6.882: Bayesian Modeling and Inference **Lecture Date: February 13, 2018**

Stochastic Variational Inference

Lecturer: Tamara Broderick *Scribe: Yuri Ahuja, Aaron Sonabend, Tamara Broderick*

1 Recap and Motivation

In previous classes, we discussed Bayesian graphical models such as Latent Dirichlet Allocation as well as various algorithms to train them. With potentially thousands to millions of latent variables and parameters - and equally massive training datasets - it is computationally infeasible to compute the high-dimensional posterior distributions of these models, forcing us to turn to approximating methods. As we have seen, variational Inference approximates the posterior with a distribution belonging to a family of "nice" (typically parametric) distributions. We solve an optimization problem to find the "closest" of the nice distribution; this optimization problem often amounts to optimizing a set of *variational parameters* that describe the "nice" distributions. Note that *variational parameters* describe the approximation and should not be confused with *parameters* in our model. However, with the ever increasing scale of data sizes, batch variational methods are may be computationally intractable on data sets of interest: indeed, Hoffman, Blei, Bach (2010) show that whereas batch mean-field variational Bayes coordinate ascent can learn an approximate LDA posterior on 10,000 documents in about 17 minutes, it takes about 8.3 hours to learn an approximate posterior on 98,000 documents.

Conversely, stochastic gradient descent (SGD) has become a widely used tool in machine learning for speeding up optimization. In this class, we ask whether SGD could usefully be applied to our mean-field variation Bayes optimization problem.

1.1 Recap: Variational Inference

Variational Inference is the method of approximating a potentially complex exact posterior distribution $p(\theta|x)$ with a "nice" distribution $q(\theta)$, where θ are the parameters of the model, and x are observations. In Bayesian inference, we treat parameters as random variables. In particular, the variational Bayes optimization problem is to find the distribution $q \in Q$ that minimizes the KL divergence $\mathrm{KL}(q||p(\cdot|x)$ across $q \in Q$. We have previously seen that minimizing the KL in this direction is equivalent to maximizing the evidence lower bound (ELBO), a lower bound on the log probability of the data, $\log p(x)$:

$$ELBO(q) = \mathbb{E}_q[\log p(x, \theta)] - \mathbb{E}_q[\log q(\theta)].$$

In particular, recall that

$$\begin{aligned} KL(q(\cdot)||p(\cdot|x)) &= \mathbb{E}_q[\log q(\theta)] - \mathbb{E}_q[\log p(\theta|x)] \\ &= \mathbb{E}_q[\log q(\theta)] - (\mathbb{E}_q[\log p(x, \theta)] - \mathbb{E}_q[\log p(x)]) \\ &= -ELBO(q) + \log p(x). \end{aligned}$$

Note that $\log p(x)$ is a constant in θ, so the expectation over θ (distributed according to q) vanishes in the final line. We see that the fact that the ELBO is a "lower bound" on the $logp(x)$ follows from the non-negativity of the KL divergence.

Recall that mean-field variational Bayes (MFVB) assumes that the "nice" distributions in Q factorize across the components of θ:

$$Q = \{q : q(\theta) = \prod_{j=1}^{J} q_j(\theta_j)\}.$$

We often omit the subscript on the q_j for convenience when deriving algorithms for mean-field variational Bayes optimization.

Recall that the mean-field variational Bayes coordinate ascent algorithm (MFVB-CA) updates each q_j in turn.

1.2 Recap: Latent Dirichlet Allocation

In the previous class, we considered the problem of discovering topics in a corpus of docuemnts. In particular, we consider parameters β (the topics), z (the assignments of words to topics), and θ (the topic proportions in each document) and data w (the words in each document).

We can now choose the LDA model (in particular the model in Figure 2b of Lecture2.pdf scribe notes) to describes the prior $p(\beta, z, \theta)$ and likelihood $p(w|\beta, z, \theta)$.

With our parameters and models in hand, our next step is to find the posterior. Since we cannot calculate the exact posterior, we must approximate the posterior.

We considered using an MFVB approximation, and last class we derived a mean-field variational Bayes coordinate ascent (MFVB-CA) algorithm. Note that we were able to derive that the components of q took on familiar parametric distributions. And we observed that this was no coincidence; the conditional conjugacy built into the model allowed us to do this. In general, we would not expect to have such nice, easy-to-execute steps for any possible generative model.

We recap our MFVB-CA algorithm for LDA here:

- Initialize ϕ, γ, λ (the variational parameters)
- Repeat until convergence (e.g., of the ELBO or variational parameters)
  - $\forall d, n, q^*(z_{dn}) = \text{Cat}(z_{dn}|\phi_{dn})$, where
    $$\forall k, \phi_{dnk} \propto \exp[\mathbb{E}_{q^*} \log \beta_{kw_{dn}} + \mathbb{E}_{q^*} \log \theta_{dk}]$$
    Note that the proportionality is across indices k since ϕ_{dnk} must form a distribution across $k \in [K]$. We discuss the computation of the expectations below.
  - $\forall d, q^*(\theta_d) = \text{Dir}(\theta_d|\gamma_d)$, where
    $$\forall k, \gamma_{dk} \leftarrow \alpha + \sum_n \phi_{dnk}$$
  - $\forall k, q^*(\beta_k) = \text{Dir}(\beta_k|\gamma_k)$
    $$\forall v, \lambda_{kv} \leftarrow \eta + \sum_{d,n} \phi_{dnk} \mathbb{1}\{w_{dn} = v\}$$

In order to compute the expectations in the ϕ_{dn} step, note that $\mathbb{E}_{q^*} \log \theta_{dk}] = \Psi(\gamma_{dk}) - \Psi(\sum_{j=1}^{K} \gamma_{dj})$, where Ψ is the *digamma function*. A similar relationship holds for the other expectation since the digamma function arises when taking the expectation of the log of a component of a Dirichlet-distributed random vector. The digamma function is also the derivative of the log gamma function.

Notice that we alternate between updating the variational parameters (ϕ, γ) corresponding to distributions over local parameters (z, θ, respectively) and updating the variational parameters (λ) corresponding to distributions over global parameters (β, the topics). For one, updating all the local parameters before updating the global parameter even once may be wasteful of computation. For another, note that all of the local updates can be accomplished once we have the λ values.

We ask ourselves the usual questions about this approximation algorithm. (1) Is it **fast**? Hoffman, Blei, Bach (2010) show that whereas batch mean-field variational Bayes coordinate ascent can learn an approximate LDA posterior on 10,000 documents in about 17 minutes, it takes about 8.3 hours to learn an approximate posterior on 98,000 documents. Wikipedia has millions of documents, and we'd like to be able to run quickly on documents at web size. So MFVB-CA may be fast here, but it's not as fast as we'd like. (2) Is it **automatic**? Notice the tedious hand derivations we had to go through to get this algorithm. We'd have to perform them anew for any new model. And the practicality of the method relied heavily on conditional conjugacy in the model. (3) Is it **accurate**? We'll revisit this question later, but we agreed in class that the results seemed useful, and so it would be desirable to improve the inference algorithm in the first two directions.

2 Detour: Stochastic gradient descent

In this section, we consider stochastic gradient descent (SGD) and whether we might be able to apply it to speed up our MFVB optimization problem. We will start by switching notation to describe a general problem solved by optimization algorithms in general and SGD in particular. We will alert the reader when we return to the LDA problem and notation.

Our typical goal in many statistical or machine learning problems is to minimize an objective of the following form:

$$f(\lambda) := \mathbb{E}_X f(X, \lambda)$$

over λ. That is λ serves as an index to the *loss* $f(X, \lambda)$. The expectation of the loss, $f(\lambda)$, is typically called the *risk*. Our goal, as you have seen e.g., in 6.867, is to choose λ to minimize the risk.

How can we minimize the risk? Well, let's think about what optimization algorithms we know. We might try **gradient descent**.

Here is a gradient descent algorithm for the risk:

- Initialize $\lambda^{(0)}$.
- Iterate $t = 1, 2, \ldots$
  - $\lambda^{(t)} = \lambda^{(t-1)} - \rho_t \nabla_\lambda f(\lambda)$

Note that since we are considering gradient descent instead of gradient ascent, we have a minus sign in the update rather than a plus sign. Also, here ρ_t is a sequence of step sizes.

One big problem with this algorithm is that we typically think of X as our data, and we don't know the distribution of our data in advance. Rather we observe some data $x_{1:N}$, which we often assume to be iid from the distribution of X. Thus, we often define the *empirical risk*:

$$\frac{1}{N} \sum_{n=1}^{N} f(x_n, \lambda).$$

By the law of large numbers, the empirical risk should approach the risk as N becomes large. And notice that the expectation of the empirical risk is exactly the risk. That is, the empirical risk is an *unbiased estimator* of the risk.

So we could instead try a gradient descent algorithm for the empirical risk:

- Initialize $\lambda^{(0)}$.
- Iterate $t = 1, 2, \ldots$
  - $\lambda^{(t)} = \lambda^{(t-1)} - \rho_t \nabla_\lambda \left[\frac{1}{N} \sum_{n=1}^{N} f(x_n, \lambda) \right]$

Note that

$$\nabla_\lambda \left[\frac{1}{N} \sum_{n=1}^{N} f(x_n, \lambda) \right] = \frac{1}{N} \sum_{n=1}^{N} \nabla_\lambda \left[f(x_n, \lambda) \right].$$

But there are still some problems with this algorithm. For one, we have to sum up over all the data at every step. If we have a huge amount of data, that could be expensive. And it could also be wasteful; we might have a good estimate of the risk, and the derivative of the risk, after a much smaller amount of data. This algorithm is also an inherently *batch* rather than *streaming* algorithm. It requires the full data set before we can run a single step, and it doesn't adapt easily when we get new data. Plus, we have to store all the data in memory.

Stochastic gradient descent addresses these concerns. Note that to move from our first algorithm (gradient descent for the risk) to our second algorithm (gradient descent for the empirical risk) above, we substitute the risk with an unbiased estimate of the risk—or more precisely, we substitute an unbiased estimate of the derivative of the risk. But in fact, $f(x_n, \lambda)$ is an unbiased estimate of the risk for any n, and $\nabla_\lambda f(x_n, \lambda)$ is an unbiased estimate of $\nabla_\lambda f(\lambda)$. (Here and in what follows, we assume we have sufficient regularity to exchange the derivative and integral.) This line of reasoning provides a very hand-wavy motivation for the stochastic gradient descent algorithm:

- Initialize $\lambda^{(0)}$.
- Iterate $t = 1, 2, \ldots$
  - $\lambda^{(t)} = \lambda^{(t-1)} - \rho_t \nabla_\lambda f(x_t, \lambda)$

This algorithm actually turns out to get arbitrarily close to minimizing the risk, rather than just the empirical risk for a fixed number of data points N, if the step sizes are chosen appropriately. Note also that now we imagine we keep getting new data points x_t in a streaming fashion.

Before we state a theoretical result along these lines, let's make one more adjustment. In each step, rather than moving in the direction of the gradient, we could have moved in the direction of the minimizer with respect to λ. We make this final adjustment to arrive at the Robbins-Monro algorithm (see, e.g., `https://en.wikipedia.org/wiki/Stochastic_approximation#Robbins%E2%80%93Monro_algorithm`):

- Initialize $\lambda^{(0)}$.
- Iterate $t = 1, 2, \ldots$
  - Let $\hat{\lambda}_t$ satisfy $\nabla_\lambda f(x_t, \lambda) = 0$.
  - $\lambda^{(t)} = \lambda^{(t-1)} - \rho_t \hat{\lambda}_t$.

An exciting aspect of this algorithm is the associated theoretical guarantees on how well it minimizes the risk.

Theorem 1 (Sketch of Robbins-Monro Theorem)**.** *Suppose* $\sum_{t=1}^{\infty} \rho_t = \infty$ *and* $\sum_{t=1}^{\infty} \rho_t^2 < \infty$*. Then, under appropriate regularity conditions, the Robbins-Monro algorithm described above converges to a local minimum of* $f(\lambda)$*.*

Now we have a fast, streaming algorithm for minimizing the risk. Sometimes, for additional stability, a *minibatch* is used instead of a single data point at each iteration. E.g., let S be the number of data points in the minibatch. Then, instead of $\nabla_\lambda f(x_t, \lambda)$, we use $\frac{1}{S}\sum_{s=1}^{S} \nabla_\lambda f(x_{S(t-1)+s}, \lambda)$, where t still indexes the iteration.

3 Can we apply SGD to (MF)VB?

Variational Bayes in general describes an optimization problem, so we might think that we could apply SGD to solve that optimization problem. But recall from the previous section that SGD solves a particular form of optimization problem (minimizing the risk), so first we need to put the VB optimization problem into that form.

Recall the general VB optimization problem can be written as

$$\begin{aligned}
&\text{argmax}_{q\in Q}\mathbb{E}_{\theta\sim q} \log \frac{p(\theta, x_{1:N})}{q(\theta)} \\
&\text{where we use the ELBO formulation of the problem} \\
&= \left[\text{argmax}_{q\in Q} \sum_{n=1}^{N} \mathbb{E}_{\theta\sim q} \log p(x_n|\theta)\right] + \mathbb{E}_{\theta\sim q} \log p(\theta) - \mathbb{E}_{\theta\sim q} \log q(\theta) \\
&= \text{argmax}_{\lambda} \left[\sum_{n=1}^{N} \mathbb{E}_{\theta\sim q_\lambda} \log p(x_n|\theta)\right] + \mathbb{E}_{\theta\sim q_\lambda} \log p(\theta) - \mathbb{E}_{\theta\sim q_\lambda} \log q_\lambda(\theta),
\end{aligned}$$

where in the last line we suppose that λ indexes the distributions in Q to make the objective look more like the risk we saw in the previous section.

Nonetheless, this objective still isn't quite a risk, or even an empirical risk. Let's think more carefully about why that is. In the risk formulation, we imagined that as we get more data, the empirical risk becomes a better and better estimator for the risk: $\frac{1}{N} f(x_n, \lambda) \xrightarrow{a.s.} f(\lambda)$ as $N \to \infty$.

But the Bayesian inference problem is fundamentally different. We actually have very different *optimization problems* for each number N of data points. For instance, consider a prior $p(\theta)$. It might be very spread out. After one data point, our posterior $p(\theta|x_1)$ is less spread out. After 10 data points, our posterior $p(\theta|x_{1:10})$ is even less spread out and a fundamentally different distribution than the first two. At every stage, we want to approximate $p(\theta|x_{1:N})$ by some $q_N^*(\theta)$, and we don't expect q_M^* should look anything like q_N^* if $M \neq N$.

A second, more subtle point is that we don't want to integrate out the data as in the risk framework. In Bayesian inference, we treat the data as fixed—to define our posterior.

OK, but we'd really like the speed gains of SGD! So can we apply SGD to our problem in some other, nonstandard way? Let's try again to rewrite our objective in the form required by SGD.

$$\mathbb{E}_{\theta\sim q_\lambda} \log \frac{p(\theta, x_{1:N})}{q_\lambda(\theta)} = \frac{1}{N}\sum_{n=1}^{N} \left[N\mathbb{E}_{\theta\sim q_\lambda} \log p(x_n|\theta) + \mathbb{E}_{\theta\sim q_\lambda} \log p(\theta) - \mathbb{E}_{\theta\sim q_\lambda} \log q_\lambda(\theta)\right]$$

This final line will be in the risk form if we define a new random variable

$$X := \begin{cases} x_n, & \text{with probability } \frac{1}{N}, \text{ for each } n \in [N] \end{cases}$$

and let the loss be defined as

$$f(x, \lambda) := N\mathbb{E}_{\theta \sim q_\lambda} \log p(x|\theta) + \mathbb{E}_{\theta \sim q_\lambda} \log p(\theta) - \mathbb{E}_{\theta \sim q_\lambda} \log q_\lambda(\theta). \tag{1}$$

Note that we are not assuming any distribution for our actual data $x_{1:N}$. The distribution of X is the empirical distribution over the observed data. Also note that these definitions depend very much on a fixed number of data points N.

With X and the loss in hand, we can use the Robbins-Monro algorithm. We substitute these choices into the general Robbins-Monro algorithm above to derive Robbins-Monro for VB:

- Initialize $\lambda^{(0)}$.
- Iterate $t = 1, 2, \ldots$
  - Draw $\tilde{x}_t \overset{iid}{\sim} \{x_n$ with probability $\frac{1}{N}$
  - $\hat{\lambda}_t$ solves $\nabla_\lambda f(\tilde{x}_t, \lambda) = 0$ for f as defined in Equation (1)
  - $\lambda^{(t)} = \lambda^{(t-1)} + \rho_t \hat{\lambda}_t$

The final line is now a plus instead of a minus since we are maximizing the ELBO rather than minimizing the objective.

A potential benefit of this algorithm is that it touches only one data point at a time, but notice that it is not streaming. Clearly, each step depends on knowing N in advance, and there is no mechanism to adjust to a different N. According to the theorem (sketch) above, if all of the Robbins-Monro regularity conditions are satisfied, this algorithm should converge to a local optimum of the exact objective (not of an approximate objective like gradient descent for the empirical risk).

4 Example: LDA

Finally, we consider an example application of the Robbins-Monro algorithm for VB; in particular, we consider using VB to find the approximate posterior for LDA. We will now revert to our LDA notation from before the discussion of various optimization algorithms. To apply our risk framework, we isolate the global parameters β, whose variational parameters are conveniently called λ (so this notation, at least, agrees with the optimization section). In particular, recall that $q^*(\beta_k) = \text{Dir}(\beta_k|\lambda_k)$.

We identify the "risk" $f(\lambda)$ and the "loss" $f(w_d, \lambda)$ for this problem using the results from the previous section. We emphasize that these are nonstandard usages of these terms, as discussed in the previous section. First, note that the optimization objective is the following, where now d indexes the documents (which correspond to the data points in the previous section) and n indexes the words within a document

(so not the same as the n index in the previous section).

$$
\begin{aligned}
f(\lambda) = \text{ELBO}_\lambda \qquad & = \mathbb{E}_{q_\lambda}\left[\sum_d \sum_n \sum_v \mathbb{1}\{z_{dn}=k\}\mathbb{1}\{w_{dn}=v\}\log\beta_{kv}\right. \\
&+ \sum_k \sum_v (\eta-1)\beta_{kv} - \sum_k \sum_v (\lambda_{kv}-1)\log\beta_{kv} \\
&\left. - \sum_k (-\log\Gamma(\sum_v \lambda_{kv}) + \sum_v \log\Gamma(\lambda_{kv})\right] \\
&+ \text{ terms that are constant in } \lambda
\end{aligned}
$$

Using the results from the previous section, we extract the data-specific "loss" terms $f(w,\lambda)$:

$$
\begin{aligned}
f(w_d,\lambda) = D\sum_n \sum_k \sum_v \phi_{dnk}\mathbb{1}\{w_{dn}=v\}\mathbb{E}_{q_\lambda}\log\beta_{kv} \\
+ \sum_k \sum_v (\eta-1)\mathbb{E}_{q_\lambda}\log\beta_{kv} \\
+ \sum_k \sum_v (\lambda_{kv}-1)\mathbb{E}_{q_\lambda}\log\beta_{kv} \\
+ \sum_k \log\Gamma(\sum_v \lambda_{kv}) \\
- \sum_k \sum_v \log\Gamma(\lambda_{kv})
\end{aligned}
$$

Now we need to find the value of λ that solves $\nabla_\lambda f(w,\lambda) = 0$ in order to finish filling in the algorithm for Robbins-Monro for VB. To that end, note that

$$
\frac{df(w_d,\lambda)}{d\lambda_{kv}} = \sum_{u=1}^{V} \frac{d\mathbb{E}_{q_\lambda}\log\beta_{ku}}{d\lambda_{kv}}\left[D\sum_n \phi_{dnk}\mathbb{1}\{w_{dn}=u\} + \eta - \lambda_{ku}\right],
$$

Note that there would be two additional terms, $+\mathbb{E}_{q_\lambda}\log\beta_{kv}$ and $\Psi(\sum_u \lambda_{kv}) - \Psi(\lambda_{kv})$ that appear when taking the derivative of the previous equation, but these terms exactly cancel.

Finally, we solve for the $\hat{\lambda}_d$ that sets this derivative to zero, which is:

$$
\hat{\lambda}_d := \eta + D\sum_n \phi_{dnk}\mathbb{1}\{w_{dn}=v\}.
$$

Finally, then, we have our Robbins-Monro algorithm for VB applies to LDA (where we incorporate coordinate ascent steps in the local parameters to finish the algorithm):

- Initialize $\lambda^{(0)}$
- Iterate $t = 1, 2, \ldots$
  - Choose $\hat{d} \overset{iid}{\sim} \{d$ with probability $\frac{1}{D}$
  - Update until converged: $\phi_{\hat{d}nk}, \gamma_{\hat{d}k}$ across n and k values
  - $\forall k, v, \hat{\lambda}_{kv} \leftarrow \eta + D\sum_n \phi_{\hat{d}nk}\mathbb{1}\{w_{\hat{d}n}=v\}$
  - $\forall k, v, \lambda_{kv}^{(t)} \leftarrow \lambda_{kv}^{(t-1)} + \rho_t\hat{\lambda}_{kv}$

Modulo some small changes in the way ρ_t appears in the update step, this is essentially SVI for LDA as described in the reading. Some ideas worth pondering are the following. We didn't use the natural gradient to derive this algorithm. Why not? Did Hoffman et al actually use the natural gradient in their case either?