Introduction to probabilistic programming

Frank Wood

fwood@cs.ubc.ca
Objectives For Today

Get you to

- Understand what probabilistic programming is
- Think generatively
- Understand inference
  - Importance sampling
  - SMC
  - MCMC
- Understand something about how modern, performant higher-order probabilistic programming systems are implemented at a very high level
Probabilistic Programming

ML: Algorithms & Applications

STATS: Inference & Theory

PL: Evaluators & Semantics

AI: Deep Learning
Intuition

Inference

\[ p(x|y) \]

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

Parameters

Program

Output

Parameters

Program

Observations

\[ p(x|y) \]

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

y

CS

Probabilistic Programming

Statistics
Probabilistic Programs

“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, Henzinger, Nori, and Rajamani
Key Ideas

Models

\[ p(x, y) \]

Programming Language Abstraction Layer

\[ p(x|y) = \frac{p(x, y)}{p(y)} \]

Evaluators that automate Bayesian *inference*
## Long History

<table>
<thead>
<tr>
<th>Year</th>
<th>PL</th>
<th>AI</th>
<th>ML</th>
<th>STATS</th>
</tr>
</thead>
<tbody>
<tr>
<td>2010</td>
<td>Hakaru, R2, Gamble</td>
<td>Figaro, HANSAI</td>
<td>webPPL, Probabilistic-C, Venture, Anglican</td>
<td>LibBi, STAN</td>
</tr>
<tr>
<td>2000</td>
<td>IBAL</td>
<td>ProbLog, Blog</td>
<td>Lambda, Church, Infer.NET, Factorie</td>
<td>JAGS, STAN</td>
</tr>
<tr>
<td>1990</td>
<td>Simula</td>
<td>Prism, KMP</td>
<td>KMP</td>
<td>WinBUGS, BUGS</td>
</tr>
<tr>
<td></td>
<td>Prolog</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Existing Languages

Graphical Models
- BUGS
- STAN

Factor Graphs
- Factorie
- Infer.NET

Infinite Dimensional Parameter Space Models
- Anglican
- WebPPL

Unsupervised Deep Learning
- PYRO
- ProbTorch
model
{
  x ~ dnorm(1, 1/5)
  for(i in 1:N) {
    y[i] ~ dnorm(x, 1/2)
  }
}

"N" <- 2
"y" <- c(9, 8)

• Language restrictions
  • Bounded loops
  • No branching

• Model class
  • Finite graphical models
  • Inference - sampling
  • Gibbs

STAN: Finite Dimensional Differentiable Distributions

parameters {
    real xs[T];
}

model {
    xs[1] ~ normal(0.0, 1.0);
    for (t in 2:T)
        xs[t] ~ normal(a * xs[t - 1], q);
    for (t in 1:T)
        ys[t] ~ normal(xs[t], 1.0);
}

• Language restrictions
  • Bounded loops
  • No discrete random variables*
• Model class
  • Finite dimensional differentiable distributions
• Inference
  • Hamiltonian Monte Carlo
    • Reverse-mode automatic differentiation
  • Black box variational inference, etc.

\[ \nabla_x \log p(x, y) \]

Goal
\[ p(x|y) \]

Modeling language desiderata

- Unrestricted language (C++, Python, Lisp, etc.)
  - “Open-universe” / infinite dim. parameter spaces
  - Mixed variable types

- Pros
  - Unfettered access to existing libraries
  - Easily extensible

- Cons
  - Inference is going to be harder
  - More ways to shoot yourself in the foot
Deterministic Simulation and Other Libraries

```lisp
(defquery arrange-bumpers []
  (let [bumper-positions []
    ;; code to simulate the world
    world (create-world bumper-positions)
      end-world (simulate-world world)
    balls (:balls end-world)
    ]
    ;; how many balls entered the box?
    num-balls-in-box (balls-in-box end-world))

  {:balls balls
    :num-balls-in-box num-balls-in-box
    :bumper-positions bumper-positions})

goal: “world” that puts ~20% of balls in box…
```
Open Universe Models and Nonparametrics

(defquery arrange-bumpers []
  (let [number-of-bumpers (sample (poisson 20))
        bumpydist (uniform-continuous 0 10)
        bumpxdist (uniform-continuous -5 14)
        bumper-positions (repeatedly
                           number-of-bumpers
                           #(vector (sample bumpxdist)
                                     (sample bumpydist)))
    ;; code to simulate the world
    world (create-world bumper-positions)
    end-world (simulate-world world)
    balls (:balls end-world)

    ;; how many balls entered the box?
    num-balls-in-box (balls-in-box end-world))

{:balls balls
 :num-balls-in-box num-balls-in-box
 :bumper-positions bumper-positions}})
(defquery arrange-bumpers []
  (let [number-of-bumpers (sample (poisson 20))
        bumpydist (uniform-continuous 0 10)
        bumpxdist (uniform-continuous -5 14)
        bumper-positions (repeatedly
                           number-of-bumpers
                           #(vector (sample bumpxdist)
                                      (sample bumpydist)))]

  ;; code to simulate the world
  world (create-world bumper-positions)
  end-world (simulate-world world)
  balls (:balls end-world)

  ;; how many balls entered the box?
  num-balls-in-box (balls-in-box end-world)

  obs-dist (normal 4 0.1))

  (observe obs-dist num-balls-in-box)

  {:balls balls
   :num-balls-in-box num-balls-in-box
   :bumper-positions bumper-positions})
New Kinds of Models

\[ p(x|y) = \frac{p(y|x)p(x)}{p(y)} \]

- program source code
- scene description
- policy and world
- cognitive process
- simulation

- program return value
- image
- rewards
- observed behavior
- simulator output
Thinking Generatively
Can you write a program to do this?

Mansinghka, Kulkarni, Perov, and Tenenbaum
Captcha Generative Model

(\[
\text{\texttt{sample-char}} \] \[
\{ \text{\texttt{symbol}} \ (\text{\texttt{sample}} \ (\text{\texttt{uniform}} \ \text{\texttt{ascii}})) \ \\
\text{\texttt{x-pos}} \ (\text{\texttt{sample}} \ (\text{\texttt{uniform-cont}} \ 0.0 \ 1.0)) \ \\
\text{\texttt{y-pos}} \ (\text{\texttt{sample}} \ (\text{\texttt{uniform-cont}} \ 0.0 \ 1.0)) \ \\
\text{\texttt{size}} \ (\text{\texttt{sample}} \ (\text{\texttt{beta}} \ 1 \ 2)) \ \\
\text{\texttt{style}} \ (\text{\texttt{sample}} \ (\text{\texttt{uniform-dis}} \ \text{\texttt{styles}})) \ \\
\ldots \}
\] 

(\[
\text{\texttt{sample-captcha}} \] \[
(\text{\texttt{let}} \ [\text{\texttt{n-chars}} \ (\text{\texttt{sample}} \ (\text{\texttt{poisson}} \ 4)) \ \\
\text{\texttt{chars}} \ (\text{\texttt{repeatedly}} \ \text{\texttt{n-chars}} \ \text{\texttt{sample-char}}) \ \\
\text{\texttt{noise}} \ (\text{\texttt{sample}} \ \text{\texttt{salt-pepper}}) \ \\
\ldots] \ \\
\text{\texttt{gen-image}}))
\]
(defquery captcha [true-image]
(let [gen-image (sample-captcha)]
  (observe (similarity-kernel gen-image true-image) gen-image))

(doquery :ipmcmc captcha true-image)
Perception / Inverse Graphics

Captcha Solving

<table>
<thead>
<tr>
<th>Input Image</th>
<th>Intermediate Iterations</th>
<th>Final Inferred Image</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="...captcha-input.png" alt="Image" /></td>
<td><img src="...captcha-intermediate.png" alt="Image" /></td>
<td><img src="...captcha-final.png" alt="Image" /></td>
</tr>
</tbody>
</table>

Scene Description

<table>
<thead>
<tr>
<th>Observed Image</th>
<th>Inferred (reconstruction)</th>
<th>Inferred model re-rendered with novel poses</th>
<th>Inferred model re-rendered with novel lighting</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="...scene-observed.png" alt="Image" /></td>
<td><img src="...scene-inferred.png" alt="Image" /></td>
<td><img src="...scene-novel-poses.png" alt="Image" /></td>
<td><img src="...scene-novel-lighting.png" alt="Image" /></td>
</tr>
</tbody>
</table>

Mansinghka, Kulkarni, Perov, and Tenenbaum.

Kulkarni, Kohli, Tenenbaum, Mansinghka
Directed Procedural Graphics

Stable Static Structures

Procedural Graphics

Ritchie, Lin, Goodman, & Hanrahan.
Generating Design Suggestions under Tight Constraints with Gradient-based Probabilistic Programming.

Ritchie, Mildenhall, Goodman, & Hanrahan.
“Controlling Procedural Modeling Programs with Stochastically-Ordered Sequential Monte Carlo.”
SIGGRAPH (2015)
Program Induction

\[ \hat{y} \sim p(\cdot|x) \]

\[ x \sim p(x) \]

\[ \tilde{y} \sim p(\cdot|x) \]

\[ x \sim p(x|y) \]

program source code

program output

Perov and Wood.

"Automatic Sampler Discovery via Probabilistic Programming and Approximate Bayesian Computation"

AGI (2016).
(defquery lin-reg [x-vals y-vals]
(let [m (sample (normal 0 1))
    c (sample (normal 0 1))
    f (fn [x] (+ (* m x) c))]
(map (fn [x y]
      (observe
       (normal (f x) 0.1) y))
x-vals y-vals))
[m c])

(doquery :ipmcmc lin-reg data options)

([0.58 -0.05] [0.49 0.1] [0.55 0.05] [0.53 0.04] ....)
A simple continuous example

• Measure the temperature of some water using an inexact thermometer

• The actual water temperature $x$ is somewhere near room temperature of 22°; we record an estimate $y$.

\[
x \sim \text{Normal}(22, 10)
\]

\[
y|x \sim \text{Normal}(x, 1)
\]

**Easy question:** what is $p(y \mid x = 25)$ ?

**Hard question:** what is $p(x \mid y = 25)$ ?
• Our data is given by $y$

• Our generative model specifies the prior and likelihood

• We are interested in answering questions about the posterior distribution of $p(x \mid y)$
Typically we are not trying to compute a probability density function for $p(x \mid y)$ as our end goal.

Instead, we want to compute expected values of some function $f(x)$ under the posterior distribution.

General problem:

$$p(x \mid y) = \frac{p(y \mid x)p(x)}{p(y)}$$

Posterior, Likelihood, Prior
Expectation

- Discrete and continuous:

\[ \mathbb{E}[f] = \sum_x p(x) f(x) \]

\[ \mathbb{E}[f] = \int p(x) f(x) \, dx. \]

- Conditional on another random variable:

\[ \mathbb{E}_x[f|y] = \sum_x p(x|y) f(x) \]
Key Monte Carlo identity

- We can approximate expectations using samples drawn from a distribution $p$. If we want to compute

$$
\mathbb{E}[f] = \int p(x) f(x) \, dx.
$$

we can approximate it with a finite set of points sampled from $p(x)$ using

$$
\mathbb{E}[f] \approx \frac{1}{N} \sum_{n=1}^{N} f(x_n)
$$

which becomes exact as $N$ approaches infinity.
How do we draw samples?

- Simple, well-known distributions: samplers exist (for the moment take as given)

- We will look at:
  1. Build samplers for complicated distributions out of samplers for simple distributions compositionally
  2. Rejection sampling
  3. Likelihood weighting
  4. Markov chain Monte Carlo
Ancestral sampling from a model

• In our example with estimating the water temperature, suppose we already know how to sample from a normal distribution.

\[ x \sim \text{Normal}(22, 10) \]
\[ y|x \sim \text{Normal}(x, 1) \]

We can sample \( y \) by literally simulating from the generative process: we first sample a “true” temperature \( x \), and then we sample the observed \( y \).

• This draws a sample from the \textbf{joint} distribution \( p(x, y) \).
Samples from the joint distribution
Conditioning via rejection

• What if we want to sample from a conditional distribution? The simplest form is via rejection.

• Use the ancestral sampling procedure to simulate from the generative process, draw a sample of $x$ and a sample of $y$. These are drawn together from the joint distribution $p(x, y)$.

• To estimate the posterior $p(x \mid y = 25)$, we say that $x$ is a sample from the posterior if its corresponding value $y = 25$.

• **Question:** is this a good idea?
Conditioning via rejection

Black bar shows measurement at $y = 25$.
How many of these samples from the joint have $y = 25$?
One option is to sidestep sampling from the posterior $p(x \mid y = 3)$ entirely, and draw from some proposal distribution $q(x)$ instead.

Instead of computing an expectation with respect to $p(x|y)$, we compute an expectation with respect to $q(x)$:

$$
\mathbb{E}_{p(x|y)}[f(x)] = \int f(x)p(x|y)dx
$$

$$
= \int f(x)p(x|y)\frac{q(x)}{q(x)}dx
$$

$$
= \mathbb{E}_{q(x)} \left[ f(x)\frac{p(x|y)}{q(x)} \right]
$$
Conditioning via importance sampling

• Define an “importance weight” \( W(x) = \frac{p(x|y)}{q(x)} \)

• Then, with \( x_i \sim q(x) \)

\[
\mathbb{E}_{p(x|y)}[f(x)] = \mathbb{E}_{q(x)}[f(x)W(x)] \approx \frac{1}{N} \sum_{i=1}^{N} f(x_i)W(x_i)
\]

• Expectations now computed using weighted samples from \( q(x) \), instead of unweighted samples from \( p(x|y) \)
Conditioning via importance sampling

• Typically, can only evaluate \( W(x) \) up to a constant (but this is not a problem):

\[
W(x_i) = \frac{p(x_i|y)}{q(x_i)} \\
\text{and} \\
w(x_i) = \frac{p(x_i, y)}{q(x_i)}
\]

• Approximation:

\[
W(x_i) \approx \frac{w(x_i)}{\sum_{j=1}^{N} w(x_j)}
\]

\[
\mathbb{E}_{p(x|y)}[f(x)] \approx \sum_{i=1}^{N} \frac{w(x_i)}{\sum_{j=1}^{N} w(x_j)} f(x_i)
\]
Conditioning via importance sampling

• We already have very simple proposal distribution we know how to sample from: the prior $p(x)$.

• The algorithm then resembles the rejection sampling algorithm, except instead of sampling both the latent variables and the observed variables, we only sample the latent variables.

• Then, instead of a “hard” rejection step, we use the values of the latent variables and the data to assign “soft” weights to the sampled values.
Likelihood weighting schematic

Draw a sample of $x$ from the prior
What does $p(y|x)$ look like for this sampled $x$?
What does $p(y|x)$ look like for this sampled $x$?
What does $p(y|x)$ look like for this sampled $x$?
Compute $p(y|x)$ for all of our $x$ drawn from the prior
Assign weights (vertical bars) to samples for a representation of the posterior
Conditioning via MCMC

- **Problem**: Likelihood weighting degrades poorly as the dimension of the latent variables increases, unless we have a very well-chosen proposal distribution \( q(x) \).

- **An alternative**: Markov chain Monte Carlo (MCMC) methods draw samples from a target distribution by performing a biased random walk over the space of the latent variables \( x \).

- Idea: create a Markov chain such that the sequence of states \( x_0, x_1, x_2, \ldots \) are samples from \( p(x \mid y) \)

![Diagram of Markov Chain](image)

\[
p(x_n \mid x_{n-1})
\]
Conditioning via MCMC

• MCMC also uses a proposal distribution, but this proposal distribution makes **local** changes to the latent variables \( x \). The proposal \( q(x' \mid x) \) defines a conditional distribution over \( x' \) given a current value \( x \).

• Typical choice: add small amount of Gaussian noise

• We use the proposal and the joint density to define an "acceptance ratio"

\[
A(x \rightarrow x') = \min \left( 1, \frac{p(x', y)q(x|x')}{p(x, y)q(x'|x)} \right)
\]

• Metropolis-Hastings: with probability \( A \) we "move" state with the new value \( x' \), otherwise we stay at \( x \).
The (unnormalized) joint distribution $p(x,y)$ is shown as a dashed line.
Initialize arbitrarily (e.g. with a sample from the prior)
Propose a local move on $x$ from a transition distribution
Here, we proposed a point in a region of higher probability density, and accepted...
Continue: propose a local move, and accept or reject.
At first, this will look like a stochastic search algorithm!
Once in a high-density region, it will explore the space
Once in a high-density region, it will explore the space
Helpful diagnostic: a “trace plot” of the path of the sampled values, as the number of MCMC iterations increases
Histogram of trace plot, overlaid on prior probability density
How It Works:
PPL Inference
(let [z (sample (bernoulli 0.5))
    mu (if (= z 0) -1.0 1.0)
    d (normal mu 1.0)
    y 0.5]
  (observe d y)
  z)

Program
The problem is then to characterize the expected return value and observes a value distribution, based on which it sets a likelihood parameter. The program first samples this program describes a two-component Gaussian mixture with a sin-

Before presenting a set of translation rules that can be used to compile any FOPPL program to a Bayesian network, we will illustrate an example model.

### Definition of a Bayesian Network

A Bayesian network represents a joint probability distribution over a set of random variables. It consists of:

- A set of vertices $V$, representing random variables.
- A set of arcs $A$, representing conditional dependencies among random variables.
- A partial map $P$, containing $V$ as a tuple $\langle V, A, P, \mathcal{Y}, E \rangle$, where:
  - $V$ is the set of random variables.
  - $A$ is the set of arcs.
  - $P$ is a partial map that represents the control flow among random variables.
  - $\mathcal{Y}$ is a partial map that indicates whether the observe expression is on the control flow path, conditioned on the values of the latent variables.
  - $E$ is a set of vertices.

In the evaluation relation $\mathcal{Y}$, we have the following definitions:

- $\text{Bernoulli}(\mu)$: the probability mass function for the Bernoulli distribution. Similarly, the density function for the normal distribution. Note that the expression $\text{Bernoulli}(\mu)$ is defined using $\text{Bernoulli}(\mu_1.0)$.
- $\text{Normal}(\mu, \sigma)$: the probability density or mass in the graph. For observed random variables, among them. For each random variable in $G$, represent random variables, and arcs dependencies between random variables; (iii) a map $A$ consisting of a deterministic set of vertices to deterministic expressions that specify the probability density or mass function for each random variable; (iv) a partial map $P$ that indicates whether the observe expression is on the control flow path, conditioned on the values of the latent variables.

For observed random variables, $E$ contains a set of vertices $V$. We define a Bayesian network $G(V, A, P, \mathcal{Y}, E)$ as a tuple $\langle V, A, P, \mathcal{Y}, E \rangle$. The vertex set $V$ that represents random variables, and arcs dependencies between random variables; (iii) a map $A$ consisting of a deterministic set of vertices to deterministic expressions that specify the probability density or mass function for each random variable; (iv) a partial map $P$ that indicates whether the observe expression is on the control flow path, conditioned on the values of the latent variables.

The vertex set $V$ that represents random variables, and arcs dependencies between random variables; (iii) a map $A$ consisting of a deterministic set of vertices to deterministic expressions that specify the probability density or mass function for each random variable; (iv) a partial map $P$ that indicates whether the observe expression is on the control flow path, conditioned on the values of the latent variables.

### Example

Here's an example of a Bayesian network:

$$V = \{z, y\}, \quad A = \{(z, y)\}, \quad P = \{z \mapsto (\text{Bernoulli}(z, 0.5)), \quad \text{Bernoulli}(z, 0.5)\}$$

$$P = \{z \mapsto (\text{Bernoulli}(z, 0.5)), \quad \text{Bernoulli}(z, 0.5)\}$$

$$V = \{z, y\}, \quad A = \{(z, y)\}, \quad P = \{z \mapsto (\text{Bernoulli}(z, 0.5)), \quad \text{Bernoulli}(z, 0.5)\}$$

$$\mathcal{Y} = \{y \mapsto 0.5\}$$

$$E = z$$

$$E = z$$

### Program

(let [z (sample (bernoulli 0.5))
       mu (if (= z 0) -1.0 1.0)
       d (normal mu 1.0)
       y 0.5]
  (observe d y)
  z)
(let [z (sample (bernoulli 0.5))
    mu (if (= z 0) -1.0 1.0)
    d (normal mu 1.0)
    y 0.5]
  (observe d y)
  z)

\[ V = \{z, y\}, \]
\[ A = \{(z, y)\}, \]
\[ P = [z \mapsto (p_{bern} z 0.5), \]
\[ y \mapsto (p_{norm} y (if (= z 0) -1.0 1.0) 1.0)] \]
\[ \mathcal{Y} = [y \mapsto 0.5] \]
\[ E = z \]

Program

Mathematic Object

\[
\begin{align*}
\rho, \phi, e_1 \Downarrow & G_1, E_1 \\
(V, A, \mathcal{P}, \mathcal{Y}) & = G_1 \oplus G_2 \\
F_1 & = \text{Score}(E_1, v) \neq \perp \\
Z & = (\text{FreeVars}(F_1) \setminus \{v\}) \cap V \\
B & = \{(z, v) : z \in Z\}
\end{align*}
\]

\[
\begin{align*}
\rho, \phi, e_2 \Downarrow & G_2, E_2 \\
F = (\text{if } \phi F_1 1) \\
\text{FreeVars}(E_2) \cap V & = \emptyset
\end{align*}
\]

\[
\rho, \phi, (\text{observe } e_1 e_2) \Downarrow (V \cup \{v\}, A \cup B, P \oplus [v \mapsto F], \mathcal{Y} \oplus [v \mapsto E_2]), E_2
\]

Big Step Operational Semantics
Intuitive Evaluation Perspective

(defquery example [y]
  (let [x (sample (beta 1 1))] ; f(x)
    (observe (bernoulli x) y) ; g(y|x)
    x))

- Syntactically denotes joint and conditioning

\[
\gamma(x) \triangleq p(x, y) = \prod_{i=1}^{N} g_i(y_i|\phi_i) \prod_{j=1}^{M} f_j(x_j|\theta_j)
\]

- Evaluator characterizes

\[
p(x|y) = \frac{p(x, y)}{p(y)}
\]
Trace Probability

• Defined as (up to a normalization constant)

\[ \gamma(\mathbf{x}) \triangleq p(\mathbf{x}, \mathbf{y}) = \prod_{i=1}^{N} g_i(y_i|\phi_i) \prod_{j=1}^{M} f_j(x_j|\theta_j) \]

• Simple notation hides complex dependency structure!

\[ \gamma(\mathbf{x}) = p(\mathbf{x}, \mathbf{y}) = \prod_{i=1}^{N} \tilde{g}_i(x_{n_i}) \left( y_i \middle| \tilde{\phi}_i(x_{n_i}) \right) \prod_{j=1}^{M} \tilde{f}_j(x_{j-1}) \left( x_j \middle| \tilde{\theta}_j(x_{j-1}) \right) \]
Execution (Trace)-Based Inference

- Sequence of $N$ observe's
  \[
  \{(g_i, \phi_i, y_i)\}_{i=1}^{N}
  \]

- Sequence of $M$ sample's
  \[
  \{(f_j, \theta_j)\}_{j=1}^{M}
  \]

- Sequence of $M$ sampled values
  \[
  \{x_j\}_{j=1}^{M}
  \]

- Conditioned on these sampled values the entire trace is deterministic
Three Base Algorithms

- Likelihood Weighting
  - Importance sampling with prior as proposal
- Metropolis Hastings
- Sequential Monte Carlo
Likelihood Weighting

• Run $K$ independent copies of program simulating from the prior

\[
q(x^k) = \prod_{j=1}^{M^k} f_j(x_j^k | \theta_j^k)
\]

• Accumulate unnormalized weights (likelihoods)

\[
w(x^k) = \frac{\gamma(x^k)}{q(x^k)} = \prod_{i=1}^{N^k} g_i^k(y_i^k | \phi_i^k)
\]

• Use in approximate (Monte Carlo) integration

\[
W^k = \frac{w(x^k)}{\sum_{\ell=1}^{K} w(x^\ell)} \quad \hat{\mathbb{E}}_{\pi}[Q(x)] = \sum_{k=1}^{K} W^k Q(x^k)
\]
Likelihood Weighting

- Run $K$ independent copies of program simulating from the prior

$$q(x^k) = \prod_{j=1}^{M_k} f_j(x^k_j | \theta^k_j)$$

- Accumulate unnormalized weights (likelihoods)

$$w(x^k) = \frac{\gamma(x^k)}{q(x^k)} = \prod_{i=1}^{N_k} g_i^k(y_i^k | \phi_i^k)$$

- Use in approximate (Monte Carlo) integration

$$W_i^k = \frac{w(x^k)}{\sum_{\ell=1}^K w(x^\ell)} \quad \hat{E}_\pi[Q(x)] = \sum_{k=1}^K W_i^k Q(x^k)$$
Likelihood Weighting

• Run $K$ independent copies of program simulating from the prior

$$q(x^k) = \prod_{j=1}^{M^k} f_j(x^k_j | \theta^k_j)$$

• Accumulate unnormalized weights (likelihoods)

$$w(x^k) = \frac{\gamma(x^k)}{q(x^k)} = \prod_{i=1}^{N^k} g_i^k(y_i^k | \phi_i^k)$$

• Use in approximate (Monte Carlo) integration

$$W^k = \frac{w(x^k)}{\sum_{\ell=1}^{K} w(x^\ell)} \quad \hat{\mathbb{E}}_{\pi}[Q(x)] = \sum_{k=1}^{K} W^k Q(x^k)$$
Likelihood Weighting Schematic

\[ z^1, w^1 \]

\[ z^2, w^2 \]

\[ \vdots \]

\[ z^K, w^K \]
Metropolis Hastings = “Single Site” MCMC = LMH

Posterior distribution of execution traces is proportional to trace score with observed values plugged in

\[ \gamma(x) \triangleq p(x, y) = \prod_{i=1}^{N} g_i(y_i | \phi_i) \prod_{j=1}^{M} f_j(x_j | \theta_j) \]

\[ \pi(x) \triangleq p(x|y) = \frac{\gamma(x)}{Z} \]

Metropolis-Hastings acceptance rule

\[ \alpha = \min \left(1, \frac{\pi(x')q(x'|x)}{\pi(x)q(x'|x')} \right) \]

Need proposal

---

Wingate, Stuhlmüller, Goodman “Lightweight Implementations of Probabilistic Programming Languages Via Transformational Compilation” AISTATS 2011
Number of samples in original trace

Probability of new part of proposed execution trace

\[ q(x'|x^s) = \frac{1}{M^s} \kappa(x'_\ell|x^s_\ell) \prod_{j=\ell+1}^{M'} f'_j(x'_j|\theta'_j) \]
LMH Acceptance Ratio

“Single site update” = sample from the prior = run program forward

\[ \kappa(x'_m | x_m) = f_m(x'_m | \theta_m), \theta_m = \theta'_m \]

MH acceptance ratio

Number of sample statements in original trace

\[ \alpha = \min \left( 1, \frac{\gamma(x') M \prod_{j=m}^{M} f_j(x_j | \theta_j)}{\gamma(x) M' \prod_{j=m}^{M'} f'_j(x'_j | \theta'_j)} \right) \]

Number of sample statements in new trace

Probability of original trace continuation restarting proposal trace at m\(^{th}\) sample

Probability of proposal trace continuation restarting original trace at m\(^{th}\) sample
LMH Variants

D. Wingate, A. Stuhlmüller, and N. D. Goodman.
"Lightweight implementations of probabilistic programming languages via transformational compilation." AISTATS (2011).

with continuations: WebPPL Anglican

"C3: Lightweight Incrementalized MCMC for Probabilistic Programs using Continuations and Callsite Caching."
2015: Probabilistic Programming

• Restricted (i.e. STAN, BUGS, infer.NET)
  • Easier inference problems -> fast
  • Impossible for users to denote some models
  • Fixed computation graph
• Unrestricted (i.e. Anglican, WebPPL)
  • Possible for users to denote all models
  • Harder inference problems -> slow
  • Dynamic computation graph

• Fixed, trusted model; one-shot inference
“Bayesian inference is computationally expensive. Even approximate, sampling-based algorithms tend to take many iterations before they produce reasonable answers. In contrast, human recognition of words, objects, and scenes is extremely rapid, often taking only a few hundred milliseconds —only enough time for a single pass from perceptual evidence to deeper interpretation. Yet human perception and cognition are often well-described by probabilistic inference in complex models. How can we reconcile the speed of recognition with the expense of coherent probabilistic inference? How can we build systems, for applications like robotics and medical diagnosis, that exhibit similarly rapid performance at challenging inference tasks?”

Resulting Trend In Probabilistic Programming

Have fully-specified model?

- **Inference?**
  - **Yes**
  - **No**

- **One-shot**
- **Repeated**

- **Probabilistic Programming**
- **Inference Compilation**

- **Unsupervised Deep Learning**
Inference Compilation
**Inference Compilation**

**Input**: an inference problem denoted in a probabilistic programming language

**Output**: a trained inference network (deep neural network “compilation artifact”)

---

Figure 1: Representative output in the polynomial regression example. Plots show 100 samples each at 5% opacity, with the mean marked as a solid dashed line. These are all proposed using the same neural network — not just the same neural network structure, but also identical learned weights. The MCMC posterior is generated by thinning 10000 samples by a factor 100, after 10000 samples of burnin. The neural network proposal density for the weights yields estimated polynomial curves very close to the true posterior solution, albeit slightly more diffuse. Any small mismatch is easily corrected via importance reweighing.

Figure 2: Here we place a Laplace prior on the regression weights, and have Student-t likelihoods, giving us $w_d \sim \text{Laplace}(0, 10^{-1})$ for $d = 0, 1, 2$; $t_n \sim \text{t}_\nu(w_0 + w_1 z_n + w_2 z_n^2, \nu^2)$ for $n = 1, \ldots, N$ for fixed $\nu = 4$, $\nu^2 = 1$, and we place a uniform prior on $(-10, 10)$ for $z_n$. The goal is to estimate the posterior distribution of weights for the constant, linear, and quadratic terms, given any possible collected dataset $\{z_n, t_n\}_{n=1}^N$. In the notation of the surrounding sections, we have latent variables $x \sim \{w_0, w_1, w_2\}$ and observed variables $y \sim \{z_n, t_n\}_{n=1}^N$.

## Table I

<table>
<thead>
<tr>
<th>Type</th>
<th>Baidu (2011)</th>
<th>Baidu (2013)</th>
<th>eBay</th>
<th>Yahoo</th>
<th>reCaptcha</th>
<th>Wikipedia</th>
<th>Facebook</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Our method</strong></td>
<td>RR 99.8%</td>
<td>99.9%</td>
<td>99.2%</td>
<td>98.4%</td>
<td>96.4%</td>
<td>93.6%</td>
<td>91.0%</td>
</tr>
<tr>
<td></td>
<td>BT 72 ms</td>
<td>67 ms</td>
<td>122 ms</td>
<td>106 ms</td>
<td>78 ms</td>
<td>90 ms</td>
<td>90 ms</td>
</tr>
<tr>
<td>Bursztein et al. [15]</td>
<td>RR 38.68%</td>
<td>55.22%</td>
<td>51.39%</td>
<td>5.33%</td>
<td>22.67%</td>
<td>28.29%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BT 3.94 s</td>
<td>1.9 s</td>
<td>2.31 s</td>
<td>7.95 s</td>
<td>4.59 s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Starostenko et al. [16]</td>
<td>RR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>91.5%</td>
<td>54.6%</td>
</tr>
<tr>
<td></td>
<td>BT &lt; 0.5 s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gao et al. [17]</td>
<td>RR 34%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>34%</td>
</tr>
<tr>
<td></td>
<td>BT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gao et al. [18]</td>
<td>RR 51%</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BT 7.58 s</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Goodfellow et al. [6]</td>
<td>RR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>99.8%</td>
</tr>
<tr>
<td>Stark et al. [8]</td>
<td>RR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>90%</td>
</tr>
</tbody>
</table>

### Facebook Captcha

- **Observed images**
  - W4kgvQ
  - uV7FeWB
  - MqhnpT
- **Inference**
  - $10^7$: W4kgvQ, uV7FeWB, MqhnpT
- **Training traces**
  - $10^6$: WA4rjvQ, uV7FeWB, MypppT
  - $10^5$: Woxewed9, mTTEMm, RIpES
  - $10^4$: BKvu2Q, C9QDsoN, rS5FP2B

---

*The full source code of our setup will be released in a public repository by the camera-ready deadline.*

---

*[40M raise]*
I’m Hiring

• Postdoc(s)

• PhD students
New Physics Via ATLAS Simulator Inversion

- Event & detector simulators
- ATLAS detector output

$\mathbf{x}$

$\mathbf{y}$

E.g. Sherpa

E.g. Geant

$\approx 2017$–$2020$
Data-Driven Discovery of Models (D3M)

Mr. Wade Shen