Tutorial on Probabilistic Programming in Machine Learning

Frank Wood
Play Along

1. Download and install Leiningen
   • [http://leiningen.org/](http://leiningen.org/)

2. Fork and clone the Anglican Examples repository
   • git@bitbucket.org:fwood/anglican-examples.git
   • https://fwood@bitbucket.org/fwood/anglican-examples.git

3. Enter repository directory and type
   • “lein gorilla”

4. Open local URL in browser

5. _OR_ [http://www.robots.ox.ac.uk/~fwood/anglican/examples/index.html](http://www.robots.ox.ac.uk/~fwood/anglican/examples/index.html)
Motivation and Background

AI
Unsupervised Automata Induction

Probabilistic Deterministic Infinite Automata (PDIA)
- World states learned
- Per-state emissions learned
- Per-state deterministic transition functions learned
- Infinite-state limit of model to right
- Unsupervised PDFA structure learning biases towards compact (few states) world-models that are fast approximate predictors

Problem
- ~4000 lines of Java code
- New student re-implementation
  - 6-12 months
Unsupervised Tracking

Dependent Dirichlet Process Mixture of Objects

- World state $\approx$ dependent infinite mixture of motile objects that may appear, disappear, and occlude
- Per-state, per-object emission model learned
- Per-state, per-object complex transition functions learned

Problem

- $\sim$5000 lines of Matlab code
- Implementation
  - $\sim$ 1 year
- Generative model
  - $\sim$1 page latex math
- Inference derivation
  - $\sim$3 pages latex math

Neiswanger, Wood, and Xing The Dependent Dirichlet Process Mixture of Objects for Detection-free Tracking and Object Modeling, AISTATS, 2014
Outline

• Supervised modeling and maximum likelihood aren’t the whole ML story

• Exact inference is great when possible but too restrictive to be the sole focus

• Continuous variables are essential to machine learning

• Unsupervised modeling is a growing and important challenge

• We can write (and actually perform inference) in some pretty advanced models using existing probabilistic programming systems now

• Inference is key, and general purpose inference for probabilistic programming is hard, but there is room for improvement and there is a big role for the PL community to play
Questions

• Is probabilistic programming really just about tool and library building for ML applications?

• Are gradients so important that we should just use HMC or stochastic gradients and neural nets for everything (re: neural Turing machine) and give up on anything resembling combinatorial search. As a corollary, for anything that does reduce to combinatorial search should we just compile to SAT solvers?

• Is it more important for purposes of language aesthetic and programmer convenience to syntactically allow constraint/equality observations than to minimize the risk of programmers writing "impossible" inference problems by forcing them to specify a "noisy" observation likelihood?

• Is it possible to develop program transformations that succeed and fail in sufficiently intuitive ways to give programmers the facility of equality constraints when appropriate while not allowing them to make measure zero observations of random variables whose support is “dangerously” complex (uncountable, high-dimensional, etc)?

• How possible and important is it to identify program fragments in which exact inference can be performed? Is it possible that the most important probabilistic programming output will be static analysis tools that identify arbitrary programs that identify fragments of models in which exact inference can be performed and then also transform them into an amenable form for running exact inference?

• Are first-class distributions a requirement? Should they be hidden from the user?

• What is “query”? What should we name it? What should it mean? We've just called "query" "distribution" because rejection-query is, in Church, just an exact sampler from a conditional distribution, vs. mh-query which is different. Can’t we unify them all and simply call them "query" if you don’t have first class distributions or "distribution" if you do and distributions support a sample interface?

• Should we expose the “internals” of the inference mechanism to the end user? Is the ability to draw a single unweighted approximate sample enough? Is a set of converging weighted samples enough? Do we need exact samples? Do we want an evidence approximation? Or converging expectation computations? Are compositions of the latter meaningful in any theoretically interpretable way?

• What is the “deliverable” from inference? How do we amortize inference across queries/tasks? Is probabilistic program compilation Rao-Blackwellization or transfer learning or both?

• Is probabilistic programming just an efficient mechanism or means for searching for models in which approximate inference is sufficiently
Outline

• Supervised modeling and maximum likelihood aren’t the whole ML story

• Exact inference is great when possible but too restrictive to be the sole focus

• Continuous variables are essential to machine learning

• Unsupervised modeling is a growing and important challenge

• We can write (and actually perform inference) in some pretty advanced models using existing probabilistic programming systems now

• Inference is key, and general purpose inference for probabilistic programming is hard, but there is room for improvement and there is a big role for the PL community to play
The Maximum Likelihood Principle

\[ \mathcal{L}(X, y, \theta) = \prod_{n} p(y_n|x_n, \theta) \]

\[ \arg \max_{\theta} \mathcal{L}(X, y, \theta) = \arg \max_{\theta} \sum_{n} \log p(y_n|x_n, \theta) \]

\[ = \theta^* \text{ s.t. } \nabla_{\theta} \sum_{n} \log p(y_n|x_n, \theta^*) = 0 \]

(defn sgd [X y f' theta-init num-iters stepsize]
(loop [theta theta-init
        num-iters num-iters]
  (if (= num-iters 0)
    theta
    (let [n (rand-int (count y))
          gradient (f' (get X n) (get y n) theta)
          theta (map - theta
                  (map #(* stepsize %) gradient))]
      (recur w (dec num-iters))))))
Reasonable Analogy?

Automatic Differentiation  Supervised Learning  Probabilistic Programming  Unsupervised Learning
Example: Logistic Regression

\[ p(y_n = 1|x_n, w) = \frac{1}{1 + \exp\{-w_0 - \sum_d w_d x_{nd}\}} \]

<table>
<thead>
<tr>
<th>( x )</th>
<th>( y )</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.0</td>
<td>2.0</td>
</tr>
<tr>
<td>3.5</td>
<td>1.0</td>
</tr>
<tr>
<td>Iris-versicolor</td>
<td></td>
</tr>
<tr>
<td>6.0</td>
<td>2.2</td>
</tr>
<tr>
<td>4.5</td>
<td>1.5</td>
</tr>
<tr>
<td>Iris-versicolor</td>
<td></td>
</tr>
<tr>
<td>6.2</td>
<td>2.2</td>
</tr>
<tr>
<td>4.5</td>
<td>1.5</td>
</tr>
<tr>
<td>Iris-versicolor</td>
<td></td>
</tr>
<tr>
<td>6.0</td>
<td>2.2</td>
</tr>
<tr>
<td>5.0</td>
<td>1.5</td>
</tr>
<tr>
<td>Iris-virginica</td>
<td></td>
</tr>
<tr>
<td>4.5</td>
<td>2.3</td>
</tr>
<tr>
<td>1.3</td>
<td>0.3</td>
</tr>
<tr>
<td>Iris-setosa</td>
<td></td>
</tr>
<tr>
<td>5.0</td>
<td>2.3</td>
</tr>
<tr>
<td>3.3</td>
<td>1.0</td>
</tr>
<tr>
<td>Iris-versicolor</td>
<td></td>
</tr>
</tbody>
</table>

DEPARTMENT)OF)ENGINEERING)SCIENCE)
Parks!Road,!Oxford!OX1!3PJ!
Telephone:!01865!273000!
Fax:!01865!273010!
Logistic Regression Maximum Likelihood Code

Sincerely,

Frank Wood
University Lecturer, Department of (Information) Engineering Science
Fellow, Kellogg College
http://www.robots.ox.ac.uk/~fwood
Pros vs. Cons

• Cons
  • Lack of convexity requires multiple restarts
  • Brittle
  • Lacking uncertainty quantification, point estimates aren’t naturally composable

• Pros
  • Fast
  • Learned parameter value is “compiled deliverable”
Questions

• Is probabilistic programming really just tool and library building for ML applications?

• Are gradients so important that we should just use differentiable models with stochastic gradients for everything (re: neural Turing machine, OpenDR, Picture, Stan) and give up on anything resembling combinatorial search?
Outline

• Supervised modeling and maximum likelihood aren’t the whole ML story

• Exact inference is great when possible but too restrictive to be the sole focus

• Continuous variables are essential to machine learning

• Unsupervised modeling is a growing and important challenge

• We can write (and actually perform inference) in some pretty advanced models using existing probabilistic programming systems now

• Inference is key, and general purpose inference for probabilistic programming is hard, but there is room for improvement and there is a big role for the PL community to play
Exact inference; pros vs. cons

• Pros
  • Exact probability computation; no worries about convergence or approximation
  • Uncertainty calculus allows principled “model” composition
  • Provides "deliverable" in terms of a complete characterization of a (usually) low-dimensional conditional probability distribution.

• Cons
  • Restrictive: Only possible in a small subset of models
  • Costly: Exponential in tree-width
Outline

• Supervised modeling and maximum likelihood aren’t the whole ML story

• Exact inference is great when possible but too restrictive to be the sole focus

• Continuous variables are essential to machine learning

• Unsupervised modeling is a growing and important challenge

• We can write (and actually perform inference) in some pretty advanced models using existing probabilistic programming systems now

• Inference is key, and general purpose inference for probabilistic programming is hard, but there is room for improvement and there is a big role for the PL community to play
Support for continuous variables is essential for ML

\begin{verbatim}
(defn 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))))
\end{verbatim}
LDS / Kalman Smoother

\[
p(z_n|z_{n-1}) = \mathcal{N}(z_n|Az_{n-1}, \Gamma) \\
p(x_n|z_n) = \mathcal{N}(x_n|Cz_n, \Sigma).
\]

\[
p(x_1, \ldots, x_N, z_1, \ldots, z_N) = p(z_1) \left[ \prod_{n=2}^{N} p(z_n|z_{n-1}) \right] \prod_{n=1}^{N} p(x_n|z_n)
\]

Kalman Smoother Example Code
Questions

• How possible and important is it to identify program fragments in which exact inference can be performed? Is it possible that the most important probabilistic programming output will be static analysis tools that identify fragments of programs/models in which exact inference can be performed and then also transform them into an amenable form for running exact inference, in effect Rao-Blackwellizing by algorithmic or symbolic integration and hiding of latent variables from inference algorithms?

• Is it worth the effort to service (via such analyses, compilation, etc.) the few special cases in which exact inference can be performed?
Related Aside

• While discrete variables can be always be grounded exactly, i.e. constrained; continuous variables often cannot.
The Allure of 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))))
Equality $\leftrightarrow$ Dirac Observe $\leftrightarrow$ Constraint

```
(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)))))
```

\[ p(x | y = o) \propto \delta(y - o)p(x, y) \]

\[ = p(x, y = o) \]
(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 (normal 0.0 tolerance) (d wet-grass true))
    (predict :s (hash-map :is-cloudy is-cloudy :is-raining is-raining :sprinkler sprinkler))))

\[ p(x|y = o) \propto p(d(y, o))p(x, y) \]
(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))))

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

• Is it more important for purposes of language aesthetic and programmer convenience to syntactically allow constraint/equality observations than to minimize the risk of programmers writing "impossible" inference problems by forcing them to specify a “noisy” observation likelihood?

• Is it possible to develop program transformations that succeed and fail in sufficiently intuitive ways to give programmers the facility of equality constraints when appropriate while not allowing them to make measure-zero observations of random variables whose support is “dangerously” complex (uncountable, high-dimensional, etc)?
Outline

• Supervised modeling and maximum likelihood aren’t the whole ML story

• Exact inference is great when possible but too restrictive to be the sole focus

• Continuous variables are essential to machine learning

• Unsupervised modeling is a growing and important challenge

• We can write (and actually perform inference) in some pretty advanced models using existing probabilistic programming systems now

• Inference is key, and general purpose inference for probabilistic programming is hard, but there is room for improvement and there is a big role for the PL community to play
Unsupervised modeling and Approximate inference

• Named by Yann LeCun @DALI as *the* most important area of research in ML

  • He probably meant deep-variational-autoencoders, but still…

• It’s what humans do; it’s what an AI must do

• Canonical pedagogical example : GMM
“Bayesian” GMM Review

\[ \pi \sim \text{Dirichlet}(\alpha) \]

\[ \Lambda_k \sim \text{Wishart}(\Lambda_0, \nu) \]

\[ \mu_k \mid \Lambda_k \sim \text{Normal}(0, (\beta \Lambda_k)^{-1}) \]

\[ z_n \mid \pi \sim \text{Categorical}(\pi) \]

\[ x_n \mid z_n = k, \mu_k, \Lambda_k \sim \text{Normal}(\mu_k, \Lambda_k^{-1}) \]

Gaussian Mixture Model Example Code
Outline

• Supervised modeling and maximum likelihood aren’t the whole ML story

• Exact inference is great when possible but too restrictive to be the sole focus

• Continuous variables are essential to machine learning

• Unsupervised modeling is a growing and important challenge

• We can write (and actually perform inference) in some pretty advanced models using existing probabilistic programming systems now

• Inference is key, and general purpose inference for probabilistic programming is hard, but there is room for improvement and there is a big role for the PL community to play
Advanced Models

- Dirichlet Process Mixture Model Example Code
- Unsupervised Hierarchical Clustering via the Hierarchical Dirichlet Process Example Code
- Automata Learning via the Probabilistic Deterministic Infinite Automata Example Code
- More; policy learning via inference, planning as MAP search, etc. (offline)
Random probability measure with Dirichlet marginals

\[(\theta(A_1), \ldots, \theta(A_k)) \sim \text{Dirichlet}(\alpha H(A_1), \ldots, \alpha H(A_k))\]

for base measure \(H\) concentration \(\alpha\) and all partitions \(A_1, \ldots, A_k\)
Constructive Definition

(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 l)))))

(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))))
Questions

• Model prototyping <-> deployment. Where will we land?
Good News

• Complexity reduction success

• New probabilistic-programming-compatible approaches to inference are possible and under development
Increased Productivity

\[
\begin{align*}
\text{Lines of Anglican Code} &= \logb \left( 1.04 \times (1 + x) \right) \\
\text{Lines of Matlab/Java Code} &= p(\cdot | \text{data})
\end{align*}
\]

Complexity Reduction Example Code
Supervised modeling and maximum likelihood aren’t the whole ML story

Exact inference is great when possible but too restrictive to be the sole focus

Continuous variables are essential to machine learning

Unsupervised modeling is a growing and important challenge

We can write (and actually perform inference) in some pretty advanced models using existing probabilistic programming systems now

Inference is key, and general purpose inference for probabilistic programming is hard, but there is room for improvement and there is a big role for the PL community to play
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

\[ g(y_n|x_{1:n}) \]
SMC

Iteratively,

- simulate
- weight
- resample
SMC for Probabilistic Programming

Intuitively
- run
- wait
- fork

Threads

continuations

observe delimiter
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

Asynchronously

- simulate
- weight
- branch

Paige, W., Doucet, Teh; NIPS 2014
Particle Cascade
The particle cascade provides an \textbf{unbiased estimator} of the \textbf{marginal likelihood}, whose variance decreases proportionally to the number of initial particles $K_0$:

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

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

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

$$\nabla[\hat{p}(y_{0:n})] < \frac{a_n}{K_0}$$
Results: Mean Squared Error

Suppose we wish to compute the posterior expectation of a function $\psi(x_{0:n})$:

$$\mathbb{E}[\psi(x_{0:n})] \approx \sum_{k} w_{n}^{k} \psi \left( x_{0:n}^{(k)} \right)$$

Under mild conditions, the mean squared error of this estimator is bounded, and also decreases as $1/K_0$.

**Theorem:** For any $n \geq 0$, there exists a constant $a_n < \infty$ such that for any $K_0 \geq 1$ and bounded function $\psi$,

$$\mathbb{E} \left[ \left\{ \left( \sum_{k=1}^{K_n} w_{n}^{k} \psi \left( x_{0:n}^{(k)} \right) \right) - \int p(dx_{0:n}|y_{0:n}) \psi(x_{0:n}) \right\}^2 \right] \leq \frac{\bar{a}_n}{K_0} \| \psi \|^2.$$
Scalability: Multiple Cores

- More cores == faster inference
- Scales to multiple cores more efficiently than other particle-based methods
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]
Questions

• What is "query"? What should we name it? What should it mean? Is "query" just a "distribution" constructor? In Church rejection-query is an exact single sampler from a conditional distribution whereas mh-query is different. Can't we unify them all and simply call them "query" if you don't have first class distributions or "distribution" if you do and distributions support a sample interface? (Andreas said yes in the hall yesterday)

• Should we expose the "internals" of a query to the end user? Is the ability to draw a single unweighted approximate sample enough? Is a set of converging weighted samples better? Do we want an evidence approximation? Or converging expectation computations?

• Are first-class distributions a requirement? Should they be hidden from the user?

Bayes Net Example Code
Questions

• What is the "deliverable" from inference?

• How do we amortize inference across queries/tasks?

• How do we define probabilistic program compilation: automatic Rao-Blackwellization or transfer learning or somehow both?
Bubble Up

Applications

Models

Probabilistic Programming Language

Probabilistic Programming System

Inference
Thank You

• David Tolpin (lead architect Anglican)
• Brooks Paige (workbooks)
• Jan Willem van de Meent (workbooks)
• Yura Perov  (workbooks)

• Chris Bishop (figures from PRML)
• Yee Whye Teh (figures and text from BNP tutorials)

• Funding: DARPA, Amazon, Microsoft
In loving memory

Mark Wood

1960-2015
Discrete RV’s Only

2000

IBAL

Prism

2010

HANSAI

Figaro

Problog

Blog

KMP

Discrete RV’s Only