Approximate Posterior Sampling via Stochastic Optimisation

Connie Trojan
Supervisor: Srshti Putcha

6th September 2019
Large scale machine learning models rely on stochastic optimisation techniques to learn parameters of interest.
Background

- Large scale machine learning models rely on stochastic optimisation techniques to learn parameters of interest.
- It is useful to understand parameter uncertainty using Bayesian inference.
**Background**

- Large scale machine learning models rely on stochastic optimisation techniques to learn parameters of interest.

- It is useful to understand parameter uncertainty using Bayesian inference.

- Usually simulate the Bayesian posterior using Markov Chain Monte Carlo (MCMC) sampling algorithms.
Large scale machine learning models rely on stochastic optimisation techniques to learn parameters of interest.

It is useful to understand parameter uncertainty using Bayesian inference.

Usually simulate the Bayesian posterior using Markov Chain Monte Carlo (MCMC) sampling algorithms.

Stochastic gradient MCMC methods combine stochastic optimisation methods with MCMC to reduce computation time.
In the Bayesian approach, the unknown parameter $\theta$ is treated as a random variable.

The Bayesian posterior distribution $\pi(\theta|x)$ has the form:

$$
\pi(\theta|x) \propto p(\theta)\ell(x|\theta) = p(\theta) \prod_{i=1}^{N} \ell(x_i|\theta),
$$

where:

- $p(\theta)$ is the prior distribution
- $\ell(x_i|\theta)$ is the likelihood associated with observation $i$
- $N$ is the size of the dataset
In particular, gradient-based MCMC algorithms use the log posterior $f(\theta)$ to propose moves:

$$f(\theta) = k + f_0(\theta) + \sum_{i=1}^{N} f_i(\theta) \equiv k + \log p(\theta) + \sum_{i=1}^{N} \log \ell(x_i|\theta)$$
Stochastic Optimisation

Efficient way of learning model parameters, typically used in machine learning.

Stochastic Gradient Ascent (SGA)

1. Take a subsample $S_t$ of size $n$ from the data
2. Estimate the gradient at $\theta_t$ by:
   $$\hat{\nabla} f(\theta_t) = \nabla f_0(\theta_t) + \frac{N}{n} \sum_{x_i \in S_t} \nabla f_i(\theta_t)$$
3. Set
   $$\theta_{t+1} = \theta_t + \epsilon_t \hat{\nabla} f(\theta_t) + \gamma (\theta_t - \theta_{t-1})$$

There are many ways of speeding up convergence, such as adding in a momentum term.
Stochastic Optimisation

Efficient way of learning model parameters, typically used in machine learning.

Stochastic Gradient Ascent (SGA)

Set starting value $\theta_0$, batch size $n \ll N$, and step sizes $\epsilon_t$. Iterate:

1. Take a subsample $S_t$ of size $n$ from the data
2. Estimate the gradient at $\theta_t$ by:
   \[ \nabla \hat{f}(\theta_t) = \nabla f_0(\theta_t) + \frac{N}{n} \sum_{x_i \in S_t} \nabla f_i(\theta_t) \]
3. Set $\theta_{t+1} = \theta_t + \epsilon_t \nabla \hat{f}(\theta_t) + \gamma (\theta_t - \theta_{t-1})$

There are many ways of speeding up convergence, such as adding a momentum term.
Stochastic Optimisation

Efficient way of learning model parameters, typically used in machine learning.

### Stochastic Gradient Ascent (SGA)

Set starting value $\theta_0$, batch size $n \ll N$, and step sizes $\epsilon_t$. Iterate:

1. Take a subsample $S_t$ of size $n$ from the data

There are many ways of speeding up convergence, such as adding in a momentum term.
Stochastic Optimisation

Efficient way of learning model parameters, typically used in machine learning.

Stochastic Gradient Ascent (SGA)

Set starting value $\theta_0$, batch size $n \ll N$, and step sizes $\epsilon_t$. Iterate:

1. Take a subsample $S_t$ of size $n$ from the data
2. Estimate the gradient at $\theta_t$ by:

$$\nabla \hat{f}(\theta_t) = \nabla f_0(\theta_t) + \frac{N}{n} \sum_{x_i \in S_t} \nabla f_i(\theta_t)$$
Stochastic Optimisation

Efficient way of learning model parameters, typically used in machine learning.

Stochastic Gradient Ascent (SGA)

Set starting value $\theta_0$, batch size $n \ll N$, and step sizes $\epsilon_t$. Iterate:

1. Take a subsample $S_t$ of size $n$ from the data
2. Estimate the gradient at $\theta_t$ by:
   \[ \nabla \hat{f}(\theta_t) = \nabla f_0(\theta_t) + \frac{N}{n} \sum_{x_i \in S_t} \nabla f_i(\theta_t) \]
3. Set $\theta_{t+1} = \theta_t + \epsilon_t \nabla \hat{f}(\theta_t)$

There are many ways of speeding up convergence, such as adding in a momentum term.
Stochastic Optimisation

Efficient way of learning model parameters, typically used in machine learning.

Stochastic Gradient Ascent (SGA)

Set starting value $\theta_0$, batch size $n \ll N$, and step sizes $\epsilon_t$. Iterate:

1. Take a subsample $S_t$ of size $n$ from the data
2. Estimate the gradient at $\theta_t$ by:
   \[
   \nabla \hat{f}(\theta_t) = \nabla f_0(\theta_t) + \frac{N}{n} \sum_{x_i \in S_t} \nabla f_i(\theta_t)
   \]
3. Set $\theta_{t+1} = \theta_t + \epsilon_t \nabla \hat{f}(\theta_t) + \gamma (\theta_t - \theta_{t-1})$

There are many ways of speeding up convergence, such as adding in a momentum term.
Robbins-Monro criteria for convergence:
If $\sum_{t=1}^{\infty} \epsilon_t = \infty$ and $\sum_{t=1}^{\infty} \epsilon_t^2 < \infty$, then $\theta_t$ will converge to a local maximum.
Robbins-Monro criteria for convergence:
If $\sum_{t=1}^{\infty} \epsilon_t = \infty$ and $\sum_{t=1}^{\infty} \epsilon_t^2 < \infty$, then $\theta_t$ will converge to a local maximum

Usually set $\epsilon_t = (\alpha t + \beta)^{-\gamma}$ with $\gamma \in (0.5, 1]$
Robbins-Monro criteria for convergence:
If \( \sum_{t=1}^{\infty} \epsilon_t = \infty \) and \( \sum_{t=1}^{\infty} \epsilon_t^2 < \infty \), then \( \theta_t \) will converge to a local maximum.

- Usually set \( \epsilon_t = (\alpha t + \beta)^{-\gamma} \) with \( \gamma \in (0.5, 1] \)

- These algorithms only converge to a point estimate of the posterior mode.
Many problems for which Bayesian inference would be useful involve non-standard distributions and a large number of parameters, making exact inference challenging.

MCMC algorithms aim to generate random samples from the posterior. These samplers construct a Markov chain, often a random walk, which converges to the desired stationary distribution.
The Langevin diffusion describes dynamics which converge to $\pi(\theta)$:

$$d\theta(t) = \frac{1}{2} \nabla f(\theta(t)) + db(t)$$

MALA uses the following discretisation to propose samples:

$$\theta_{t+1} = \theta_t + \frac{\sigma^2}{2} \nabla f(\theta_t) + \sigma \eta_t$$

A Metropolis-Hastings accept/reject step is then used to correct discretisation errors, ensuring convergence to the desired stationary distribution.
MALA algorithm

Set starting value $\theta_0$ and step size $\sigma^2$. Iterate the following:

1. Set $\theta^* = \theta_t + \sigma^2 \nabla f(\theta_t) + \sigma \eta_t$, where $\eta_t \sim N(0, I)$.

2. Accept and set $\theta_{t+1} = \theta^*$ with probability $a(\theta^*, \theta_t) = \min\{1, \pi(\theta^*) q(\theta_t|\theta^*) / \pi(\theta_t) q(\theta^*|\theta_t)\}$.

3. If rejected, set $\theta_{t+1} = \theta_t$.
MALA algorithm

Set starting value $\theta_0$ and step size $\sigma^2$. Iterate the following:

1. Set $\theta^* = \theta_t + \frac{\sigma^2}{2} \nabla f(\theta_t) + \sigma \eta_t$, where $\eta_t \sim N(0, I)$.
MALA algorithm

Set starting value $\theta_0$ and step size $\sigma^2$. Iterate the following:

1. Set $\theta^* = \theta_t + \frac{\sigma^2}{2} \nabla f(\theta_t) + \sigma \eta_t$, where $\eta_t \sim N(0, I)$
2. Accept and set $\theta_{t+1} = \theta^*$ with probability
   
   $a(\theta^*, \theta_t) = \min \left\{ 1, \frac{\pi(\theta^*)q(\theta_t|\theta^*)}{\pi(\theta_t)q(\theta^*|\theta_t)} \right\}$,

   where $q(x|y) = P(\theta^* = x|\theta_t = y)$
MALA algorithm

Set starting value $\theta_0$ and step size $\sigma^2$. Iterate the following:

1. Set $\theta^* = \theta_t + \frac{\sigma^2}{2} \nabla f(\theta_t) + \sigma \eta_t$, where $\eta_t \sim N(0, I)$

2. Accept and set $\theta_{t+1} = \theta^*$ with probability
   
   $$a(\theta^*, \theta_t) = \min \left\{ 1, \frac{\pi(\theta^*) q(\theta_t | \theta^*)}{\pi(\theta_t) q(\theta^* | \theta_t)} \right\},$$
   
   where $q(x|y) = P(\theta^* = x | \theta_t = y)$

3. If rejected, set $\theta_{t+1} = \theta_t$
\[ \sigma = 0.03 \quad a = 0.99 \]

\[ \sigma = 0.13 \quad a = 0.57 \]

\[ \sigma = 0.20 \quad a = 0.13 \]
SGLD aims to reduce the computational cost of MALA by replacing the full gradient calculation in the proposal with the stochastic approximation $\nabla \hat{f}(\theta)$:

$$\theta_{t+1} = \theta_t + \frac{\epsilon_t}{2} \nabla \hat{f}(\theta_t) + \sqrt{\epsilon_t} \eta_t$$

Here, the $\epsilon_t$ are decreasing to 0 as in SGA.

Since the Metropolis-Hastings acceptance rate tends to 1 as the step size decreases, the costly accept/reject step is ignored.
SGLD algorithm

Set starting value $\theta_0$, batch size $n$, and step sizes $\epsilon_t$. Iterate:

1. Take a subsample $S_t$ of size $n$ from the data
2. Estimate the gradient at $\theta_t$ by $\nabla \hat{f}(\theta_t) = \nabla f_0(\theta_t) + \frac{1}{n} \sum_{x_i \in S_t} \nabla f_i(\theta_t)$
3. Set $\theta_{t+1} = \theta_t + \epsilon_t \nabla \hat{f}(\theta_t) + \sqrt{\epsilon_t} \eta_t$, where $\eta_t \sim N(0, I)$

In practice, a fixed step size often works and is far easier to tune.
SGLD algorithm

Set starting value $\theta_0$, batch size $n$, and step sizes $\epsilon_t$. Iterate:

1. Take a subsample $S_t$ of size $n$ from the data
SGLD algorithm

Set starting value \( \theta_0 \), batch size \( n \), and step sizes \( \epsilon_t \). Iterate:

1. Take a subsample \( S_t \) of size \( n \) from the data
2. Estimate the gradient at \( \theta_t \) by

\[
\nabla \hat{f}(\theta_t) = \nabla f_0(\theta_t) + \frac{N}{n} \sum_{x_i \in S_t} \nabla f_i(\theta_t)
\]
SGLD algorithm

Set starting value $\theta_0$, batch size $n$, and step sizes $\epsilon_t$. Iterate:

1. Take a subsample $S_t$ of size $n$ from the data
2. Estimate the gradient at $\theta_t$ by
   \[
   \nabla \hat{f}(\theta_t) = \nabla f_0(\theta_t) + \frac{N}{n} \sum_{x_i \in S_t} \nabla f_i(\theta_t)
   \]
3. Set $\theta_{t+1} = \theta_t + \frac{\epsilon_t}{2} \nabla \hat{f}(\theta_t) + \sqrt{\epsilon_t} \eta_t$, where $\eta_t \sim N(0, I)$
SGLD algorithm

Set starting value $\theta_0$, batch size $n$, and step sizes $\epsilon_t$. Iterate:

1. Take a subsample $S_t$ of size $n$ from the data
2. Estimate the gradient at $\theta_t$ by
   \[
   \nabla \hat{f}(\theta_t) = \nabla f_0(\theta_t) + \frac{N}{n} \sum_{x_i \in S_t} \nabla f_i(\theta_t)
   \]
3. Set $\theta_{t+1} = \theta_t + \frac{\epsilon_t}{2} \nabla \hat{f}(\theta_t) + \sqrt{\epsilon_t} \eta_t$, where $\eta_t \sim N(0, I)$

In practice, a fixed step size often works and is far easier to tune.
\( \epsilon = 0.0001 \)

\( \epsilon = 0.0005 \)

\( \epsilon = 0.0013 \)

\( \epsilon = 0.0050 \)
SGLD with Control Variates (SGLD-CV)

- The gradient estimate in SGLD is simple
SGLD with Control Variates (SGLD-CV)

- The gradient estimate in SGLD is simple
- The variance of the gradient estimator can be reduced using control variates
SGLD with Control Variates (SGLD-CV)

- The gradient estimate in SGLD is simple

- The variance of the gradient estimator can be reduced using control variates

- This is achieved by finding $\hat{\theta}$, a value of $\theta$ close to the mode, called the centering value. The gradient estimates in the sampler will condition on $\hat{\theta}$
Since
\[ \nabla f(\theta_t) = \nabla f(\hat{\theta}) + \left[ \nabla f(\theta_t) - \nabla f(\hat{\theta}) \right], \]
Since

\[ \nabla f(\theta_t) = \nabla f(\hat{\theta}) + \left[ \nabla f(\theta_t) - \nabla f(\hat{\theta}) \right], \]

we can take a subsample \( S_t \) of the data and estimate \( \nabla f(\theta_t) \) by

\[ \nabla \tilde{f}(\theta_t) = \nabla f(\hat{\theta}) + \left[ \nabla \hat{f}(\theta_t) - \nabla \hat{f}(\hat{\theta}) \right]. \]

Here, \( \nabla \hat{f} \) is the simple estimate used in SGLD.
Since
\[ \nabla f(\theta_t) = \nabla f(\hat{\theta}) + \left[ \nabla f(\theta_t) - \nabla f(\hat{\theta}) \right], \]
we can take a subsample \( S_t \) of the data and estimate \( \nabla f(\theta_t) \) by
\[ \nabla \tilde{f}(\theta_t) = \nabla f(\hat{\theta}) + \left[ \nabla \hat{f}(\theta_t) - \nabla \hat{f}(\hat{\theta}) \right]. \]
Here, \( \nabla \hat{f} \) is the simple estimate used in SGLD.
In full our new estimate \( \nabla \tilde{f} \) is:
\[
\nabla f(\hat{\theta}) + \left[ \nabla f_0(\theta_t) - \nabla f_0(\hat{\theta}) \right] + \frac{N}{n} \sum_{x_i \in S_t} \left[ \nabla f_i(\theta_t) - \nabla f_i(\hat{\theta}) \right]
\]
Use stochastic optimisation to find $\hat{\theta}$, a value close to a mode
SGLD-CV algorithm

- Use stochastic optimisation to find $\hat{\theta}$, a value close to a mode
- Calculate the full gradient $\nabla f(\hat{\theta})$
SGLD-CV algorithm

- Use stochastic optimisation to find \( \hat{\theta} \), a value close to a mode
- Calculate the full gradient \( \nabla f(\hat{\theta}) \)
- Set starting value \( \hat{\theta} \), batch size \( n \), and step sizes \( \epsilon_t \). Iterate:
SGLD-CV algorithm

- Use stochastic optimisation to find $\hat{\theta}$, a value close to a mode
- Calculate the full gradient $\nabla f(\hat{\theta})$
- Set starting value $\hat{\theta}$, batch size $n$, and step sizes $\epsilon_t$. Iterate:

1. Take a subsample $S_t$ of size $n$ from the data
SGLD-CV algorithm

- Use stochastic optimisation to find $\hat{\theta}$, a value close to a mode
- Calculate the full gradient $\nabla f(\hat{\theta})$
- Set starting value $\hat{\theta}$, batch size $n$, and step sizes $\epsilon_t$. Iterate:

1. Take a subsample $S_t$ of size $n$ from the data
2. Estimate the gradient at $\theta_t$ by $\nabla \tilde{f}(\theta_t)$
SGLD-CV algorithm

- Use stochastic optimisation to find $\hat{\theta}$, a value close to a mode
- Calculate the full gradient $\nabla f(\hat{\theta})$
- Set starting value $\hat{\theta}$, batch size $n$, and step sizes $\epsilon_t$. Iterate:

1. Take a subsample $S_t$ of size $n$ from the data
2. Estimate the gradient at $\theta_t$ by $\nabla \tilde{f}(\theta_t)$
3. Set $\theta_{t+1} = \theta_t + \frac{\epsilon_t}{2} \nabla \tilde{f}(\theta_t) + \sqrt{\epsilon_t} \eta_t$, where $\eta_t \sim N(0, I)$
Comparison

MALA

SGLD (n=10)

SGLD−CV

Kernel Stein Discrepency

MALA

SGLD (n=100)

SGLD (n=50)

SGLD (n=10)

SGLD−CV

Passes through data

MALA

SGLD (n=100)

SGLD (n=50)

SGLD (n=10)

SGLD−CV

Kernel Stein Discrepency

MALA

SGLD (n=100)

SGLD (n=50)

SGLD (n=10)

SGLD−CV

Passes through data

Approximate Posterior Sampling via Stochastic Optimisation

Connie Trojan
Supervisor: Srshiti Putcha
Comparison of the samplers for a more complicated multimodal target distribution

Data distribution: $\mathbf{x} \sim \frac{1}{2} N(\mu_1, \sigma) + \frac{1}{2} N(\mu_2, \sigma)$

Each sampler was given 500 passes through the data and 20 passes of burn-in or optimisation
The sampling algorithms discussed above were used to fit a binary logistic regression model to the **covertype** dataset. The aim was to predict the class of tree cover from 54 forest terrain factors.
The Covertype Dataset

Elevation (m)
Aspect (degrees azimuth)
Slope (degrees)
Horizontal distance to nearest surface water (m)
Vertical distance to nearest surface water (m)
Horizontal distance to nearest roadway (m)
Hillshade 9am (0-255)
Hillshade Noon (0-255)
Hillshade 3pm (0-255)
Horizontal distance to wildfire ignition points (m)
Wilderness area designation \( \times 4 \) (binary)
Soil type \( \times 40 \) (binary)

Class (1-7)
The problem was converted to a binary classification problem aiming to separate class 2 from the others.
The problem was converted to a binary classification problem aiming to separate class 2 from the others.

Instead of class, used the response variable $y$ where:

$$y_i = \begin{cases} 
1, & \text{if class}(x_i) = 2 \\
0, & \text{else}
\end{cases}$$

$$P(y_i = 1|x_i) = \sigma(\beta_0 + \beta^T x_i) \equiv \frac{1}{1 + \exp[-(\beta_0 + \beta^T x_i)]}$$
The problem was converted to a binary classification problem aiming to separate class 2 from the others.

Instead of class, used the response variable $y$ where:

$$
y_i = \begin{cases} 
1, & \text{if } \text{class}(x_i) = 2 \\
0, & \text{else}
\end{cases}
$$

$$P(y_i = 1|x_i) = \sigma(\beta_0 + \beta^T x_i) \equiv \frac{1}{1 + \exp[-(\beta_0 + \beta^T x_i)]}$$

The training dataset had 570,000 observations and an additional 10,000 were used to test the model.
Performance measure: log loss

\[ \frac{1}{|T|} \sum_{y_i \in T} [y_i \log(\hat{p}_i) + (1 - y_i) \log(1 - \hat{p}_i)] \]
Conclusions and Further Work

- MALA is very impractical with large datasets
Conclusions and Further Work

- MALA is very impractical with large datasets
- SGLD-CV consistently outperforms the other algorithms
Conclusions and Further Work

- MALA is very impractical with large datasets
- SGLD-CV consistently outperforms the other algorithms
- Tuning SGLD is very difficult - have to test a wide range of stepsizes and use a metric like KSD to assess performance

Approximate Posterior Sampling via Stochastic Optimisation
Conclusions and Further Work

- MALA is very impractical with large datasets
- SGLD-CV consistently outperforms the other algorithms
- Tuning SGLD is very difficult - have to test a wide range of stepsizes and use a metric like KSD to assess performance
- SGLD-CV also has a high tuning burden, since both the optimisation and the sampling stages have to be tuned
Conclusions and Further Work

- MALA is very impractical with large datasets
- SGLD-CV consistently outperforms the other algorithms
- Tuning SGLD is very difficult - have to test a wide range of stepsizes and use a metric like KSD to assess performance
- SGLD-CV also has a high tuning burden, since both the optimisation and the sampling stages have to be tuned
- Gradient calculations had to be done by hand, making it difficult to implement more complicated models
  - It is more practical to use numerical differentiation for this (e.g. sgmcmc for R)

Connie Trojan
Supervisor: Srshiti Putcha
Approximate Posterior Sampling via Stochastic Optimisation
References

https://www.jstor.org/stable/3318418

http://jmlr.org/papers/v18/15-205.html

Max Welling and Yee W. Teh. *Bayesian Learning via Stochastic Gradient Langevin Dynamics.*

Jack Baker, Paul Fearnhead, Emily B. Fox, and Christopher Nemeth. *Control Variates for Stochastic Gradient MCMC.*
Any Questions?