Compiling Stan to Generative Probabilistic Languages

Guillaume Baudart (IBM), Javier Burroni (UMass Amherst), Martin Hirzel (IBM), Kiran Kate (IBM), Louis Mandel (IBM), Avraham Shinnar (IBM)

Is it possible to compile any Stan program to a generative probabilistic program?

Stan

- Declarative style
- Very large community

Generative PPLs

- Many instances: WebPPL, Pyro, ...
- General purpose programming language with `sample`, `observe`, and `factor`

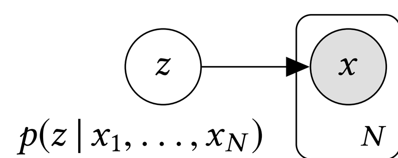
Contributions

- Comprehensive compilation scheme
- Correctness proof
- A new Pyro backend for Stanc3
- Extending Stan with explicit variational guides and neural networks

Benefits

- Stan users have access to a new backend with different inference engines and new features
- PPLs developers have access to a large number of models

Example




```
data { int N; int<lower=0,upper=1> x[N]; }
parameters { real<lower=0,upper=1> z; }
model { z ~ beta(1, 1);
        for (i in 1:N) x[i] ~ bernoulli(z); }
```

References

- Baudart, Burroni, Hirzel, Kate, Mandel, Shinnar. Extending Stan for Deep Probabilistic Programming. arxiv:1810.00873.
- Bingham, et al. Pyro: Deep Universal Probabilistic Programming. JMLR 2019.
- Carpenter, et al. Stan: A probabilistic programming language.JSS 2017.
- Gorinova, Gordon, and Sutton. Probabilistic programming with densities in SlicStan. POPL 2019.
- Staton. Commutative Semantics for Probabilistic Programming. ESOP 2017.

Generative compilation

- ~ on parameters: sampling
- ~ on data: conditioning
- *Cannot handle all Stan models!*

```
def model(N, x):
  z = sample(beta(1.,1.))
  for i in range(0, N):
    observe(bernoulli(z), x[i])
  return z
```

Stan features: example, prevalence and compilation

Feature	%	Example	Compilation
Left expression	7.7	`sum(phi) ~ normal(0, 0.001*N);`	`observe(Normal(0.,0.001*N), sum(phi))`
Multiple updates	3.9	`phi_y ~ normal(0,sigma_py);` `phi_y ~ normal(0,sigma_pt)`	`observe(Normal(0.,sigma_py), phi_y);` `observe(Normal(0.,sigma_pt), phi_y)`
Implicit prior	60.7	`real alpha0;` `/* missing 'alpha0 ~ ...' */`	`alpha0 = sample(ImproperUniform())`
Target update	16.3	`target += -0.5 * dot_self(` `phi[node1] - phi[node2]);`	`factor(-0.5 * dot_self(` `phi[node1] - phi[node2]))`

Comprehensive compilation

- All ~ statements are conditioning
- Parameters are initialized with uniform priors

```
def model(N, x):
  z = sample(uniform(0.,1.))
  observe(beta(1.,1.), z)
  for i in range(0, N):
    observe(bernoulli(z), x[i])
  return z
```

Correctness proof

- The semantics of Stan is based on an extension of [Gorinova et al. 2018]
- The semantics of the generative PPL is based on [Staton 2017]
- The compilation is formalized as a continuation passing style transformation

$$C(p) = \mathcal{P}_{\mathcal{S}_{\texttt{return}(params(p))}(model(p))}\,(params(p))$$

$$\mathcal{P}_k\,(params(p)) \;=\; \texttt{let}\; x_1 = D_1\; \texttt{in} \ldots \texttt{let}\; x_n = D_n\; \texttt{in}\; k$$

- Proof:

$$\begin{aligned}
\{\!\!\{C(p)\}\!\!\}_D &\propto \lambda U. \int_U \{\!\!\{\mathcal{S}_{\texttt{return}(())}\,(model(p))\}\!\!\}_{D,\theta}(\{()\})\,d\theta \\
&= \lambda U. \int_U \exp([\![model(p)]\!]_{D,\theta}\,(\texttt{target})) \times \{\!\!\{\texttt{return}(())\}\!\!\}(\{()\})\,d\theta \\
&= \lambda U. \int_U \exp([\![model(p)]\!]_{D,\theta}\,(\texttt{target}))\,d\theta \\
&= \{\!\!\{p\}\!\!\}_D
\end{aligned}$$

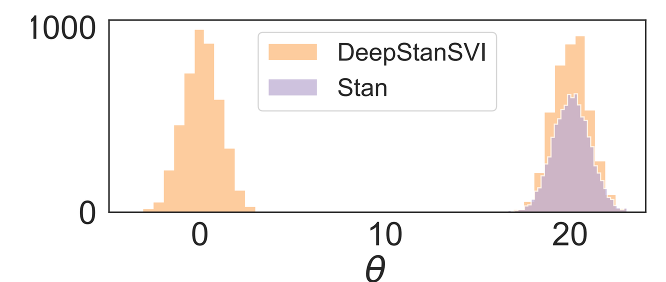
Evaluation

- Compiler implemented as a fork of Stanc3
- Tested based on the 97 Stan models provided by PosteriorDB
- 96 models are compiling (the 1 error also fails to compile with Stan 3)
- Inference runs on 77 models
- Yield distributions similar to Stan on 8 classic models

Extensions: SVI guides and NN

Stochastic Variational Inference (SVI)

- Explicit guides to specify the family of target distributions




```
parameters {
  real cluster;
  real theta; }
model {
  real mu;
  cluster ~ normal(0, 1);
  if (cluster > 0) mu = 20;
  else mu = 0;
  theta ~ normal(mu, 1); }
guide parameters {
  real mc;
  real m1; real m2;
  real ls1; real ls2; }
guide {
  cluster ~ normal(mc, 1);
  if (cluster > 0) theta ~ normal(m1, exp(ls1));
  else theta ~ normal(m2, exp(ls2)); }
```

Neural Networks

- Neural networks defined in PyTorch
- Deep probabilistic models: models using deep neural networks
- Bayesian Networks: parameters of the network are random variables

```
networks {
  Decoder decoder; Encoder encoder; }
data {
  int nz;
  int<lower=0, upper=1> x[28, 28]; }
parameters { real z[*]; }
model {
  real mu[_, _];
  z ~ normal(0, 1);
  mu = decoder(z);
  x ~ bernoulli(mu); }
guide {
  real encoded[2, nz] = encoder(x);
  real mu_z[*] = encoded[1];
  real sigma_z[*] = encoded[2];
  z ~ normal(mu_z, sigma_z); }
```