Sequential Stein Variational Gradient Descent for Time Series Model Estimation

Gibson, Reich, and Ray in some order

December 3, 2017

Introduction

State-Space models have become popular tools in the analysis of time-series. They allow for arbitrary transition and observation dynamics. The researcher can assign a latenet data generating process, while simultaneously allowing for observational error on that process. The classic algorithm for fitting non-Gaussian SSMs is given by the particle filter. Although many variations exist, we generally refer to the sampling importance resampling (SIR) filter when discussing particle filtering. Although a powerful inference tool, particle filtering suffers from several well known drawbacks. The first is the problem of filter degeneracy. This occurs when the obervations are far from the state predicted by the latent dynamics. The second is the excessive run-times on longer time series with complex dynamics. We propose an alternative approach that we hope will do better than particle filtering in practice. In this approach, Stein Variational Gradient Descent (SVGD) is used to sequentially estimate the distribution of state variables in each time step, conditional on observed data up through that time.

Overview of SVGD

Stein Variational Gradient Descent can be used to estimate a continuous distribution by a set of samples. By iteratively transporting samples from an initial distribution in the direction of the likelihood, we are able to generate a sample from the target distribution. The usefullness of this approximation is apparent in Bayesian statistics, where the usually intractable normalizing constant disappears in the gradient. The particles are subject to the following gradient ascent procedure.

$$x_t^{(i)} \leftarrow x_{t-1}^{(i)} + \frac{1}{n} \sum_{j=1}^{n} [k(x_j, x_{t-1}^{(i)}) * \nabla log \ p(x_j) + \nabla k(x_j, x_{t-1}^{(i)})]$$

for an arbitrary postivie definite kernel function $k(.,.)$.

Sequential Stein Variational Gradient Descent

Suppose we are given a time series $Y_1, Y_2, ..., Y_t$ for $Y \in \mathbb{R}$. We model the sequence as a state-space model parameterized by an observation density $p(y_t|x_t)$ and a transition density $p(x_t|x_{t-1})$ Figure 1.

We are interested in the filtering distribution $p(x_1, ..., x_n|y_1, ..., y_n)$ which by Bayes formula is

$$p(x_1, ..., x_n|y_1, ..., y_n) = \frac{p(y_1, ..., y_n|x_1, ..., x_n)p(x_1, ..., x_n)}{Z}$$

.

Because computing the normalizing constant Z is intractable for many choices of $p(y_t|x_t)$ and $p(x_t|x_{t-1})$, we must resort to monte carlo algorithms. The classic approach that incorporates the sequential nature of the data is given by the particle filtering algorithm. Particle filtering approximates the filtering density using sequential importance sampling. We instead focus on the following recursion.
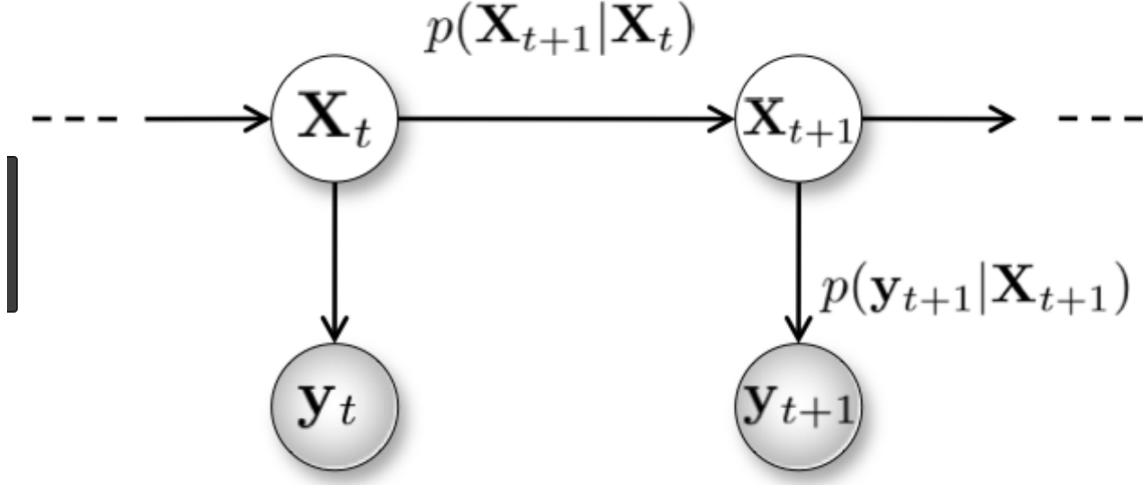
Figure 1: State-Space Model Setup

$$p(x_t|y_{1:t}) = \int p(x_{0:t}|y_{1:t})d_{x_0:t-1}$$

$$= \frac{p(y_t|x_t)}{\int p(y_t|x_t)p(x_t|y_{1:t-1})dx_t}p(x_t|y_{1:t-1})$$

$$\propto p(y_t|x_t)p(x_t|y_{1:t-1})$$

$$\propto p(y_t|x_t)p(x_t|y_{1:t-1})$$

$$\propto p(y_t|x_t)\int_{x_{t-1}} p(x_t, x_{t-1}|y_{1:t-1})d_{x_{t-1}}$$

$$\propto p(y_t|x_t)\int_{x_{t-1}} p(x_t|x_{t-1})p(x_{t-1}|y_{1:t-1})d_{x_{t-1}}$$

which we can approximate using svgd as

$$\approx p(y_t|x_t)\frac{1}{n}\sum_{i=1}^{n} p(x_t|x_{t-1}^{(i)})$$

We can now estimate $p(x_{t+1}|y_{1:t+1})$ using the same algebra as above. (proof in apendix A)

Model Structure

States:

- $X_1 \sim g_1(x_1; \xi)$

- $X_t | X_{t-1} \sim g(x_t | x_{t-1}; \xi)$ for all $t = 2, \ldots, T$

Observations:

- $Y_t | X_t \sim h(y_t | x_t; \zeta)$

Here, $g_1(\cdot)$ and $g(\cdot)$ are appropriately defined probability density functions depending on parameters ξ and $h(\cdot)$ is an appropriately defined probability density function or probability mass function depending on parameters ζ.

Define $\theta = (\xi, \zeta)$ to be the full set of model parameters.

Filtering

There are two types of filtering:

1. sample of particles $x_{1:T}^{(k)} \sim f(x_{1:T} | y_{1:T})$
2. sample of particles $x_t^{(k)} \sim f(x_t | y_{1:t})$ for each $t = 1, \ldots, T$

Let's look at the second one. Assume we have a sample $x_{t-1}^{(k)} \sim f(x_{t-1} | y_{1:t-1})$

$$
\begin{aligned}
p(x_t | y_{1:t}) &= \frac{f(x_t, y_t | y_{1:t-1})}{f(y_t | y_{1:t-1})} \\
&\propto f(x_t, y_t | y_{1:t-1}) \\
&= f(y_t | x_t) f(x_t | y_{1:t-1}) \\
&= f(y_t | x_t) \int f(x_t, x_{t-1} | y_{1:t-1}) dx_{t-1} \\
&= f(y_t | x_t) \int f(x_t | x_{t-1}) f(x_{t-1} | y_{1:t-1}) dx_{t-1} \\
&\approx f(y_t | x_t) \sum_{x_{t-1}^{(k)}} f(x_t | x_{t-1}^{(k)})
\end{aligned}
$$

So $\log\{p(x_t | y_{1:t})\}$ is approximately proportional to $\log\{f(y_t | x_t)\} + \log\{\sum_{x_{t-1}^{(k)}} f(x_t | x_{t-1}^{(k)})\}$
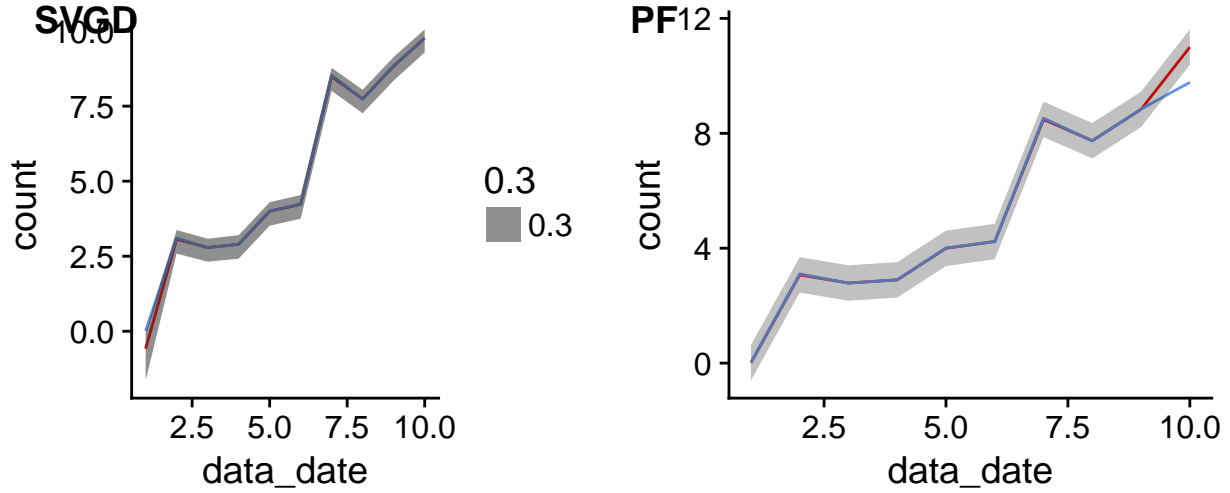
Locally Level Gaussian Noise Model

In order to demonstrate that the approximation is reasonable we evaluate the predictive accuracy under an analytically tractable model, the locally level Gaussian model. This model takes the form

$$X_t \sim N(X_{t-1}, \sigma_1^2)$$

$$Y_t \sim N(X_t, \sigma_2^2)$$

[1] "python /Users/gcgibson/Stein-Variational-Gradient-Descent/python/locally_level_gaussian.py 0.0:

SVGD 10.0

count

7.5

5.0

2.5

0.0

0.3

0.3

2.5   5.0   7.5   10.0

data_date

PF 12

count

8

4

0

2.5   5.0   7.5   10.0

data_date

Poisson Observation Model With Seasonal State-Space Dynamics

In order to evaluate the performance on more involved dynamics we consider the following state-space model.

$$\begin{pmatrix} X_{t,1} \\ X_{t,2} \end{pmatrix} = \begin{pmatrix} cos(2\pi/s) & sin(2\pi/s) \\ -sin(2\pi/s) & cos(2\pi/s) \end{pmatrix} \begin{pmatrix} X_{t-1,1} \\ X_{t-1,2} \end{pmatrix}$$

$$Y_t \sim Pois(e^{X_{t,1}})$$

```
## * Parsing model in: /Users/gcgibson/Stein-Variational-Gradient-Descent/bug_files/seasonal_pois.bug

## Warning in biips_model(seasonal_model_file, data = seasonal_data,
## sample_data = FALSE): Unused variables in data: G, mean_sigma_init,
## cov_sigma_init, mean_x_init

## * Compiling model graph
##   Declaring variables
##   Resolving undeclared variables
##   Allocating nodes
##   Graph size: 121

## * Assigning node samplers
## * Running SMC forward sampler with 100000 particles
##   |--------------------------------------------------| 100%
##   |**************************************************| 11 iterations in 20.78 s

## [1] "python /Users/gcgibson/Stein-Variational-Gradient-Descent/python/seasonal.py  17, 18, 10, 16, 1
```

4
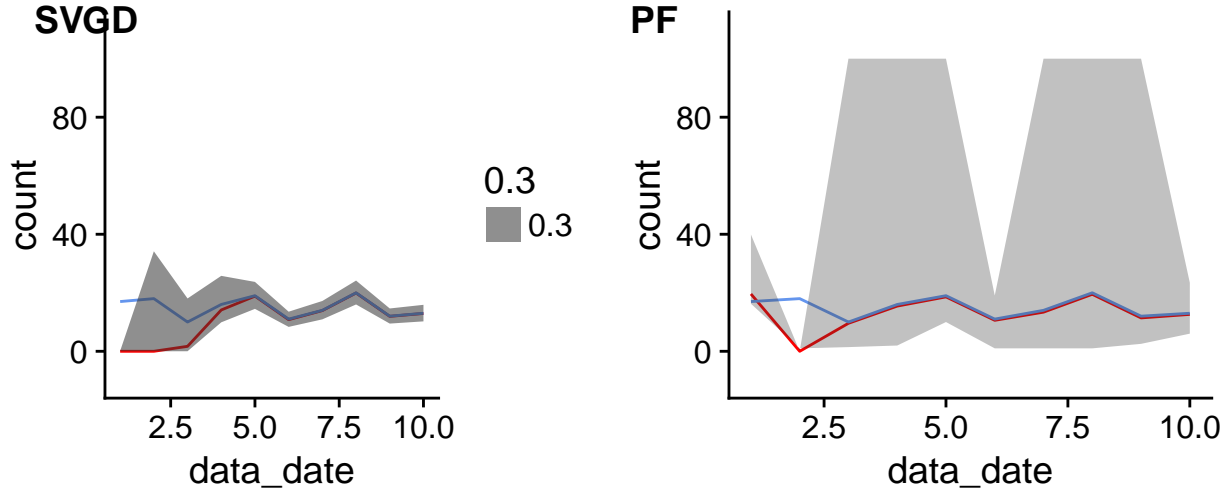
Divergent Particle Filter

We next investigate the ability of SSVGD to perform in the presence of poor initialization. This is a well known issue with current particle filter implemntations: starting far from a plausabile value of x_0 forces all particles to receive weight 0 under the likelihood, leading to a degenerate filtering distribution. However, under SSVGD, we can simply increase the number of iterations, allowing for arbitrarily poor starting points. Standard particle filtering algorithms use effective sample size as a measure of degeneracy. This is commonly defined as

$$S_{eff}^{pf} = \frac{1}{\sum_i (w_t^i)^2}$$

. The common rule of thumb is to not allow this quantity to drop below 50. The natural translation of this metric into particle filtering is compute the same metric based on the samples obtained by SSVGD.

```
## * Parsing model in: /Users/gcgibson/Stein-Variational-Gradient-Descent/bug_files/locally_level_1.bug
## * Compiling model graph
##   Declaring variables
##   Resolving undeclared variables
##   Allocating nodes
##   Graph size: 18

## * Assigning node samplers
## * Running SMC forward sampler with 100 particles
##   |--------------------------------------------------| 100%
##   |**************************************************| 5 iterations in 0.00 s

## * Diagnosis of variable: x[1:5]
##   Filtering: POOR
##     The minimum effective sample size is too low: 1.82146
##     Estimates may be poor for some variables.
##     You should increase the number of particles
## .  Smoothing: POOR
##     The minimum effective sample size is too low: 1
##     Estimates may be poor for some variables.
##     You should increase the number of particles
## .

## [1] 0.0001431633 0.0032389841 0.0036750256 0.0633277320 0.3430663214

## [1] "python /Users/gcgibson/Stein-Variational-Gradient-Descent/python/locally_level.py  1, 4, 0, 3, :
```
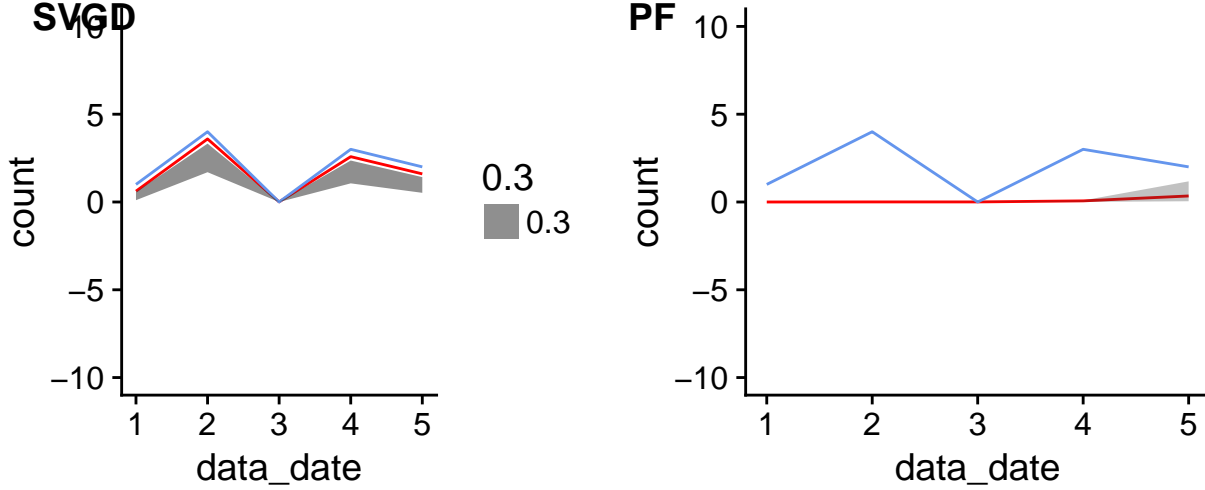
```
## [1] 1.2455414 0.2450772 9.0004783 0.3402830 0.5344517
```

Diagnostics

Standard particle filtering algorithms use effective sample size as a measure of degeneracy. This is commonly defined as

$$S_{eff} = \frac{1}{\sum_i (w_i)^2}$$

. The common rule of thumb is to not allow this quantity to fall below 50. Indeed, software implementations such as Biips throws an error if the number of effective particles falls below 50. We compute the effective sample size in an analogous way to the particle filter, where w_i is defined as in the SIR particle filter.

```
library(knitr)

ess_df <- data.frame(seq(1:5),effect_size,out_smc$x$f$ess)
colnames(ess_df) <- c("t","SSVGD ESS", "PF ESS")
kable(ess_df,caption="Effective Sample size")
```

Table 1: Effective Sample size

t	SSVGD ESS	PF ESS
1	8.660931	33.465772
2	9.735688	1.821460
3	11.876790	99.996175
4	10.229694	4.717344
5	12.842775	10.914045

Results

In order to assess the accuracy of SSVGD we consider multiple different evaluation metrics. We evaluate the forecasts on both mean-square-error and log-score. Log-scores were computed as follows,

$$p(y_{1:n}) = \int_{x_{1:n}} p(y_{1:n}|x_{1:n})p(x_{1:n})d_{x_{1:n}}$$

This can be approximated as

$$\frac{1}{k}\sum_{i=1}^{k}p(y_{1:n}|x_{1:n}^{(i)})$$

That is, we take a full trajectory $x_{1:n}$ and compute the log-probability of $y_{1:n}$. Note that this is equivalent to taking the average weight at each step.

```
log_p <- c()
for (i in 1:nrow(weights)){
  log_p <- c(log_p,mean(weights[i,]))
}



ll_df <- data.frame(sum(log_p),out_smc$log_marg_like)
colnames(ll_df) <- c("SSVGD","PF")
kable(ll_df,caption="Log Score")
```

Table 2: Log Score

SSVGD	PF
-7.373609	-56.57699

Discussion

SSVGD seems to perform well in terms of effective sample size and log score when compared to traditional sequential monte carlo particle filtering.

Bibliography