Encapsulating models and approximate inference programs in probabilistic modules

Marco F. Cusumano-Towner and Vikash K. Mansinghka

Probabilistic modules: an interface for stochastic computations with auxiliary random choices

An existing interface for stochastic computations that requires we can compute the density on outputs:

\[ z \leftarrow p.\text{SIMULATE}(x) \text{ for } z \sim p(z|x) \]

\[ p(z|x) \leftarrow p.\text{DENSITY}(x, z) \]

The new probabilistic modules interface, a generalization of the above interface designed for stochastic computations with uncollapsed auxiliary random choices \( \mathcal{U} \):

\[
\begin{align*}
& \quad \left( z, \frac{p(u, z|x)}{q(u|x, z)} \right) \leftarrow (p, q) \text{SIMULATE}(x) \text{ for } u, z \sim \frac{p(u, z|x)}{q(u|x, z)} \\
& \quad \frac{p(u, z|x)}{q(u|x, z)} \leftarrow (p, q) \text{REGENERATE}(z, x) \text{ for } u|x, z \sim q(u|x, z) \\
& \quad \text{Simulate and regenerate both return estimates of the output density. The accuracy of the estimates depends on how well the regenerator } q(u|x, z) \text{ approximates } p(u|x, z). \\
\end{align*}
\]

Example: Stochastic inverse regeneration

For a fragment of a Bayesian network, a 'stochastic inverse' of [Stuhlmüller et al. 2013] can be learned and used as the regenerator.

Example: Resimulation regeneration

If the density \( p(z|x) \) can be computed, a generic (generally poor) choice of regenerator involves running the sampler from scratch, ignoring the output.

Example: Stochastic inference regeneration

\[
\begin{align*}
1 & : \text{Construct probabilistic module } (p’, q’) \text{ with auxiliary variables } u := (u, v) \\
2 & : \text{procedure } (p’, q’) \text{SIMULATE}(x) \text{ for } u, v \sim p’(u, v) \\
3 & : \quad v \sim p’(z’\mid x) \\
4 & : \quad u \sim v’(z, u) \\
5 & : \quad \text{Run meta-inference sampler, producing output } w \\
6 & : \quad \text{return } (w, v) \\
7 & : \text{end procedure} \\
8 & : \text{procedure } (p’, q’) \text{REGENERATE}(z, x) \text{ for } u|x, z \sim q(u|x, z) \\
9 & : \quad \text{Run inference sampler, producing output } w \\
10 & : \quad \text{return } (w, u) \\
11 & : \text{end procedure} \\
12 & \text{end} \\
\end{align*}
\]

We can bundle together a stochastic inference program for inference in a sampler by extending the auxiliary data-flow during simulate. Module A uses a 'stochastic inverse' regenerator and module B uses an SMC stochastic inference regenerator.

Example: Variational autoencoder as a probabilistic module

\[
\begin{align*}
\text{requires} & \quad \text{Autorencoder with generative network } u, z \sim p(\cdot|x) \text{ and recognition network } u \sim q(z|x) \\
1 & : \text{procedure } (p_\theta, q_\theta) \text{SIMULATE}(x) \text{ for } u \sim p_\theta(z|x) \\
2 & : \quad \text{Run generative network} \\
3 & : \quad \text{return } (u, q(z|x))(u/q(z|x)) \\
4 & : \text{end procedure} \\
5 & : \text{procedure } (p_\theta, q_\theta) \text{REGENERATE}(z, x) \text{ for } u\mid z \sim q_\theta(u|z) \\
6 & : \quad \text{Run recognition network} \\
7 & : \quad \text{return } (u, q(z|x))(u/q(z|x)) \\
8 & : \text{end procedure} \\
\end{align*}
\]

A trained variational autoencoder naturally implements the probabilistic modules interface, and serves as an example of a sampler that is optimized alongside its regenerator.

Example: Nonparametric Bayesian regeneration

\[
\begin{align*}
\text{requires} & \quad \text{Bayesian network } p(x|\theta) \text{ and stick-breaking prior } q(\theta_1) \\
1 & : \text{procedure } \text{SIMULATE}(x) \text{ for } \theta \sim \text{DBP} \\
2 & : \quad \text{Samples from regeneration with } \text{DBP} \text{ non-parametric model} \\
3 & : \quad \text{Inferring } \phi = \text{g}(\theta) \\
4 & : \quad \text{Generating samples from } p(x|\phi) \\
5 & : \quad \text{return } \phi \\
6 & : \text{end procedure} \\
\end{align*}
\]

A generic adapter for improving regeneration accuracy

\[
\begin{align*}
\text{requires} & \quad \text{Probabilistic module } (p, q), \text{Number of regenerations } K \geq 1 \\
1 & : \text{Construct probabilistic module } (p, q') \\
2 & : \text{procedure } (p, q') \text{SIMULATE}(x) \text{ for } u \sim q'(u|z, x) \\
3 & : \quad v \sim q'(z|x) \\
4 & : \quad \text{Compute } p(u|x, z) \\
5 & : \quad q'(u|x, z) \quad \text{for } u \sim p(u|x, z) \\
6 & : \quad \text{return } (u, v) \\
7 & : \text{end for} \\
8 & : \text{end procedure} \\
9 & : \text{procedure } (p, q') \text{REGENERATE}(z, x) \text{ for } u|x, z \sim q(u|x, z) \\
10 & : \quad u \sim q(u|x, z) \\
11 & : \quad \text{return } u \\
12 & : \text{end procedure} \\
\end{align*}
\]

Given a probabilistic module we can construct a new probabilistic module with the same simulation distribution but more accurate regeneration, by averaging the weights returned by K independent regenerations of the original module. The simulator of the new module also runs K - 1 regenerations alongside the primary simulation.

Encapsulating latent variable models in a probabilistic module

Encapsulating a stochastic computation such as a simulator for a fragment of a generative probabilistic model with an accurate regenerator in a module serves to ‘approximately collapse out’ the latent variables in the computation. As the accuracy of the regenerator increases, the model behaves more like a collapsed computation with an available density. The declarative semantics of the module are identical to that of the collapsed computation (g).

Single-site Metropolis-Hastings in a network of modules

A network of two probabilistic modules showing the data-flow during simulate. Module A uses a ‘stochastic inverse’ regenerator and module B uses an SMC stochastic inference regenerator.

The output of module A (orange) is proposed and regeneration is run in both modules to compute the acceptance probability in the MH algorithm.