Design and Implementation of Anglican Probabilistic Programming Language

David Tolpin Jan Willem van de Meent Hongseok Yang
Frank Wood

September 1, 2016

https://bitbucket.org/probprog/anglican-white-paper
https://bitbucket.org/probprog/anglican
http://www.robots.ox.ac.uk/~fwood/anglican/index.html

Outline

Intuition

Probabilistic program:

- A program with random computations.
- Distributions are conditioned by 'observations'.
- Values of certain expressions are 'predicted' — **the output**.

Can be written in any language (extended by `sample` and `observe`).

Example: Model Selection

```
1  (let [;; Guessing a distribution
2        dist (sample (categorical
3                      [[normal 1] [gamma 1]
4                       [uniform-continuous 1]
5                       [uniform-discrete 1]]))
6        a (sample (gamma 1 1))
7        b (sample (gamma 1 1))
8        d (dist a b)]
9    ;; Observing samples from the distribution
10   (loop [data data]
11     (when (seq data)
12       (let [[x & data] data]
13         (observe d x))
14       (recur data)))
15   ;; Predicting a, b and the distribution
16   (predict :a a)
17   (predict :b b)
18   (predict :d d))
```

More examples

- *Intruder detection* — given a log of **times** and **amounts** of payments in a bank account, how likely that the baccount was compromised?

More examples

- *Intruder detection* — given a log of **times** and **amounts** of payments in a bank account, how likely that the baccount was compromised?
- *Counterfactual reasoning* — There are **two routes** from Jerusalem to Tel Aviv: 1 and 443. Based on traffic reports, I chose route 1 and was late. Would I arrive on time If I chose 443 instead?

More examples

- *Intruder detection* — given a log of **times** and **amounts** of payments in a bank account, how likely that the baccount was compromised?
- *Counterfactual reasoning* — There are **two routes** from Jerusalem to Tel Aviv: 1 and 443. Based on traffic reports, I chose route 1 and was late. Would I arrive on time If I chose 443 instead?
- *(Due to Stuart Russell)* If you observe that a student GPA is exactly 4.0 in a model of transcripts of students from the USA (GPA's from 0.0 to 4.0) and India (GPA's from 0.0 to 10.0) what is the probability that the student is from India?

Inference Objective

- Suggest **most probable explanation** (MPE) - most likely assignment for all non-evidence variable given evidence.

Inference Objective

- Suggest **most probable explanation** (MPE) - most likely assignment for all non-evidence variable given evidence.
- Approximately **compute integral** of the form

$$\Phi = \int_{-\infty}^{\infty} \varphi(x) p(x) dx$$

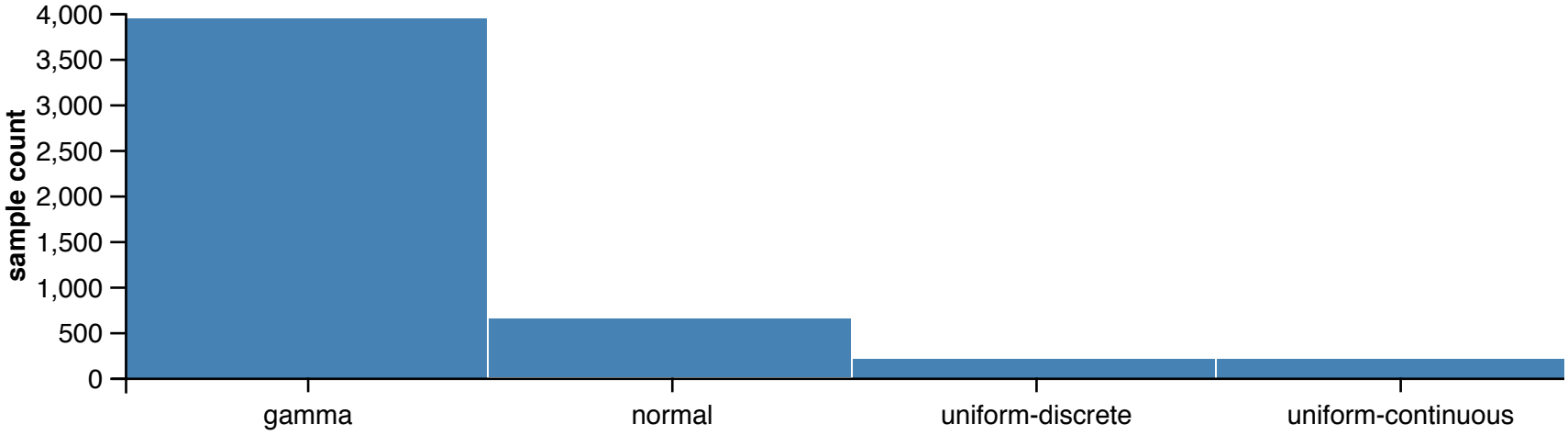
Inference Objective

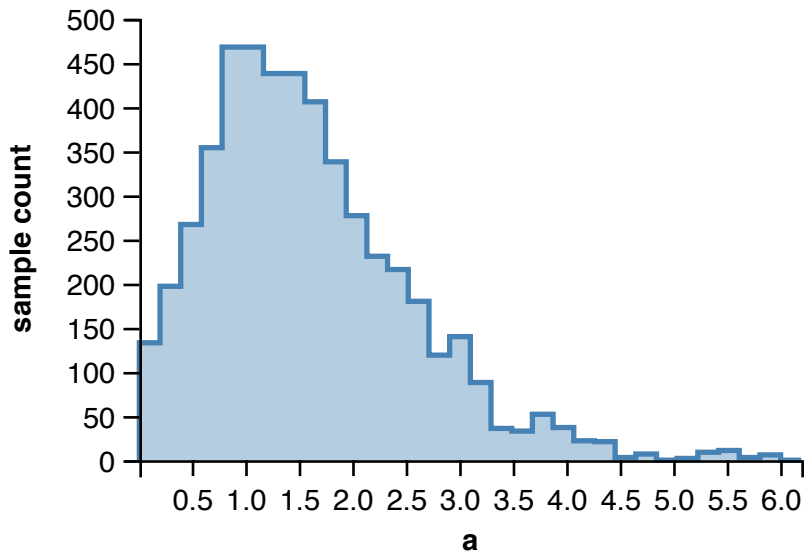
- Suggest **most probable explanation** (MPE) - most likely assignment for all non-evidence variable given evidence.
- Approximately **compute integral** of the form

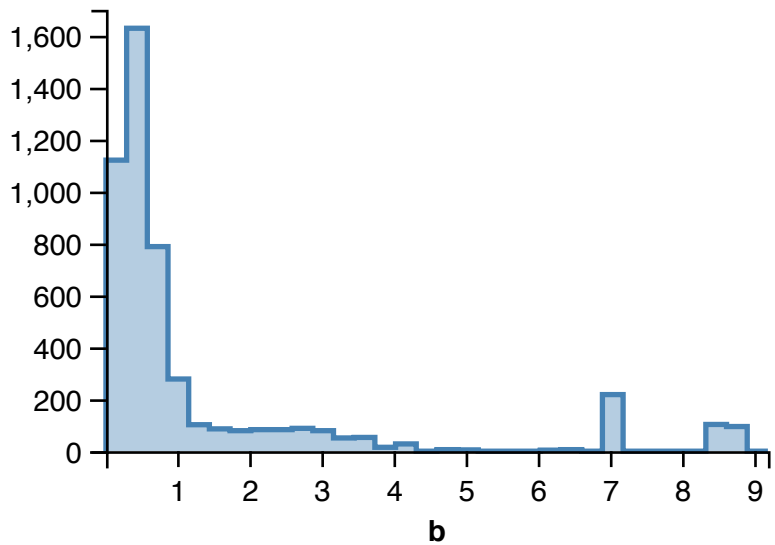
$$\Phi = \int_{-\infty}^{\infty} \varphi(x) p(x) dx$$

- Continuously and **infinitely generate a sequence of samples** drawn from the distribution of the output expression — so that someone else puts it in good use (vague but common). ✓

Example: Inference Results







Importance Sampling

loop

Run program, computing weight based on observations.

Output result and weight.

end loop

- Simple — good.
- Slow convergence (unless one knows the answer) — bad.

Can we do better?

Lightweight Metropolis-Hastings (LMH)

Run program once, remembering random choices.
loop
    Uniformly select one random choice.
    Propose a new value for the choice.
    Re-run the program.
    Accept or reject with MH probability.
    Output result.
end loop

Can we do better?

- Particle methods
- Variational inference
- ...

Why functional?

We want a functional language because an inference algorithm controls the execution:

- A program is run many (often many hundreds of thousands) of times (with almost any algorithm).
- A program must be partially re-executed multiple times from different positions (particle methods).
- We want to reason about the distribution defined by the program.

Why Closure?

- Runs on JVM — easy deployment and access to libraries.

Why Closure?

- Runs on JVM — easy deployment and access to libraries.
- A Lisp — we (ab)use the macro facility.

Why Closure?

- Runs on JVM — easy deployment and access to libraries.
- A Lisp — we (ab)use the macro facility.
- Church (`https://en.wikipedia.org/wiki/Church_(programming_language)`) is derived from Scheme.

Why Closure?

- Runs on JVM — easy deployment and access to libraries.
- A Lisp — we (ab)use the macro facility.
- Church (`https://en.wikipedia.org/wiki/Church_(programming_language)`) is derived from Scheme.

Others use:

- Scheme — (Church, Venture).
- Scala — Figaro.
- Haskell — Hakaru, Model-Bayes.
- ...

Why Closure?

- Runs on JVM — easy deployment and access to libraries.
- A Lisp — we (ab)use the macro facility.
- Church (`https://en.wikipedia.org/wiki/Church_(programming_language)`) is derived from Scheme.

Others use:

- Scheme — (Church, Venture).
- Scala — Figaro.
- Haskell — Hakaru, Model-Bayes.
- ...

As well as Python, C#, and other languages.

Outline

Anglican and Clojure

On language on top of (or besides) another:

- Interpeter.
- Source-to-source compiler.
- Integrated language. ✓

Anglican and Clojure

On language on top of (or besides) another:

- Interpeter.
- Source-to-source compiler.
- Integrated language. ✓

Integrated language

- either extends the syntax of the host language;
- or changes the semantics of the host language constructs ✓.

Anglican and Clojure

On language on top of (or besides) another:

- Interpeter.
- Source-to-source compiler.
- Integrated language. ✓

Integrated language

- either extends the syntax of the host language;
- or changes the semantics of the host language constructs ✓.

Anglican is

- Integrated with Clojure.
- Shares syntax.
- Alters operational semantics.

Anglican

A subset of Clojure, wrapped inside `defquery`:

- `if`, `when`, `cond`, `case`, `let`, `and`, `or`, `fn`.
- Vector destructuring in bindings of `let` and `fn`.
- Compound literals for vectors, hash maps, and sets.
- `loop`/`recur` — a convenience.

Core library:

- All of Clojure core library, except for higher-order functions.
- `map`, `reduce`, `filter`, `some`, `repeatedly`, `comp`, `partial`.

Any Clojure function can be called from Anglican.

Macro-based compilation

Anglican code macro-compiled into Clojure:

```
1   (loop [data data]
2     (if (seq data)
3       (let [[x & data] data]
4         (observe (gamma a b) x)
5         (recur data))
6       (predict :a a)))
```

```
1   (fn loop [C23151 state data]
2     (if (seq data)
3       (let [[x & data] data]
4         (->observe 'O23153 (gamma a b) x
5           (fn do23152 [_ state]
6             (loop C23151 state data))
7           state))
8       (fn []
9         (C23151
10          nil
11          (add-predict state :a a)))))
```

Outline

Managing stack size

Probabilistic forms

Memoization

Outline

Outline

Thank you!
Questions?