Towards Inference Amortization for BUGS models: BUGS to Anglican compilation

Adam Goliński, Frank Wood
Department of Engineering Science
University of Oxford
{adamg,fwood}@robots.ox.ac.uk

Outline

- **Probabilistic Programming Languages** (PPL) are a special class of programming languages which allow users to specify probabilistic models and run inference on them, i.e. find $p(x|y)$.
  - $x$ are latents and $y$ are observed variables
- BUGS is a popular probabilistic programming language allowing to describe graphical models
- **Inference amortization** is a technique that greatly reduces the computational cost of run-time inference by training a neural network approximating the posterior distribution $q(x|y; \phi) \sim p(x|y)$ ahead of the time of the system operation.
  - $\phi$ are the learnt parameters of the neural network
- Anglican is a universal, research-oriented PPL which implements some of the cutting-edge inference techniques including inference amortization.
- To enable BUGS models to use inference amortization we have created a compiler translating models from BUGS to Anglican

Next steps

- completing the translation of the entire feature set of the BUGS language
- application and further improvement of the inference amortization approach which takes advantage of the structure of the forward graphical model [3] to automate the design of the neural network and is perfectly suited for the class of models expressible in BUGS

Pump failure model

Hierarchical model for failure rates of power plant pumps

$$p(x, y) = \alpha \sim \text{Exponential}(1.0) \beta \sim \text{Gamma}(0.1, 1.0) \lambda_n \sim \text{Gamma}(\alpha, \beta) \quad y_n \sim \text{Poisson}(\lambda_n t_n)$$

**Figure 1.** Forward graphical model [3]

$$x = \{\alpha, \beta\} \cup \{\lambda_n\}_N \quad y = \{t_n, y_n\}_N \quad \lambda_n \text{ rate of failure for pump } n \quad y_n \text{ number of failures for pump } n \quad t_n \text{ length of operation time for pump } n$$

Inference amortization

**Compilation**

- Training data $\{x^{(m)}, y^{(m)}\}$
- Probabilistic program $p(x, y)$
- NN architecture

**Inference**

- Test data $y$
- Test $\alpha$
- Inference $q(x|y; \phi)$
- Learning $D_{KL}(p(x|y)||q(x|y; \phi))$
- Training

Expensive / slow

Cheap / fast

**Figure 2.** Inference amortization framework [2]

SIS stands for Sequential Importance Sampling

**Figure 3.** Inverted graphical model [3]

**Figure 4.** Inference network with MADE-like neural networks [3]

References


**Figure 5.** Masked Autoencoder for Distribution Estimation [1]