2\bar{v} + (1-\beta_2)v_t.$$
  - *Compute* $\alpha_t = \min(\varepsilon_0, \varepsilon_0\frac{\tau}{t})$ *and update*
$$\lambda^{(t+1)} = \lambda^{(t)} + \alpha_t\bar{g}/\sqrt{\bar{v}}$$
  - *Compute the lower bound estimate*
$$\widehat{LB}(\lambda^{(t)}) := \frac{1}{S}\sum_{s=1}^S h(\theta_s) + \frac{1}{2}\log|L^{(t)}L^{(t)^\top}| + \frac{d}{2} + \frac{d}{2}\log(2\pi).$$
  - *If* $t \geq t_W$*: compute the moving averaged lower bound*
$$\overline{LB}_{t-t_W+1} = \frac{1}{t_W}\sum_{k=1}^{t_W}\widehat{LB}(\lambda^{(t-k+1)}),$$
    and if $\overline{LB}_{t-t_W+1} \geq \max(\overline{\text{LB}})$ *patience = 0; else patience:=patience+1.*
  - *If patience*$\geq P$, `stop=true`.
  - *Set* $t := t+1$.

Example 4: Bayesian logistic regression

Consider a Bayesian logistic regression problem with design matrix $X=[x_1,...,x_n]^\top$ and vector of binary responses y. The log-likelihood is

$$\log p(y|X,\theta)=y^\top X\theta-\sum_{i=1}^n\log\big(1+\exp(x_i^\top\theta)\big)$$

with θ the vector of d coefficients. Suppose that a normal prior $\mathcal{N}(0,\sigma_0^2 I)$ is used for θ. To implement the GVB method in Algorithm 6, all we need is the function

$$h(\theta)=\log p(\theta)+\log p(y|X,\theta)=-\frac{d}{2}\log(2\pi)-\frac{d}{2}\log(\sigma_0^2)-\frac{\theta^\top\theta}{2\sigma_0^2}+y^\top X\theta-\sum_{i=1}^n\log\big(1+\exp(x_i^\top\theta)\big),$$

and its gradient

$$\nabla_\theta h(\theta)=-\frac{1}{\sigma_0^2}\theta+X^\top\big(y-\pi(\theta)\big)$$

with

$$\pi(\theta)=\Big(\frac{1}{1+\exp(-x_1^\top\theta)},...,\frac{1}{1+\exp(-x_n^\top\theta)}\Big)^\top.$$

The Labour Force Participation dataset contains information of 753 women with one binary variable indicating whether or not they are currently in the labour force together with seven covariates such as number of children under 6 years old, age, education level, etc. Figure 4 plots the VB approximation for each coefficient θ_i together with the lower bound estimates over the iterations.

3.5.2 GVB with factor decomposed covariance

An alternative to the Cholesky decomposition is the factor decomposition

$$\Sigma=BB^\top+C^2,$$

where B is the factor loading matrix of size $d\times f$ with $f\ll d$ the number of factors and $C=\text{diag}(c_1,...,c_d)$. GVB with this factor covariance structure is useful in high-dimensional settings where d is large. This section describes the case with one factor, $f=1$, which achieves a great computational speed-up for approximate Bayesian inference in big models such as deep neural networks. With $f=1$, we rewrite the factor decomposition as

$$\Sigma=bb^\top+C^2,\ C=\text{diag}(c),$$

where $b=(b_1,...,b_d)^\top$ and $c=(c_1,...,c_d)^\top$ are vectors. The variational parameter is $\lambda=(\mu^\top,b^\top,c^\top)^\top$. The lower bound using the reparameterization trick can be written as

$$\text{LB}(\lambda)=\mathbb{E}_{q_\varepsilon}\big[h_\lambda(\mu+\varepsilon_1 b+c\circ\varepsilon_2)\big],\quad h_\lambda(\theta):=\log\big(p(\theta)p(y|\theta)\big)-\log q_\lambda(\theta)$$

where $\varepsilon=(\varepsilon_1,\varepsilon_2^\top)^\top\sim\mathcal{N}_{d+1}(0,I)$, and $c\circ\varepsilon_2$ denotes the component-wise product of vectors c and ε_2. Note that

$$\nabla_\mu(\mu+\varepsilon_1 b+c\circ\varepsilon_2)=I_d,\ \nabla_b(\mu+\varepsilon_1 b+c\circ\varepsilon_2)=\varepsilon_1 I_d,\ \nabla_c(\mu+\varepsilon_1 b+c\circ\varepsilon_2)=\text{diag}(\varepsilon_2),$$

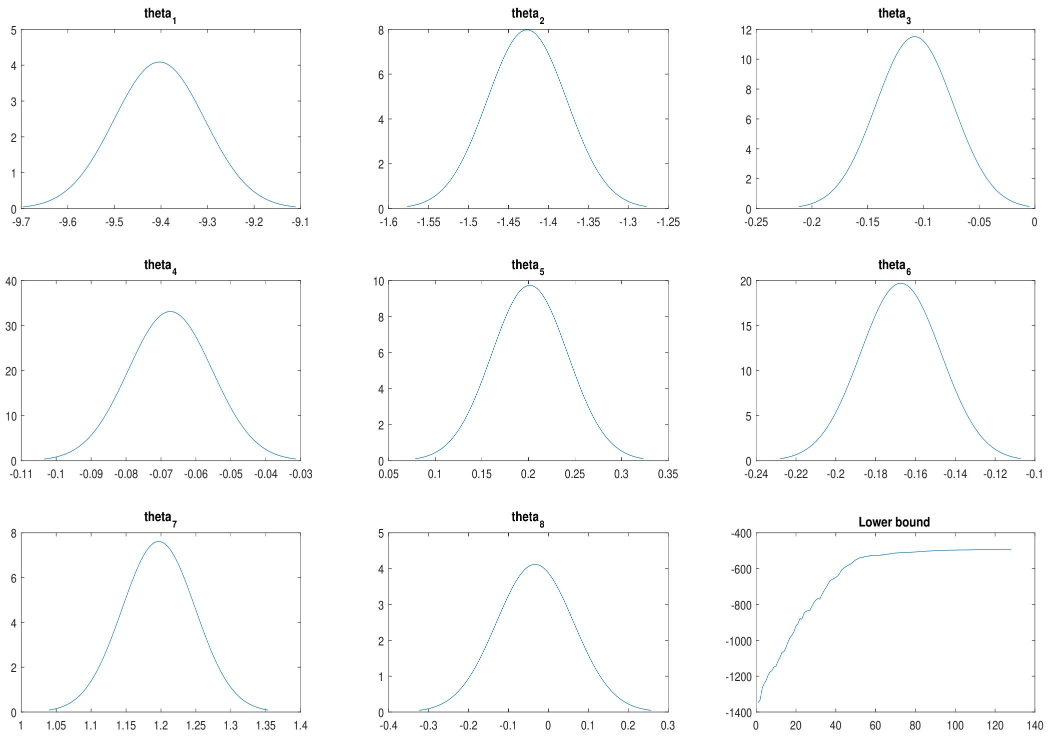

Figure 4: GVB with the Cholesky decomposition for approximating the posterior in logistic regression. The CPU time was roughty 3 seconds.

hence

$$\nabla_\lambda \text{LB}(\lambda) = \mathbb{E}_{q_\varepsilon}\begin{pmatrix} \nabla_\theta h_\lambda(\mu+\varepsilon_1 b + c\circ\varepsilon_2) \\ \varepsilon_1 \nabla_\theta h_\lambda(\mu+\varepsilon_1 b + c\circ\varepsilon_2) \\ \varepsilon_2 \circ \nabla_\theta h_\lambda(\mu+\varepsilon_1 b + c\circ\varepsilon_2)\end{pmatrix}. \tag{16}$$

The gradient of function $h_\lambda(\theta)$ is

$$\nabla_\theta h_\lambda(\theta) = \nabla_\theta \log\big(p(\theta)p(y|\theta)\big) - \nabla_\theta \log q_\lambda(\theta),$$

where the first term is model-specific and the second term is

$$\nabla_\theta \log q_\lambda(\theta) = (\theta-\mu)\circ c^{-2} - \frac{(b\circ c^{-2})^\top(\theta-\mu)}{1+(b\circ c^{-1})^\top(b\circ c^{-1})}(b\circ c^{-2}),$$

with $c^{-1} := (1/c_1,...,1/c_d)^\top$ and $c^{-2} := (1/c_1^2,...,1/c_d^2)^\top$. Finally, it can be shown that the natural gradient in (8) can be approximately computed in closed form (see Tran et al. (2020)).

Algorithm 7 (Computing the natural gradient)**.** *Input: Vector b, c and ordinary gradient of the lower bound $g = (g_1^\top, g_2^\top, g_3^\top)^\top$ with g_1 the vector formed by the first d elements of g, g_2 formed by the next d elements, and g_3 the last d elements. Output: The natural gradient $g^{nat} = I_F^{-1} g$.*

- *Compute the vectors* $v_1=c^2-2b^2\circ c^{-4}$, $v_2=b^2\circ c^{-3}$, *and the scalars* $\kappa_1=\sum_{i=1}^d b_i^2/c_i^2$, $\kappa_2=\frac{1}{2}(1+\sum_{i=1}^d v_{2i}^2/v_{1i})^{-1}$.
- *Compute*

$$g^{nat}=\begin{pmatrix}(g_1^\top b)b+c^2\circ g_1\\ \frac{1+\kappa_1}{2\kappa_1}\Big((g_2^\top b)b+c^2\circ g_2\Big)\\ \frac{1}{2}v_1^{-1}\circ g_3+\kappa_2\big[(v_1^{-1}\circ v_2)^\top g_3\big](v_1^{-1}\circ v_2)\end{pmatrix}.$$

Algorithm 8 (GVB with factor decomposition)**. Input***: Initial* $\lambda^{(0)}:=(\mu^{(0)},b^{(0)},c^{(0)})$*, momentum weight* α_m*, fixed learning rate* ϵ_0*, learning rate threshold* τ*, rolling window size* t_W *and maximum patience* P*.* **Model-specific requirement***: function* $h(\theta)$ *and* $\nabla_\theta h(\theta)$*.*

- *Initialization*
  - *Generate* $\varepsilon_{1,s}\sim\mathcal{N}(0,1)$ *and* $\varepsilon_{2,s}\sim\mathcal{N}_d(0,I_d)$*,* $s=1,...,S$*.*
  - *Compute the LB gradient estimate* $\widehat{\nabla}_\lambda\text{LB}(\lambda^{(0)})$ *as in* (16)*, and then compute the natural gradient* $\widehat{\nabla_\lambda\text{LB}}(\lambda^{(0)})^{nat}$ *using Algorithm 7.*
  - *Set momentum gradient* $\overline{\nabla_\lambda\text{LB}}:=\widehat{\nabla_\lambda\text{LB}}(\lambda^{(0)})^{nat}$*.*
  - *Set* $t=0$*, patience*$=0$ *and* `stop=false`*.*
- *While* `stop=false`*:*
  - *Generate* $\varepsilon_{1,s}\sim\mathcal{N}(0,1)$ *and* $\varepsilon_{2,s}\sim\mathcal{N}_d(0,I_d)$*,* $s=1,...,S$*.*
  - *Compute the LB gradient estimate* $\widehat{\nabla}_\lambda\text{LB}(\lambda^{(t)})$ *as in* (16)*, and then compute the natural gradient* $\widehat{\nabla_\lambda\text{LB}}(\lambda^{(t)})^{nat}$ *using Algorithm 7.*
  - *Compute the momentum gradient*

$$\overline{\nabla_\lambda\text{LB}}=\alpha_m\overline{\nabla_\lambda\text{LB}}+(1-\alpha_m)\widehat{\nabla_\lambda\text{LB}}(\lambda^{(t)})^{nat}.$$

  - *Compute* $\alpha_t=\min(\epsilon_0,\epsilon_0\frac{\tau}{t})$ *and update*

$$\lambda^{(t+1)}=\lambda^{(t)}+\alpha_t\overline{\nabla_\lambda\text{LB}}.$$

  - *Compute the lower bound estimate*

$$\widehat{LB}(\lambda^{(t)}):=\frac{1}{S}\sum_{s=1}^S h_{\lambda^{(t)}}(\theta_s).$$

  - *If* $t\geq t_W$*: compute the moving averaged lower bound*

$$\overline{LB}_{t-t_W+1}=\frac{1}{t_W}\sum_{k=1}^{t_W}\widehat{LB}(\lambda^{(t-k+1)}),$$

    and if $\overline{LB}_{t-t_W+1}\geq\max(\overline{\text{LB}})$ *patience = 0; else patience:=patience+1.*
  - *If patience*$\geq P$*,* `stop=true`*.*
  - *Set* $t:=t+1$*.*

Example 5: Bayesian deep neural net

The census dataset extracted from the U.S. Census Bureau database is available on the UCI Machine Learning Repository `https://archive.ics.uci.edu/ml/index.php`. The prediction task is to determine whether a person's income is over \$50K per year, based on 14 attributes including age, workclass, race, etc, of which many are categorical variables. After using dummy variables to represent the categorical variables, there are 103 input variables. The training dataset has 24,129 observations and the validation set has 6032 observations. As is typical in Deep Learning applications, here we use the minus log-likelihood computed on the validation set rather than using the lower bound to judge when to stop the VB training algorithm. The structure of the neural net is [104,100,100]: input layer with 104 variables including the intercept, and two hidden layers each with 100 units. The size of parameters θ is 20,500. Algorithm 8 for training this deep learning model stopped after 2812 iterations. Figure 5 plots the validation loss over the iterations.

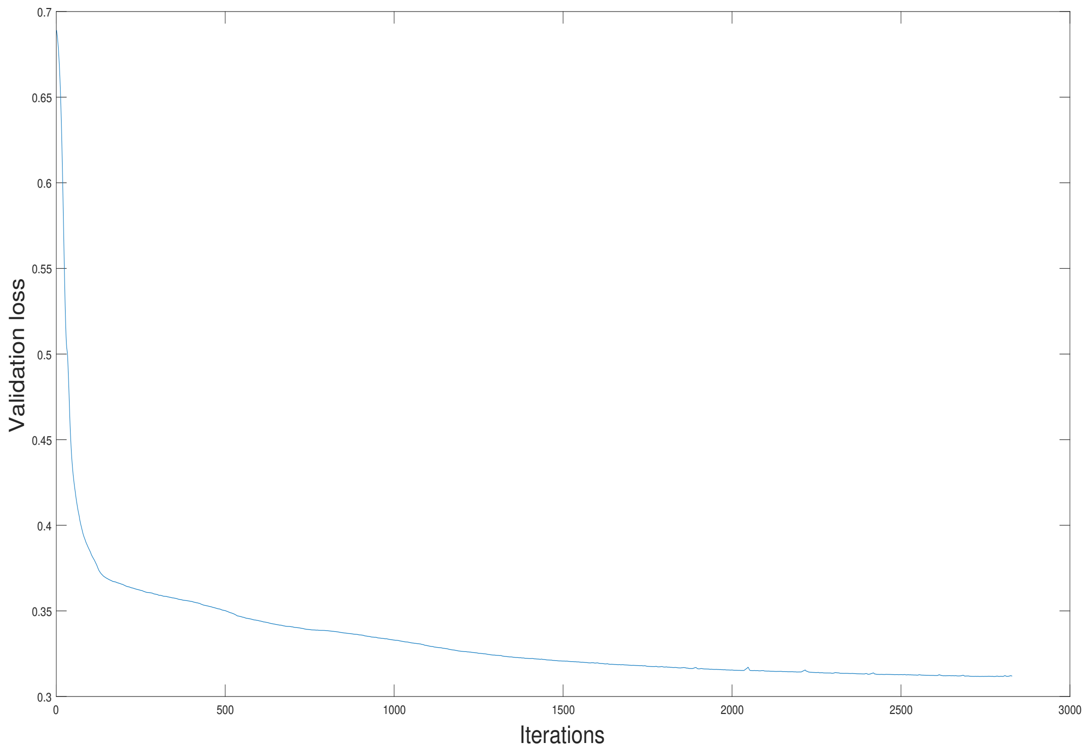

Figure 5: Example 5: GVB with factor decomposition method in Algorithm 8 for deep neural net modelling. The plot shows the validation loss over iterations.

4 A quick note on the bibliography

This isn't a review paper, I therefore made no attempt to give a comprehensive literature review on Variational Bayes. This section is to give a short list of further reading for the interested reader.

One of the earliest textbook introductions to VB is Bishop (2006), and Ormerod and Wand (2010) provide an introduction of VB to statisticians. These two references provide an excellent focus on MFVB. Compared to MFVB, FFVB is developed more recently with great contributions not only from machine learning but also the statistics community. Further reading on control variate can be found in Paisley et al. (2012); Nott et al. (2012); Ranganath et al. (2014); Tran et al. (2017). See Kingma and Ba (2014); Duchi et al. (2011) and Zeiler (2012) for the adaptive learning methods. The natural gradient is first introduced in statistics by Rao (1945), popularized in machine learning by Amari (1998), and developed further in Martens (2014); Mishkin et al. (2018); Lin et al. (2019) and Tran et al. (2020). The reparameterization trick can be found in Kingma and Welling (2014); Titsias and Lázaro-Gredilla (2014). The GVB with the Cholesky decomposition in Section 3.5.1 is borrowed from Titsias and Lázaro-Gredilla (2014) and Tan and Nott (2018), and more details of Algorithm 8 together with the DL Example 5 can be found in Tran et al. (2020).

References

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

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

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

Kingma, D. and Welling, M. (2014). Auto-encoding Variational Bayes. *Proceedings of the 2nd International Conference on Learning Representations (ICLR).*

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980.*

Lin, W., Khan, M. E., and Schmidt, M. (2019). Fast and simple natural-gradient variational inference with mixture of exponential-family approximations. *Proceedings of the 36th International Conference on Machine Learning (ICML).*

Martens, J. (2014). New insights and perspectives on the natural gradient method. *arXiv preprint arXiv:1412.1193.*

Mishkin, A., Kunstner, F., Nielsen, D., Schmidt, M., and Khan, M. E. (2018). Slang: Fast structured covariance approximations for Bayesian deep learning with natural gradient. *Neural Information Processing Systems (NIPS).*

Nott, D. J., Tan, S., Villani, M., and Kohn, R. (2012). Regression density estimation with variational methods and stochastic approximation. *Journal of Computational and Graphical Statistics*, 21:797–820.

Ormerod, J. T. and Wand, M. P. (2010). Explaining variational approximations. *American Statistician*, 64:140–153.

Paisley, J., Blei, D., and Jordan, M. (2012). Variational Bayesian inference with stochastic search. In *International Conference on Machine Learning*, Edinburgh, Scotland, UK.

Ranganath, R., Gerrish, S., and Blei, D. M. (2014). Black box variational inference. In *International Conference on Artificial Intelligence and Statistics*, volume 33, Reykjavik, Iceland.

Rao, C. R. (1945). Information and accuracy attainable in the estimation of statistical parameters. *Bull. Calcutta. Math. Soc*, 37:81–91.

Tan, L. and Nott, D. (2018). Gaussian variational approximation with sparse precision matrices. *Stat Comput*, (28):259–275.

Titsias, M. and Lázaro-Gredilla, M. (2014). Doubly stochastic Variational Bayes for non-conjugate inference. *Proceedings of the 29th International Conference on Machine Learning (ICML).*

Tran, M., Nott, D., and Kohn, R. (2017). Variational Bayes with intractable likelihood. *Journal of Computational and Graphical Statistics*, 26(4):873–882.

Tran, M.-N., Nguyen, N., Nott, D., and Kohn, R. (2020). Bayesian deep net GLM and GLMM. *Journal of Computational and Graphical Statistics.*

Zeiler, M. D. (2012). Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701.*