Probabilistic Programming; Ways Forward

Frank Wood
Outline

• What is probabilistic programming?
• What are the goals of the field?
• What are some challenges?
• Where are we now?
• Ways forward…
What is probabilistic programming?
An Emerging Field

- ML: Algorithms & Applications
- STATS: Inference & Theory
- PL: Compilers, Semantics, Analysis
- Probabilistic Programming
Conceptualization

Parameters -> Program -> Output

Parameters -> Program -> Observations

\[ p(\theta|x) \]

\[ p(x|\theta)p(\theta) \]

CS       Probabilistic Programming       Statistics
Operative Definition

“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.”

Gordon et al, 2014
What are the goals of probabilistic programming?
Increased Productivity

```
(fn [x] (logb 1.04 (+ 1 x)))
```

Lines of Matlab/Java Code

Lines of Anglican Code

- HPYP, [Wood 2007]
- DDPMO, [Neiswanger et al 2014]
- PDIA, [Pfau 2010]
- Collapsed LDA
- DP Conjugate Mixture
Latent Dirichlet Allocation

Wood (University of Oxford)

is formally written as:

\[ \text{independently from Mult}( \text{Conditional on the assignment variable } i) \]

Each of the mixture generative model for the number of documents, \( D \) to indicate the number of words in the vocabulary, and \( z \) to denote the number of words in the vocabulary, and \( z \), an assignment variable \( k \)

The model is parameterized by the vector valued parameters \( \{ \theta \} \)

\( \text{Discrete}( \text{Dir}(K)) \)

\( \text{Mult}(\theta \cdot | \cdot) \)

Under the uniform deletion model, the number of alive allocation variables at time \( t \) of alive allocation variables at time \( t \), respectively.

To ensure we obtain a first-order stationary Pitman-Yor process mixture model, we also need to

3.3 Properties of the Models

Figure 1: Inference Engine(s)

Programming Language Representation / Abstraction Layer
What are some challenges?
Challenges

- Unbounded recursion
- Equality and continuous variables
Unbounded Recursion

\begin{verbatim}
(defn geometric
  "generates geometrically distributed values in \{0,1,2,...\}\)
  ([p] (geometric p 0))
  ([p n] (if (sample (flip p))
    n
    (geometric p (+ n 1))))
\end{verbatim}
(defm pick-a-stick [stick v l k]
  "picks a stick given a stick generator
  given a value v ~ uniform-continuous(0,1)
  should be called with l = 0.0, k=1"
  (let [u (+ l (stick k))]
    (if (> u v)
      k
      (pick-a-stick stick v u (+ k 1)))))

v (sample (uniform-continuous 0 1))
Semantics and Termination

(defn p []
  (if (sample (flip 0.5))
    1
    (if (sample (flip 0.5))
      (p)
      (infinite-loop))))

(def infinite-loop
  #(loop [] (recur)))

\[ p(x = 1) = \sum_{n=1}^{\infty} \frac{1}{2}^{2n-1} = \frac{2}{3} ? \]
Equality and Continuous Variables

Why are your probabilistic programming systems anti-equality?
Equality

(defquery bayes-net []
  (let [is-cloudy (sample (flip 0.5))
        is-raining (cond (= is-cloudy true) (sample (flip 0.8)) (= is-cloudy false) (sample (flip 0.2)))
        sprinkler (cond (= is-cloudy true) (sample (flip 0.1)) (= is-cloudy false) (sample (flip 0.5)))
        wet-grass (cond (and (= sprinkler true) (= is-raining true)) (sample (flip 0.99)) (and (= sprinkler false) (= is-raining false)) (sample (flip 0.0)) (or (= sprinkler true) (= is-raining true)) (sample (flip 0.9)))]
    (observe (= wet-grass true))
    (predict :s (hash-map :is-cloudy is-cloudy :is-raining is-raining :sprinkler sprinkler)))
(defquery bayes-net [ ]
  (let [is-cloudy (sample (flip 0.5))]
    is-raining (cond (= is-cloudy true)
      (sample (flip 0.8))
      (= is-cloudy false)
      (sample (flip 0.2)))
    sprinkler (cond (= is-cloudy true)
      (sample (flip 0.1))
      (= is-cloudy false)
      (sample (flip 0.5)))
    wet-grass (cond (and (= sprinkler true) (= is-raining true))
      (sample (flip 0.99))
      (and (= sprinkler false) (= is-raining false))
      (sample (flip 0.0))
      (or (= sprinkler true) (= is-raining true))
      (sample (flip 0.9)))]
  (observe (dirac wet-grass) true)
  (predict :s (hash-map :is-cloudy is-cloudy
                        :is-raining is-raining
                        :sprinkler sprinkler)))))

\[ p(x|y = o) \propto \delta(y - o)p(x, y) \]
\[ = p(x, y = o) \]
\[
p(x | y = o) \propto p(d(y, o))p(x, y)
\]
(defquery bayes-net [])
(let [is-cloudy (sample (flip 0.5))]
  is-raining (cond (= is-cloudy true)
                 (sample (flip 0.8))
                 (= is-cloudy false)
                 (sample (flip 0.2)))
  sprinkler (cond (= is-cloudy true)
               (sample (flip 0.1))
               (= is-cloudy false)
               (sample (flip 0.5)))
  wet-grass (cond (and (= sprinkler true) (= is-raining true))
                       (flip 0.99)
                       (and (= sprinkler false) (= is-raining false))
                       (flip 0.0)
                       (or (= sprinkler true) (= is-raining true))
                       (flip 0.9)))

(observe wet-grass true)

(predict :s (hash-map :is-cloudy is-cloudy
                       :is-raining is-raining
                       :sprinkler sprinkler)))

\[ p(x|y = o) \propto p(o|x)p(x) \]
Continuous Variables

(defquery unknown-mean []
  (let [sigma (sqrt 2)
        mu (marsaglia-normal 1 5)]
    (observe (normal mu sigma) 9)
    (observe (normal mu sigma) 8)
    (predict :mu mu)))
Measure Theoretic Challenges

The “Indian GPA problem”

\[ p(\text{nationality} = \text{"USA"} | \ gpa = 4.0) = ? \]

(defquery which-nationality [gpa]
  (let [nationality (sample categorical [["USA" 0.25] ["India" 0.75]])
      simulated_gpa (if (= nationality "USA")
                      (american-gpa)
                      (indian-gpa))]
    (observe (dirac simulated_gpa) gpa)
    (predict :nationality nationality)))
American GPA Distribution $[0,4]$

(defn american-gpa []
  (if (sample (flip 0.95))
    (* 4 (sample (beta 8 2)))
    (if (sample (flip 0.85))
      4.0
      0.0))))
Indian GPA Distribution \([0,10]\)

```
(defn indian-gpa []
  (if (sample (flip 0.99))
    (* 10 (sample (beta 5 5)))
    (if (sample (flip 0.1))
      0.0
      10.0)))
```
Mixed GPA Distribution

(defn student-gpa []
    (if (sample (flip 0.25))
        (american-gpa)
        (indian-gpa))))
The “Indian GPA problem” by Russell

\[ p(\text{nationality} = "USA" | \text{gpa} = 4.0) = ? \]

```
(defquery which-nationality [gpa tolerance]
  (let [nationality (sample (categorical [["USA" 0.25] ["India" 0.75]])
        simulated_gpa (if (= nationality "USA")
                        (american-gpa)
                        (indian-gpa))]
    (observe (normal simulated_gpa tolerance) gpa)
    (predict :nationality nationality)))
```
Where are we now?
Discrete RV’s Only

1990

2000

2010

PL

AI

ML

STATS

Simula

Prolog
Ways forward...
Trace Probability

- **observe** data points $y_n$
- internal random choices $x_n$
- simulate from
  
  $f(x_n|x_{1:n-1})$

by running the program forward

- weight execution traces by
  
  $g(y_n|x_{1:n})$
Iteratively,

- simulate
- weight
- resample
SMC for Probabilistic Programming

Intuitively:
- run
- wait
- fork

Threads

observe delimiter

continuations
• Sequential Monte Carlo is now a building block for other inference techniques

• Particle MCMC
  - PIMH : “particle independent Metropolis-Hastings”
  - iCSMC : “iterated conditional SMC”

[Andrieu, Doucet, Holenstein 2010]
[W., van de Meent, Mansinghka 2014]
SMC Parallelism Bottleneck

SMC slowed down for clarity
Particle Cascade

Paige, W., Doucet, Teh; NIPS 2014
Particle Cascade
Particle Cascade
Particle Cascade
Theoretical Properties

The particle cascade provides an **unbiased estimator** of the **marginal likelihood**, whose variance decreases proportionally to the number of initial particles $K_0$:

$$\hat{p}(y_{0:n}) := \frac{1}{K_0} \sum_{k=1}^{K_n} W_n^k$$

**Theorem:** For any $K_0 \geq 1$ and $n \geq 0$, $\mathbb{E}[\hat{p}(y_{0:n})] = p(y_{0:n})$.

**Theorem:** For any $n \geq 0$, there exists a constant $a_n$ such that

$$\mathbb{V}[\hat{p}(y_{0:n})] < \frac{a_n}{K_0}$$
Conclusion
Thank You

• Questions?

• Funding: DARPA, Amazon, Microsoft
Opportunities

• Parallelism

  “Asynchronous Anytime Sequential Monte Carlo” [Paige, W., Doucet, Teh NIPS 2014]

• Backwards passing

  “Particle Gibbs with Ancestor Sampling for Probabilistic Programs” [van de Meent, Yang, Mansinghka, W. AISTATS 2015]

• Search

  “Maximum a Posteriori Estimation by Search in Probabilistic Models” [Tolpin, W., SOCS, 2015]

• Adaptation

  “Output-Sensitive Adaptive Metropolis-Hastings for Probabilistic Programs” [Tolpin, van de Meent, Paige, W.; in submission]

• Novel proposals

  “Adaptive PMCMC” [Paige, W.; in submission]
Probabilistic-C

```c
#include "probabilistic.h"
#define K 3
#define N 17

/* Markov transition matrix */
static double T[K][K] = {
    { 0.1, 0.5, 0.4 },
    { 0.2, 0.2, 0.6 },
    { 0.15, 0.15, 0.7 }
};

/* Prior distribution on initial state */
static double initial_state[K] = { 1.0/3, 1.0/3, 1.0/3 }

/* Generative program for a Markov model */
int main(int argc, char **argv) {

    int states[N];
    for (int n=0; n<N; n++) {
        states[n] = (n==0) ? discrete_rng(initial_state, K)
                          : discrete_rng(T[states[n-1]], K);
        predict("state[%d],%d\n", n, states[n]);
    }

    return 0;
}
```
Paige & W.; ICML 2014
How can you participate?
Ways to Participate

• Contribute applications
  • https://bitbucket.org/fwood/anglican-examples

• Contribute inference algorithms
  • https://bitbucket.org/dtolpin/embang
An Analogy

Automatic Differentiation
Supervised Learning

→

Probabilistic Programming
Unsupervised Learning
defquery sat-solver [N formula]
    "explores an N-dimensional universe for worlds that satisfy the formula"
    (let [state (repeatedly N (fn [] (sample (flip 0.5)))))
        (observe (dirac (formula state)) true)
        (predict :state state))

defdist dirac
    "Dirac distribution"
    [x] []
        (sample [this] x)
        (observe [this value] (if (= x value) 0.0 NegInf)))

defm satisfiable-3cnf-formula [state]
    (let [v (fn [i] (nth state i))]
        (and (or (v 0) (not (v 1))) (not (v 2)))
            (or (not (v 0)) (v 1) (v 2))
            (or (not (v 0)) (not (v 1)) (not (v 2))))
(defquery md5-inverse [L md5str]
  "conditional distribution of strings that map to the same MD5 hashed string"
  (let [mesg (sample (string-generative-model L))]
    (observe (dirac md5str) (md5 mesg))
    (predict :message mesg))))
Particle Cascade
Not Sum-Product: Bayesian HMM

Suppose the transition matrix is unknown: $T_k \sim \text{Dirichlet}(\alpha_k)$

```c
#define N 17

/* Markov transition matrix */
static double T[K][K] = {
    { 0.1, 0.5, 0.4 },
    { 0.2, 0.2, 0.6 },
    { 0.15, 0.15, 0.7 }
};

/* Observed data */
static double data[N] = {
    NAN, .9, .8, .7, 0, -.025,
    -5, -2, -.1, 0, 0.13, 0.45,
    6, 0.2, 0.3, -1, -1
};

/* Prior distribution on initial state */
static double initial_state[K] = { 1.0/3, 1.0/3, 1.0/3 }

/* Per-state mean of Gaussian emission distribution */
static double state_mean[K] = { -1, 1, 0 }

/* Generative program for a HMM */
int main(int argc, char **argv) {
    int states[N];
    for (int n=0; n<N; n++) {
        states[n] = (n==0) ? discrete_rng(initial_state, K) :
            discrete_rng(T[states[n-1]], K);
    }
}
```
Range of Effectiveness

PL
HANSAI
IBAL
Figaro

AI
ML
webChurch
Probabilistic-C
Venture
Anglican

ML
Factorie
Church
Infer.NET

STATS
LibBi
STAN
JAGS

IBAL
Prolog
Blog
Prism
KMP

1990
2000
2010
Continuous Variables

(defm marsaglia-normal [mean var]
  (let [d (uniform-continuous -1.0 1.0)
        x (sample d)
        y (sample d)
        s (+ (* x x) (* y y))]
    (if (< s 1)
        (+ mean (* (sqrt var)
                   (* x (sqrt (* -2 (/ (log s) s))))))
        (marsaglia-normal mean var)))))
Scalability: Particle Count

- Comparison across particle-based inference approaches: raw speed of drawing samples
Unbounded Recursion

Expressivity

Efficiency
Credits

- Code highlighting: http://hilite.me
Forward Inference (SMC)
Bayesian Nonparametrics

```emacs-lisp
(defm pick-a-stick [stick v l k]
    ; picks a stick given a stick generator
    ; given a value v ~ uniform-continuous(0,1)
    ; should be called with l = 0.0, k=1
(let [u (+ l (stick k))]
    (if (> u v)
        k
        (pick-a-stick stick v u (+ k 1))))

(defm remaining [b k]
    [if (<= k 0)
        1
        (* (- 1 (b k)) (remaining b (- k 1))))

(defm polya [stick]
    ; given a stick generating function
    ; polya returns a function that samples
    ; stick indexes from the stick lengths
(let [uc01 (uniform-continuous 0 1)]
    (fn []
        (let [v (sample uc01)]
            (pick-a-stick stick v 0.0 1))))

(defm dirichlet-process-breaking-rule [alpha k] (sample (beta 1.0 alpha)))

(defm stick [breaking-rule]
    ; given a breaking-rule function which
    ; returns a value between 1 and 0 given a
    ; stick index k returns a function that
    ; returns the stick length for index k
(let [b (mem breaking-rule)]
    (fn [k]
        (if (< 0 k)
            (* (b k) (remaining b (- k 1))
            0))))
```

---

The image contains a page from a document discussing Bayesian Nonparametrics in Emacs Lisp. The text includes code examples, with comments explaining the functions `pick-a-stick`, `remaining`, `polya`, `dirichlet-process-breaking-rule`, and `stick`. The code is designed to simulate stick-breaking processes, which are fundamental in Bayesian nonparametric models. The figure illustrates the stick-breaking process visually, showing how sticks of varying lengths are broken and distributed according to the given functions.
Syntax & Implementation Considerations

• Embedded vs. Standalone

• Imperative vs. functional

• Lisp vs. Python vs. C vs.