Stein Variational Descent Methods for Non-parametric Sampling in Inverse Problems

Felipe Suárez
Motivation

Bayesian Inverse Problems

\[
\begin{cases}
\nabla \cdot (\theta \nabla u) = f & u \text{ partially known} \\
\frac{\partial u}{\partial n} = 0 & f \text{ known} \\
\end{cases}
\rightarrow \theta?
\]

\[
\begin{cases}
(\theta \omega^2 + \Delta)u = f & u \text{ partially known} \\
\frac{\partial u}{\partial n} = 0 & f \text{ known} \\
\end{cases}
\rightarrow \theta?
\]

Setup

Naïve Approach:
- Guess \( \theta \)
- Solve the forward model \( F(\theta) = u \)
- Iterate until \( F(\theta) \) matches \( u \) measured

Bayesian Approach

\( \theta \sim \text{Prior} \)
\( D = F(\theta) + \xi \rightarrow \text{Gaussian Noise} \)
\( \theta | D \sim \text{Posterior} \)

\( \pi(\theta) = \frac{\mathbb{P}(D|\theta)\mathbb{P}(\theta)}{\mathbb{P}(D)} \)

\[ \min_{\theta} L(F(\theta), D) + R(\theta) \]

- Infinite-dimensional Gradient Descent
- Adjoint-State method
Goal

\[ \mathbb{E}_{\theta \sim \pi}[h(\theta)] \]

- Low variance
- Low sample complexity
- Low cost per sample

Difficulties

- \( \theta \in \mathbb{R}^n, n \gg 1 \)
- \( \pi(\theta) \propto \mathbb{P}(D|\theta)\mathbb{P}(\theta) \rightarrow \) Un-normalized
- \( \mathbb{P}(D|\theta) \) often costly

Purpose

Design a tool for sampling \( \pi \):
- Particle transport from a
- Cheaper distribution \( q \) toward \( \pi \),
- Using cross-entropy minimization by
- A small RKHS perturbations and
- Simpler than MCMC! (Or not?)
Stein Method

\[ g = \underset{h \in \langle \kappa \rangle}{\text{argmin}} D_{KL}((I + h)_*q||\pi) \]

\[ Z \rightarrow \pi \]
Three Key Lemmas

\[
\frac{\delta}{\delta h} D_{KL}( (I + h)q || p ) \bigg|_{h=0} = -tr \mathbb{E}_q \nabla_p h
\]

Steepest Descent

\[
\arg\max_{h \in \langle k \rangle} tr \mathbb{E}_q \nabla_p h = \mathbb{E}_q \nabla_p k(x, \cdot)
\]

Algorithm

Input: Sample \( \{x_i\}_i \sim q \), target \( \pi \), kernel \( k \)
Output: Mapped Sample \( \{Tx_i\}_i \sim \pi \)

For \( i = 1, \ldots, n \)

1. Compute a Descent direction \( G(x_i) \)
2. Apply transport map \( T = I - \eta G \)

endFor

First Order

- Steepest Descent
- GD with Momentum
- AdaGrad
- RMSprop

Second Order(ish)

- Gauss Newton
Implementation in Julia

Type
- Struct <: Type
- Function(::Struct)

StochasticModel
- newModel
- newModel2

kernel
- isoRBF
- genRBF
- Kxy(...)
- Kxy!(...)

SteinVariate
- SVGD
- SVN

Optimizer
- FirstOrderOptimizer
  - ConstantRate
  - Momentum
  - AdaGrad
  - RMSprop
- SecondOrderOptimizer
  - Constant
  - Stabilized

Optimize!(...)
Some Results

References

