Probabilistic Programming

Frank Wood
fwood@robots.ox.ac.uk

AGI 2015
Inverse Graphics

Captcha Solving

Mesh Fitting


Compromise can assist with this exploration by generating fixation and overcome their unconscious biases. The whole space of creative options seems to help people avoid quality final designs. In this framework, the user provides a model of the design space by expressing her preferences as soft constraints. As color selection and furniture layout are popular for computer-aided suggestion in domains as diverse as interior design, we synthesize examples that their users might never have thought of independently.

1. Introduction

We present a system for generating suggestions from highly-constrained, continuous design spaces. We formulate Stable Static Structures

Stochastically-Ordered Sequential Monte Carlo. In this framework, the user provides a model of the design space by expressing her preferences as soft constraints.

Figure 1: Physical realizations of stable structures generated by our system. To create these structures, we write programs that are used to synthesize examples that their users might never have thought of independently.

<table>
<thead>
<tr>
<th>SOSMC</th>
<th>SMC</th>
<th>MH</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>Score: -20.59</td>
<td>Score: -38.73</td>
<td>Score: -34.07</td>
<td></td>
</tr>
<tr>
<td>Time: 6.99s</td>
<td>Time: 3.39s</td>
<td>Time: 1.14s</td>
<td></td>
</tr>
</tbody>
</table>

(a)

<table>
<thead>
<tr>
<th>SOSMC</th>
<th>SMC</th>
<th>MH</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>Score: -10.38</td>
<td>Score: -17.65</td>
<td>Score: -18.65</td>
<td></td>
</tr>
<tr>
<td>Time: 8.32s</td>
<td>Time: 3.42s</td>
<td>Time: 7.53s</td>
<td></td>
</tr>
</tbody>
</table>

(b)

<table>
<thead>
<tr>
<th>SOSMC</th>
<th>SMC</th>
<th>MH</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>Score: -10.97</td>
<td>Score: -16.74</td>
<td>Score: -15.91</td>
<td></td>
</tr>
<tr>
<td>Time: 6.15s</td>
<td>Time: 8.47s</td>
<td>Time: 10.00s</td>
<td></td>
</tr>
</tbody>
</table>

(c)

Ritchie, D., Lin, S., Goodman, N. D., & Hanrahan, P.

D. Ritchie, B. Mildenhall, N. D. Goodman, & P. Hanrahan.
Probabilistic Program Induction

Yura Perov and Frank Wood.
"Learning Probabilistic Programs."
Policy Learning in POMDPs

Fig. 1: Learned policies for the Canadian traveler problem. Line widths indicate the frequency at which the policy travels each edge, averaged over random combinations of open and closed edges.

5. Case Studies

We demonstrate the proposed policy learning method on three problem domains: (1) the Canadian Traveller Problem, (2) a modified version of the RockSample POMDP, and (3) an optimal diagnosis benchmark inspired by the classic children's game Guess Who. Each of these domains can be formulated as a POMDP. This means that there is some form of unobserved state in the problem instance, and the agent must choose actions based on contextual information $x_t = (u_0, o_1, ..., u_t, o_t)$. Even for discrete problems, the cardinality of the set of possible information states $x_t$ grows exponentially with the horizon $T$.

The aim of these studies is to explore how probabilistic programs can be used to define policies tailored to the structure of each domain. Fundamentally, some information must be discarded when making a decision. Program policies encode our intuition about what information is most relevant in a given context. As such, these studies are not intended to achieve results that are competitive with current state-of-the-art specialized techniques for POMDPs (see Shani et al. [2013] for a recent overview). Rather, we consider probabilistic programs as a concise algorithmic representation of domain-specific probabilistic mappings from information states to actions, in order to describe the search space over policies in terms of a moderate yet not unwieldy number of parameters.

5.1 Evaluation Setup

We use the same experimental setup in each of the three domains. A trial begins with a learning phase, in which BBEM is used to learn the policy hyperparameters, followed by a number of testing episodes in which the agent chooses actions according to a fixed learned policy. At each gradient update step, we use 1000 samples to calculate a gradient estimate. Each testing phase consists of 1000 episodes. All shown results are based on test-phase simulations.

Stochastic gradient methods can be sensitive to the learning rate parameters. Results reported here use a RMSProp style rescaling of the gradient [Hinton et al.], which normalizes the gradient by a discounted rolling decaying average of its magnitude with decay factor $\lambda$. We use a step size schedule $\epsilon_k = \epsilon_0 / (\tilde{\epsilon} + k)$ as reported in [Hoffman et al., 2013], with $\tilde{\epsilon} = 1$, $\epsilon_0 = 0.1$ in all experiments. We use a relatively conservative base learning rate $\epsilon_0 = 0.1$ in all reported experiments. For independent trials performed across a range $1, 2, 5, 10, ..., 1000$ of total gradient steps, consistent convergence was observed in all runs using over 100 gradient steps.

5.2 Canadian Traveller Problem

In the Canadian Traveller Problem [Papadimitriou and Yannakakis, 1991], an undirected graph $G = (V, E)$ is given, along with the cost $w_e$ of traversing every edge $e \in E$, and the probability $p_e$ that the edge is open. The agent must traverse the graph from the initial node $s$ to the goal node $t$ at the lowest possible cost. The agent does not know the state of an edge until it reaches one of the edge's vertices. The problem is NP-hard [Fried et al., 2013], and heuristic online and offline approaches [Eyerich et al., 2010] are used to solve problem instances.

Here we learn a policy based on the depth-first search (DFS) — the agent traverses the graph in the depth-first order until the goal node is reached (only connected instances are considered). Depth-first search

Landscape

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

Inference

Parameters

Program

Output

Parameters

Program

Observations

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

x

y

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?
Increase Programmer Productivity

http://www.robots.ox.ac.uk/~fwood/anglican/examples/viewer/?worksheet=complexityreduction
Latent Dirichlet Allocation (Wood, University of Oxford) is formally written as:

parameters \( \theta \) independently from Mult(

Conditional on the assignment variable \( i \), each of \( \alpha \) is a Dirichlet distribution over the \( K \) dimensions, where \( K \) denotes the number of topics in the generative model for the documents. Each document \( d \) is a mixture of \( K \) topics through the use of a static dictionary. This is denoted as \( z_d \in \{1, \ldots, K\} \), and \( z_d \) is drawn independently from Dir(\( \alpha \)).

The graphical model for the process can be seen in Figure 1. It can also be summarized using a Chinese Restaurant metaphor (see Figure 2).

Model by the following Bayesian network in Figure 1. It can also be summarized using a Chinese Restaurant metaphor (see Figure 2).

Model by the following Bayesian network in Figure 1. It can also be summarized using a Chinese Restaurant metaphor (see Figure 2).

Combining the stationary Pitman-Yor and cluster locations models, we can summarize the full model by the following Bayesian network in Figure 1. It can also be summarized using a Chinese Restaurant metaphor (see Figure 2).

It is possible to define a slightly modified version of our model that is consistent under marginal distributions of the latent allocation variables at time \( t \) and \( t+1 \). This lack of consistency is shared by other models based on the Pitman-Yor process.

This lack of consistency is shared by other models based on the Pitman-Yor process. It is possible to define a slightly modified version of our model that is consistent under marginal distributions of the latent allocation variables at time \( t \) and \( t+1 \).

This lack of consistency is shared by other models based on the Pitman-Yor process. It is possible to define a slightly modified version of our model that is consistent under marginal distributions of the latent allocation variables at time \( t \) and \( t+1 \).

This lack of consistency is shared by other models based on the Pitman-Yor process. It is possible to define a slightly modified version of our model that is consistent under marginal distributions of the latent allocation variables at time \( t \) and \( t+1 \).

This lack of consistency is shared by other models based on the Pitman-Yor process. It is possible to define a slightly modified version of our model that is consistent under marginal distributions of the latent allocation variables at time \( t \) and \( t+1 \).

This lack of consistency is shared by other models based on the Pitman-Yor process. It is possible to define a slightly modified version of our model that is consistent under marginal distributions of the latent allocation variables at time \( t \) and \( t+1 \).

This lack of consistency is shared by other models based on the Pitman-Yor process. It is possible to define a slightly modified version of our model that is consistent under marginal distributions of the latent allocation variables at time \( t \) and \( t+1 \).

This lack of consistency is shared by other models based on the Pitman-Yor process. It is possible to define a slightly modified version of our model that is consistent under marginal distributions of the latent allocation variables at time \( t \) and \( t+1 \).

This lack of consistency is shared by other models based on the Pitman-Yor process. It is possible to define a slightly modified version of our model that is consistent under marginal distributions of the latent allocation variables at time \( t \) and \( t+1 \).

This lack of consistency is shared by other models based on the Pitman-Yor process. It is possible to define a slightly modified version of our model that is consistent under marginal distributions of the latent allocation variables at time \( t \) and \( t+1 \).

This lack of consistency is shared by other models based on the Pitman-Yor process. It is possible to define a slightly modified version of our model that is consistent under marginal distributions of the latent allocation variables at time \( t \) and \( t+1 \).

This lack of consistency is shared by other models based on the Pitman-Yor process. It is possible to define a slightly modified version of our model that is consistent under marginal distributions of the latent allocation variables at time \( t \) and \( t+1 \).

This lack of consistency is shared by other models based on the Pitman-Yor process. It is possible to define a slightly modified version of our model that is consistent under marginal distributions of the latent allocation variables at time \( t \) and \( t+1 \).

This lack of consistency is shared by other models based on the Pitman-Yor process. It is possible to define a slightly modified version of our model that is consistent under marginal distributions of the latent allocation variables at time \( t \) and \( t+1 \).

This lack of consistency is shared by other models based on the Pitman-Yor process. It is possible to define a slightly modified version of our model that is consistent under marginal distributions of the latent allocation variables at time \( t \) and \( t+1 \).

This lack of consistency is shared by other models based on the Pitman-Yor process. It is possible to define a slightly modified version of our model that is consistent under marginal distributions of the latent allocation variables at time \( t \) and \( t+1 \).

This lack of consistency is shared by other models based on the Pitman-Yor process. It is possible to define a slightly modified version of our model that is consistent under marginal distributions of the latent allocation variables at time \( t \) and \( t+1 \).
Anglican

• Higher order, pure functional
• Compiled (CPS -> Clojure -> JVM bytecode)
  • Complete JVM language family interoperability
• First class distributions
• 15+ composable inference algorithms
  • SMC
  • CASCADE
  • PMCMC (PIMH, PGIBBS, PGAS)
  • (Adaptive) LMH
  • …

http://www.robots.ox.ac.uk/~fwood/anglican/
Anglican

• http://www.robots.ox.ac.uk/~fwood/anglican/

• Open source (GPLv3)
  • core: https://bitbucket.org/probprog/anglican
  • user: https://bitbucket.org/probprog/anglican-user
  • tutorial: https://bitbucket.org/probprog/mlss2015
Traditional Bayesian Statistics

\[
\mu \sim \text{Normal}(1, \sqrt{5})
\]

\[
y_i | \mu \sim \text{Normal}(\mu, \sqrt{2})
\]

\[
y_1 = 9, y_2 = 8
\]

\[
\mu | y_{1:2} \sim \text{Normal}(7.25, 0.9129)
\]

\[
(\text{defquery} \ \text{gaussian-model} \ [\text{data}])
\]

\[
(\text{let} \ [\mu (\text{sample} (\text{normal} \ 1 \ (\text{sqrt} \ 5)))])
\]

\[
\sigma (\text{sqrt} \ 2)]
\]

\[
(\text{map} \ (\text{fn} \ [x]) (\text{observe} (\text{normal} \ \mu \ \sigma) \ x)) \ \text{data})
\]

\[
(\text{predict} :\mu \ \mu)))
\]

\[
(\text{def} \ \text{dataset} \ [9 \ 8])
\]

\[
(\text{def} \ \text{posterior}
\]

\[
(\text{conditional} \ \text{gaussian-model}
\]

\[
:pgibbs
\]

\[
:number-of-particles \ 1000) \ \text{dataset})
\]

\[
(\text{def} \ \text{posterior-samples}
\]

\[
(\text{repeatedly} \ 20000 \ #(\text{sample} \ \text{posterior})))
\]

\[
(-10 -8 -6 -4 -2 0 2 4 6 8 10)
\]

\[
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8
\]
(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))

(predict :balls balls)

goal: ~20% of balls in box…
(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)
en-end-world (simulate-world world)
balls (:balls end-world)

;; how many balls entered the box?
num-balls-in-box (balls-in-box end-world)]
(predict :balls balls)
(predict :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 2 0.1)]

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

(predict :balls balls)
(predict :bumper-positions bumper-positions))
Inference

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

\[ p(y|h) = \int p(y|z, h)p(z|h)dz \]
Automatic Complexity Regularization

\[ p(y|h) \]

\[ h = \mathcal{H}_{simple} \]

\[ h = \mathcal{H}_{complex} \]

\[ p(h|y) \propto p(y|h)p(h) \]

Bayesian Occam’s Razor

Probabilistic Programming Is Fully Generative

\[ x = z \cup h \]

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

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>program source code</td>
<td>program output</td>
</tr>
<tr>
<td>scene description</td>
<td>image</td>
</tr>
<tr>
<td>policy</td>
<td>reward</td>
</tr>
<tr>
<td>world</td>
<td>simulator output</td>
</tr>
<tr>
<td>automata</td>
<td>sequence</td>
</tr>
</tbody>
</table>
How Does it Work?
The Gist

- Explore as many “traces” as possible, intelligently
  - Each trace contains all random choices made during the execution of a generative model
- Compute trace “goodness” (probability) as side-effect
- Combine weighted traces probabilistically coherently
- Report projection of posterior over traces
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

$$ g(y_n|x_{1:n}) $$
Traces

(let [x-1-1 3
      x-1-2 (sample (discrete (range x-1-1)))]
  (if (not= x-1-2 1)
      (let [x-2-1 (+ x-1-2 7)]
        (sample (poisson x-2-1))))
(let [x-1-1 3
      x-1-2 (sample (discrete (range x-1-1)))]
  (if (not= x-1-2 1)
    (let [x-2-1 (+ x-1-2 7)]
      (sample (poisson x-2-1))))))
(observe (gaussian x-2-1 0.0001) 7)
Iteratively,
- simulate
- weight
- resample
SMC for Probabilistic Programming

$$p(x_{1:n-1}|y_{1:n-1}) \approx \sum_{\ell=1}^{L} w_{n-1}^{\ell} \delta_{x_{1:n-1}^{\ell}}(x_{1:n-1})$$

$$p(x_{1:n}|y_{1:n}) \propto g(y_n|x_{1:n}) f(x_n|x_{1:n-1}) p(x_{1:n-1}|y_{1:n-1})$$

$$q(x_{1:n}|y_{1:n}) = f(x_n|x_{1:n-1}) p(x_{1:n-1}|y_{1:n-1})$$

$$p(x_{1:n}|y_{1:n}) \approx \sum_{\ell=1}^{L} g(y_n|x_{1:n}^{\ell}) \delta_{x_{1:n}^{\ell}}(x_{1:n}), \quad x_{1:n}^{\ell} = x_n^{\ell} x_{1:n-1}^{a_{n-1}^{\ell}} \sim f$$

Fischer, Kiselyov, and Shan “Purely functional lazy non-deterministic programming” ACM Sigplan 2009
W., van de Meent, and Mansinghka “A New Approach to Probabilistic Programming Inference” AISTATS 2014
Paige and W. “A Compilation Target for Probabilistic Programming Languages” ICML 2014
SMC for Probabilistic Programming

Intuitively:
- run
- wait
- fork

Threads

continuations

observe delimiter
Issues

- Degeneracy
- Not iterable (naively)
PMCMC for Probabilistic Programming

[Wood, van de Meent, Mansinghka “A new approach to probabilistic programming inference” AISTATS 2014]

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

• Iterable SMC
  - PIMH: “particle independent Metropolis-Hastings”
  - PGIBBS: “iterated conditional SMC”

Andrieu, Doucet, Holenstein “Particle Markov chain Monte Carlo methods.” JRSSB 2010
Better Inference Per Unit Energy
PMCMC (and SMC) Methods Only Require

- Initialization (sample)
  \[ p(x_1) \]

- Forward simulation (sample)
  \[ f(x_n | x_{1:n-1}) \]

- Observation likelihood computation
  - pointwise evaluation up to normalization
  \[ g(y_n | x_{1:n}) \]
Stop Making New Probabilistic Programming Languages

sort-of
Probabilistic C

- Standard C with two new directives: observe and predict
- Is compiled to parallel machine code by standard compilers
- Relies on standard operating system functionality: processes, forking, mutexes, shared memory
- Compiled programs automatically do inference
- Emits posterior samples of predicted quantities
Simple example program

Posterior mean of a Gaussian, given i.i.d. draws

**observe** constrains program execution

**predict** emits sampled values

```c
#include "probabilistic.h"

int main(int argc, char **argv) {
    double var = 2;
    double mu = normal_rng(1, 5);
    observe(normal_lnp(9, mu, var));
    observe(normal_lnp(8, mu, var));
    predict("mu,%f\n", mu);
    return 0;
}
```

mean, 8.013323
mean, 8.013323
mean, 6.132654
mean, 7.229289
mean, 7.027069
mean, 7.194609
mean, 7.194609
mean, 5.218672
mean, 6.184513
A Markov model

\[ z_0 \sim \text{Discrete}([1/K, \ldots, 1/K]) \quad z_n|z_{n-1} \sim \text{Discrete}(T_{z_{n-1}}) \]

```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;
} 
```
Conditioning on observed data

\[ z_0 \sim \text{Discrete}([1/K, \ldots, 1/K]) \quad z_n | z_{n-1} \sim \text{Discrete}(T_{z_{n-1}}) \quad y_n | z_n \sim \text{Normal}(\mu_{z_n}, \sigma^2) \]
Changing the generative model is easy
Suppose the transition matrix were 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);
```

```
Implementation

• Inference: forward simulation (SMC, particle MCMC, particle cascade, …)

• POSIX **fork:**
  - operating-system level call to clone a running process: branch on program execution state, explore many downstream paths
  - duplicates *entire* memory address space
  - efficient: lazy copy-on-write behaviour
  - parallel: each downstream path is explored by an independent OS process
The Next 700 Probabilistic Programming Languages?

W., Jeffrey Mark Siskind and Brooks Paige
(in prep. 2015)
Probabilistic Scheme

Gaussian example, in probabilistic scheme

;;; Define a (random) mean, mu
(define mu (normal 1 (sqrt 5)))
(define sigma (sqrt 2))

;;; Define a likelihood function
(define (likelihood x) (normal-lnp x mu sigma))

;;; Condition on the data
(define data (list 8 9))
(map observe-lnp (map likelihood data))

;;; Emit samples of the mean
(predict-float "mean" mu)
All we need for probabilistic scheme

• existing scheme compiler (i.e. STALIN)

• existing C compiler (i.e. GCC, clang)

;;; ERPs
(define poisson-rng
  (foreign-procedure (double) long "poisson_rng" "probabilistic"))

(define normal-lnp
  (foreign-procedure (double double double) double "normal_lnp" "probabilistic"))
;;; plus more -rng and -lnp

;;; directives
(define observe (foreign-procedure (double) void "observe" "probabilistic")

(define predict-value
  (foreign-procedure (char* double) void "predict_value" "probabilistic"))

;;; necessary boilerplate
(vector-ref argv 0)
#include <probabilistic.h>

int main(int argc, char *argv[]) {
    long a = poisson_rng(100.0) - 100;
    long b = poisson_rng(100.0) - 100;
    observe(normal_lnp(7.0, (double)(a+b), 0.00001));
    predict_value("a", (double)a);
    predict_value("b", (double)b);
}
C (GCC, CLANG)

```c
int main(int argc, char *argv[]) { 
    long a = poisson_rng(100.0) - 100;
    long b = poisson_rng(100.0) - 100;
    observe(normal_lnp(7.0, 
        (double)(a+b), 0.00001));
    predict_value("a", (double)a);
    predict_value("b", (double)b);
}
```

Scheme (STALIN)

```scheme
(define a (- (poisson-rng 100.0) 100))
(define b (- (poisson-rng 100.0) 100))
(observe (normal-lnp 7.0
    (exact->inexact (+ a b)) .00001))
(predict-value "a" (exact->inexact a))
(predict-value "b" (exact->inexact b))
```

Standard ML (MLTON)

```ml
val a = (poisson_rng 100.0)-100
val b = (poisson_rng 100.0)-100
val _ = observe (normal_lnp 7.0,
    (int64ToReal (a+b)), 0.00001))
val _ = predict_value ("a", (int64ToReal a))
val _ = predict_value ("b", (int64ToReal b))
val _ = return_from_main 0
```

Haskell (GHC)

```haskell
model = do
    a <- (+(-100)) <$> poisson_rng 100.0
    b <- (+(-100)) <$> poisson_rng 100.0
    observe $ normal_lnp 7
    (realToFrac (a+b)) 0.00001
    predict_value "a" (realToFrac a)
    predict_value "b" (realToFrac b)
    return ()
```
Figure 2: Four input images from our CAPTCHA corpus, along with the final results exhibiting extreme letter overlap. The second row is a CAPTCHA from TurboTax, the third row on some runs. Our probabilistic graphics program did not originally support rotation, which was needed for the AOL CAPTCHAs; adding it required only 1 additional line of probabilistic code. See Figure 2 for representative inputs and outputs. In this program, $P_{\text{inferred}} = 1$ for every input image.

The stochastic likelihood model is a multivariate Gaussian whose mean is the blurry rendering; formally, $\mathbf{x}_i \sim \mathcal{N}(\mathbf{y}_i, \Sigma_i)$. The control variables experimented with enabling enumerative (griddy) Gibbs sampling for uniform discrete variables $\{1, 2\}$, and the standard deviation of the Gaussian likelihood was $\sigma = 0.5$. The control variables also used in $[35, 47]$. We use a step size $\alpha = 0.001$.

For quantitative metrics, refer to section 4.1. Effects can be combined much more richly than is typical for non-parametric prior models. We use a step size $\alpha = 0.001$ as reported in [Hoffman et al., 2013], with annealing (e.g. global or local blur variables in GPGP $[\ldots]$).

In this section, we will explain the essential architectural features of GPGP, compilation) instead of interpreting, making program execution outputs an image of laser-scanned faces from $[\ldots]$. The program $I$ is an imperative programming language, where expressions can take on either deterministic or stochastic values. The program $I$ defines a stochastic procedure that can be described in terms of a hierarchy of abstract features. The program $I$ Defines a structured noise process over the rendering and the set of all random choices $\mathbf{g}$. The program $I$ uses general-purpose graphics simulators (Blender\$\ldots\$, and uses general-purpose graphics simulators (Blender\$\ldots\$, which is a general method of trans-compilation) instead of interpreting, making program execution outputs an image

Program policies encode our intuition about what information is most relevant for making a decision. Program policies encode our intuition about what information is most relevant for making a decision. Program policies encode our intuition about what information is most relevant for making a decision. Program policies encode our intuition about what information is most relevant for making a decision. Program policies encode our intuition about what information is most relevant for making a decision.

We demonstrate the proposed policy learning method on three problem domains: (1) the Canadian Traveller Problem, (2) a modified version of the RockSample POMDP, and (3) an optimal diagnosis of novel lighting effects can be combined much more richly than is typical for non-parametric prior models. We use a step size $\alpha = 0.001$ as reported in [Hoffman et al., 2013], with annealing (e.g. global or local blur variables in GPGP $[\ldots]$).

In this section, we will explain the essential architectural features of GPGP, compilation) instead of interpreting, making program execution outputs an image

Program policies encode our intuition about what information is most relevant for making a decision. Program policies encode our intuition about what information is most relevant for making a decision. Program policies encode our intuition about what information is most relevant for making a decision. Program policies encode our intuition about what information is most relevant for making a decision.

We demonstrate the proposed policy learning method on three problem domains: (1) the Canadian Traveller Problem, (2) a modified version of the RockSample POMDP, and (3) an optimal diagnosis of novel lighting effects can be combined much more richly than is typical for non-parametric prior models. We use a step size $\alpha = 0.001$ as reported in [Hoffman et al., 2013], with annealing (e.g. global or local blur variables in GPGP $[\ldots]$).

In this section, we will explain the essential architectural features of GPGP, compilation) instead of interpreting, making program execution outputs an image

Program policies encode our intuition about what information is most relevant for making a decision. Program policies encode our intuition about what information is most relevant for making a decision. Program policies encode our intuition about what information is most relevant for making a decision. Program policies encode our intuition about what information is most relevant for making a decision. Program policies encode our intuition about what information is most relevant for making a decision.

We demonstrate the proposed policy learning method on three problem domains: (1) the Canadian Traveller Problem, (2) a modified version of the RockSample POMDP, and (3) an optimal diagnosis of novel lighting effects can be combined much more richly than is typical for non-parametric prior models. We use a step size $\alpha = 0.001$ as reported in [Hoffman et al., 2013], with annealing (e.g. global or local blur variables in GPGP $[\ldots]$).

In this section, we will explain the essential architectural features of GPGP, compilation) instead of interpreting, making program execution outputs an image

Program policies encode our intuition about what information is most relevant for making a decision. Program policies encode our intuition about what information is most relevant for making a decision. Program policies encode our intuition about what information is most relevant for making a decision. Program policies encode our intuition about what information is most relevant for making a decision. Program policies encode our intuition about what information is most relevant for making a decision.

We demonstrate the proposed policy learning method on three problem domains: (1) the Canadian Traveller Problem, (2) a modified version of the RockSample POMDP, and (3) an optimal diagnosis of novel lighting effects can be combined much more richly than is typical for non-parametric prior models. We use a step size $\alpha = 0.001$ as reported in [Hoffman et al., 2013], with annealing (e.g. global or local blur variables in GPGP $[\ldots]$).

In this section, we will explain the essential architectural features of GPGP, compilation) instead of interpreting, making program execution outputs an image

Program policies encode our intuition about what information is most relevant for making a decision. Program policies encode our intuition about what information is most relevant for making a decision. Program policies encode our intuition about what information is most relevant for making a decision. Program policies encode our intuition about what information is most relevant for making a decision. Program policies encode our intuition about what information is most relevant for making a decision.

We demonstrate the proposed policy learning method on three problem domains: (1) the Canadian Traveller Problem, (2) a modified version of the RockSample POMDP, and (3) an optimal diagnosis of novel lighting effects can be combined much more richly than is typical for non-parametric prior models. We use a step size $\alpha = 0.001$ as reported in [Hoffman et al., 2013], with annealing (e.g. global or local blur variables in GPGP $[\ldots]$).

In this section, we will explain the essential architectural features of GPGP, compilation) instead of interpreting, making program execution outputs an image

Program policies encode our intuition about what information is most relevant for making a decision. Program policies encode our intuition about what information is most relevant for making a decision. Program policies encode our intuition about what information is most relevant for making a decision. Program policies encode our intuition about what information is most relevant for making a decision. Program policies encode our intuition about what information is most relevant for making a decision.

We demonstrate the proposed policy learning method on three problem domains: (1) the Canadian Traveller Problem, (2) a modified version of the RockSample POMDP, and (3) an optimal diagnosis of novel lighting effects can be combined much more richly than is typical for non-parametric prior models. We use a step size $\alpha = 0.001$ as reported in [Hoffman et al., 2013], with annealing (e.g. global or local blur variables in GPGP $[\ldots]$).

In this section, we will explain the essential architectural features of GPGP, compilation) instead of interpreting, making program execution outputs an image

Program policies encode our intuition about what information is most relevant for making a decision. Program policies encode our intuition about what information is most relevant for making a decision. Program policies encode our intuition about what information is most relevant for making a decision. Program policies encode our intuition about what information is most relevant for making a decision. Program policies encode our intuition about what information is most relevant for making a decision.

We demonstrate the proposed policy learning method on three problem domains: (1) the Canadian Traveller Problem, (2) a modified version of the RockSample POMDP, and (3) an optimal diagnosis of novel lighting effects can be combined much more richly than is typical for non-parametric prior models. We use a step size $\alpha = 0.001$ as reported in [Hoffman et al., 2013], with annealing (e.g. global or local blur variables in GPGP $[\ldots]$).

In this section, we will explain the essential architectural features of GPGP, compilation) instead of interpreting, making program execution outputs an image

Program policies encode our intuition about what information is most relevant for making a decision. Program policies encode our intuition about what information is most relevant for making a decision. Program policies encode our intuition about what information is most relevant for making a decision. Program policies encode our intuition about what information is most relevant for making a decision. Program policies encode our intuition about what information is most relevant for making a decision.

We demonstrate the proposed policy learning method on three problem domains: (1) the Canadian Traveller Problem, (2) a modified version of the RockSample POMDP, and (3) an optimal diagnosis of novel lighting effects can be combined much more richly than is typical for non-parametric prior models. We use a step size $\alpha = 0.001$ as reported in [Hoffman et al., 2013], with annealing (e.g. global or local blur variables in GPGP $[\ldots]$).

In this section, we will explain the essential architectural features of GPGP, compilation) instead of interpreting, making program execution outputs an image
It’s All About Inference

- **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**
  

- **Novel proposals**
  
  “Neural Adaptive Inference for Probabilistic Programming” [Paige, W.; in submission]
Thank You

• Questions?

• Funding: DARPA, Amazon, Microsoft