Bean Machine

A Declarative Probabilistic Programming Language For Efficient Programmable Inference

Nazanin Tehrani, Nimar Arora, Yucen Li, Kinjal Shah, David Noursi, Michael Tingley, Narjes Torabi, Sepehr Masouleh, Eric Lippert, and Erik Meijer

1. Imperative (used by Church, Pyro, Gen, Turning and many others)
2. Declarative (used by BLOG and Bean Machine)

Declarative	Imperative
1. Random variables are defined with a separate code blocks.	1. Random variables are defined in one main code block.
2. Model does not provide an order to draw samples from each random variable.	2. Model provides an order to draw samples from each random variable.
3. Directed Acyclic Graph (DAG) is explicit.	3. Directed Acyclic Graph (DAG) is implicit.

Single-Site Inference

Inference

- Markov chain Monte Carlo
- Single-Site Metropolis Hastings
- Performed over worlds

Single-Site Metropolis Hastings

for each iteration of inference:
  for each unobserved random variable X:
    perform an MH update:
    propose a new value x' for X using proposal Q
    update the world σ to σ'
    accept / reject using MH rule

Metropolis Hastings acceptance probability:

$$\min\left(1, \frac{P(\sigma')Q_x(\sigma_X \mid \sigma')}{P(\sigma)Q_x(\sigma'_X \mid \sigma)}\right)$$

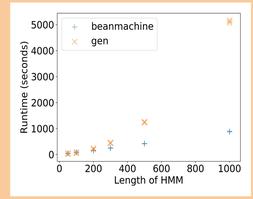
Key Features

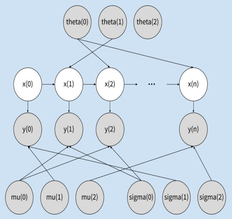
- $P(\sigma')/P(\sigma)$ can be computed locally
  - the only update in probability is from X and its immediate children
- runtime complexity of one MH update for X is proportional to the size of X's Markov blanket
- second-order gradient-based inference methods are tractable

Inference Example: HMM

Bean Machine
- explicit DAG
- runtime of inference per variable is proportional to size of its Markov blanket for all models
- runtime per variable is constant for HMM
- linear runtime complexity for HMM

Imperative Languages
- no explicit dependency structure
- re-executes trace to resample one variable for all models
- runtime per variable is linear for HMM
- quadratic runtime complexity for HMM





$$\mu_k \sim \text{Normal}(\alpha, \beta)$$
$$\sigma_k \sim \text{Gamma}(\nu, \rho)$$
$$\theta_k \sim \text{Dirichlet}(\kappa)$$
$$x_i \sim \begin{cases} \text{Categorical}(init) & \text{if } i = 0 \\ \text{Categorical}(\theta_{x_{i-1}}) & \text{if } i > 0 \end{cases}$$
$$y_i \sim \text{Normal}(\mu_{x_i}, \sigma_{x_i})$$

Hidden Markov Model (HMM)

```
@random_variable
def mu(k):
    return Normal(alpha, beta)

@random_variable
def sigma(k):
    return Gamma(nu, rho)

@random_variable
def theta(k):
    return Dirichlet(kappa)

@random_variable
def x(i):
    if i == 0:
        return Categorical(init)
    return Categorical(theta(x(i - 1))

@random_variable
def y(i):
    return Normal(mu(x(i)), sigma(x(i)))
```

```
mu, sigma, theta, x, y = {}, {}, {}, {}, {}

With Model():
    for i in range(K):
        mu[k] = Normal(f"mu[{k}]", alpha, beta)
        sigma[k] = Gamma(f"sigma[{k}]", nu, rho)
        theta[k] = Dirichlet(f"theta[{k}]", kappa)

x[0] = Categorical("x[0]", init)
    for i in range(1, N):
        x[i] = Categorical(f"x[{i}], theta[x[i-1]])
        y[i] = Normal(
                f"y[{i}]",
                mu[x[i]],
                sigma[x[i]],
                observed =data[i]
            )
```

FACEBOOK

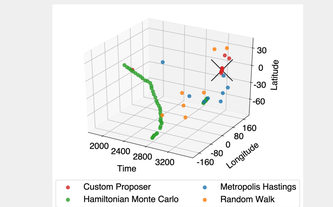
Composition Example: Annotation Model

$$\pi \sim \text{Dirichlet}\left(\frac{1}{C}, \ldots, \frac{1}{C}\right)$$
$$z_i \sim \text{Categorical}(\pi)$$
$$\theta_{lm} \sim \text{Dirichlet}(\alpha_m)$$
$$y_{li} \sim \text{Categorical}(\theta_{lz_i})$$

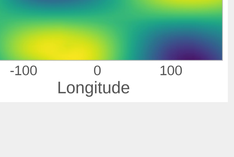
```
mh = CompositionalInference({
  pi: SingleSiteNewtonianMonteCarlo(),
  theta: SingleSiteNewtonianMonteCarlo(),
  z: SingleSiteUniformMetropolisHastings()
})
```

Composition

- Each variable can have its own proposer
- Discrete variables do not need to be integrated out



Method	Effective Sample Size
Single-Site	622
Global	487



FACEBOOK

Blocking

- Single-site may not be suitable for models with highly correlated variables
- Block inference allows highly correlated variables to be updated together

Metropolis Hastings For [x, mu, sigma]

propose new x' for x and update the world
add x and x' Markov blanket to *Blanket*
for mu in *Blanket*:
  propose new mu' for mu and update the world
  add mu and mu' Markov blanket to *Blanket*
for sigma in *Blanket*:
  propose new sigma' for sigma and update the world
  add sigma and sigma' Markov blanket to *Blanket*
accept / reject using MH rule

```
mh = CompositionalInference()
mh.add_sequential_proposer([x, mu, sigma])
queries = [x(N-1)]
        + [theta(k) for k in range(K)]
        + [mu(k) for k in range(K)]
        + [sigma(k) for k in range(K)]
obs = {y(i): data[i] for i in range(N)}
samples = mh.infer(queries, obs)
```

Results

100 iterations on an HMM of length 200

Method	Hidden States	Effective Sample Size
Block	25	109
Non-Block	25	89
Block	50	93
Non-Block	50	30

FACEBOOK

Seismic Event Model

Custom Proposers

- **Domain Knowledge:** invert the detection attributes to find the most likely attributes of an event
- **Custom Proposer Idea:** use a Gaussian mixture model proposer around the inverted attributes

```
@random_variable
def event_attr():
  return SeismicEventPrior()

@random_variable
def is_detected(station):
  prob = calculate_prob(station, event_attr())
  return Bernoulli(prob)

@random_variable
def det_attr(station):
  det_loc = calculate_dec(station, event_attr())
  return Laplace(det_loc, scale)
```

```
class SeismicProposer(Proposer):

  # return proposed value and probability of
proposing
  def propose(self, variable, curr_world):
    det_attrs = [child.value for child in
variable.children
      if child.dependency_fn = det_attr]
    event_attrs = [seismic_invert(det) for det in
det_attrs]
    self.gmm = construct_gmm(event_attrs)
    new_value = self.gmm.sample()
    return new_value, self.gmm.log_prob(new_value)

  # return probability of proposing original value
  def post_process(self, variable, new_world):
    return self.gmm.log_prob(variable.value)
```


FACEBOOK