Introduction to probabilistic programming

David Duvenaud and James Lloyd

University of Cambridge

Thanks to
Daniel M Roy (Cambridge)

How to write a Bayesian modeling paper

1. Write down a generative model in an afternoon

2. Get 2 grad students to implement inference for a month

3. Use technical details of inference to pad half of the paper

Can we do better?

Example: Graphical Models

Application Papers

1. Write down a graphical model
2. Perform inference using general-purpose software
3. Apply to some new problem

Inference papers

1. Identify common structures in graphical models (e.g. chains)
2. Develop efficient inference method
3. Implement in a general-purpose software package

Modeling and inference have been disentangled

Not all models are graphical models

What is the largest class of models available?

Probabilistic Programs

- A probabilistic program (PP) is any program that can depend on random choices. Can be written in any language that has a random number generator.
- You can specify any computable prior by simply writing down a PP that generates samples.
- A probabilistic program implicitly defines a distribution over its output.

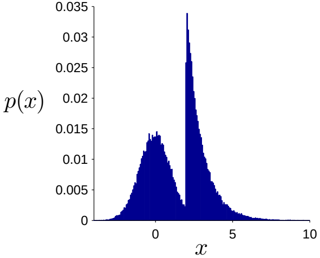
An Example Probabilistic Program

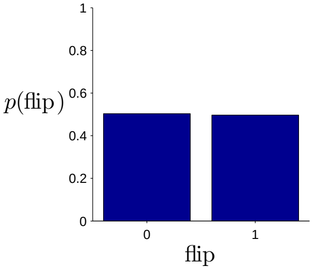
```
flip = rand < 0.5
if flip
    x = randg + 2     % Random draw from Gamma(1,1)
else
    x = randn         % Random draw from standard Normal
end
```

An Example Probabilistic Program

```
flip = rand < 0.5
if flip
    x = randg + 2    % Random draw from Gamma(1,1)
else
    x = randn        % Random draw from standard Normal
end
```

Implied distributions over variables

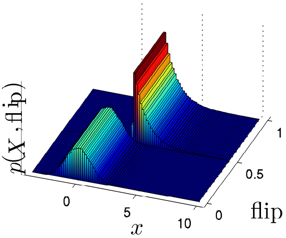


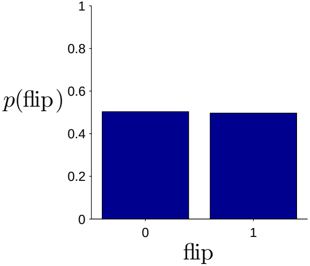


An Example Probabilistic Program

```
flip = rand < 0.5
if flip
    x = randg + 2    % Random draw from Gamma(1,1)
else
    x = randn        % Random draw from standard Normal
end
```

Implied distributions over variables





Probabilistic Programming: Conditioning

Once we've defined a prior, what can we do with it?

The stochastic program defines joint distribution $P(D, N, H)$

- D to be the subset of variables we observe (condition on)
- H the set of variables we're interested in
- N the set of variables that we're not interested in, (so we'll integrate them out).

We want to know about $P(H|D)$

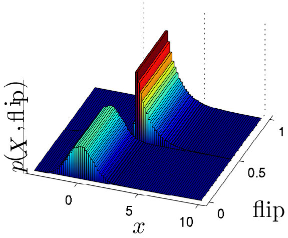
Probabilistic Programming

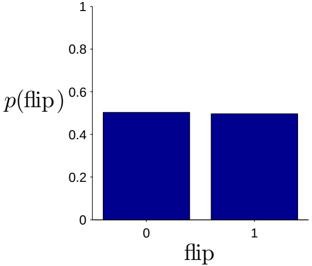
- Usually refers to doing conditional inference when a probabilistic program specifies your prior.
- Could also be described as automated inference given a model specified by a generative procedure.

An Example Probabilistic Program: Conditioning

```
flip = rand < 0.5
if flip
    x = randg + 2    % Random draw from Gamma(1,1)
else
    x = randn        % Random draw from standard Normal
end
```

Implied distributions over variables

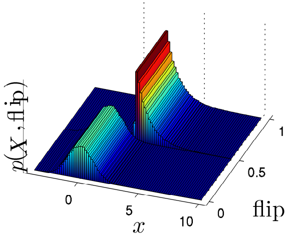


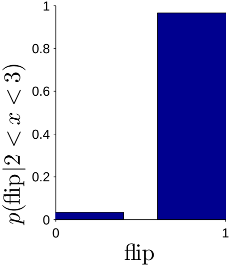


An Example Probabilistic Program: Conditioning

```
flip = rand < 0.5
if flip
    x = randg + 2    % Random draw from Gamma(1,1)
else
    x = randn        % Random draw from standard Normal
end
```

Implied distributions over variables





Can we develop generic inference for all PPs?

Rejection sampling

1. Run the program with a fresh source of random numbers
2. If condition D is true, record H as a sample, else ignore the sample
3. Repeat

Example

```
flip = rand < 0.5
if flip
    x = randg + 2
else
    x = randn
end
```

This produces samples over the execution trace

e.g.

CAN WE DEVELOP GENERIC INFERENCE FOR ALL PPS?

Rejection sampling

1. Run the program with a fresh source of random numbers
2. If condition D is true, record H as a sample, else ignore the sample
3. Repeat

Example

```
flip = rand < 0.5
if flip
    x = randg + 2
else
    x = randn
end
```

```
»True
```

This produces samples over the execution trace

e.g.

CAN WE DEVELOP GENERIC INFERENCE FOR ALL PPS?

Rejection sampling

1. Run the program with a fresh source of random numbers
2. If condition D is true, record H as a sample, else ignore the sample
3. Repeat

Example

```
flip = rand < 0.5
if flip
    x = randg + 2
else
    x = randn
end
```

```
»True

»2.7
```

This produces samples over the execution trace

e.g. `(True, 2.7)`,

Rejection sampling

1. Run the program with a fresh source of random numbers
2. If condition D is true, record H as a sample, else ignore the sample
3. Repeat

Example

```
flip = rand < 0.5
if flip
    x = randg + 2
else
    x = randn
end
```

This produces samples over the execution trace

e.g. `(True, 2.7),`

CAN WE DEVELOP GENERIC INFERENCE FOR ALL PPS?

Rejection sampling

1. Run the program with a fresh source of random numbers
2. If condition D is true, record H as a sample, else ignore the sample
3. Repeat

Example

```
flip = rand < 0.5
if flip
    x = randg + 2
else
    x = randn
end
```

```
»True
```

This produces samples over the execution trace

e.g. `(True, 2.7)`,

Can we develop generic inference for all PPs?

Rejection sampling

1. Run the program with a fresh source of random numbers
2. If condition D is true, record H as a sample, else ignore the sample
3. Repeat

Example

```
flip = rand < 0.5
if flip
    x = randg + 2
else
    x = randn
end
```

```
»True

»3.2
```

This produces samples over the execution trace

e.g. `(True, 2.7),`

CAN WE DEVELOP GENERIC INFERENCE FOR ALL PPS?

Rejection sampling

1. Run the program with a fresh source of random numbers
2. If condition D is true, record H as a sample, else ignore the sample
3. Repeat

Example

```
flip = rand < 0.5
if flip
    x = randg + 2
else
    x = randn
end
```

This produces samples over the execution trace

e.g. `(True, 2.7)`,

Can we develop generic inference for all PPs?

Rejection sampling

1. Run the program with a fresh source of random numbers
2. If condition D is true, record H as a sample, else ignore the sample
3. Repeat

Example

```
flip = rand < 0.5
if flip
    x = randg + 2
else
    x = randn
end
```

```
»True
```

This produces samples over the <u>execution trace</u>

e.g. `(True, 2.7)`,

Can we develop generic inference for all PPs?

Rejection sampling

1. Run the program with a fresh source of random numbers
2. If condition D is true, record H as a sample, else ignore the sample
3. Repeat

Example

```
flip = rand < 0.5
if flip
    x = randg + 2
else
    x = randn
end
```

```
»True

»2.1
```

This produces samples over the execution trace

e.g. `(True, 2.7), (True, 2.1),`

Can we develop generic inference for all PPs?

Rejection sampling

1. Run the program with a fresh source of random numbers
2. If condition *D* is true, record *H* as a sample, else ignore the sample
3. Repeat

Example

```
flip = rand < 0.5
if flip
    x = randg + 2
else
    x = randn
end
```

This produces samples over the execution trace

e.g. `(True, 2.7), (True, 2.1),`

Can we develop generic inference for all PPs?

Rejection sampling

1. Run the program with a fresh source of random numbers
2. If condition D is true, record H as a sample, else ignore the sample
3. Repeat

Example

```
flip = rand < 0.5
if flip
    x = randg + 2
else
    x = randn
end
```

»False

This produces samples over the execution trace

e.g. `(True, 2.7), (True, 2.1),`

Can we develop generic inference for all PPs?

Rejection sampling

1. Run the program with a fresh source of random numbers
2. If condition D is true, record H as a sample, else ignore the sample
3. Repeat

Example

```
flip = rand < 0.5
if flip
    x = randg + 2
else
    x = randn
end
```

```
»False
```

```
»-1.3
```

This produces samples over the execution trace

e.g. `(True, 2.7),(True, 2.1),`

CAN WE DEVELOP GENERIC INFERENCE FOR ALL PPS?

Rejection sampling

1. Run the program with a fresh source of random numbers
2. If condition D is true, record H as a sample, else ignore the sample
3. Repeat

Example

```
flip = rand < 0.5
if flip
    x = randg + 2
else
    x = randn
end
```

This produces samples over the execution trace

e.g. `(True, 2.7),(True, 2.1),`

CAN WE DEVELOP GENERIC INFERENCE FOR ALL PPS?

Rejection sampling

1. Run the program with a fresh source of random numbers
2. If condition D is true, record H as a sample, else ignore the sample
3. Repeat

Example

```
flip = rand < 0.5
if flip
    x = randg + 2
else
    x = randn
end
```

»False

This produces samples over the execution trace

e.g. `(True, 2.7),(True, 2.1),`

Can we develop generic inference for all PPs?

Rejection sampling

1. Run the program with a fresh source of random numbers
2. If condition D is true, record H as a sample, else ignore the sample
3. Repeat

Example

```
flip = rand < 0.5
if flip
    x = randg + 2
else
    x = randn
end
```

```
»False

»2.3
```

This produces samples over the execution trace

e.g. `(True, 2.7),(True, 2.1),(False, 2.3),...`

Can we be more efficient?

Metropolis-Hastings

1. Start with a trace
   - `(True, 2.3)`

Can we be more efficient?

Metropolis-Hastings

1. Start with a trace
   - `(True, 2.3)`
2. Change one random decision, discarding subsequent decisions
   - `(False,)`

Can we be more efficient?

Metropolis-Hastings

1. Start with a trace
   - `(True, 2.3)`
2. Change one random decision, discarding subsequent decisions
   - `(False,)`
3. Sample subsequent decisions
   - `(False, -0.9)`

Metropolis-Hastings

1. Start with a trace
   - `(True, 2.3)`
2. Change one random decision, discarding subsequent decisions
   - `(False,)`
3. Sample subsequent decisions
   - `(False, -0.9)`
4. Accept with appropriate (RJ)MCMC acceptance probability
   - Reject, does not satisfy observation (i.e. likelihood is zero)

CAN WE BE MORE EFFICIENT?

Metropolis-Hastings

1. Start with a trace
   - `(True, 2.3)`

Metropolis-Hastings

1. Start with a trace
   - `(True, 2.3)`
2. Change one random decision, discarding subsequent decisions
   - `(True, 2.9)`

Can we be more efficient?

Metropolis-Hastings

1. Start with a trace
   - `(True, 2.3)`
2. Change one random decision, discarding subsequent decisions
   - `(True, 2.9)`
3. Sample subsequent decisions
   - Nothing to do

Can we be more efficient?

Metropolis-Hastings

1. Start with a trace
   - `(True, 2.3)`
2. Change one random decision, discarding subsequent decisions
   - `(True, 2.9)`
3. Sample subsequent decisions
   - Nothing to do
4. Accept with appropriate (RJ)MCMC acceptance probability
   - Accept, maybe

PP VIA METROPOLIS-HASTINGS - NOTATION

Evaluating a program results in a sequence of random choices

$$
\begin{aligned}
x_1 &\sim p_{t_1}(x_1). \\
x_2 &\sim p_{t_2}(x_2|x_1). \\
x_3 &\sim p_{t_3}(x_3|x_2, x_1). \\
x_k &\sim p_{t_k}(x_k|\underbrace{x_1, \ldots, x_{k-1}}_{\text{execution trace}}).
\end{aligned}
$$

The density/probability of a particular evaluation is then

$$
p(x_1, \ldots, x_K) = \prod_{k=1}^{K} p_{t_k}(x_k|x_1, \ldots, x_{k-1}).
$$

We then perform MH over the the execution trace $x = (x_1, \ldots, x_K)$

MH OVER EXECUTION TRACES

1. Select a random decision in the execution trace x
   - e.g. x_k

2. Propose a new value
   - e.g. $x'_k \sim K_{t_k}(x'_k|x_k)$

3. Run the program to determine all subsequent choices $(x'_l : l > k)$, reusing current choices where possible

4. Propose moving from the state $(x_1, \ldots, x_K)$ to $(\underbrace{x_1, \ldots, x_{k-1}}_{\text{old choices}}, \underbrace{x'_k, \ldots, x'_{K'}}_{\text{new choices}})$

5. Accept the change with the appropriate MH acceptance probability

$$\frac{K_{t_k}(x_k|x'_k) \prod_{i=k}^{K'} p_{t'_i}(x'_i|x_1, \ldots, x_{k-1}, x'_k, \ldots, x'_{i-1})}{K_{t_k}(x'_k|x_k) \prod_{i=k}^{K} p_{t_i}(x_i|x_1, \ldots, x_{k-1}, x_k, \ldots, x_{i-1})}$$

Demo: Regression with an Interesting Prior

NONPARAMETRIC MODELS

- If we can sample from the prior of a nonparametric model using finite resources with probability 1, then we can perform inference automatically using the techniques described thus far

- We can sample from a number of nonparametric processes/models with finite resources (with probability 1) using a variety of techniques
  - Gaussian processes via marginalisation
  - Dirichlet processes via stick breaking
  - Indian Buffet processes via urn schemes

- Active research to produce finite sampling algorithms for other nonparametric processes (e.g. hierarchical beta processes, negative binomial process)

Example: Mixture of Gaussians

Generative model

$$
\begin{aligned}
(\mu_k)_{k=1\ldots K} &\overset{\text{iid}}{\sim} \mathcal{N}(0,1) \\
(\pi_k)_{k=1\ldots K} &\sim \mathrm{Dir}(\alpha/K) \\
\Theta &:= \sum_{k=1}^{K} \pi_k \delta_{\mu_k} \\
(\theta_n)_{n=1\ldots N} &\overset{\text{iid}}{\sim} \Theta \\
(x_i)_{n=1\ldots N} &\sim \mathcal{N}(\theta_n, 1)
\end{aligned}
$$

(Pseudo) MATLAB code

```
mu = randn(K,1)
pi = dirichlet(K, alpha/K)

for n = 1:N
  theta = mu(mnrnd(1,pi))
  x(n)  = theta + randn
end
```

Example: Infinite mixture of Gaussians

Limit of generative model is a DP

$$(\mu_k)_{k=1\dots K} \overset{\text{iid}}{\sim} \mathcal{N}(0,1)$$
$$(\pi_k)_{k=1\dots K} \sim \mathrm{Dir}(\alpha/K)$$
$$\Theta := \sum_{k=1}^{K} \pi_k \delta_{\mu_k} \underset{K\to\infty}{\to} \Theta \sim \mathrm{DP}(\alpha, \mathcal{N}(0,1))$$

Avoiding infinity

- Θ is now infinitely complex, and can only be represented approximately with finite resources
- However, we can sample a finite number of samples $(\theta_n)_{n=1\dots N}$ from some unknown Θ in finite time (with probability 1) using a stick-breaking algorithm

EXAMPLE: INFINITE MIXTURE OF GAUSSIANS

MATLAB stick breaking construction

```
sticks = [];
atoms = [];
for i = 1:n
  p = rand;
  while p > sum(sticks)
    sticks(end+1) = (1-sum(sticks)) * betarnd(1, alpha);
    atoms(end+1) \ = randn;
  end
  theta(i) = atoms(find(cumsum(sticks)>=p, 1, 'first'));
end
x = theta' + randn(n, 1);
```

DEMOS: NONPARAMETRIC MODELS

Advanced Automatic Inference

- Now that we have separated inference and model design, can use any inference algorithm.
- Free to develop inference algorithms independently of specific models.
- Once graphical models identified as a general class, many model-agnostic inference methods:
  - Belief Propagation
  - Pseudo-likelihood
  - Mean-field Variational
  - MCMC
- What generic inference algorithms can we implement for more expressive generative models?

ADVANCED AUTOMATIC INFERENCE: GIBBS

- BUGS: **B**ayesian inference **U**sing **G**ibbs **S**ampling
  - An early, limited form of automated inference in generative models.
  - Began in 1989 in the MRC Biostatistics Unit, Cambridge, UK.
  - A workhorse of applied statisticians. Also JAGS (open-source)

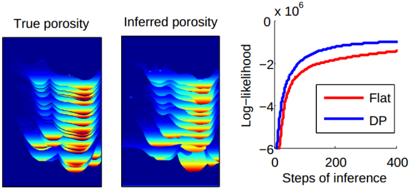
```
model{
  for( i in 1 : N ) {
    S[i] ~ dcat(pi[])
    mu[i] <- theta[S[i]]
    x[i] ~ dpois(mu[i])
    for (j in 1 : C) {
      SC[i, j] <- equals(j, S[i])}}
    # Precision Parameter
    alpha~ dgamma(0.1,0.1)
    # Constructive DPP
    p[1] <- r[1]
    for (j in 2 : C) {
      p[j] <- r[j] * (1 - r[j - 1]) * p[j -1 ] / r[j - 1]}
    p.sum <- sum(p[])
    for (j in 1:C){
      theta[j] ~ dgamma(A, B)
      r[j] ~ dbeta(1, alpha)
      # scaling to ensure sum to 1
      pi[j] <- p[j] / p.sum }
    # hierarchical prior on theta[i] or preset parameters
    A ~ dexp(0.1)     B ~dgamma(0.1, 0.1)
    # total clusters
    K <- sum(cl[])
    for (j in 1 : C) {
      sumSC[j] <- sum(SC[ , j])
      cl[j] <- step(sumSC[j] -1)}}

Data:
list(x=c(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,2,2,2,2,2,2,2,2,
2,3, 3, 3, 3, 4, 4, 5, 5, 5, 6, 6, 6,7,7,7,8,9,9,10, 10,
11, 11, 12, 12, 14, 15, 15, 17, 17, 22, 24, 34), N=101, C=10)
```

Advanced Automatic Inference: Metropolis-Hastings

- Bher, MIT-Church
  (Goodman, Mansinghka, Roy, Bonawitz and Tenenbaum, 2008)
  - (Automatic inference in Scheme)
- Stochastic Matlab
  - Lightweight Implementations of Probabilistic Programming Languages Via Transformational Compilation
    (Wingate, Stuhlmüller, Goodman, 2011)

A layer model of rock porosity

```
function dp_render_rock( data )
  num_layers = poissrnd();
  dp_sample_layer = dpmem( @sample_layer, 1.0 );
  for i=1:num_layers
    layer(i,:) = dp_sample_layer();
  end;
  rock_volume = render_layers( layers );
  data = rock_volume + randn( size(rock_volume) );
return;
```



Advanced Automatic Inference: HMC

- Automatic Differentiation in Church:
  Nonstandard Interpretations of Probabilistic Programs for Efficient Inference
  (Wingate, Goodman, Stuhlmuller, Siskind, 2012)
- Stan (Gelman et al)
  http://mc-stan.org/

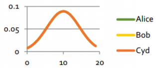
```
// Predict from Gaussian Process Logistic Regression
// Fixed covar function: eta_sq=1, rho_sq=1, sigma_sq=0.1

data {
  int<lower=1> N1;
  vector[N1] x1;
  int<lower=0,upper=1> z1[N1];
  int<lower=1> N2;
  vector[N2] x2;}
transformed data {
  int<lower=1> N;
  vector[N1+N2] x;
  vector[N1+N2] mu;
  cov_matrix[N1+N2] Sigma;
  N <- N1 + N2;
  for (n in 1:N1) x[n] <- x1[n];
  for (n in 1:N2) x[N1 + n] <- x2[n];
  for (i in 1:N) mu[i] <- 0;
  for (i in 1:N)
    for (j in 1:N)
      Sigma[i,j] <- exp(-pow(x[i] - x[j],2))
                    + if_else(i==j, 0.1, 0.0);}
parameters {
  vector[N1] y1;
  vector[N2] y2;}
model {
  vector[N] y;
  for (n in 1:N1) y[n] <- y1[n];
  for (n in 1:N2) y[N1 + n] <- y2[n];
```

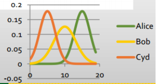
Advanced Automatic Inference: Expectation Propagation

- Infer.NET (Minka, Winn, Guiver, Knowles, 2012)
  - EP in graphical models:
  - Now works in functional language F#:

TrueSkill in Fun

```
// prior distributions, the hypothesis
let skill() = sample (Gaussian(10.0,20.0))
let Alice,Bob,Cyd = skill(),skill(),skill()
// observe the evidence
let performance player = sample (Gaussian(player,1.0))
observe (performance Alice > performance Bob) //Alice beats Bob
observe (performance Bob > performance Cyd) //Bob beats Cyd
observe (performance Alice > performance Cyd) //Alice beats Cyd
// return the skills
Alice,Bob,Cyd
```





MICROSOFT RESEARCH

ADVANCED AUTOMATIC INFERENCE: VARIATIONAL

- Infer.NET has it too.
- Automated Variational Inference in Probabilistic Programming (Wingate, Weber, 2012)

Probabilistic program A

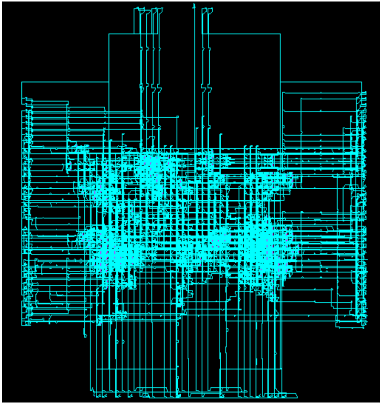
```
M = normal();
if M>1
  mu = complex_deterministic_func( M );
  X = normal( mu );
else
  X = rand();
end;
```

Mean-Field variational program A

```
M = normal( θ_1 );
if M>1
  mu = complex_deterministic_func( M );
  X = normal( θ_3 );
else
  X = rand(θ_4, θ_5);
end;
```

- Learning phase: Forward sample, then stochastically update θs to minimize expected KL from true distribution.
- Dependency of variatonal dist on control logic remains.

- Natively Probabilistic Computation (Mansinghka, 2009)
- Lyric Semiconductor? (Error correcting codes)
- Main idea: If we know we're going to be sampling, some errors in computation can be OK.
  - Samplers can be made robust to computational error.
  - Run at low voltage on (cheap?) FPGA
- Compile from generative model to FPGA (9x9 Ising model sampler):



Automated Modeling

- Automated inference helpful for human modelers.
- Essential for machine-generated models
  - For example, approximate Solomonoff induction.
- Essential for more general version of automated Bayesian statistician.

Theoretical Directions

Inference in stochastic programs opens up a new branch of computer science, new generalizations of computability:

- "Computable de Finetti measures"
  (Freer, Roy, 2012)
- "Noncomputable conditional distributions"
  (Ackerman, Freer, Roy, 2011)
- "Computable exchangeable sequences have computable de Finetti measures"
  (Freer, Roy, 2009
- "On the computability and complexity of Bayesian reasoning"
  (Roy, 2012)

Main takeaways:

- No general computable algorithm exists for conditioning
- Representational choices important
  - i.e. stick-breaking vs CRP latent representation changes computability

Compiler Development

- 1950s: Ask a programmer to implement an algorithm efficiently: They'll write it on their own in assembly.
  - No good compilers; problem-dependent optimizations that only human expert could see.

- 1970s: Novice programmers use high-level languages and let compiler work out details, experts still write assembly.
  - Experts still write custom assembly when speed critical.

- 2000s: On most problems, even experts can't write faster assembly than optimizing compilers.
  - can automatically profile (JIT).
  - can take advantage of paralellization, complicated hardware, make appropriate choices w.r.t. caching.
  - Compilers embody decades of compiler research

Inference Methods in the Future

- 2010: Ask a grad student to implement inference efficiently: They'll write it on their own.
  - No good automatic inference engines; problem-dependent optimizations that only human expert can see.
- 2015: Novice grad students use automatic inference engines and let compiler work out details, experts still write their own inference.
  - Experts still write custom inference when speed critical.
- 2020: On most problems, even experts can't write faster inference than mature automatic inference engines.
  - Can use paralellization, sophisticated hardware
  - Can automatically choose appropriate methods (meta-reasoning?).
  - Inference engines will embody 1 decade (!) of PP research.

How to write a Bayesian modeling paper: 2010

1. Write down a generative model in an afternoon

2. Get 2 grad students to implement inference for a month

3. Use technical details of inference to pad half of the paper

How to write a Bayesian modeling paper: 2020

1. Write down a generative model in an afternoon
2. ~~Get 2 grad students to implement inference for a month~~
   Run automatic inference engine.
3. Use technical details of inference to pad half of the paper

1. Write down a generative model in an afternoon

2. ~~Get 2 grad students to implement inference for a month~~
   Run automatic inference engine.

3. ~~Use technical details of inference to pad half of the paper~~
   Discuss strengths and weaknesses of the model in the extra 4 pages.

HOW TO WRITE A BAYESIAN MODELING PAPER: 2020

1. Write down a generative model in an afternoon

2. ~~Get 2 grad students to implement inference for a month~~
   Run automatic inference engine.

3. ~~Use technical details of inference to pad half of the paper~~
   Discuss strengths and weaknesses of the model in the extra 4 pages.

Thanks!