École des Ponts ParisTech
Department of Statistics, University of Oxford

2017
Master's Internship Report

Émile Mathieu
Élève ingénieur, Third year

Bayesian Nonparametric Inference within Probabilistic Programming Languages

Internship carried out at Department of Statistics, University of Oxford
From the 22nd of May, to the 15th of September 2017.

Company tutor: Teh, Yee Whye
Training supervisor: Obozinski, Guillaume

Acknowledgments

First of all, I would like to express my indebtedness appreciation to my departmental supervisor Prof. Yee Whye Teh. His belief in me and his advices played a decisive role in making the execution of my work and thus this report.

I also express my deepest thanks to Benjamin Bloem-Reddy, who as a postdoc, oversaw me during this internship and with whom I frequently work.

Moreover, my gratitude goes to Guillaume Obozinski, my school training supervisor, whose guidance has continually shaped my career path since I have been at Ecole des Ponts ParisTech.

Abstract

Bayesian Nonparametric (BNP) model have been gaining attraction because of their flexibility. Indeed, these models, automatically adapt with the number of data which avoid having to define a priori the number of parameters of the model, such as the number of components for a mixture model for instance. Probabilistic Programming Languages (PPLS) allow practitioners to express probabilistic programs in a universal way, and bring generic inference algorithms. These systems avoid designing specific inference schemes, which is error-prone and time consuming.

Therefore, a natural question is how can BNP models be represented in such systems ? BNP models live in infinite dimensional space which is problematic for machines having finite memory and computational resources. Secondly, how should the inference schemes be designed so that by using Probabilistic Programming Language (PPL)s (and thus gaining in flexibility) performance is not too much affected ?

Keywords : Probabilistic Programming, Bayesian Non-parametric, Bayesian Inference, Sampling methods

Résumé

Contents

List of Figures

Glossary

- **BNP**: *Bayesian Non-Parametric*, explained in Section 5.1.
- **PPL**: *Probabilistic Programming Language*, explained in Section 4.1.

Introduction

cf Research Proposal ?

Presentation of the Department of Statistics

2.1 Creation

The Department of Statistics [1] is part of the University of Oxford, along with the other departments and the 38 constituent colleges. The University of Oxford was founded in the 11th century, which makes it the oldest university in the English-speaking world and the world's second-oldest university in continuous operation.

The Department of Statistics was officially created in 1988, even though first moves in the development of Oxford statistics can be dated to the 19th century.

Indeed, In the 1870s, Florence Nightingale – the pioneer of modern nursing – discussed the possibility of endowing a Professorship of Statistics in Oxford, but the proposal eventually foundered. However, Oxford did appoint a statistician to a chair in 1891, although not to a chair in statistics.

The next significant moves in the development of Oxford statistics were by economists, who were increasingly keen to build economic theory on a foundation of sound data analysis. This led to the creation in 1935 of an Institute of Statistic, swhich was then renamed as the Institute of Economics and Statistics in 1962.

The sequence of events which led directly to the establishment of the present Department of Statistics began with the appointment in 1945 of David Finney as the universitys first Lecturer in the Design and Analysis of Scientific Experiment (LIDASE).

Then in the 1980s, after the Department of Biomathematics' head increasingly felt that Oxford was losing out in the face of developments in statistics, a working party appointed by the general board of the university to assess a careful analysis of the organisation of statistics in Oxford. They found fragmentation to be the dominant feature of Oxford statistics and concluded that fragmentation has serious disadvantages ..... The working partys report recommended the creation of a university statistics department, which were to include the former Department of Biomathematics, together with a new Professorship in

[1] https://www.stats.ox.ac.uk

Statistical Science and the two existing lecturerships in statistics within the Mathematical Institute.

These major recommendations were all accepted by the university and the new Department of Statistics was created in 1988.

2.2 Activities

The Department of Statistics at Oxford is a world leader in research including computational statistics and statistical methodology, applied probability, bioinformatics and mathematical genetics. The main research groups in the Department are Computational statistics and machine learning, Probability, Statistical genetics and bioinformatics, Protein Informatics and Statistical Genetics.

I am part of the Computational Statistics and Machine Learning Group (OxCSML) [2], which have research interests spanning Statistical Machine Learning, Monte Carlo Methods and Computational Statistics, Statistical Methodology and Applied Statistics.

The department offers an undergraduate degree (BA or MMath) in Mathematics and Statistics, jointly with the Mathematical Institute. At postgraduate level there is an MSc course in Applied Statistics (MSc in Statistical Science from 2017), as well as a lively and stimulating environment for postgraduate research (DPhil or MSc by Research). The department also has a consulting activity called *Oxford University Statistical Consulting*.

[2] `http://csml.stats.ox.ac.uk/people/mathieu/`

Mission

3.1 Themes of research

Prof. Yee Whye Teh [1] has worked for a long time on inference sampling schemes for BNP mixture models [20, 19, 31, 32], but also on stick-breaking constructions [48, 18]. He also has recently been interested in PPL and consequently in inference schemes within PPL for BNP models.

This theme requires knowledge in several fields – Probabilities, Computational Statistics, Programming Languages – which makes it deeply interesting. He proposed me working with him on this topic as part of a 3-years DPhil program, and to start earlier as an intern.

3.2 Context

In addition to inviting me to work with him, Prof. Yee Whye Teh also opened two postdoctoral positions for working on the same project, which have been filled by Tom Rainforth and Benjamin Bloem-Reddy.

Tom Rainforth [2] is finishing is third year of DPhil in the Dept. of Engineering Science in Oxford, supervised by Prof. Frank Wood. His interests include probabilistic programming, Bayesian optimization, probabilistic numerics, sequential Monte Carlo and particle Markov Chain Monte Carlo methods. He will join the group in October, but he has already attended several reading group meetings.

On the other hand, Benjamin Bloem-Reddy [3] arrived in May in Oxford and has already started working on the project. He was supervised by Peter Orbanz at Columbia University, and his research were focused on probabilistic and statistical analysis of networks and other discrete data.

[1] https://www.stats.ox.ac.uk/~teh
[2] http://www.robots.ox.ac.uk/~twgr
[3] http://www.stats.ox.ac.uk/~bloemred/

3.3 Reading group

Four reading groups are organised with a bi-weekly period: Kernel methods, Deep Learning, Bayesian Nonparametrics and Probabilistic Inference. I have been leading the Probabilistic Inference reading group [4] since July. Since, Ben and I have presented four papers [44, 49, 45, 12] with an emphasise on probabilist programming.

3.4 Organisation

In this section I develop my current organisation and workflow as a researcher. At first, I had much trouble to organise my workflow, I wrote my papers' review and new ideas on flying sheets, papers were saved in my computer's folders, citations for report was time-consuming, my code was locally saved, etc...

Thus, I worked on a better workflow and after trials and errors, I eventually arrived on what I describe below. I aim to modify this process with time, so as to continually enhance my productivity and be able to focus on the interesting part of the job.

3.4.1 Managing papers

My biggest trouble was keeping organised the dozens of new articles I read each week. I was saving them in a tree-like structure of folders, but with the number of articles saved growing, it became more and more difficult to find specific article. Moreover, this structure inherently prohibits cross-categories articles which is annoying for a project situated at the intersections of several fields. Furthermore, I had no fast way to cite an article, neither in plain format (for markdown [5] files for instance) nor in *BibTeX* format.

Then, I have heard of papers managing library such as Papers3 [6] or Mendeley [7]. I have eventually opted for Papers3 but Mendeley is also a popular choice in the academic community. These applications features many tools easing the life of a researcher, the main one being from my point of view:

- Synchronisation: between multiple computers or devices.
- Multi-labels: these are used in the search tool.
- Local search: search in titles, authors, labels and even papers' content.
- Online search: can import articles in a fast manner by being connected with online search engines such as *arXiv*.
- Collections: create a reading list, or group of papers which can be cited at once
- Citations: get *BibTeX* reference or *BibTeX* cite command in clipboard

[4] https://github.com/BigBayes/oxsml/wiki/Probabilistic-Inference-meetings
[5] https://en.wikipedia.org/wiki/Markdown
[6] https://www.readcube.com/papers/mac/
[7] https://www.mendeley.com

3.4.2 Managing research

Another of my organizational issue was keeping track of ideas. I happened to find that research is a result of a long chain of ideas which were continually iterated upon. I am now maintaining a single *master document* for keeping tracks of this chain of ideas.

It has a bulleted list of all ideas, problems, and topics that Id like to think more carefully about. This list is succinct but subsequent sections go in depth on particular entries. This list is sorted according to what Id like to work on next, but I continually revise my priorities according to whether I think the direction aligns with my broader research vision, and if I think the direction is necessarily impactful for the community at large.

3.4.3 Managing projects

Then, when an idea has matured enough and I have seriously started working on it, I create a Github [8] repository for the project. Each project has its separated repository. It contains a `/readme.md` file maintaining a list of todos, with also questions (and sometimes answers!) both for myself and collaborators. This makes it transparent how to keep moving forward and whats blocking the work.

There is also a `/doc/` folder for all the write-ups, usually in LaTeXformat. The `etc/` folder is used for everything not relevant to other directories such as pictures of whiteboards during conversations about the project. Finally, the `/src/` folder is where all code is written. Runnable scripts are written directly in `/src/` , and classes and utilities are written in `/src/codebase/`.

[8] `https://github.com`

Probabilistic programming

4.1 What is it ?

At a high level, PPLs are Programming Language (PL) techniques to abstract inference algorithms from statistics such that they apply automatically and correctly to the broadest possible set of model-based reasoning applications.

A bit more precisely, Probabilistic programming systems [23, 24, 36, 55] represent generative models as programs written in a specialized language that provides syntax for the definition and conditioning of random variables.

Indeed, "probabilistic programs are usual functional or imperative programs with two added constructs: (1) the ability to draw values at random from distributions, and (2) the ability to condition values of variables in a program via observations." [25]

Probabilistic programs define probability distributions over sequences of values, implicitly by means of program execution.

Schema of inference / CS / Stats ?

4.2 Why is it useful ?

For data science practitioners, statistical inference is typically just one step in a more elaborate analysis workflow. The first stage of this work involves data acquisition, preprocessing and cleaning. This is often followed by several iterations of exploratory model design and testing of inference algorithms. Once a sufficiently robust statistical model and a corresponding inference algorithm have been identified, analysis results must be post-processed, visualized, and in some cases integrated into a wider production system.

The main goal of PPLs is to increase productivity. The code for models written with these systems is generally concise, modular, and easy to modify or extend. Thus, one of the savings is to be found in the amount of code that needs to written in order to prototype and develop models.

Secondly, PPLs remove the burden of having to develop inference code for each new model which is error-prone and time consuming. This is done by providing a modeling language

abstraction layer in which developers can denote their models. Once denoted, generic inference is provided for free.

4.3 Existing languages

The first generation of PPLs had limitation in the range of models that could be represented and in which inference could be performed. BUGS [33] and STAN [10] can only work with graphical models. Similarly Factorie [37] and Infer.NET [39] only handle factor graphs. These so-called *First-Order* PPLS, can only represent finite dimensional model and have bounded loops. Schema of first order and high order PPLs ?

On the other hand, *High Order* PPLS which arrived a bit before the 2010s, are Turing complete, allow complex control flow, including stochastic recursion, thus can denote infinite dimensional objects. They are easy to program in, natural to express certain models, but it is hard to perform inference in these PPLs. Anglican [55], Venture [36], Church [23] and WebPPL [24] are *High Order* PPLS.

Recently, a new PPL named Edward [49] has been developed. It is different from the classical PPLs since it focuses on variational inference (VI) and Hamiltonian methods.

4.4 Design

ADD SIMPLE SAMPLE PPL CODE ? Probabilistic programs denote probabilistic generative models as programs that include `sample` and `observe` statements. Both `sample` and `observe` are functions that specify random variables in this generative model using probability distribution objects as an argument, while `observe`, in addition, specifies the conditioning of this random variable upon a particular observed value in a second argument. These observed values induce a conditional probability distribution over the execution traces whose approximations and expected values we aim to characterize by *performing inference*.

A good way to understand this is to imagine the following interpreter of probabilistic programs. Starting from a fixed initial state, the interpreter runs the deterministic parts of a program according to the standard semantics, executes the `sample` statement by generating a random sample, and treats the `observe` statement by skip. More importantly, the interpreter keeps a log that records information about all the `sample` and `observe` forms encountered during execution. The information recorded for `sample` is a triple (F, x, α) of (i) a primitive probability distribution F, such as the standard normal, for which we have the probability density f ; (ii) a value x sampled from the distribution F ; and (iii) an address α that uniquely and systematically identifies the random choice made. The information recorded for `observe` is a pair (G, y) where G is a primitive probability distribution with density g and y is an observed value.

An execution trace $\mathbf{x}$ is defined to be a sequence of triples (F, x, α) and $\mathbf{x}$ is said feasible if the trace is precisely the triple part of the log of some execution. The observed values are denoted by $\mathbf{y} := (y_j)_{j=1}^{N}$.

In most probabilistic programming systems any variable may be declared as being the output of a random procedure. Such variables can take different values in independent

interpretations of the program. This leads to a "many-worlds" computational trace tree in which, at interpretation time, there is a branch at every random procedure application.

A (almost-surely terminating) probabilistic program defines a probability distribution over finite feasible traces $\mathbf{x}$ with probability density $\pi(\mathbf{x}) := \gamma(\mathbf{x})/Z$ where

$$\gamma(\mathbf{x}) := p(\mathbf{x}, \mathbf{y}) = \prod_{i=1}^{|\mathbf{x}|} f_i(x_i \mid x_{1:i-1}) \prod_{j=1}^{|\mathbf{y}|} g_i(y_j \mid x_{1:\tau(j)})$$

where Z is the normalizing constant $Z := \int \gamma(\mathbf{x}) d(\mathbf{x})$ and τ is a mapping from the index j of the `observe` statement to the index of the last `sample` statement encountered before this `observe` statement during the execution of the program. Without any `observe` statement, the probability distribution over traces $\mathbf{x}$ is simply the prior

$$p(\mathbf{x}) := \prod_{i=1}^{|\mathbf{x}|} f_i(x_i \mid x_{1:i-1})$$

4.5 Inference

Add pseudo-code for each algo ?

Inference in probabilistic programming characterizes the conditional distribution of execution traces $\mathbf{x}$ given observed data $\mathbf{y}$ assumed to have been generated by executing the probabilistic program.

Concretely, using the `sample` statements, the PPL's model first defines a so called prior distribution on these execution traces, and then it adjusts this prior distribution based on observations in data using the `observe` statement. Samples from this conditioned distribution (also called posterior distribution) can be obtained by running the model under one of the PPLs inference algorithms.

The inference algorithms may make random choices that do not correspond to any statements in the program, and decide which parts of the program code are executed and how often. Some inference algorithms re-run the program multiple times partially, from a certain point on, while reusing random choices made in the previous runs as much as possible. Upon encountering a sample or observe record, the inference algorithm computes the updated program state and the value to be passed to the continuation. How the state is updated, the number of times the continuation is called, and the value passed to the continuation of sample depend on the inference algorithm executing the program. Implementing an inference algorithm in PPLs amounts to defining checkpoint handlers for `sample` and `observe`.

Typically inference can be performed for any probabilistic program using one or more generic inference techniques provided by the system back end, such as Metropolis-Hastings [53, 36], Hamiltonian Monte Carlo [10], expectation propagation [39], and extensions of Sequential Monte Carlo [50, 42, 56] methods.

4.5.1 Use-case

ADD PPL CODE FOR THIS SSM ? Even if as highlighted before, inference in PPLs should be able to deal with arbitrary series of targets, for simplicity we will focus on a non-Markovian State-Space Model (SSM).

SSMs are probabilistic models over a set of latent variables $X_t \in \mathcal{X}_t, \forall t = 1 : T$ and observed variables $Y_t \in \mathcal{Y}_t, \forall t = 1 : T$. We can further consider a model to be parameterized by $\theta \in \Theta$. The SSM is then characterized by an initial density $\mu_\theta(x_1)$, a series of transition densities $f_{t,\theta}(x_t|x_{1:t-1})$, and a series of emission densities $g_{t,\theta}(y_t|x_{1:t})$.

$$
\begin{aligned}
&X_1 \sim \mu_\theta(\cdot) \\
&X_t|(X_{1:t-1} = x_{1:t-1}) \sim f_{t,\theta}(\cdot|x_{1:t-1}) \\
&Y_t|(X_{1:t} = x_{1:t}) \sim g_{t,\theta}(\cdot|x_{1:t})
\end{aligned}
$$

The joint density of the SSM is then as follows

$$
\gamma_\theta(\mathbf{x}) := p_\theta(x_{1:T}, y_{1:T}) = \mu_\theta(x_1) \prod_{t=2}^{T} f_{t,\theta}(x_t|x_{1:t-1}) \prod_{t=1}^{T} g_{t,\theta}(y_t|x_{1:t})
$$

We are free to choose any density for $\mu_\theta(x_1)$ and each $f_{t,\theta}(x_t|x_{1:t-1})$ and $g_{t,\theta}(y_t|x_{1:t})$. One is usually interested characterizing the posterior

$$
p_\theta(x_{1:T}|y_{1:T}) \propto p_\theta(x_{1:T}, y_{1:T})
$$

Or expectations of some function ϕ under this posterior

$$
I(\phi) = \int \phi(x_{1:T}) p_\theta(x_{1:T}|y_{1:T}) dx_{1:T}
$$

4.5.2 Enumeration

The easiest way one could think of to perform inference in a probabilistic program is via *rejection sampling*. An execution trace can be thought of a *path* in a *tree* implicitly defined by a discrete model. This tree could then be explored using depth-first search, breadth-first search, or a probability-based priority queue. Then only the paths matching the observations $\mathbf{x}$ are retained and the others are rejected. These retained execution traces form the targeted posterior distribution.

4.5.3 MCMC

Yet, for many models with large state spaces, enumeration is infeasible. This is particularly clear for models with continuous random variables, where the state space is infinite. In the case of a large number of execution paths, one should avoid exploring all paths individually but only a subset of paths.

A popular way to estimate a difficult distribution is to sample from it by constructing a random walk that will visit each state in proportion to its probability. This class of algorithms are called MCMC. In our case, we are interested in random walk in the space of execution traces of a computation.

For discrete models, an easy way to a build random walk in the space of executions is to execute the probabilistic program by sampling variables at `sample` AND `observe` statements. The proposed execution trace $\mathbf{x}^\star$ shall then be rejected if at least one of the true observation y_k does not match the associated generated observation $y \sim G$. If accepted, that means that this trace is one of the trace that can lead to the observations $\mathbf{y}$, i.e. it belongs to the support of the targeted posterior distribution $p(\mathbf{x}|\mathbf{y})$. This method is called *rejection sampling*. Yet, for highly dimensional space, this method will accept quite rarely a proposed trace. What is more, for continuous models, it will almost-surely never be accepted.

A more complex Markov Chain needs to be built to have a reasonable acceptance rate. Given a current trace $\mathbf{x}$ and score $p(\mathbf{x})$, we proceed by reconsidering one random choice x_k. Each `sample` statement is equipped with a proposal kernel $\mathcal{K}_k(x^\star|x, \psi)$, which is used to generate proposals to x_k. A random walk can the be built by (i) randomly choosing $k \in 1, \ldots, |\mathbf{x}|$, (ii) sample a new value $x_k^\star$ with $\mathcal{K}_k(x_k^\star|x_k, \psi)$ and (iii) re-run the program starting from x_k which generate a new trace $\mathbf{x}^\star$. This new execution trace is accepted with probability

$$1 \wedge \frac{\gamma(\mathbf{x}^\star)\mathcal{K}_k(x_k|x_k^\star, \psi)}{\gamma(\mathbf{x})\mathcal{K}_k(x_k^\star|x_k, \psi)}$$

This is better than our previous Metropolis-Hastings (MH) algorithm, but when the chosen k is close to 1, it is almost as sampling from the prior since we only reuse the random choices made before the point of regeneration. So as to avoid having a small acceptance rate, it is generally better to make smaller steps by reusing as many choices as possible. If we knew which sampled value was which, then we could look into the previous trace as the execution runs and reuse its values. That is, imagine that each call to sample was passed a (unique) name: `sample(name, dist)`. Then the `sample` function could try to look up and reuse values. Notice that, in addition to reusing existing sampled choices, we add the name and mark whether this choice has been resampled. We must account for this in the MH acceptance calculation. We now hope to reuse most of the choices from the old trace in making a proposal. This algorithm is called Lightweight MCMC [53], and has been improved in [45].

More generally, for an excellent introduction to MCMC and particle inference in PPLs, see [24].

4.5.4 Importance Sampling

Importance Sampling (IS) Importance sampling is an example of a Monte Carlo sampling scheme that provides approximately independent and identically distributed samples from a distribution of interest or target distribution, such as a posterior distribution, by generating a candidate sample from a proposal or importance distribution $q(\mathbf{x}|y_1, \ldots, y_T)$. The fact that the weights $w^k = \frac{p(\mathbf{x}, y_1, \ldots, y_T)}{q(\mathbf{x}|y_1, \ldots, y_T)} \propto \frac{p(\mathbf{x}|y_1, \ldots, y_T)}{q(\mathbf{x}|y_1, \ldots, y_T)}$, with $k \in 1, \ldots, K$ can be

computed is exploited, and samples from the target are obtained by sampling from the following weighted empirical distribution

$$\hat{p}(\mathbf{x}|y_1,\dots,y_T) = \sum_{k=1}^{K} \bar{w}^k \delta_{\tilde{\mathbf{x}}^k}(\mathbf{x})$$

where $\bar{w}^k = \frac{w^k}{\sum_{k=1}^{K} w^k}$ is the normalized weight and δ_z is a Dirac measure centered on z. The expectation $I(\phi)$ can also be approximated using

$$\hat{I}(\phi) = \sum_{k=1}^{K} \bar{w}^k \phi(\tilde{\mathbf{x}})$$

One problem with this method is that it is not easy to choose the proposal distribution q. A good proposal should share most of the support of the target distribution and have the same number of modes, i.e. it should be close to the target. A second problem is that it is a batch estimation method. To tackle this latter issue, in the next section, an extension to a sequential scenario is described.

Sequential Importance Sampling (SIS) SIS exploits the structure of a model by breaking down the overall inference problem into a series of target distributions which get incrementally closer to the distribution of interest. These targets are then approximated by propagating a population of samples known as particles. If each intermediary target is kept similar to its predecessor, approximating one target given the last forms a significantly simpler problem than the overall inference.

More formally, SIS performs approximate inference on a sequence of target distributions $(\pi_t(x_{1:t}))_{t=1}^{T}$ of increasing spaces $(\mathcal{X}_1 \times \cdots \times \mathcal{X}_t)_{t=1}^{T}$. In the context of SSMs, the target distributions are often taken to be $(p_\theta(x_{1:t}|y_{1:t}))_{t=1}^{T}$. At each time step t, we have a set of K particles $\left(\tilde{x}_{1:t}^k\right)_{t=1}^{T}$, corresponding to samples of the latents, and respective particle weights $\left(w_t^k\right)_{t=1}^{T}$. Similarly to IS, using these weighted particles, one can approximate each posterior $p_\theta(x_{1:t}|y_{1:t})$. In particular, the posterior for the complete model $p_\theta(x_{1:T}|y_{1:T})$ and the expectation $I(\phi)$ can be approximated using the following estimators

$$\hat{p}(x_{1:T}|y_{1:T}) = \sum_{k=1}^{K} \bar{w}_T^k \delta_{\tilde{x}_{1:T}^k}(x_{1:T})$$

$$\hat{I}(\phi) = \sum_{k=1}^{K} \bar{w}_T^k \phi(\tilde{x}_{1:T})$$

where $\bar{w}_T^k = \frac{w_T^k}{\sum_{k=1}^{K} w_T^k}$ is the normalized weight.

Let us assume that the importance distribution q at time t depends on all data points until time t and not on the future data points, the joint posterior can be written in the following factorised form

$$q(x_{1:T}|y_{1:T}) = q(x_1)\prod_{t=2}^{T} q(x_t|x_{1:t-1}, y_{1:t})$$

The corresponding importance weight is

$$\begin{aligned} w_t =& \frac{p(x_{1:T}|y_{1:T})}{q(x_{1:T}|y_{1:T})} \\ w_t =& \frac{\mu(x_1)\prod_{t=2}^{T} p(x_t|x_{1:t-1}, y_{1:t})}{q(x_1)\prod_{t=2}^{T} q(x_t|x_{1:t-1}, y_{1:t})} \end{aligned} \tag{4.1}$$

The target distribution is the posterior distribution up to time t, which changes sequentially as we observe more data.

The posterior distribution can be estimated recursively due to

$$p(x_{1:t+1}|y_{1:t+1}) = p(x_{1:t}|y_{1:t}) \times \frac{p(y_{t+1}|x_{1:t+1})p(x_{t+1}|x_{1:t})}{p(y_{t+1}|y_{1:t})} \tag{4.2}$$

Substituting the numerator of Equation 4.2 in 4.1 we obtain a recursive equation for the importance weight at time $t+1$

$$w_{t+1} = w_t \times \frac{p(y_{t+1}|x_{1:t+1})p(x_{t+1}|x_{1:t})}{p(y_{t+1}|y_{1:t})}$$

Even if we are not dealing with a SSM, SIS can be used as a general-purpose algorithm. The data are assumed to be observed sequentially so the observations index is the time index.

PPL breakpoints *breakpoints* are needed. Continuation-Passing Style (CPS) in *Anglican* [55] and *WebPPL* [24] coroutine copying in Turing [21]

Sequential Monte Carlo (SMC) Illustration of particles wrt time t ?

The main problem with SIS is that the weights become more skewed as the number of data points increases [15], after a few steps only one particle will have a significant weight. To remedy this, a resampling step can be introduced which allows to eliminate particles with small weights and replicate particles with high weights. This selection step can be introduced only occasionally or at every step of the algorithm. We resample from the empirical posterior at step t

$$\hat{p}(x_{1:T}|y_{1:T}) = \sum_{k=1}^{K} \bar{w}_t^k \delta_{\tilde{x}_{1:T}^k}(x_{1:T})$$

If the selection step is to be performed occasionally, a possible criterion is when the Effective Sample Size (ESS) is below a given threshold, which is a function of the number of particles, a popular choice is $0.5T$. The ESS for the unnormalised weights is given by

$$ESS_t = \frac{\left(\sum_{k=1}^{K} w_i^k\right)^2}{\sum_{k=1}^{K} {w_i^k}^2}$$

The resampling step is achieved by, at each time step $t = 2, \ldots, T$, selecting the ancestor index a_{t-1}^k for the kth particle from a discrete distribution $\mathcal{F}(\cdot|\bar{w}_{t-1}^1, \ldots, \bar{w}_{t-1}^K)$ over parent indices $1, \ldots, K$ with probabilities equal to the normalized weights at the previous time step $(\bar{w}_{t-1}^1, \ldots, \bar{w}_{t-1}^K)$. [14] provides a comparison of numerous different schemes for sampling from $\mathcal{F}(\cdot|\bar{w}_{t-1}^1, \ldots, \bar{w}_{t-1}^K)$ that reduce the variance of the SMC estimates compared with naïve multinomial resampling.

4.5.5 PMCMC

In a Bayesian setting, it is usual to consider the parameter θ in $p_\theta(x_{1:T}|y_{1:T})$ as a random variable by specifying a prior $p(\theta)$. In this subsection, we are therefore interested on algorithms targeting $p(\theta, x_{1:T}|y_{1:T})$. Let's think about MCMC algorithms targeting the distribution $p(\theta, x_{1:T}|y_{1:T})$ which rely on sampling exactly from $p_\theta(x_{1:T}|y_{1:T})$, called idealized algorithms. Such algorithms are purely conceptual but a natural idea consists of approximating these idealized algorithms by using the output of an SMC algorithm targeting $p_\theta(x_{1:T}|y_{1:T})$ using $K \geq 1$ particles as a proposal distribution for an MH update. Intuitively this could allow us to approximate with arbitrary precision such idealized algorithms while only requiring the design of low dimensional proposals for the SMC algorithm.

A direct implementation of this idea is impossible as the marginal density of a particle that is generated by an SMC algorithm is not available analytically but would be required for the calculation of the MH acceptance ratio. Yet the SMC algorithm yields an unbiased estimate of the marginal likelihood

$$\hat{p}(y_{1:T}) = \prod_{t=1}^{T} \frac{1}{K} \sum_{k=1}^{K} w_t^k$$

which can be used for the calculation of the MH acceptance ratio. These PMCMC updates have been introduced in [2]. The key feature of PMCMC algorithms is that they are in fact exact approximations to idealized MCMC algorithms targeting either $p(\theta, x_{1:T}|y_{1:T})$ in the sense that for any fixed number $N \geq 1$ of particles their transition kernels leave the target density of interest invariant.

Particle Marginal Metropolis-Hastings (PMMH) PMMH makes use of the standard decomposition $p(\theta, x_{1:T}|y_{1:T}) = p(\theta|y_{1:T})p_\theta(x_{1:T}|y_{1:T})$. It is natural to suggest the following form of proposal density for an MH update:

$$q\left(\theta^\star, x_{1:T}^\star|\theta, x_{1:T}\right) = q(\theta^\star|\theta)p_{\theta^\star}(x_{1:T}^\star|y_{1:T})$$

for which the proposed $x_{1:T}^\star$ is given by a SMC algorithm targeting $p_{\theta^\star}(x_{1:T}|y_{1:T})$. Thus, the only degree of freedom of the algorithm (which will affect its performance) is $q(\theta^\star|\theta)$. The resulting MH acceptance ratio is given by

$$1 \wedge \frac{p_{\theta^\star}(y_{1:T})p(\theta^\star)q(\theta|\theta^\star)}{p_\theta(y_{1:T})p(\theta)q(\theta^\star|\theta)}$$

PMMH uses $\hat{p}(y_{1:T})$ to compute the acceptance ratio. It has been proven [2] that the PMMH update leaves $p(\theta, x_{1:T}|y_{1:T})$ invariant and that under weak assumptions the PMMH sampler is ergodic.

Particle Gibbs (PG) An alternative to the previous algorithm to sample from $p(\theta, x_{1:T}|y_{1:T})$ consists of using the Gibbs sampler which samples iteratively from $p(\theta|x_{1:T}, y_{1:T})$ and $p_\theta(x_{1:T}|y_{1:T})$. It is often possible to sample from $p(\theta|x_{1:T}, y_{1:T})$ and thus the potentially tedious design of a proposal density for θ that is necessary in the PMMH update can be bypassed.

It has been shown [2] that the naïve particle approximation to the Gibbs sampler where sampling from $p_\theta(x_{1:T}|y_{1:T})$ is replaced by sampling from an SMC approximation $\hat{p}_\theta(x_{1:T}|y_{1:T})$ does not admit $p(\theta, x_{1:T}|y_{1:T})$ as invariant density.

A valid particle approximation to the Gibbs sampler requires the use of a special type of PMCMC update called the *conditional* SMC update. This update is similar to a standard SMC algorithm but is such that a prespecified path $x^\star_{1:T}$ is ensured to survive all the resampling steps, whereas the remaining $N-1$ particles are generated as usual.

Particle Gibbs with Ancestor Sampling (PGAS) A drawback of PG is that it can be particularly adversely affected by path degeneracy in the Conditional Sequential Monte Carlo (CSMC) step. Conditioning on an existing trajectory means that whenever resampling of the trajectories results in a common ancestor, this ancestor must correspond to this trajectory. Consequently, the mixing of the Markov chain for the early steps in the state sequence can become very slow when the particle set typically coalesces to a single ancestor during the CSMC sweep.

[30] introduces PGAS, which alleviates the problem with path degeneracy by modifying the original PG kernel with a so-called Ancestor Sampling (AS) step. The idea is to sample a new value for the index variable a_t^N in an ancestor sampling step. While this is a small modification of the algorithm, the improvement in mixing can be quite considerable. The task is to artificially assign a history to the partial path $(x^\star_{t:T})$ of the reference path. This is done by connecting $(x^\star_{t:T})$ to one of the particles $(x^k_{1:t-1})$. The ancestor index of the reference path $(x^\star_{t:T})$ at time t $a_t^N \in \{1, \ldots, K\}$ encodes the ancestry of this particle.

To assign a history to this partial path, first the following weights are computed

$$\tilde{w}^k_{t-1|T} := w^k_{t-1} \frac{p((x^k_{1:t-1}, x^\star_{t:T}), y_{1:T})}{p(x^k_{1:t-1}, y_{1:t-1})}$$

for $k = 1, \ldots, K$. Then, a_t^K is sampled via $\mathbb{P}\left(a_t^K = k\right) \propto \tilde{w}^k_{t-1|T}$.

Interacting Particle Markov Chain Monte Carlo (IPMCMC) IPMCMC [43] is another way of tackling the path degeneracy issue. In IPMCMC, a pool of CSMC and unconditional SMC algorithms are ran as parallel processes (referred as nodes. After each

run of this pool, successive Gibbs updates are applied to the indexes of the CSMC nodes, such that the indices of the CSMC nodes changes. Hence, the nodes from which retained particles are sampled can change from one MCMC iteration to the next. This lets us trade off exploration (SMC) and exploitation (CSMC) to achieve improved mixing of the Markov chains.

4.5.6 Hamiltonian

Can be quite brief for Hamiltonian

Hamiltonian Monte Carlo (HMC)

No-U-Turn Sampler (NUTS) In [27], the authors address the issue of choosing the two hyperparameters of HMC: a step size ϵ and a desired number of steps L, since HMC's performance is highly sensitive on those. [41]

Stochastic Gradient Langevin Dynamics (SGLD) mini-batch / online setting, scale to bug dataset [1] [52]

Stochastic Gradient Hamiltonian Monte Carlo (SGHMC) Same setting as SGLD Naive version is wrong (posterior is not the invariant distribution), see [34] friction term [11]

4.5.7 Variational Inference

Introduction to VI ? or in appendix ?

MCMC methods can be slow to converge and their convergence can be difficult to diagnose. To my knowledge, *Edward* [49] is the only PPL handling variational inference.

4.6 Contributions

During this internship I have taken the time to actually implement several inference algorithms, and by so, I contributed to two existing PPLs. Some only for the sake of learning more about sampling schemes and PPLs, but others as specifically part of the project.

First, I implemented [2] both the SGLD and SGHMC inference algorithms in Turing.jl [21], a PPL based on Julia [4] and developed at the University of Cambridge. Then, I implemented [3] the Dual Averaging extension [27] of HMC for Edward [49], a PPL built on top of Tensorflow [1] by Blei's group [4] at Columbia University.

[1] See for instance, SGLD applied to a Bayesian logistic regression at `https://github.com/yebai/Turing.jl/blob/master/example-models/sgld-paper/lr_sgld.jl`

[2] See `https://github.com/yebai/Turing.jl/tree/master/src/samplers`

[3] See `https://github.com/blei-lab/edward/pull/728`

[4] `http://www.cs.columbia.edu/~blei/`

More recently, I have worked on PMCMC methods for Turing. PMMH [5] is implemented but not merged yet, and I am currently working on PGAS and IPMCMC. Consequently I became a *Collaborator* of the Turing's repository.

I have also written a stick-breaking representation of the Dirichlet Process which inherits the Distribution.jl [6] type so as to be easily used in Turing.

To develop ?

[5] https://github.com/yebai/Turing.jl/pull/339
[6] https://github.com/JuliaStats/Distributions.jl

Bayesian nonparametric

5.1 Definition

Use general introductions of famous papers

5.2 Usefulness

5.3 Canonical models

???

DP, PYP, etc

Mixture models

Others ?

5.4 MCMC Inference

Constructing MCMC schemes for models with one or more Bayesian nonparametric components is an active research area since dealing with the infinite dimensional component P forbids the direct use of standard simulation-based methods.These methods usually require a finite-dimensional representation. The general idea for designing inference schemes is to find finite dimensional representations to be able to store the model in a computer with finite capacity.

There are two main sampling approaches to facilitate simulation in the case of Bayesian nonparametric models: random truncation and marginalisation. These two schemes are known in the literature as conditional and marginal samplers.

5.4.1 Marginal Samplers

Marginal samplers bypass the need to represent the infinite-dimensional component by marginalising it out. These schemes have lower storage requirements than conditional

samplers because they only store the induced partition, but could potentially have worse mixing properties.

5.4.2 Conditional Samplers

Conditional samplers replace the infinite-dimensional prior by a finite-dimensional representation chosen according to a truncation level. Since these samplers do not integrate out the infinite-dimensional component, their output provides a more comprehensive representation of the random probability measure. thinning vs stick-breaking

5.4.3 Hybrid Samplers

YW paper on PK ?

5.4.4 SMC

Review of SMC ? cf Maria Lomeli thesis

5.5 Variational inference

Use future work writing

BNP sampling in PPL

Stochastic Memoization with DPmem: $\alpha = 0$, deterministic memoization, $\alpha = \inf$ no memoization
https://probmods.org/chapters/12-non-parametric-models.html

6.1 Link between BNP and High order PPL

See Frank Wood meeting Frank's Theorem: "The Lambda abstraction is necessary in a PPL to be able to denote BNP in this language." The Lambda abstraction seems to be synonymous to function recursion and thus allowing stochastic loop, which are useful for BNP.

Stick-breaking process = control flow + recursion = stochastic recursion ?

6.1.1 Implementation

In Clojure there is a recursive operator for tail-recursive functions that uses constant space. Without such a specific operator, we will have a Maximum call stack size exceeded error for heavy tail BNP such as for some PY stick-breaking process. Unfortunately, Tail Call Optimization (TCO) is not implemented in Julia (since it's not implemented in LLVM on which Julia relies). But other solutions in Julia exist to avoid stack overflow.

Clojure provides special forms loop and recur for writing tail-recursive programs. Anglican programs are CPS-onverted and do not use the stack; recursive calls in Anglican cannot lead to stack overflow.

A function is tail-recursive when the recursive call happens as the final action in a function, in which case it can happen without the function call stack growing. In continuation-passing style, there is no stack all function calls are tail calls, hence all recursive functions are tail-recursive.

6.2 Stick-breaking process

PPL -> be able to sample from prior

6.2.1 DP and PY

well known already implemented in most PPLs

6.2.2 PK

Theorem 2.1 of Perman et al. states that the sequence of surplus masses $(T_k)_(k >= 0)$ forms a Markov chain and gives the corresponding initial distribution and transition kernels.

Could we sample the Kth stick length in generic way since we know the density ? For instance by restricting to Levy measure intensity and Total mass density which are tractable (not -stable PK). One may use a simple rejection sampling with proposal $U(0, t_K)$ (what would be M ?).

There exist other models based directly on specifying a valid distribution for the stick breaks (e.g. Ottowa sequence, Beta-Stacy, others?); these should fit in without a problem.

6.2.3 PG

cf article Ben

6.3 Sampler

highly dimensional -> particle methods write model as streaming/online to avoid IS and take advantage of resampling

prior on parameters -> PMCMC PG -> path degeneracy PPMH -> ? PGAS / IPMCMCM

6.4 Theoretical aspects

calculus for SMC for BNP mixture with fixed parameters fixed parameters are not a assumption, since can be made random then with PMCMC

6.5 Open questions

Is a stick-breaking-like Markov Chain necessary and sufficient for doing inference with particle methods? For lazy instantiation (possibly equivalent to denotable / lambda abstraction?)?

What should be the representation of the state in the PPL (theoretically and efficiently concerned) ? Unique components + assignments | all components | sticks lengths ?

Future Work

In this chapter are presented several ideas which I intend to further developed. When matured enough, this ideas may become a project on itself.

7.1 Learning parameters in PPL

7.1.1 Motivation

Since variational methods have arisen, ideas of mixing sampling with VI have emerged, including in the PPL literature.

In [54], the authors introduce the idea of automatically learning the parameters of proposals for SMC within a PPL. A lower bound on the KL divergence between the proposal and the true posterior distribution is optimize via gradient descent. In [44, 28], this idea is further developed using neural networks (such as LSTMs [26]) to parametrize these proposal distributions. These networks are fed with the previous latent and observed variables. In AESMC [29], FIVO [35] and VSMC [40], both the model and the SMC's proposal are learned by maximizing the marginal likelihood estimator given by the SMC.

The interest in learning parameters (for proposals and for the model) and performing inference on some random variables at once is thus great. PPLs allow to easily write probabilistic models and perform inference on latent variables. One may be interested in building a PPL with the capability of automatically optimizing some parameters given a loss/estimator.

Automatic Differentiation (AD) methods [3] enable the computation of gradients of some variables with respect to some parameters. The reverse differentiation is particularly popular in the machine learning community, where the history of each variable (how it has been constructed) is saved as a computational graph, and gradients can then be computed via the *chain rule*.

Some libraries such as TensorFlow [1] require the users to define static computation graphs within the syntactic and semantic constraints of a domain-specific mini language with limited support for control flow whereas the lineage of projects leading from autograd [1]

[1] https://github.com/HIPS/autograd

to PyTorch [2] provide truly general-purpose reverse mode AD capability. The latter mode is to be preferred for fully and easy support of control flow such as stochastic recursion which is needed for stick-breaking processes.

7.1.2 Choice of language/library

We this idea in mind, one can now think how to pragmatically build such a AD PPL.

Python: One of the most famous language for scientific computing is Python [46]. As *Edward* [49] is built on top of *Tensorflow*, one could build a PPL layer on top of PyTorch. *Edward* implements each MCMC step (specific for each algorithms) as a computational graph in *Tensorflow* which is thereafter run with the updated input so as to sample a new value. Similarly for VI, *Edward* implements a loss function as a computational graph, for which its gradient can be computed via auto-differentiation.

However, *Edward* focuses on VI and HMC-like schemes and does not handle particle algorithms. Indeed, so as to handle such algorithms, a PPL must have access to *breakpoints* at `assume` statement. This can be implemented via CPS [3] or coroutine [4] copying. Unfortunately implementing CPS is something non-trivial.

Julia: One could also think of using Julia [4], which has been specifically built for scientific usage. Julia has the advantage of natively handling coroutine copying, which is used in Turing [21] to implement particle methods.

Reverse mode AD libraries exist in Julia, *ReverseDiff.jl* [5] and *Knet.jl* [6] which respectively build a static and dynamic graph.

I am particularly interested in the perspective of adapting a AD library for Turing [21].

7.1.3 New models

With such as PPL in mind, one can think of new model or algorithms to be developed.

The idea of AESMC [29] might be extended to PG and PMMH so as to learn proposals' parameters for their SMC and for $p(\theta^\star|\theta)$ parameters (specific of PMMH).

7.1.4 Difficulties

Yet this is not a trivial task, one have to put proper care when computing unbiased gradient of a loss function defined by an expectation over a collection of random variables.

[2] `http://pytorch.org/`
[3] `http://matt.might.net/articles/by-example-continuation-passing-style/`
[4] `https://en.wikipedia.org/wiki/Coroutine`
[5] `https://github.com/JuliaDiff/ReverseDiff.jl`
[6] `https://github.com/denizyuret/Knet.jl`

Hopefully, a stochastic computation graph [47] can be converted into a deterministic computation graph, to which the backpropagation algorithm can then be applied on a surrogate loss function which results in an unbiased gradient estimator of the loss.

7.2 Piecewise Deterministic Markov Processes

A novel class of non-reversible Markov chain Monte Carlo schemes relying on continuous-time piecewise deterministic Markov Processes has recently emerged [51]. In these algorithms, the state of the Markov process evolves according to a deterministic dynamics which is modified using a Markov transition kernel at random event times. A general framework is presented in [5], and includes among others the Zig-Zag Process [6], the Bouncy Particle Sampler [9] and the Generalized Bouncy Particle Sampler [57].

It has been claimed [5] that the non-reversibility property of these algorithms enhances the mixing rate of the chain. I am consequently interested in understanding to what extent this class of MCMC schemes could fit the PPL's setting.

7.3 Variational Inference for BNP in PPL

To my knowledge, the first article tackling inference in a BNP setting is [7], where a truncated proposal is introduced to approximate a Dirichlet Process Mixture model.

Truncation-free VI methods have also been introduced [8]. These methods adapts model complexity on the fly by lazily representing clusters assignments. Yet, the sticks proportions and mixture components are marginalize out to obtain a closed form distribution for the mixture assignment hidden variables z_i. This marginalization is unfortunately only tractable for few models, such as the Dirichlet Process and the Pitman-Yor process.

However, we may be able to use a similar approach for more flexible BNP models, by extending the latent space with the sticks proportions and mixture components (since they cannot be marginalized out). Moreover, there might be a deeper link between *Truncation-free* VI and stick-breaking processes.

7.4 Adversarial Inference for BNP in PPL

Adversarial inference methods [17, 13, 38] inspired by GANs [22] jointly learns a generation network and an inference network using an adversarial process.

The decoder/generator network $x' \sim p(x|z)$ maps samples from stochastic latent variables to the data space while the encoder/inference network $z' \sim q(z|x)$ maps training examples in data space to the space of latent variables.

An adversarial game is cast between these two networks and a discriminative network is trained to distinguish between joint latent/data-space samples (x', z) from the generative network and joint samples (x, z') from the inference network.

Adversarial inference seems to be closely related to VI. Yet, in adversarial inference the model can also be learned as opposed to VI where only the proposal is learned. Moreover,

in VI the marginal likelihood is optimised via a lower bound (ELBO) whereas in adversarial inference, a classification loss is optimised.

This approach could be interesting in the BNP setting, if a tractable and tight lower bound on the marginal likelihood cannot be found.

Conclusion and personal review

8.1 Conclusion

Ouverture

8.2 Personal review

Responsability: organization of the probabilistic inference reading group. Research environment Research way of working / thinking: Finding the good questions, etc AI Summer School ?

Appendices

A Variational Inference ?

B Automatic Differentiation ?

Bibliography

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, *Tensorflow: A system for large-scale machine learning*, in 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), 2016, pp. 265–283.

[2] C. Andrieu, A. Doucet, and R. Holenstein, *Particle Markov chain Monte Carlo methods*, Journal of the Royal Statistical Society: Series B (Statistical Methodology), 72 (2010), pp. 269–342.

[3] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, *Automatic differentiation in machine learning: a survey*, arXiv.org, (2015).

[4] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, *Julia: A Fresh Approach to Numerical Computing*, SIAM Review, 59 (2017), pp. 65–98.

[5] J. Bierkens, A. Bouchard-Côté, A. Doucet, A. B. Duncan, P. Fearnhead, G. Roberts, and S. J. Vollmer, *Piecewise Deterministic Markov Processes for Scalable Monte Carlo on Restricted Domains*, arXiv.org, (2017).

[6] J. Bierkens, P. Fearnhead, and G. Roberts, *The Zig-Zag Process and Super-Efficient Sampling for Bayesian Analysis of Big Data*, arXiv.org, (2016).

[7] D. M. Blei and M. I. Jordan, *Variational inference for Dirichlet process mixtures*, Bayesian Analysis, 1 (2006), pp. 121–143.

[8] D. M. Blei and C. Wang, *Truncation-free Stochastic Variational Inference for Bayesian Nonparametric Models*, Advances in Neural Information Processing Systems th Annual Conference on Neural Information Processing Systems, (2012), pp. 422–430.

[9] A. Bouchard-Côté, S. J. Vollmer, and A. Doucet, *The Bouncy Particle Sampler: A Non-Reversible Rejection-Free Markov Chain Monte Carlo Method*, Journal of the American Statistical Association, 4 (2017), pp. 0–0.

[10] B. Carpenter, D. Lee, M. A. Brubaker, A. Riddell, A. Gelman, B. Goodrich, J. Guo, M. Hoffman, M. Betancourt, and P. Li, *Stan: A probabilistic programming language*.

[11] T. D. Chen and C. Fox, Emily B.and Guestrin, *Stochastic gradient hamiltonian monte carlo*, in International Conference on Machine Learning, 2014.

[12] P. Del Moral and L. M. Murray, *Sequential Monte Carlo with Highly Informative Observations*, SIAM/ASA Journal on Uncertainty Quantification, 3 (2015), pp. 969–997.

[13] J. Donahue, P. Krähenbühl, and T. Darrell, *Adversarial Feature Learning*, arXiv.org, (2016).

[14] R. Douc, O. Cappé, and E. Moulines, *Comparison of Resampling Schemes for Particle Filtering*, arXiv.org, (2005).

[15] A. Doucet, N. de Freitas, and N. Gordon, *Sequential Monte Carlo Methods in Practice*, Springer New York, New York, NY, 2001.

[16] S. Duane, A. D. Kennedy, B. J. Pendleton, and D. Roweth, *Hybrid monte carlo*, Physics Letters B, 195 (1987), pp. 216 – 222.

[17] V. Dumoulin, I. Belghazi, B. Poole, O. Mastropietro, A. Lamb, M. Arjovsky, and A. Courville, *Adversarially Learned Inference*, arXiv.org, (2016).

[18] S. Favaro, M. Lomeli, B. Nipoti, and Y. W. Teh, *On the stick-breaking representation of σ-stable Poisson-Kingman models*, Electronic Journal of Statistics, 8 (2014), pp. 1063–1085.

[19] S. Favaro, M. Lomeli, and Y. W. Teh, *On a class of σ-stable Poisson–Kingman models and an effective marginalized sampler*, Statistics and Computing, 25 (2014), pp. 67–78.

[20] S. Favaro and Y. W. Teh, *MCMC for Normalized Random Measure Mixture Models*, Statistical Science, 28 (2013), pp. 335–359.

[21] H. Ge, A. Ścibior, K. Xu, and Z. Ghahramani, *Turing: A fast imperative probabilistic programming language.*, (2016).

[22] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, *Generative Adversarial Networks*, arXiv.org, (2014).

[23] N. Goodman, V. Mansinghka, D. M. Roy, K. Bonawitz, and J. B. Tenenbaum, *Church: a language for generative models*, arXiv.org, (2012).

[24] N. D. Goodman and A. Stuhlmüller, *The Design and Implementation of Probabilistic Programming Languages*. `http://dippl.org`, 2014. Accessed: 2017-8-29.

[25] A. D. Gordon, T. A. Henzinger, A. V. Nori, and S. K. Rajamani, *Probabilistic programming*, in Proceedings of the on Future of Software Engineering, FOSE 2014, New York, NY, USA, 2014, ACM, pp. 167–181.

[26] S. Hochreiter and J. Schmidhuber, *Long Short-Term Memory*, Neural Computation, 9 (1997), pp. 1735–1780.

[27] M. D. Homan and A. Gelman, *The no-u-turn sampler: Adaptively setting path lengths in hamiltonian monte carlo*, J. Mach. Learn. Res., 15 (2014), pp. 1593–1623.

[28] T. A. Le, A. G. Baydin, and F. Wood, *Inference Compilation and Universal Probabilistic Programming*, arXiv.org, (2016).

[29] T. A. Le, M. Igl, T. Jin, T. Rainforth, and F. Wood, *Auto-Encoding Sequential Monte Carlo*, arXiv.org, (2017).

[30] F. Lindsten, M. I. Jordan, and T. B. Schön, *Particle Gibbs with Ancestor Sampling*, arXiv.org, (2014).

[31] M. Lomeli, S. Favaro, and Y. W. Teh, *A hybrid sampler for Poisson-Kingman mixture models*, arXiv.org, (2015).

[32] ——, *A Marginal Sampler for σ-Stable Poisson–Kingman Mixture Models*, Journal of Computational and Graphical Statistics, 26 (2017), pp. 44–53.

[33] D. J. Lunn, A. Thomas, N. Best, and D. Spiegelhalter, *Winbugs - a bayesian modelling framework: Concepts, structure, and extensibility*, Statistics and Computing, 10 (2000), pp. 325–337.

[34] Y.-A. Ma, T. Chen, and E. B. Fox, *A complete recipe for stochastic gradient mcmc*, in Proceedings of the 28th International Conference on Neural Information Processing Systems, NIPS'15, Cambridge, MA, USA, 2015, MIT Press, pp. 2917–2925.

[35] C. J. Maddison, D. Lawson, G. Tucker, N. Heess, M. Norouzi, A. Mnih, A. Doucet, and Y. W. Teh, *Filtering Variational Objectives*, arXiv.org, (2017).

[36] V. Mansinghka, D. Selsam, and Y. Perov, *Venture: a higher-order probabilistic programming platform with programmable inference*, arXiv.org, (2014).

[37] A. McCallum, K. Schultz, and S. Singh, *FACTORIE: Probabilistic programming via imperatively defined factor graphs*, in Neural Information Processing Systems (NIPS), 2009.

[38] L. M. Mescheder, S. Nowozin, and A. Geiger, *Adversarial variational bayes: Unifying variational autoencoders and generative adversarial networks*, in Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017, 2017, pp. 2391–2400.

[39] T. Minka, J. Winn, J. Guiver, S. Webster, Y. Zaykov, B. Yangel, A. Spengler, and J. Bronskill, *Infer.NET 2.6*, 2014. Microsoft Research Cambridge. http://research.microsoft.com/infernet.

[40] C. A. Naesseth, S. W. Linderman, R. Ranganath, and D. M. Blei, *Variational Sequential Monte Carlo*, arXiv.org, (2017).

[41] Y. Nesterov, *Primal-dual subgradient methods for convex problems*, Math. Program., 120 (2009), pp. 221–259.

[42] B. Paige, F. Wood, A. Doucet, and Y. W. Teh, *Asynchronous Anytime Sequential Monte Carlo*, arXiv.org, (2014).

[43] T. Rainforth, C. A. Naesseth, F. Lindsten, B. Paige, J.-W. van de Meent, A. Doucet, and F. Wood, *Interacting Particle Markov Chain Monte Carlo*, arXiv.org, (2016).

[44] D. Ritchie, P. Horsfall, and N. D. Goodman, *Deep Amortized Inference for Probabilistic Programs*, arXiv.org, (2016).

[45] D. Ritchie, A. Stuhlmüller, and N. D. Goodman, *C3: Lightweight Incrementalized MCMC for Probabilistic Programs using Continuations and Callsite Caching*, arXiv.org, (2015).

[46] G. Rossum, *Python reference manual*, tech. rep., Amsterdam, The Netherlands, The Netherlands, 1995.

[47] J. Schulman, N. Heess, T. Weber, and P. Abbeel, *Gradient Estimation Using Stochastic Computation Graphs*, arXiv.org, (2015).

[48] Y. Teh, D. Görür, and Z. Ghahramani, *Stick-breaking construction for the indian buffet process*, in JMLR Workshop and Conference Proceedings Volume 2: AISTATS 2007, Cambridge, MA, USA, Mar. 2007, Max-Planck-Gesellschaft, MIT Press, pp. 556–563.

[49] D. Tran, M. D. Hoffman, R. A. Saurous, E. Brevdo, K. Murphy, and D. M. Blei, *Deep probabilistic programming*, in International Conference on Learning Representations, 2017.

[50] J.-W. van de Meent, H. Yang, V. Mansinghka, and F. Wood, *Particle Gibbs with Ancestor Sampling for Probabilistic Programs*, arXiv.org, (2015).

[51] P. Vanetti, A. Bouchard-Côté, G. Deligiannidis, and A. Doucet, *Piecewise Deterministic Markov Chain Monte Carlo*, arXiv.org, (2017).

[52] M. Welling and Y. W. Teh, *Bayesian learning via stochastic gradient langevin dynamics.*, in ICML, L. Getoor and T. Scheffer, eds., Omnipress, 2011, pp. 681–688.

[53] D. Wingate, N. D. Goodman, and A. Stuhlmüller, *Lightweight Implementations of Probabilistic Programming Languages Via Transformational Compilation*, in Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, Fort Lauderdale, USA, Apr. 2011, pp. 770–778.

[54] D. Wingate and T. Weber, *Automated Variational Inference in Probabilistic Programming*, arXiv.org, (2013).

[55] F. Wood, J. W. van de Meent, and V. Mansinghka, *A new approach to probabilistic programming inference*, in Proceedings of the 17th International conference on Artificial Intelligence and Statistics, 2014, pp. 1024–1032.

[56] F. Wood, J.-W. van de Meent, and V. Mansinghka, *A New Approach to Probabilistic Programming Inference*, arXiv.org, (2015).

[57] C. Wu and C. P. Robert, *Generalized Bouncy Particle Sampler*, arXiv.org, (2017).