Introduction to Stan and Hamiltonian Monte Carlo

Space-Time Reading Group

Sahar Z Zangeneh

November 7, 2017

Outline of Talk

- Overview of Stan programming language
- Challenges in high-dimensional settings
- Intuition behind Hamiltonian Monte Carlo

References

- Betancourt, M., 2017. A Conceptual Introduction to Hamiltonian Monte Carlo. *arXiv preprint arXiv:1701.02434*.
- Carpenter, B., Gelman, A., Hoffman, M., Lee, D., Goodrich, B., Betancourt, M., Brubaker, M.A., Guo, J., Li, P. and Riddell, A., 2016. Stan: A probabilistic programming language. *Journal of Statistical Software*, *20*, pp.1-37.
- Gelman, A., Lee, D. and Guo, J., 2015. Stan: A probabilistic programming language for Bayesian inference and optimization. *Journal of Educational and Behavioral Statistics*, *40*(5), pp.530-543.
- Neal, R.M., 2011. MCMC using Hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo*, *2*(11).

What is Stan?

- Open-source probabilistic programming language for specifying statistical models
- Named after Stanislaw Ulam, a mathematician who was one of the developers of the Monte Carlo method in the 1940s
- Allows a user to write a Bayesian model in a convenient language whose code looks like statistics notation
  - User writes a Stan code that directly computes the log-posterior density
  - Result is a set of posterior simulations of the parameters in the model
- Uses Hamiltonian Monte Carlo instead of MCMC

How does Stan work?

- Perform Bayesian inference via C++ program
- No-U-turn sampler, an adaptive variant of Hamiltonian Monte Carlo
- Could only perform inference for continous parameters
  - Main limitation of Stan
  - Allows for discrete data and discrete-data models such as logistic regressions, but it cannot perform inference for discrete unknowns
  - Re-express discrete parameter models as mixture models with continuous parameters. Sometimes this sort of re-expression does not exist

Motivation for Stan software

- Motivated by attempt to apply full Bayesian inference to multilevel generalized linear models, structured with
  - grouped and interacted predictors at multiple levels
  - hierarchical covariance priors
  - nonconjugate coefficient priors
  - latent effects as in item-response models
  - varying output link functions and distributions
- Needed a better sampler, rather than more efficient implementation of Gibbs sampling

Algorithmic challenges with HMC

1. Hamiltonian dynamics simulation requires gradient of the log posterior
   - Computing by hand on a model-by-model basis very tedious and prone to error error
   - Reverse-mode algorithmic differentiation (Carpenter et al, 2017)
2. Variables with constrained support
3. Sensitivity to two tuning parameters
   - Discretization interval (i.e., step size) – tune during warm-up based on Metropolis rejection rates
   - Total simulation time (i.e., number of steps) -- difficult to tune without sacrificing the detailed balance of the sampler.
   - No U-Turn (NUTS) sampler (Hoffman and Gelman, 2011)

Sequences of statements and execution order

- Stan allows sequences of statements wherever they may occur, e.g., statements are executed imperatively in the order in which they occur in a program
- Blocks and variable scope
- Sequences of statements surrounded by curly braces ({ and }) form blocks.
- Blocks may start with local variable declarations. scope of local variables is limited to that specific block
- Other variables, e.g., those declared as data or parameters, need to specifically be assigned. May be used in (i) the block in which they are declared or (ii) any block after the block in which they are declared

Blocks and variable scope in Stan

- A Stan program starts with an (optional) data block, which declares the data required to fit the model
- Sequences of statements surrounded by curly braces ({ and }) form blocks
- Blocks may start with local variable declarations. Scope of local variables is limited to that specific block
- Other variables, e.g., those declared as data or parameters, need to specifically be assigned. May be used in (i) the block in which they are declared or (ii) any block after the block in which they are declared

Stan Data Blocks -- ctd

- (Transformed) data block
  - Define new variables that can be computed based on the data
- (Transformed) parameters block
  - Executed after the parameter block.
  - Constraints are validated after all of the statements defining the transformed parameters have been executed.
  - If transformed parameters are used on the left-hand side of a sampling statement, up to user to add appropriate log Jacobian adjustment to the log probability accumulator
- Model block
  - Defines the log probability function on the constrained parameter space

Example: Generating lognormal variate in Stan

- Generate without the built-in lognormal density function
- Transform is $f(u) = \log(u)$, so $f^{-1}(v) = \exp(v)$, so absolute log Jacobian is $|d(\exp(v)/dv| = \exp(v) = u$

```
parameters {
  real<lower=0> u;
  ...
transformed parameters {
  real v;
  v <- log(u);
  increment_log_prob(u);      // log absolute Jacobian adjustment
}
model {
  v ~ normal(0,1);
}
```

Implicit change of variables to unconstrained space

- In Stan, probability distribution intended to have unconstrained support (i.e., no points of zero probability) -- simplifies the task of writing samplers or optimizers
  - Transform variables declared with constrained support to an unconstrained space, e.g. log-transform variables defined on [0,1]
  - Dimensionality of resulting probability function may change as a result of the transform
  - Inverse transform unconstrained parameters over which the model is defined back to their constrained forms before executing the model code
  - Log absolute Jacobian determinant of the inverse transform is added to the overall log probability function to account for change of variables

Data block summary

Variable Categorization	Declaration Block
unmodeled data	`data`, `transformed data`
modeled data	`data`, `transformed data`
missing data	`parameters`, `transformed parameters`
modeled parameters	`parameters`, `transformed parameters`
unmodeled parameters	`data`, `transformed data`
generated quantities	`transformed data`, `transformed parameters`, `generated quantities`
loop indices	any loop statement
local variables	any statement block

Figure 5: *Variables of each categorization must be declared in specific blocks. Data may also be expressed using numeric literals.*

Assignment and Sampling

- Log density accumulator
  - Implicitly defined through function target()
- Sampling Statements
  - Merely shorthand for incrementing log denisty accumulator
- Define variables before sampling statements
- Direct definition of probability functions

Missing Data in Stan

- Missing data models may still be coded in Stan, but the missing values must be declared as parameters

Function and distribution library

- Basic operators – C++
- Special functions – C++ and beyond
- Matrix and linear algebra functions
- Probability functions
  - Growing list of built-in univariate and multivariate densities
  - Everything defined on log scale to avoid underflow
  - All named with suffix _lpdf or _lpmf
  - Up to a proportion calculations
  - Univariate also accept arrays or vectors as arguments

Hamiltonian Monte Carlo – Some Intuition

- Open-source probabilistic programming language for specifying statistical models
- Named after Stanislaw Ulam, a mathematician who was one of the developers of the Monte Carlo method in the 1940s
- Allows a user to write a Bayesian model in a convenient language whose code looks like statistics notation
  - User writes a Stan code that directly computes the log-posterior density
  - Result is a set of posterior simulations of the parameters in the model
- Uses Hamiltonian Monte Carlo instead of MCMC