Stan's ADVI

Alexandre Piche

May 1, 2016

1 Introduction

Variational inference (VI) is an approximate Bayesian inference technique that gained a lot of ground in the recent years. It differs from the MCMC techniques by posing the estimation of the conditional distribution as an optimization problem. VI is usually a lot faster than conventional MCMC methods, however it can be tedious to derive the algorithm for a new model.

In this project, we will explore how the Stan's ADVI algorithm compares to exact MCMC methods in term of performance accuracy and computation time. We will explore three types of model: logistic regression, Bayesian neural network, and Gaussian process on the Dorothea dataset.

2 Variational Inference

In Bayesian statistics, we are often interested in approximating the conditional distribution of hidden variables z given the observed variables x.

$$p(z|x) \quad = \quad \frac{p(z,x)}{p(x)}$$

Unfortunately, often $p(x) = \int_z p(z,x)$ is intractable or very expensive to compute. VI is an optimization technique to approximate $p(z|x)$.

2.1 Evidence Lower Bound

In VI to approximate the conditional distribution, we minimize a notion of distance between $p(z|x)$ and a simple distribution. We first need to specify a family of distributions $\mathcal{Q}$ over the latent variables z. Then, we optimize $q(z) \in \mathcal{Q}$ to minimize the KL divergence:

$$q^*(z) \quad = \quad \arg\min_{q(z)\in\mathcal{Q}} KL(q(z)||p(z|x))$$

where $KL(q(z)|p(z|x))$ is given by:

$$\begin{aligned} KL(q(z)||p(z|x)) &= E[\log q(z)] - E[\log p(z|x)] \\ &= E[\log q(z)] - E[\log p(z,x)] + \log p(x) \end{aligned}$$

where $p(x)$ is intractable in most cases, but is a constant wrt to $q(z)$. Instead, we optimize the evidence lower bound (ELBO):

$$\begin{aligned} ELBO(q) &= E[\log p(z,x)] - E[\log q(z)] \\ &= E[\log p(x|z)] - KL(q(z)||p(z)) \end{aligned}$$

From the second equation, it is easy to see that maximizing the ELBO is the same as minimizing the KL divergence.

The ELBO also has a statistical intuitive interpretation. Specifically, writing it as:

$$ELBO(q) = E[\log p(z)] + E[\log p(x|z)] - E[\log q(z)]$$

The first term is the expected likelihood, thus it encourages the distribution to place the mass on configurations of the latent variables that explain the observed data[1]. While the second term is the negative divergence between the variational distribution and the prior, thus encouraging the distribution to be close to the prior. Those properties are aligned with the Bayesian framework.

The $ELBO$ is that it lower-bounds the log evidence, $\log p(x) \geq ELBO(q)$ for any $q(z)$ [1], since:

$$\log p(x) = KL(q(z)||p(z|x)) + ELBO(q)$$

2.2 Mean-Field Variational Inference

To ease the computation, we usually make the assumption that the latent variables are mutually independent, specifically:

$$q(z) = \prod_{j=1}^{m} q_j(z_j)$$

Note that each latent variable z_j has its own distribution $q_j \in \mathcal{Q}$. The mean-field approximation can capture any marginal distribution of the latent variables, but not the correlation between them.

2.3 Coordinate Ascent Variational Inference

We usually maximize the ELBO using coordinate ascent variational inference (CAVI). It iteratively optimizes each distribution q_j, while holding q_{-j} fix. Taking advantage of using mean-field VI, we can write the optimal $q*_j(z_j)$ as:

$$\begin{aligned} q*_j(z_j) &\propto \exp\{E_{-j}[\log p(z_j|z_{-j},x)]\} \\ &\propto \exp\{E_{-j}[\log p(z_j,z_{-j},x)]\} \end{aligned}$$

The ELBO can be written as:

$$ELBO(q_j) = E_j[E_{-j}[\log p(z_j,z_{-j},x)]] - E_j[\log q_j(z_j)] + c$$

where $\log q_{-j}(z_{-j})$ is absorbed by the constant c, since it does not depend on z_j.

2.4 Stochastic Variational Inference

Going through the entire large dataset to compute the ELBO is expensive. Instead, we can use batch of data to maximize the ELBO, a method call stochastic variational inference [3]. Specifically, we can use batch of data to obtain noisy, but unbias gradients, and to converge to a local minimum. The step-size must satisfy [7] :

$$\sum_t \epsilon_t = \infty \quad ; \quad \sum_t \epsilon_t^2 < \infty$$

3 Automatic Differentiation Variational Inference

We now have the tools to explore Stan Automatic Differentiation Variational Inference (ADVI) algorithm.

3.1 Transformation of Constrained Variables

ADVI first transforms the latent variables to the real space. Doing so removed constraints such as the variance being positive. Let T be a one-to-one differentiable mapping from the constrained space to the real space and $\zeta = T(\theta)$ [4]. The transformed joint density is given by:

$$g(X,\zeta) = p(X,T^{-1}(\zeta))|detJ_{T^{-1}}|$$

where p is the original joint density of the latent variables, and $J_{T^{-1}}$ is the jacobian of the inverse of the transformation T.

$$L(\phi) = E_{q(\zeta|\phi)}[\log p(y, T^{-1}(\zeta)) + \log |detJ_{T^{-1}}(\zeta)|] - E_{q(\zeta|phi)}[\log q(\zeta|\phi)]$$

3.2 Variational Approximation

ADVI then approximates the variational distribution $q(\zeta)$ with a mean-field Gaussian, such that:

$$q(\zeta|\phi) = N(\zeta|\mu, \sigma) = \prod_{k=1}^{K} N(\zeta_k|\mu_k, \sigma_k)$$

where the vector $\phi = (\mu_1, \ldots, \mu_K, \sigma_1, \ldots, \sigma_K)$. The ELBO is thus given by:

$$\mathcal{L}(\mu, \sigma) = E_{q(\zeta)}[\log p(X, T^{-1}(\zeta)) + \log |detJ_{T^{-1}}(\zeta)|] + \frac{K}{2}(1 + \log 2\pi) + \sum_{k=1}^{K} \log \sigma_k$$

3.3 Stochastic Optimization

We could maximize the ELBO on the real space as:

$$\begin{aligned} \mu*, \sigma* &= \text{arg max}_{\mu,\sigma} \mathcal{L}(\mu, \sigma) \\ &\quad \text{such that } \sigma > 0 \end{aligned}$$

but the expectation is intractable. Instead, we transform σ to be unconstrained as $\omega = \log \sigma$. We can use the elliptical standardization, by defining $\eta = S_{\mu,\omega}(\zeta) = \text{diag } (\exp(\omega)^{-1})(\zeta - \mu)$ The density is now given by:

$$\begin{aligned} q(\eta; 0, I) &= N(\eta; 0, I) \\ &= \prod_k N(\eta_k; 0, 1) \end{aligned}$$

and the optimization problem is now given by:

$$\begin{aligned} \mu*, \omega* &= \text{arg max}_{\mu,\omega} \mathcal{L}(\mu, \omega) \\ &= \text{arg max}_{\mu,\omega} E\left[\log p(X, T^{-1}(S^{-1}_{\mu,\omega}(\eta))) + \log |\det\ J_{T^{-1}}(S^{-1}_{\mu,\omega}(\eta))| \right] + \sum_{k=1}^{K} \omega_k \end{aligned}$$

Where the gradients are given by:

$$\begin{aligned} \nabla_\mu \mathcal{L}(\mu, \omega) &= E_{N(\eta)}[\nabla_\theta \log p(X, \theta) \nabla_\zeta T^{-1}(\zeta) + \nabla_\zeta \log |\det\ J_{T^{-1}}(\zeta)|] \\ \nabla_{\omega_k} \mathcal{L}(\mu, \omega) &= E_{\eta_k}[(\nabla_{\theta_k} \log p(X, \theta) \nabla_{\zeta_k} T^{-1}(\zeta) + \nabla_{\zeta_k} \log |\det\ J_{T^{-1}}(\zeta)|) \eta_k \exp(\omega_k)] + 1 \end{aligned}$$

We use MC integration to get an unbias estimate of the gradient that we can use in SVI.

ADVI has complexity $O(BMK)$ per iteration, where B is the batch size, M the number of MC samples while CAVI has complexity $O(NK)$. Note that the optimization of the ELBO stop when the change is under a certain threshold, 0.01 in our case.

4 Experiments on the Dorothea Dataset

In 2003, NIPS had a features selection challenge consisting of 5 datasets. The biggest of all was the Dorothea dataset with 100000 structural molecule features. The task require predicting whether a chemical compound is binding to thrombin or not. This is an important task related to drug discovery. Given that half the features were designed to be irrelevant, feature selection is at the center of the problem. We will explore three methods: logistic regression (LR), a Bayesian neural network (BNN), and a Gaussian process (GP). For every models, we experimented with a variant of automatic relevance determination (ARD). To make computations reasonable, we used PCA to reduce the dimension to 75 inputs.

We will use the balance error rate (BER) to measure the accuracy of our models:

$$BER = 0.5 \times \left(\frac{TP}{TP + FN} + \frac{TN}{TN + FP} \right)$$

where TP is true positive, FN is false negative, TN is true negative, and FP is false positive.

4.1 Logistic Regression

Logistic regression is one of the simplest classification tool, but is performing well on real world problems. It can divide correctly any linearly divisible dataset.

4.1.1 LR Results

I used a normal prior bounded on $[-0.5, 0.5]$ for the β to avoid some awkward convergence issues, and an unbounded normal prior for the bias term α. Both normal priors have unknown mean.

LR Comparative Results		
Methods	BER	Computation time
HMC	0.223	41.87
ADVI	0.230	4.73

The accuracy and computation time are fairly simillar in both cases, but we can note that ADVI is faster but does not reach the same accuracy level than HMC.

4.1.2 LR-ARD Results

We now let each β_j come from $N(0, \sigma_j)$, where σ_j is unknown. As the posterior for $\sigma_j \to 0$, β_j will be near 0 and the feature will have almost no impact on the prediction.

LR-ARD Comparative Results		
Methods	BER	Computation time
HMC	0.216	94.38
ADVI	0.226	5.50

In both the LR and the LR-ARD, the ADVI is a lot faster in proportion to HMC, but comparable in absolute term, and the balance error rate is comparable for the two methods.

4.2 Bayesian Neural Network

Radford Neal used NewBayes, a Bayesian neural network, in the 2003 competition to attain the best average predictive performance over the 5 datasets. He used a Dirichlet Diffusion tree, to average different predictions, thus we won't be able to replicate his results here.

We build a neural network with one hidden layer and 25 neurons. The stan code for the neural network is inspired by Herra Huu's code available at: https://groups.google.com/forum/!topic/stan-users/3QBBpo11Lus . It has been modified to use ARD and tanh activation function [5].

4.2.1 BNN Results

For every weights, I used a normal prior with mean 0 and unknown variance specific for each level.

BNN Comparative Results		
Methods	BER	Computation time
HMC	0.2704	36325.61
ADVI	0.5	359.39

The results clearly highlight the trade-off between accuracy and speed. Using the HMC sampling results in a much higher accuracy, but the computation are 10 times more expensive. For the next sections, the HMC sampling methods were not possible due to computation constraints. They are presented since they provide us with insight on the ADVI method.

4.2.2 BNN-ARD Results

In addition to the normal prior used for the previous BNN, I used a different variance σ_j for every input weight β_j [2].

BNN-ARD Comparative Results		
Methods	BER	Computation time
HMC	-	-
ADVI	0.3542	7523.39

Using ARD improved ADVI results, but is still not as accurate at the simple BNN using HMC.

4.3 Gaussian Process

Experimenting with the BNN, I noticed that as a grow the number of nodes the performance improve, however after a certain threshold it was not computationally sustainable. Given that infinitely wide NN tends to Gaussian processes (GPs), it was natural to include the later to our analysis. GPs can model any continuous function. Since the HMC computation were too expensive using Stan, I also included results using the GPy package [8].

4.3.1 GP Results

We modeled the probability $p(y|x) = 1$ using the GP, written as:

$$p(y_i|x_i) \quad = \quad \sigma(f(x_i))$$

where σ is the logistic function, and f is the GP posterior [6]. We will use the squared exponential kernel:

$$k(x, x') \quad = \quad \tau^2 \exp\left(-\sum_{d=1}^{D} \frac{(x_d - x'_d)}{2\sigma^2} \right)$$

GP Comparative Results		
Methods	BER	Computation time
HMC	-	-
ADVI	0.4242	2354.45
GPy	0.2595	34.48

With the GPy's model, we are achieving a better result that the BNN using HMC, but surprisingly we are not achieving the accuracy results from the logistic regression. The ADVI results are nowhere as good as the one attained by GPy.

4.3.2 GP-ARD Results

Finally for the GP-ARD, we will use the ARD kernel:

$$k(x, x') \quad = \quad \tau^2 \exp\left(-\sum_{d=1}^{D} \frac{(x_d - x'_d)}{2\sigma_d^2} \right)$$

Note that as $\sigma_d^2 \to \infty$ the feature d will become irrelevant.

GP-ARD Comparative Results		
Methods	BER	Computation time
HMC	-	-
ADVI	0.4835	8835.94
GPy	0.1860	41.87

Surprisingly, increasing the flexibility of the model decrease the prediction accuracy for the ADVI. It might be due to an increasing difficulty in approximating the posterior. Using GPy's classification GP, we are attaining our best results overall which is what we expected since the GP-ARD is the most flexible model we used, and can model any function.

5 Conclusion

We compared the ADVI method to the conventional MCMC method used by Stan on three bayesian models: logistic regression, bayesian neural network and Gaussian process. From our results, it is obvious that HMC is always reaching the best balance error rate on the test set, but for more complex models HMC is very expensive and might not be computationally feasible. I think the poor accuracy performance of the ADVI are due to complex posteriors. It would be interesting to tweak the learning rate or the optimization methods, to see if they actually improve the accuracy performance. On a different note, the good performance of the logistic regression, and computation speed from GPy are both impressive.

References

[1] D. M. Blei, A. Kucukelbir, and J. D. McAuliffe. Variational inference: A review for statisticians. *arXiv preprint arXiv:1601.00670*, 2016.

[2] I. Guyon, S. Gunn, M. Nikravesh, and L. A. Zadeh. *Feature extraction: foundations and applications*, volume 207. Springer, 2008.

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

[4] A. Kucukelbir, R. Ranganath, A. Gelman, and D. Blei. Automatic variational inference in stan. In *Advances in Neural Information Processing Systems*, pages 568–576, 2015.

[5] R. M. Neal. *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media, 2012.

[6] C. E. Rasmussen. Gaussian processes for machine learning. 2006.

[7] H. Robbins and S. Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[8] The GPy authors. GPy: A gaussian process framework in python. `http://github.com/SheffieldML/GPy`, 2012–2015.