# Markov Chain Monte Carlo - Hamiltonian Monte Carlo

Alvin Shi

IAP-APPBML

January 26, 2017

## Outline

- Introduction to approximate inference and review of MCMC
- Introduction to variational inference
- Modern variational inference
  - Coordinate Ascent Variational Inference (CAVI)
  - Automatic differentiation (AD)
  - Black box variational inference (BBVI)
  - Automatic differentiation variational inference (ADVI)
- Demos in Stan, Edward (and maybe PyMC3)

## Basics

Chain Rule

$$p(\theta, x) = p(x \mid \theta)p(\theta) \tag{1}$$

Bayes' Rule

$$p(\theta \mid x) = \frac{p(\theta, x)}{p(x)} = \frac{p(x \mid \theta)p(\theta)}{p(x)} \propto p(x \mid \theta)p(\theta) \tag{2}$$

We typically refer to $p(\theta \mid x)$ as the posterior, $p(\theta)$ as the prior, and $p(x \mid \theta)$ as the likelihood.

## Problems when trying to solve for the posterior

$$p(\theta \mid x) = \frac{p(\theta, x)}{p(x)} = \frac{p(x \mid \theta)p(\theta)}{p(x)} \propto p(x \mid \theta)p(\theta) \tag{3}$$

Tthe $p(x)$ term is often difficult or impossible to evaluate analytically. Specifically, we often need to evaluate the following integral:

$$p(x) = \int_{\Theta} p(x, \theta)d\theta = \int_{\Theta} p(x \mid \theta)p(\theta)d\theta \tag{4}$$

This is either difficult or $\theta$ can takes values between $(-\infty, +\infty)$, a common situation for real-valued latent parameters such as the mean of a normal distribution, we have to evaluate an integral with bounds $\int_{-\infty}^{+\infty} p(x \mid \theta)p(\theta)d\theta$.
We need to approximate $p(x)$ some how. Two common methods: Variational Inference (VI) and Markov Chain Monte Carlo (MCMC)

## Summary of MCMC

- Construct a Markov chain whose stationary (limiting) distribution is the same as the posterior distribution of interest
- We can do so by making sure our Markov chain fulfills two important properties with respect to our posterior distribution $p(\theta \mid x)$: ergodicity and detailed balance.
- Several algorithms do this, including Metropolis-Hastings, Gibbs, and HMC. They vary in their efficiency in terms of exploring the posterior distribution.
- Stan (and most modern probabilistic programming languages) use Hamiltonian Monte Carlo (HMC) - which borrows concepts from Hamiltonian dynamics to efficiently sample from the posterior.

## Variational Inference (VI)

Use a different distribution $q(x)$ to approximate our true posterior.

- $q(x)$ chosen such that it's easy to work with, such as independent normal distributions
- Maximize how "close" $q(x)$ is to $p(\theta \mid x)$, by a measure of similarity between $q(x)$, one example is KL divergence.
- **Turns posterior inference into an optimization problem!**
- A fancier version of this happens whenever you use the `stanmodel.vb()` method.
- Very active field of research; a lot of methods I'll be presenting were just published a year or two ago.

## TLDR (Too long didn't read) summary of VI

- Turn inference into an optimization problem (many developed methods to perform optimization, such as gradient descent)
- Approximate posterior $p(\theta \mid x)$ with a (often simpler) distribution $q(\theta)$.
- Minimize a similarity metric between $p(\theta \mid x)$ and $q(\theta)$, often using a metric known as the Kullback-Leibler (KL) divergence $KL(q(\theta) \parallel p(\theta \mid x))$.
- We can parameterize our approximate distributions $q(\theta; \lambda)$ with variational parameters $\lambda$ and obtain estimates for these variational parameters $\lambda$ that minimize the KL divergence.
- Variational inference

## Intuition behind VI

Instead of try to find the exact posterior $p(\theta \mid x)$ (which, as mentioned before, is hard or often intractable because of the $p(x) = \int_{\Theta} p(\theta, x) d\theta$ term at the bottom of Bayes rule), we will approximate the posterior distribution $p(\theta \mid x)$ with an another distribution $q(\theta)$. Often $q(\theta)$ is chosen such that it is very easy to work with.

Once we choose an appropriate $q(\theta)$, we will minimize the difference between the true posterior $p(\theta \mid x)$ and $q(\theta)$, often through a metric known as the Kullback-Leibler (KL) divergence. We can minimize this metric through **optimization** to obtain a very close approximation to our posterior. Note that since we're using an approximation to $p(\theta \mid x)$, we have no theoretical guarentees that $q(\theta)$ will recapitulate the posterior exactly.

Again, just like MCMC, this is a general method that can be used to approximate any probability distribution (not just Bayesian posteriors).

## Kullbkack-Leibler divergence

KL divergence (also known as information divergence, discrimination divergence) is a measure of difference between probability distributions $p$ and $q$. It is often denoted in literature as either $KL(p \parallel q)$ or $D_{KL}(p \parallel q)$.

### KL-divergence

$$KL(P \parallel Q) = D_{KL}(P \parallel Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)}$$

$$KL(P \parallel Q) = D_{KL}(P \parallel Q) = \int_{-\infty}^{\infty} p(x) \log \frac{p(x)}{q(x)} dx = \mathbb{E}_p \log \frac{p(x)}{q(x)}$$

### KL divergence is generally not symmetric

**Generally,** $KL(Q \parallel P) \neq KL(P \parallel Q)$. Make sure you get the direction correct!

## KL divergence in Variational Inference

Recall that we want to minimize the distance between our $q(\theta)$ and $p(\theta \mid x)$. To do this, we can write the KL divergence for variational inference:

**KL-divergence for variational inference**

$$KL(q(\theta) \parallel p(\theta \mid x) = \mathbb{E}_q \left[ \log \frac{q(\theta)}{p(\theta \mid x)} \right]$$

In literature, you might often encounter $\theta$ rewritten as $Z$, they both denote the latent parameters of interest.
You might ask, why do we minimize $KL(q(\theta) \parallel p(\theta \mid x))$ instead of $KL(p(\theta \mid x) \parallel q(\theta))$? Generally the answer is that taking $\mathbb{E}_q$ makes for easier computations and generally more tractable optimization.

## Minimizing the KL-divergence between $q$ and $p$

**KL-divergence for variational inference**

$$KL(q(\theta) \parallel p(\theta \mid x) = \mathbb{E}_q \left[ \log \frac{q(\theta)}{p(\theta \mid x)} \right]$$

You might be wondering, can we just directly optimize the $KL(q(\theta) \parallel p(\theta \mid x))$ term in the equation above with respect to $q$? The answer is no, because we still have to evaluate the difficult term $p(x) = \int_\Theta p(x, \theta) d\theta$ when finding the posterior term in the denominator of the expectation, $p(\theta \mid x)$. Need a different approach...

**Why is it called variational inference?** We are looking for $q^*(\theta) = \arg\min_{q(z) \in \mathbb{Q}} KL(q(\theta) \parallel p(\theta \mid x)$. The optimal $q*$ is a function in a space of functions $\mathbb{Q}$ - so we need to borrow concepts from calculus of variations to perform optimization over functionals - hence variational inference.

## Jensen's inequality

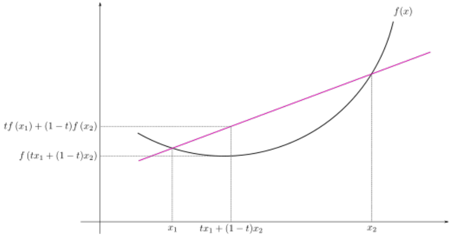
### Jensen's Inequality

If $f_{convex}$ is a convex function and $X$ is a random variable, then the following statement always holds:

$$f_{convex}(\mathbb{E}[X]) \leq \mathbb{E}[f_{convex}(X)] \tag{5}$$

Opposite is true for a concave function $f_{concave}$

$$f_{concave}(\mathbb{E}[X]) \geq \mathbb{E}[f_{concave}(X)] \tag{6}$$



## Derivation of the Evidence Lower Bound (ELBO)

Can't directly minimize the KL-variational divergence $KL(q \parallel p)$), instead, we can minimize a function that is equal to it up to a constant. To derive this function, we can start by writing the log probability of observations:

$$\begin{aligned}\log p(x) &= \log \int_{\Theta} p(x,\theta) d\theta \\ &= \log \int_{\Theta} p(x,\theta) \frac{q(\theta)}{q(\theta)} d\theta \\ &= \log \left( \mathbb{E}_q \left[ \frac{p(x,\theta)}{q(\theta)} \right] \right)\end{aligned}$$

Now we can use concave form of (log function is concave) Jensen's inequality to transpose the $\mathbb{E}_q$ and log terms to derive the ELBO.

$$\begin{aligned}\log p(x) = \log \left( \mathbb{E}_q \left[ \frac{p(x,\theta)}{q(\theta)} \right] \right) &\geq \mathbb{E}_q \log \left( \left[ \frac{p(x,\theta)}{q(\theta)} \right] \right) \\ &\geq \mathbb{E}_q \left[ \log p(x,\theta) \right] - \mathbb{E}_q \left[ \log q(\theta) \right]\end{aligned}$$

## Evidence lower bound (ELBO)

**ELBO**

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

**Recall that this is a lower bound for** $\log p(x)$ **- we haven't touched the KL divergence part yet**.

Maximizing the ELBO is the equivalent of minimizing the variational KL divergence between $q(\theta)$ and $p(\theta \mid x)$.

$$\begin{aligned}
KL(q(\theta)||p(\theta \mid x)) &= \mathbb{E}_q\left(\log\left(\frac{q(\theta)}{p(\theta \mid x)}\right)\right) \\
&= \mathbb{E}_q[\log q(\theta)] - \mathbb{E}_q[\log p(\theta \mid x)] \\
&= \mathbb{E}_q[\log q(\theta)] - \mathbb{E}_q[\log p(\theta, x)] + \log p(x) \\
&= -\left(\mathbb{E}_q[\log p(\theta, x)] - \mathbb{E}_q[\log q(\theta)]\right) + \log p(x) \\
&= -\text{ELBO} + \log p(x)
\end{aligned}$$

## Maximizing ELBO = Minimizing KL divergence

**Maximizing the ELBO is the equivalent of minimizing the KL divergence**

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

If we want to minimize the KL divergence, we can just maximize the ELBO.
Note that $\log p(x)$ **does not depend on** $q$, so we can focus on the ELBO term only.

## Summary of KL divergence and ELBO

- We approximate true posterior $p(\theta \mid x)$ with another distribution $q(\theta)$.
- We can minimize the difference through a metric known as the KL divergence, which is not symmetric.
- We generally minimize the $KL(q(\theta) \parallel p(\theta \mid x)$ in VI to find $q^*(\theta) = \arg\min_{q(z)\in\mathbb{Q}} KL(q(\theta) \parallel p(\theta \mid x)$.
- Can't directly minimize the KL due to $p(\theta \mid x)$ term (still need to do $p(x) = \int_\Theta p(x, \theta) d\theta$).
- We rearrange the KL divergence so that we can maximize a value known as the Evidence Lower Bound (ELBO) is equivalent to minimizing the KL divergence up to a constant that does not depend on $q$.
- Since $q(\theta)$ is an approximate function, **we are generally not guaranteed exact theoretical convergence to the true posterior** $p(\theta \mid x)$, which is not the case in MCMC approaches (assuming infinite sampling).
- VI is preferred in large models due to increased speed.

## Mean-field Variational Inference (MFVI) and good approximating families

Having detailed the nitty-gritty of Variational Inference, we will now examine how we construct the family of variational distributions from which we want to draw $q(\theta)$ from.

The simplest family is where each latent parameter $\theta_j$ has its own independent distribution. This family of functions are known as **Mean-Field Variational Family**. Inference using such a factorization is known as **Mean-Field Variational Inference**. A generic member of the MFVI family is:

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

Each latent variable in our posterior is governed by its own independent distribution. Note that the data $x$ doesn't even appear here. It is the ELBO maximization that dictates how we can get the best member $q_j^*(\theta)$ from our data.

## Variational parameters

We can further parameterize the approximating distributions $q(\theta)$ with variational parameters $\lambda$ so that we can express our approximating distribution as $q(\theta_j; \lambda_j)$.

For example, if we set our family of approximating distributions as a set of independent normals $q_j(\theta_j) = N(\mu_j, \sigma_j^2)$, we can parameterize this distribution with the mean and variance $q_j(\theta_j; \mu_j, \sigma_j^2)$, and we usually denote the set of variational parameters $\lambda_j = (\mu_j, \sigma_j^2)$.

This makes our lives much easier, as we can set nice variational parameterizations and optimize the ELBO with respect to the variational parameters $\lambda$.

## How do we actually do Variational Inference...?

Several algorithms have been developed to do Variational Inference, they differ in how user-friendly they are and the complexity needed to carry them out.

- **Coordinate Ascent Variational Inference** - derive variational updates by hand and perform coordinate ascent (iteratively updating each latent variable $\theta_j$) on the latent until convergence of the ELBO.
- **Black box variational inference**: optimizes $q(\theta; \lambda)$ using a stochastic gradient based method. Takes unbiased noisy gradient of ELBO by drawing Monte Carlo samples from variational distribution.
- **Automatic Differentiation Variational Inference (ADVI)** - automatically transforms your model into one with unconstrained real latent variables - it transforms $p(x, \theta)$ into $p(x, \zeta)$ and performs minimization. Puts all of your variables into same parameter space - enables ADVI to use **single variational family for all models**.

## Coordinate Ascent Variational Inference (CAVI)

**CAVI Intuition**

Consider each latent parameter $\theta_j$ in turn. We can compute an analytic update for each $q_j^*(\theta_j) = p(\theta_j, \mid \theta_{-j}, \mathbf{x})$ - very similar to Gibbs sampling. Mostly relevant for exponentially family members with conjugate (exponentially family) priors.

Specifically, this exploits the fact that conditionals in the exponential family have a closed form expectation. This provides you with **closed form/analytic updates** for all your variational updates. Extremely fast, but extremely tedious to derive (since you need to do so by hand).

See here for a derivation of mean-field CAVI for generic exponential family member, as well as an example for a Bayesian Gaussian Mixture Model. Bayesian Gaussian Mixture Model VI Demo: here

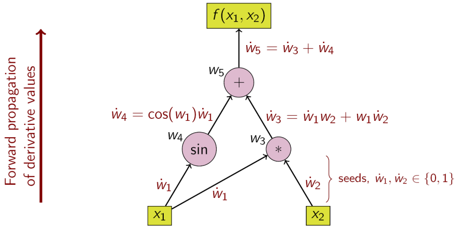
## Automatic Differentiation

Both Black-box and Automatic Variational Inference depend on Automatic Differentiation, which is a method of finding the **exact** (up to numerical precision of your machine) derivative of computer code by taking advantage of the fact that most computational functions are built up from basic elementary operations (additions, subtraction, etc...) and the chain rule.

$$\begin{tikzcd}
\boxed{\texttt{f(x) \{...\};}} \arrow[r, "\text{Automatic differentiation}"] & \boxed{\texttt{df(x) \{...\};}} \\
\boxed{y = f(x)} \arrow[u, "\text{human programmer}"'] \arrow[r, dashed, "\text{symbolic differentiation (human/computer)}"'] & \boxed{y' = f'(x)} \arrow[u, dashed]
\end{tikzcd}$$

## A simple example of AD

Turns your function into a computational graph made of simple elementary operations. Say my function was $f(x_1, x_2) = x_1 x_2 + \sin x$. I can form the following graph and use chain rule to find gradient for each node:

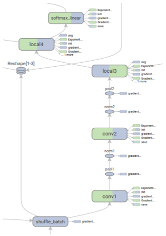


Forward propagation of derivative values
$f(x_1, x_2)$
$\dot{w}_5 = \dot{w}_3 + \dot{w}_4$
$w_5$
+
$\dot{w}_4 = \cos(w_1)\dot{w}_1$
$\dot{w}_3 = \dot{w}_1 w_2 + w_1 \dot{w}_2$
$w_4$
sin
$w_3$
$*$
$\dot{w}_1$
$\dot{w}_1$
$\dot{w}_2$
seeds, $\dot{w}_1, \dot{w}_2 \in \{0, 1\}$
$x_1$
$x_2$

| Operations to compute value | Operations to compute derivative |
|---|---|
| $w_1 = x_1$ | $\dot{w}_1 = 1$ (seed) |
| $w_2 = x_2$ | $\dot{w}_2 = 0$ (seed) |
| $w_3 = w_1 \cdot w_2$ | $\dot{w}_3 = w_2 \cdot \dot{w}_1 + w_1 \cdot \dot{w}_2$ |
| $w_4 = \sin w_1$ | $\dot{w}_4 = \cos w_1 \cdot \dot{w}_1$ |
| $w_5 = w_3 + w_4$ | $\dot{w}_5 = \dot{w}_3 + \dot{w}_4$ |

## Modern deep learning and compute frameworks are built on computational graphs and AD

Reverse-mode AD goes from output to input (see here) for a simple example. More efficient memory usage.
Both scale linearly with the size of your program, and both allow for control flow statements (if, for etc...). Tensorflow, Theano, Torch etc... all exploit building a computational graph and uses automatic differentiation to do backpropagation (for neural networks).



We can use it to take gradients as well! Useful for Hamiltonian Monte-Carlo and ADVI/BBI.

## Comparing Automatic Differentiation vs. Numerical and Symbolic Differentiation

**Numerical differentiation** - computes gradients by finite differences:

$$\frac{\partial f}{\partial x_n}(x) \approx \frac{f(x_1 + \ldots x_n + \epsilon, x_M) - f(x_1, \ldots, x_M)}{\epsilon}$$

Numerical stability issues - especially if $\epsilon$ is small. Computing gradient takes $M + 1$ evaluations of $f$.

**Symbolic differentiation** - computes gradients **analytically** by applying chain rule. Suppose we have $f = \phi_3 \circ \phi_2 \circ \phi_1$

$$f'(x) = \phi_3'(\phi_2(\phi_1(x)))\phi_2'(\phi_1(x))\phi_1'(x)$$

If we compose $k$ functions, we need approximately $O(k^2)$) function evaluations. **Does not allow for conditionals, for loops. etc....**

## Automatic Differentiation Variational Inference (ADVI)

Automatically (using pre-built library of transformations such that the supports match) transforms your latent variables $p(\theta)$ into a real coordinate space $\mathbb{R}^K$, such that your joint density is transformed into $g(\mathbf{X}, \zeta) = p(\mathbf{X}, T^{-1}(\zeta)) \left|\det J_{T^{-1}}(\zeta)\right|$.
After transformation, fit the mean-field Gaussian variational approximation:

$$q(\zeta; \lambda) = N(\zeta; \mu, \sigma^2) = \prod_{k=1}^{K} N(\zeta_k; \mu_k, \sigma_k^2)$$

Optimize the variational parameters $\lambda$ with respect to the ELBO (in transformed space) using Automatic Differentiation and then transform it back using $T^{-1}$. Allows for non-conjugacy in model (which CAVI doesn't allow).

## ADVI visualization

Advantages: Don't have to select approximating variational family; everything is automatically a Gaussian in a transformed space. Also allows for

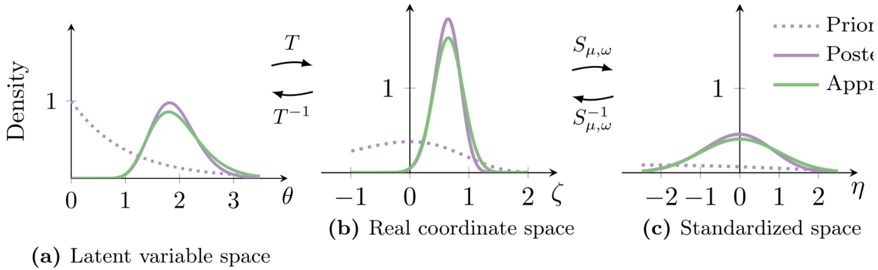


**Figure 3:** Transformations for ADVI. The purple line is the posterior. The green line is the approximation. **(a)** The latent variable space is $\mathbb{R}^+$. **(a→b)** $T$ transforms the latent variable space to $\mathbb{R}$. **(b)** The variational approximation is a Gaussian. **(b→c)** $S_{\mu,\omega}$ absorbs the parameters of the Gaussian. **(c)** We maximize the ELBO in the standardized space, with a fixed standard Gaussian approximation.

## Black Box Variational Inference

Basically uses computational tricks to allow us to stochastically optimize the ELBO by taking an unbiased **noisy** gradient and use Monte Carlo sampling. See paper here

$$\nabla_\lambda L = \mathbb{E}_q \left[\nabla_\lambda \log q(\theta \mid \lambda) \left(\log p(x, \theta) - \log q(\theta_s \mid \lambda)\right)\right]$$

Where we draw Monte Carlo $\theta_s$ from $\theta_s \sim q(\theta \mid \lambda)$. Lots of math to prove that this is unbiased. Also use Automatic Differentiation to evaluate the $\nabla_\lambda$.

**Stochastic Variational Inference** takes a very similar approach but instead uses the natural gradient (Elucidean distance is a bad measurement of gradients sometimes, defines a new measure). More details here.

**Downsides compared to ADVI:** you still have to specify an approximating variational family, whereas you don't have to in ADVI. Is this better? Depends on your knowledge of the actual posterior distributions.

## Differences between Probabilistic Programming Languages

We've only taught Stan for the beginning of this class. Stan has their own automatic differentiation library implemented in C++. Stan offers two major inference modes:

- No-U-Turn-Sampler Hamiltonian Monte Carlo
- Automatic Differentiation Variational Inference

If you want to use/construct other inference algorithms, you have to interface with C++.

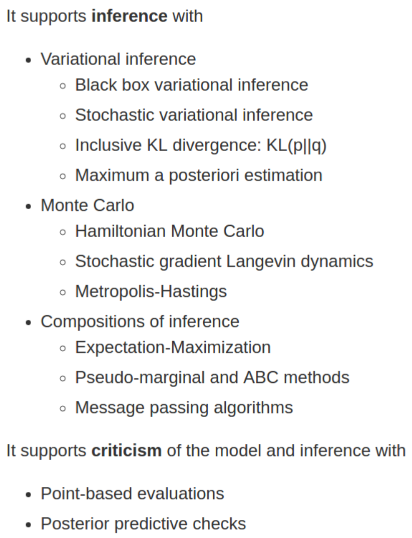
No CUDA (GPU) acceleration.

## Edward (Tensorflow Backend)

Edward is a new library from the Blei group at Columbia that uses Tensorflow as a backend.

- Arguably more powerful probabilistic language than Stan
  - Allows import of models composed in other languages including: Stan, Keras (for Bayesian Neural Networks), and PyMC3.
  - More suited for (Bayesian) deep learning and generative modeling if you're interested in that area.
  - Tensorflow backend allows for CUDA/GPU-acceleration (useful for massively parallel tasks)
  - Tensorflow also optimizes your computational graph automatically, reducing redundancies and increasing run time.
- Lots of implemented inference algorithms!
- **Downsides:** Need to learn some Tensorflow; not compatible with latest Tensorflow stable release (0.12); missing some algorihms: no ADVI, no NUTS-HMC.

# Edward Supported Algorithms



It supports **inference** with

- Variational inference
  - Black box variational inference
  - Stochastic variational inference
  - Inclusive KL divergence: KL(p||q)
  - Maximum a posteriori estimation
- Monte Carlo
  - Hamiltonian Monte Carlo
  - Stochastic gradient Langevin dynamics
  - Metropolis-Hastings
- Compositions of inference
  - Expectation-Maximization
  - Pseudo-marginal and ABC methods
  - Message passing algorithms

It supports **criticism** of the model and inference with

- Point-based evaluations
- Posterior predictive checks

## PyMC3

Similar to Edward, uses Theano (very similar to Tensorflow - but slower compile/faster running due to more aggressive optimizations of your computational graph). Theano is arguably easier if you're starting out in deep learning frameworks as API is somewhat easier to parse (in my opinion).

Implemented algorithms

- NUTS-HMC
- HMC
- Metropolis-Hastings
- Slice
- ADVI

See here for PyMC3 tutorial/documentation. See here for variational inference in PyMC3.